Technische Universität München
Institut für Informatik

Planung kollisionsfreier Bewegungen für Manipulatoren durch Kombination von zielgerichteter Suche und zufallsgesteuerter Zwischenzielerzeugung

Bernhard Glavina

Vollständiger Abdruck der von der Fakultät für Mathematik und Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. O. Giering
Prüfer der Dissertation:
1. Univ.-Prof. Dr. H.-J. Siegert
2. Univ.-Prof. Dr. Chr. Zenger

Die Dissertation wurde am 16.1.1991 bei der Technischen Universität München eingereicht und durch die Fakultät für Mathematik und Informatik am 26.2.1991 angenommen.

Für Sabine und Arista

Inhaltsverzeichnis

Zusammenfassung 4

1 Einleitung und Überblick 5
- 1.1 Bedeutung der Bewegungsplanung 5
- 1.2 Motivation 5
- 1.3 Ziel der Arbeit 6
- 1.4 Überblick über den Inhalt der folgenden Kapitel 7
- 1.5 Ergebnisse der Arbeit 8

2 Suchaufgabe 10
- 2.1 Abstrakte Formulierung der Suchaufgabe 10
- 2.2 Stand der Technik 17
- 2.3 Folgerungen und ZZ-Suche 19

3 ZZ-Suchalgorithmus für n Freiheitsgrade 21
- 3.1 Zielsetzung und Vorgehensweise 21
- 3.2 Zielgerichtete Suche 23
- 3.3 Gleitschritt 26
- 3.4 Zwischenzielerzeugung 32
- 3.5 Stellungsgraph 33

4 Kollisionstest 35
- 4.1 Anforderungen 35
- 4.2 Umweltrepräsentation 35
- 4.3 Algorithmus 36

5 Ergebnisse 40
- 5.1 Implementierung 40
- 5.2 Beispielläufe mit repräsentativen Aufgaben 41
- 5.3 Vergleich mit Ergebnissen anderer Autoren 52
- 5.4 Zusammenfassende Bewertung 54

6 Ausblick 56

Anhang 58

Literatur 62

Zusammenfassung

Es wird ein neues heuristisches Verfahren zur Wegesuche in hochdimensionalen Konfigurationsräumen vorgestellt. Dieser Algorithmus ist in der Lage, schnell kollisionsfreie Bewegungen für Manipulatoren mit sechs Freiheitsgraden in dreidimensionalen Umwelten zu planen. Die praktische Brauchbarkeit wird in realitätsnahen Beispielen gezeigt.

Aufgabenorientierte Programmierung erleichtert dem menschlichen Programmierer die Steuerung bei Manipulatoren. Die Verbesserung gegenüber herkömmlichen Robotersprachen liegt in den einfacheren und kürzeren Programmen, in der schnelleren Programmerstellung und in der erhöhten Sicherheit. Beispielsweise muß in den aufgabenorientierten Sprachen nicht mehr angegeben werden, auf welchem Weg der Arm zu bewegen ist, sondern einfach nur noch das Ziel der Bewegung. Die Details der Armbewegung werden automatisch so erzeugt, daß Kollisionen mit den Umweltobjekten nicht auftreten — und der Programmierer kann seine Aufmerksamkeit auf das Wesentliche der Aufgabe konzentrieren.

Die bekannten Ansätze zur Lösung des Wegesuchproblems haben entweder sehr lange Rechen- oder Vorverarbeitungszeiten und sind damit praktisch nicht einsetzbar, oder sie schränken die Anzahl der Freiheitsgrade des Manipulators künstlich stark ein und können dann nur noch sehr spezielle Bewegungen ausführen. Beide Nachteile sind in vielen realen Anwendungen nicht tragbar.

Das hier vorgestellte Verfahren zur automatischen Planung von kollisionsfreien Bewegungen erbringt bei sechs Freiheitsgraden in dreidimensionalen, realistischen Umgebungen Lösungen innerhalb kurzer Zeit (typisch: unter einer Minute auf einer modernen Workstation). Es vereint eine einfache zielgerichtete Suche — direkt auf das Ziel zu, falls nötig Entlanggleiten an einem Hindernis — mit einer übergeordneten zufallsgesteuerten Zwischenzielerzeugung zur Aufteilung der Gesamtaufgabe in mehrere Teilaufgaben, falls die zielgerichtete Suche allein nicht erfolgreich ist.

Bei beiden Komponenten ist wesentlich, daß die Rechenzeit nicht exponentiell von der Anzahl der Freiheitsgrade abhängt, was durch Verzicht auf eine explizite Darstellung des Suchraums und der Konfigurationsraum-Hindernisse erreicht wird. Jeder Test, ob eine gewisse Konfiguration erlaubt ist oder eine Kollision ergibt, wird durch geometrische Schnittbildung berechnet, die auf einer hierarchischen Umweltrepräsentation sehr effizient ausgeführt werden kann.

Das Verfahren wurde implementiert und seine Leistungsfähigkeit durch mehrere repräsentative Beispielaufgaben unter Beweis gestellt.

Kapitel 1

Einleitung und Überblick

1.1 Bedeutung der Bewegungsplanung

Aufgabenorientierte Programmierung erlaubt einem menschlichen Programmierer, einen Manipulator einfacher, schneller und sicherer zu steuern, als dies mit herkömmlichen Roboterprogrammiersprachen möglich ist.

Die gerätespezifischen und geometrischen Details der Abarbeitung des Roboterprogramms brauchen nicht im Programm angegeben werden, sondern werden von der Steuerung automatisch geeignet ergänzt, etwa aus einer globalen Wissensbasis. Das aufgabenorientierte Programm enthält nur noch Angaben, die die Lösung der betrachteten Aufgabe auf hohem abstraktem Niveau beschreiben, es ist dementsprechend kürzer, übersichtlicher und weniger gerätespezifisch als Programme, die in üblichen Robotersprachen geschrieben sind [32]. Zum Beispiel braucht man, um den Roboterarm an eine bestimmte Stelle zu bewegen, nur mehr diese Stellung anzugeben; eine Vorgabe von mehr oder weniger vielen Zwischenpunkten zur Vermeidung von Kollisionen auf dem Weg zur Zielstellung entfällt.

Wesentlicher Bestandteil eines jeden aufgabenorientierten Programmiersystems ist deshalb die automatische Planung von kollisionsfreien Bewegungen für den Arm des Manipulators.

Der Schritt von den heutigen Robotersprachen hin zu aufgabenorientierten Sprachen ist vergleichbar mit der Ablösung der Maschinensprachen durch höhere Programmiersprachen wie Algol, Pascal oder C, und wird die Roboterprogrammierung und -anwendung stark beeinflussen. Die Entwicklung solcher fortschrittlicher Steuerungen ist aber bisher über das Laborstadium noch nicht hinausgekommen.

1.2 Motivation

Beim üblichen Bestimmen einer Manipulatorbewegung durch Teach-In werden nur einzelne Punkte der Bahn von Hand angefahren und dann abgespeichert. Die Kollisionsicherheit hängt bei dieser Methode davon ab, wie gut der Programmierer die zwischen den Einzelpunkten interpolierende Bewegung des Roboters im Kopf abschätzen kann.

Solche Programme sind nicht nur vom verwendeten Robotertyp abhängig, sondern auch von der Lage, Größe und Anzahl der Hindernisse im Arbeitsraum. Durch Teach-In festgelegte Bewegungen beziehen sich auf feste Koordinaten und müssen bei Änderungen der Umwelt erneut auf Kollisionsfreiheit überprüft werden, oft auch neu festgelegt werden, was zeitaufwendig ist.

Dagegen können die Bewegungsanweisungen in einem aufgabenorientierten Programm unverändert bleiben, auch wenn die Umwelt variiert wird. Unter der Voraussetzung, daß zwei verschiedene Roboter die gleiche aufgabenorientierte Sprache verstehen, kann sogar der Roboter ausgetauscht werden, ohne daß eine Programmänderung nötig wird.

Mit der Zunahme der Roboteranwendungen, etwa in der flexiblen Fertigung, erlangen die schnelle Erstellung und die Kollisionssicherheit eines Roboterprogramms immer mehr Bedeutung. Eine maschinelle Unterstützung des Programmierers ist hier sehr wünschenswert, aber solche Systeme sind kommerziell noch nicht erhältlich.

1.3 Ziel der Arbeit

Trotz der Bedeutung der Bewegungsplanung für zukünftige aufgabenorientierte Programmiersysteme machen die bisher veröffentlichten Arbeiten zu diesem Problemkreis Voraussetzungen, die einen Einsatz in allgemeinen praktischen Anwendungen nicht erlauben. Typische Einschränkungen sind beispielsweise die Modellierung der Manipulatorarme als dünne Linien [3], Rechenzeiten im Stundenbereich [9, 38] oder die Bedingung, daß die Hand und die Last des Roboters nur ein vernachlässigbares Volumen haben müssen [21, 32, 13].

Ziel dieser Arbeit ist die Beschreibung eines neuen Verfahrens zur automatischen Bewegungsplanung für Manipulatorarme. Besonderer Wert wird darauf gelegt, daß allgemein in praktischen Anwendungen auftretende Probleme ohne die oben genannten Einschränkungen und schnell gelöst werden. Der Schwerpunkt ist die schnelle Erzeugung einer brauchbaren Bewegung, im Gegensatz zur Erzeugung einer geometrisch oder dynamisch optimalen Bewegung.

Der beschriebene Algorithmus zur schnellen Planung kollisionsfreier Bewegungen ist gedacht zum Einsatz in einem interaktiven Roboter-Programmierplatz oder in einem Hochsprachenübersetzer für Roboterprogramme. Bei entsprechender Weiterentwicklung des Verfahrens oder weiterer Erhöhung der Rechengeschwindigkeit von Prozessoren ist auch eine Verwendung direkt in der Steuerung eines autonomen Roboters praktikabel.

Die verwandte Aufgabenstellung, einen kollisionsfreien Weg für einen mobilen (fahrbaren) Roboter zu finden, soll hier nicht untersucht werden, weil die Anforderungen und Randbedingungen dieses Problems wesentlich verschieden sind.

1.4 Überblick über den Inhalt der folgenden Kapitel

Kapitel 2 führt eine Beschreibung des Problems der Bewegungsplanung als formale Suchaufgabe im Konfigurationsraum ein. Die Begriffe Freiheitsgrad und Konfigurationsraum werden erklärt. Dann werden die bereits veröffentlichten und bekannten Ansätze zur Lösung des Bewegungplanungsproblems in dreidimensionaler Umwelt vorgestellt. Keines dieser Verfahren ist in der Lage, die Planung von Bewegungen für allgemeine Manipulatoren mit sechs Freiheitsgraden in realistischen dreidimensionalen Arbeitsräumen mit vernünftigem Rechenaufwand zu leisten. Aus den Unzulänglichkeiten der bekannten Ansätze wird die Folgerung gezogen, daß man die Konfigurationsraum-Hindernisse nicht explizit berechnen und darstellen darf, wenn man exponentielle Komplexität vermeiden will. Aus dieser Erkenntnis wird das Prinzip der ZZ-Suche (zielgerichte und zufallsgesteurte Suche) entwickelt.

In Kapitel 3 wird der Algorithmus der ZZ-Suche für n Freiheitsgrade im Detail vorgestellt. Die Zielsetzung bei der Entwicklung des Suchverfahrens war, daß praktisch relevante Bewegungsplanungsaufgaben für sechs Freiheitsgrade eines Manipulators in dreidimensionaler Umgebung möglichst schnell gelöst werden. Nach grundlegenden Betrachtungen über die Zielsetzung und Vorgehensweise bei der Suche werden die beiden Hauptkomponenten — zielgerichtete Suche mit kollisionsvermeidenden Gleitschritten und zufallsgesteuerte Zwischenzielerzeugung — ausführlich erklärt.

Die zielgerichtete Suche testet im n-dimensionalen Konfigurationsraum den direkten Weg vom Start zum Ziel. Trifft man dabei auf ein Hindernis, gleitet man an dessen Oberfläche entlang weiter auf das Ziel zu, bis man das Hindernis umgangen hat oder in einer „Sackgasse" stecken bleibt. Im Fall des Steckenbleibens wird zufallsgesteuert ein Zwischenziel erzeugt und versucht, das Ziel vom Start aus über das Zwischenziel zu erreichen. Falls nötig, werden weitere Zwischenziele hinzugefügt. Es wird beschrieben, wie im sogenannten Stellungsgraph die erzeugten Zwischenziele und die bisher gefundenen Teilwege gespeichert werden, und wie dadurch die gesamte Suche gesteuert werden kann.

Wesentlich ist an der ZZ-Suche, daß in allen n-dimensionalen Konfigurationsräumen nur eindimensionale Unterräume erforscht werden. Dadurch ist die Zeit- und Speicherkomplexität nicht exponentiell abhängig von der Anzahl n der Freiheitsgrade des Manipulators.

Kapitel 4 befaßt sich mit der formalen Repräsentation der Geometrie der Umweltobjekte und dem Kollisionstest. Zur Darstellung der Umwelt wird ein hierarchisches Polyeder-Flächen-Modell gewählt. Die speziellen Anforderungen der ZZ-Suche und der daraus resultierende Algorithmus zur Erkennung von Kollisionen zwischen dreidimensionalen Objekten werden besprochen. Da die ZZ- Suche nur ein Ja-Nein-Ergebnis benötigt, können im hier eingesetzten Kollisionstest wesentliche Vereinfachungen angewandt werden, die den Ablauf signifikant beschleunigen.

Kapitel 5 stellt die prototypische Implementierung des auf der ZZ-Suche aufbauenden Bewegungsplaners vor. Vier repräsentative Beispielumwelten mit entsprechenden Aufgaben werden gezeigt, und die Ergebnisse der Testläufe erläutert. Darunter sind auch zwei Aufgaben — Bewegungsplanung für einen schlangenartigen sechsgelenkigen Roboter und Bewegen eines langen dünnen Stabes durch ein Tor —, die mit anderen Verfahren prinzipiell nicht gelöst werden können, weil sie eine Betrachtung aller sechs Freiheitsgrade des Manipulators erfordern.

Die Ergebnisse der Beispielaufgaben werden mit den Ergebnissen anderer Autoren verglichen. Dabei zeigt sich, daß die Bewegungsplanung nach der ZZ-Suche in einfachen Fällen, wie Transportaufgaben mit kleiner Last in großen Freiräumen, durchaus mit den schnellen Verfahren für drei Freiheitsgrade konkurrieren kann. Darüber hinaus ist das ZZ-Verfahren aber in der Lage, auch Bewegungen für komplexe Aufgaben mit sechs Freiheitsgraden in kurzer Zeit zu planen, welche für die anderen Planer prinzipiell unlösbar sind. Eine zusammenfassende Bewertung der Ergebnisse schließt dieses Kapitel ab.

Kapitel 6 beendet die Arbeit mit einem Ausblick auf weitere mögliche Entwicklungen. Es wird kurz angedeutet, wie man die Qualität der erzeugten Bewegung nach bestimmten Kriterien verbessern könnte, und wie man die Berechnung einer Bewegung durch Parallelisierung des Ablaufs beschleunigen könnte.

1.5 Ergebnisse der Arbeit

Die wesentlichen Ergebnisse dieser Arbeit sind

- die Vorstellung eines neuen effizienten Verfahrens zur Wegesuche in hochdimensionalen Suchräumen,
- der experimentelle Nachweis, daß dieses Verfahren für die Bewegungsplanung von sechsachsigen Manipulatoren in praktischen Anwendungen leistungsfähiger ist als vorhandene Verfahren, und
- die Infragestellung der Behauptung, daß die Bewegungsplanung (in praktischen Anwendungen) durch die Vernachlässigung von Orientierungsfreiheitsgraden erleichtert werde.

Das ZZ-Verfahren (zielgerichtete und zufallsgesteuerte Suche) erhält seine hohe Effizienz aus der Tatsache, daß auf explizite Darstellung des freien Konfigurationsraums verzichtet werden kann und Information über die Raumbelegung (Hindernis oder Freiraum) nur an diskreten Punkten von gewissen eindimensionalen Kurven berechnet werden muß. Dadurch wird eine exponentielle Abhängigkeit der Suche von der Anzahl der Freiheitsgrade vermieden.

In mehreren realistischen Beispielaufgaben wird mit Hilfe der Implementierung des Bewegungsplaners gezeigt, daß praktische Anwendungen gelöst

werden können mit Rechenzeiten von wenigen Sekunden — bei Transportbewegungen durch einen großen Freiraum — bis wenigen Minuten bei Bewegen eines Stabes durch ein Tor mit Umorientierung. Zur Zeit ist mir kein Verfahren oder Programm bekannt, das Vergleichbares leistet.

Kapitel 2

Suchaufgabe

2.1 Abstrakte Formulierung der Suchaufgabe

Ein Manipulator besteht aus m beweglichen, starren Gliedern, die Lage jedes dieser Glieder ist in Bezug auf sein Vorgängerglied in der kinematischen Kette oder bezüglich eines festen Koordinatensystems festgelegt. Die Mindestanzahl skalarer Werte, die zur eindeutigen Bestimmung der Lage eines Gliedes notwendig ist, nennt man den Freiheitgrad dieses Gliedes.

Meistens sind die Glieder eines Manipulators durch einfache Rotations- oder Translationsgelenke miteinander verbunden, die nur Drehungen um eine Achse oder Verschiebungen entlang einer Achse erlauben. Der Freiheitsgrad ist dann 1, weil zur Angabe der Stellung des Nachfolgegliedes bezüglich des Vorgängergliedes nur ein Wert erforderlich ist, nämlich der Drehwinkel oder die Entfernung.

Komplexere Fälle, wie Kugelgelenke oder fahrbare Plattformen (mobile Roboter) können durch Einführung gedachter Zwischenglieder (mit jeweils einfacher Achse) auf obigen Fall zurückgeführt werden. Der Freiheitsgrad ist dann die Summe der Freiheitsgrade der einfachen Komponenten. Beispielsweise kann die Lage eines in der Ebene beweglichen Fahrzeugs bezüglich eines festen Koordinatensystems eindeutig beschrieben werden durch die Abstände seines Mittelpunktes von der x-Achse und von der y-Achse und durch die Winkelabweichung seiner Längsachse von der x-Achse. Es sind auch andere Festlegungen möglich — etwa Polarkoordinaten —, aber in jedem Fall braucht man mindestens drei Werte, also ist der Freiheitsgrad 3.

Um die Lage eines jeden Punktes des beweglichen Manipulators mit n Freiheitsgraden im Raum festzulegen, benutzen wir ein Tupel c,

$$c = (c_1, c_2, \ldots, c_n), \quad c_i \in C_i = [a_i, b_i] \subset \mathbf{R},$$

das wir Konfiguration nennen. Die Konfiguration eines Manipulators mit m beweglichen Gliedern enthält für jeden unabhängigen Freiheitsgrad des Manipulators einen skalaren Wert c_i, der diesen Freiheitsgrad festlegt. Manchmal sind Glieder kinematisch gekoppelt, so daß sie nicht unabhängig voneinander bewegt werden können; dann genügt die Angabe eines Wertes zur Festlegung von mehreren Gliedern. Im allgemeinen gilt aber, daß jede Komponente einer

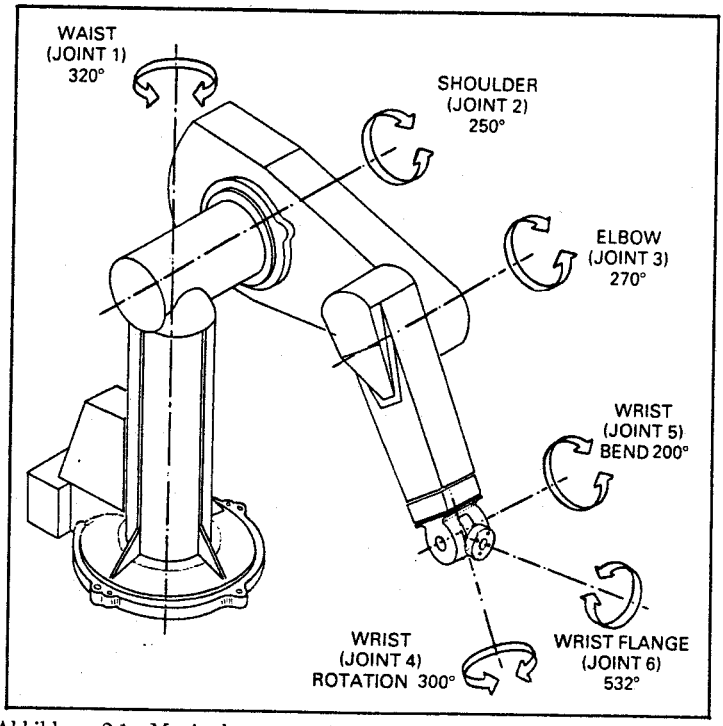

Abbildung 2.1: Manipulator vom Typ PUMA 560 mit sechs Drehgelenken. Alle Gelenke sind unabhängig voneinander, damit hat der Roboter den Freiheitsgrad 6.

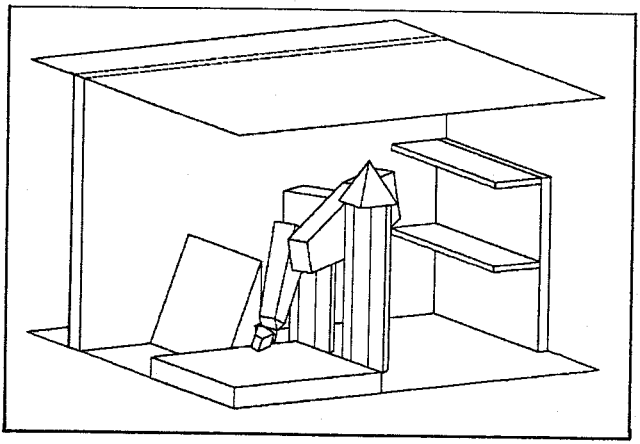

Abbildung 2.2: Ein Manipulator in seinem Arbeitsraum. Die Objekte der Umwelt — hier Boden, Decke, Wand, Regal, Turm, Podest, Keil — bilden für den Roboter Hindernisse beim Bewegen. Abgebildet ist die Stellung $c = (25, 195, 40, 0, 40, 60)$.

Konfiguration genau einem Winkel- oder Verschiebungswert einer Roboterachse entspricht ($m = n$); dann kann man c_i auch anschaulich Gelenkvariable nennen. Übliche Manipulatoren haben sechs Freiheitsgrade ($n = 6$), wobei meist die ersten drei Gelenke die Position des Effektors festlegen und die restlichen dann nur noch die Orientierung des Effektors ändern.

Das kartesische Produkt der zulässigen Bereiche C_i aller (unabhängigen) Gelenkvariablen c_i bildet den sogenannten Stellungsraum oder Konfigurationsraum C,

$$C = C_1 \times C_2 \times \ldots \times C_n \subset \mathbf{R}^n,$$

des Manipulators. Jeder Stellung des Manipulators im anschaulichen dreidimensionalen Objektraum entspricht genau ein Punkt im n-dimensionalen Konfigurationsraum.

Diejenigen Stellungen $c \in C$, die im Objektraum zu einer Kollision mit dem Umwelthindernis h_i führen würden, fassen wir zu der Menge $H_i \subset C$ zusammen, die wir auch Konfigurationsraum-Hindernis nennen. Die Menge aller verbotenen Stellungen ist dann $H = \bigcup H_i$; das Komplement, die Menge aller erlaubten Stellungen $F = C \setminus H$ wird auch freier Konfigurationsraum genannt.

Die sogenannte Locus-Methode ist in der Computer-Geometrie ein übliches Vorgehen, um komplexe geometrische Fragestellungen in äquivalente, aber algorithmisch leichter zu beschreibende Punkt-Probleme in geeigneten Suchräumen umzuformen [33].

Bei einem beweglichen Manipulator wird jede Stellung im Objektraum eineindeutig in einen Punkt im Konfigurationsraum C abgebildet. Eine Bewegung

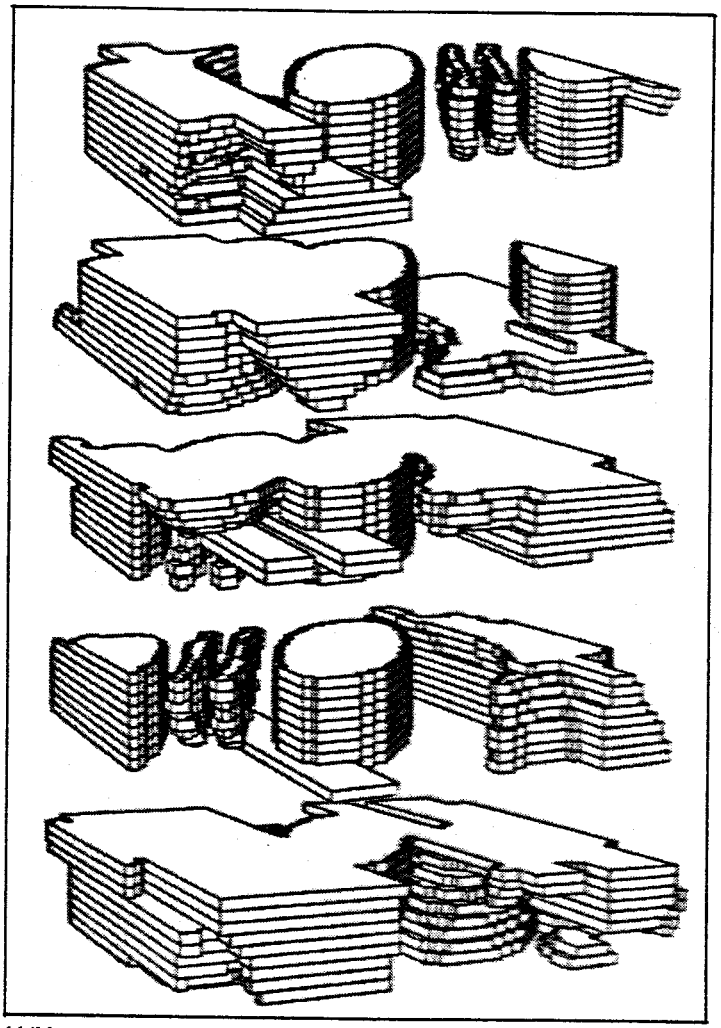

Abbildung 2.3: Konfigurationsraum des Roboters aus Abbildung 2.2. Aus dem sechsdimensionalen Raum wurde ein dreidimensionaler Bereich ausgeschnitten ($c_4 = c_5 = c_6 = 0$, was einer gestreckten Hand entspricht). Die Gelenkvariable c_1 geht in 45 Schritten von unten nach oben (-90 bis 180 Grad), c_2 in 57 Schritten von links nach rechts (-80 bis 260 Grad), und c_3 in 50 Schritten von vorne nach hinten (-60 bis 240 Grad).

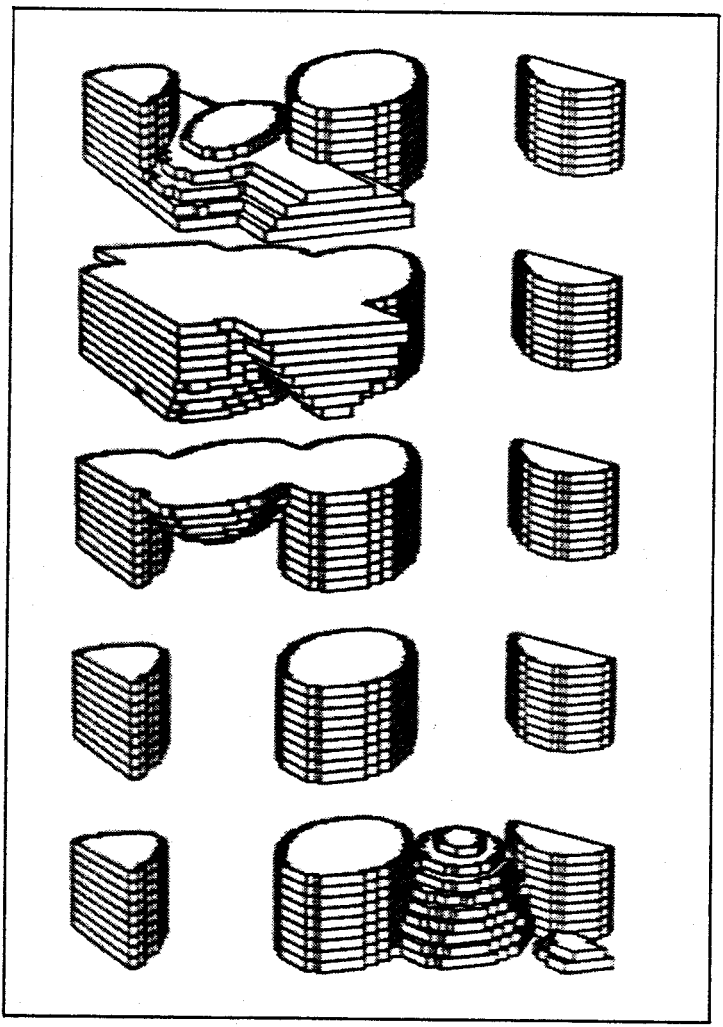

Abbildung 2.4: Ausschnitt aus dem Konfigurationsraum. Hier sind nur die Hindernisse Boden, Decke und Wand berücksichtigt. Man sieht, daß die im Objektraum (Abbildung 2.2) horizontalen Flächen unabhängig von der Stellung des Rumpfgelenks c_1 immer die gleiche Behinderung darstellen: sie sind im Konfigurationsraum senkrechte Säulen. Die übrigen Gebilde (links oben und rechts unten) stammen von der Wand.

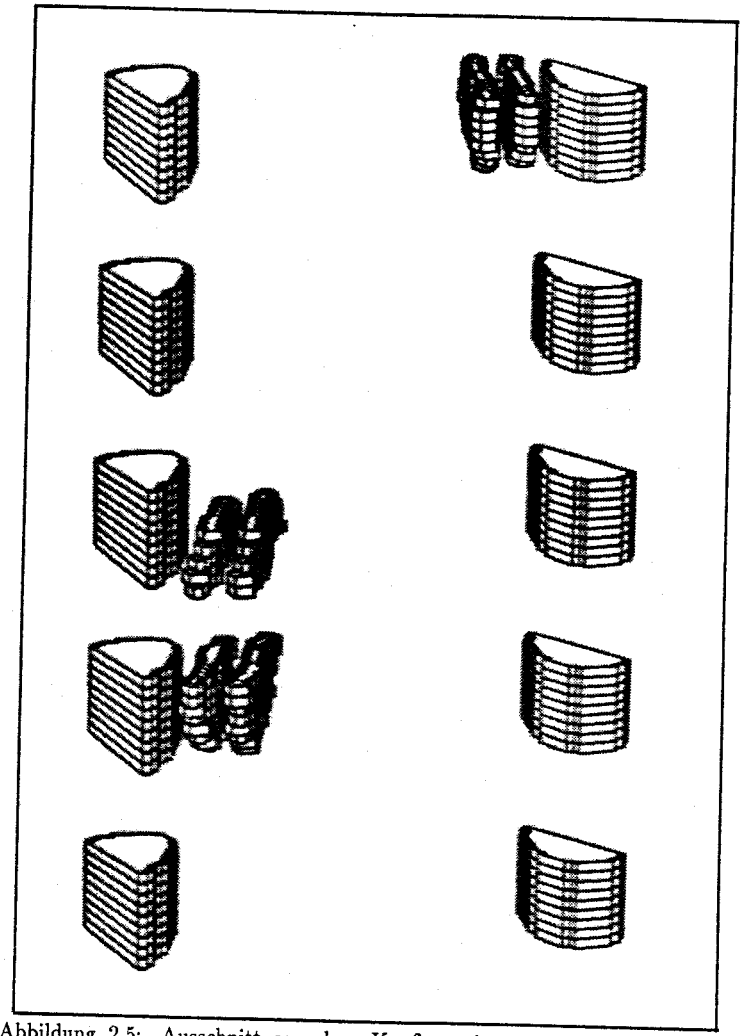

Abbildung 2.5: Ausschnitt aus dem Konfigurationsraum. Hier sind nur die Transformationen von Boden (durchgehende senkrechte Säulen) und Regal gezeigt. Das Regal erzeugt zwei Hindernis-Paare: einmal im Bereich $c_1 \in [-29, 45]$ Grad (unten links), und im Bereich $c_1 \in [137, 180]$ Grad (oben rechts). Das untere Hindernis wird im linksarmigen Zustand („lefty“) vom Roboter erreicht, das obere im rechtsarmigen („righty“).

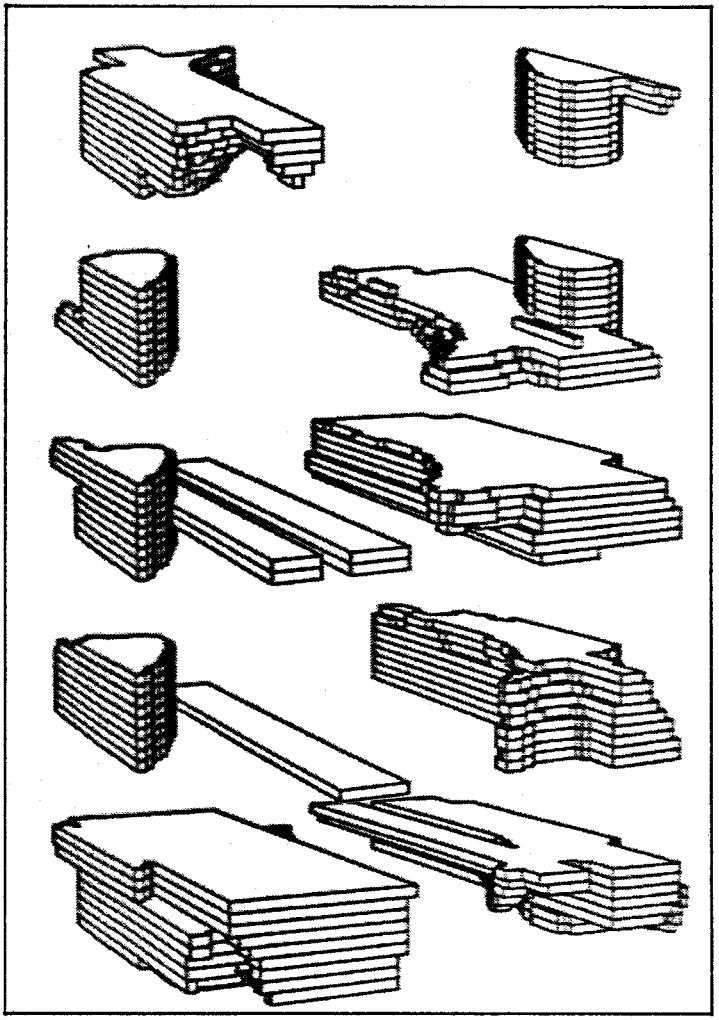

Abbildung 2.6: Ausschnitt aus dem Konfigurationsraum. Hier sind die Transformationen von Boden, Turm, Podest und Keil abgebildet. Die Vereinigung der hier gezeigten Hindernis-Mengen mit denen der Abbildungen 2.4 und 2.5 ergibt die Gesamtmenge der Konfigurationsraum-Hindernisse, die einen für die Suche verbotenen Bereich darstellt und in Abbildung 2.3 dargestellt ist.

des Roboters in seiner Umwelt kann dann durch diese Locus-Transformation eineindeutig auf einen Kurvenzug im zugehörigen Konfigurationsraum abgebildet werden. Das ursprüngliche Problem der Bewegungsplanung im Objektraum stellt sich dann so dar: Suche im Konfigurationsraum einen stetigen Kurvenzug w,

$$w : I \rightarrow C,$$

(mit $I = [a, b] \subset \mathbf{R}$) vom Startpunkt S,

$$S = w(a) \in F,$$

zum Endpunkt G,

$$G = w(b) \in F,$$

so daß

$$w(I) \cap H = \emptyset$$

erfüllt ist, also die Kurve außerhalb sämtlicher Konfigurationsraum-Hindernisse liegt.

Die Vereinfachung durch „Schrumpfung"des Roboters zu einem Punkt zieht eine entsprechende „Aufblähung" der Umwelthindernisse im Konfigurationsraum nach sich, die für den gesuchten Kurvenzug verbotene Bereiche darstellen. Diese Konfigurationsraum-Hindernisse H_i sind Gebilde, deren Oberflächen durch $(n-1)$-dimensionale algebraische Mannigfaltigkeiten beschrieben werden können; die exakte rechnerische Behandlung und Darstellung dieser Oberflächen ist aber sehr aufwendig.

2.2 Stand der Technik

Wegen seiner Bedeutung für die aufgabenorientierte Programmierung besteht ein großes Interesse an einer Lösung des Bewegungsplanungsproblems — entsprechend viele Arbeiten wurden auch schon zu diesem Thema veröffentlicht. Einen guten Überblick über die umfangreiche Literatur bieten die Artikel von Whitesides [42] und Sharir [37]. Leider behandeln die meisten Autoren nur zweidimensionale, ebene Umwelten, die zur Modellierung realistischer dreidimensionaler Anwendungen nicht brauchbar sind.

Einige der dreidimensionalen Ansätze sind im Prinzip für beliebig viele Freiheitsgrade des Manipulators geeignet, aber ihre inhärente exponentielle Zeit- und Speicherkomplexität — zur expliziten Darstellung des freien Konfigurationsraums F durch Zellenaufteilung und zur Suche im Zellengraph — beschränkt ihre praktische Brauchbarkeit auf drei oder maximal vier Freiheitsgrade.

So verwenden Hasegawa und Terasaki [22], wie auch Lozano-Pérez und seine Mitarbeiter [32] jeweils nur die ersten drei Gelenke zur Planung von Transportbewegungen, lediglich in der engsten Umgebung der Aufnahme- und Ablageposition wird die Orientierung der Hand und der Last berücksichtigt. Siméon [38] hat einen solchen Planer für vier Freiheitsgrade implementiert. Die Berechnung des freien Konfigurationsraums F für eine Umgebung, die aus

einer Wand mit einem Loch zum Durchgreifen besteht, benötigt bei ihm 50 Minuten, die darauf aufbauende Planung eines Wegs 10 Sekunden (jeweils auf SUN 3).

Alle diese Verfahren haben den Nachteil, daß man während des Transports um die Hand herum soviel Abstand zu Hindernissen lassen muß, daß jede Orientierung kollisionsfrei bleibt, weil man ja nicht weiß, welche Orientierung nun gerade vorliegt. Anschaulich stülpt man eine Hüllkugel um die Hand mit der Last, innerhalb derer sie sich frei bewegen kann. Es ist klar, daß das Hüllkugelverfahren nur für verhältnismäßig kleine Lasten sinnvoll ist, weil sonst die Hüllkugel bereits einen Großteil des Arbeitsraums ausfüllen würde, und der Roboter kaum noch bewegt werden könnte.

Andere Autoren schränken die Kinematik des Manipulators so ein, daß nur noch vier Freiheitsgrade übrig bleiben [5, 16, 41]. Die verminderte Beweglichkeit des Roboters — zum Beispiel: der Greifer muß immer senkrecht nach unten ausgerichtet sein — ist in vielen Anwendungen ein großer Nachteil. Sinnvoll eingesetzt werden können diese Systeme nur für einfachste Transportbewegungen, wenn über den Objekten und Hindernissen genügend Platz vorhanden ist, und wenn die zu bewegenden Lasten verhältnismäßig klein sind.

Obwohl eine Behandlung aller sechs (oder mehr) Freiheitsgrade eines Manipulators bei vielen Aufgaben wesentlich ist — etwa beim Transport sperriger Lasten oder bei der Steuerung redundanter Manipulatoren —, gibt es bisher nur wenige Veröffentlichungen, die über Implementierungen von Bewegungsplanern für Manipulatoren mit sechs oder mehr Achsen in dreidimensionalen Umgebungen berichten.

Donald [9] hat ein Programm entwickelt, das mittels sogenannter lokaler Experten kollisionsfreie Bahnen für ein frei im Raum bewegliches („fliegendes") Objekt erzeugt. Die Laufzeiten der Planung betragen mehrere Stunden, und eine Übertragung auf Manipulatoren mit Drehgelenken ist nicht ohne weiteres möglich, da der Algorithmus wesentlich auf der Entkopplung von Position und Orientierung bei dem frei fliegenden Objekt aufbaut.

Barraquand und Latombe [3] geben ein Beispiel für eine Bewegungsplanung mit einem 31-Gelenk-Arm in dreidimensionaler Umwelt an (Rechenzeit 15 Minuten auf DECstation 3100). Diese Leistung ist bisher konkurrenzlos. Der Manipulatorarm besteht aber nur aus Strichen (Linien), und die Umwelt ist als Voxelfeld (128 mal 128 mal 128 Bit) sehr grob modelliert, so daß dieser Planer — zumindest im momentanen Zustand — eher theoretisch interessant als praktisch relevant ist.

Die sequentielle Strategie von Gupta [20] zerlegt ein n-dimensionales Bewegungsplanungsproblem in n zweidimensionale, und damit leicht lösbare Probleme. Die Planung wird erst nur für das erste Glied durchgeführt, danach — von der Lösung des Vorgängerglieds ausgehend — für das jeweils nächste Glied. Dadurch wird die exponentielle Komplexität des Problems vermieden, aber die Suche ist natürlich nicht mehr vollständig und findet in manchen Fällen keine Lösung, obwohl eine existiert.

Faverjon und Tournassoud [13] vereinen in ihrem sogenannten gemischten Ansatz die Vorteile von lokalen und globalen Suchmethoden. Um den exponentiellen Aufwand zu beschränken, benutzen sie für die globale Suche nur eine sehr grob angenäherte Darstellung des freien Konfigurationsraums F der ersten drei Gelenke. Die übrigen Freiheitsgrade werden erst in der lokalen Suche berücksichtigt, die sehr effizient ist, aber wegen ihrer Lokalität durch „Sackgassen" gefährdet ist, in denen die Suche steckenbleibt. Die globale Komponente muß also die Aufgabe so in Teilaufgaben zerlegen, daß diese jeweils vom lokalen Teil gelöst werden können. Diese Methode scheint gut zu funktionieren, solange eine ausreichende Entkopplung zwischen den ersten drei Gelenken (für die globale Suche) und den restlichen vorliegt, wie das bei einem PUMA-Roboter mit kleiner Last der Fall ist. Dafür werden Beispiele mit Laufzeiten von wenigen Minuten angegeben. Sobald diese Bedingung aber nicht mehr erfüllt wird — etwa bei einem PUMA mit sperriger Last — führt die gewählte Strategie zu einer unvollständigen Suche, die aber nach Angabe der Autoren „viele praktische Fälle löst".

Keines der bekannten Verfahren leistet also die automatische Planung von Bewegungen für allgemeine Manipulatoren mit sechs Freiheitsgraden in realistischer dreidimensionaler Umwelt mit vernünftigem Rechenaufwand.

2.3 Folgerungen und ZZ-Suche

Um den Nachteil der exponentiellen Komplexität zu vermeiden, muß man unbedingt auf die explizite Darstellung der Konfigurationsraum-Hindernisse H_i und des freien Konfigurationsraums F verzichten. Dadurch entfällt auch die Aufteilung des Raums in Zellen, deren Anzahl exponentiell mit der Anzahl der Freiheitsgrade wächst und damit die Suche im Zellengraph langwierig macht. Gleichzeitg muß man den Anspruch auf eine optimale Lösung aufgeben, weil zu deren Konstruktion natürlich der gesamte Konfigurationsraum bekannt sein müßte.

Bei dem hier vorgestellten Verfahren zur Bewegungsplanung wird Information über die Lage von Konfigurationsraum-Hindernissen nur bei Bedarf und nur an einzelnen Punkten entlang von eindimensionalen Kurven berechnet. Dieses Vorgehen ermöglicht die schnelle Erzeugung kollisionsfreier Bewegungen auch für komplexe sechsachsige Manipulatoren in dreidimensionalen Umgebungen.

Im folgenden wird dieses neue Suchverfahren kurz umrissen. Da es die Kombination einer zielgerichteten und einer zufallsgesteuerten Komponente darstellt, nennen wir es kurz ZZ-Suche.

Zuerst wird eine zielgerichtete Suche im Konfigurationsraum C vom Start S aus direkt auf das Ziel G zu eingeleitet. Dabei wird in bestimmten Abständen auf Kollision mit Hindernissen H_i getestet. Bei Auftreffen auf ein Konfigurationsraum-Hindernis gleitet man an der Oberfläche des Hindernisses entlang. Das Gleiten erfolgt immer so, daß dabei die Entfernung zum

Ziel verringert wird. Entweder kommt man so um das Hindernis herum oder man bleibt in einem lokalen Minimum („Sackgasse") stecken. Im Fall des Steckenbleibens wird zufallsgesteuert ein Zwischenziel $Z_1 \in F$ erzeugt und dann versucht, dieses Zwischenziel direkt und durch Gleiten zu erreichen, und zwar von Start S aus und vom eigentlichen Ziel G aus. Bei Mißerfolg werden sukzessiv weitere, zusätzliche Zwischenziele $Z_2, Z_3, \ldots \in F$ erzeugt, und es wird versucht, jedes neue Zwischenziel Z_i jeweils vom Startpunkt S, vom Zielpunkt G und von allen bisherigen Zwischenzielen $Z_1, Z_2, \ldots, Z_{i-1}$ aus zu erreichen. Die Sammlung aller gefundenen Teilwege erfolgt in den Kanten eines Graphen, dessen Knoten Startpunkt S und Zielpunkt G und die Zwischenziele $Z_1, \ldots, Z_i$ repräsentieren. Sobald Startknoten und Zielknoten in diesem Graph zusammenhängen, ist ein Weg gefunden, und die Suche wird beendet.

Die wesentliche Grundannahme besteht darin, daß viele Lösungen möglich sind, aber nur eine einzige benötigt wird. Das ZZ-Verfahren hat nur Aussicht auf schnellen Erfolg bei der Suche, wenn es so viele Lösungen des Problems gibt, daß die Chance, zufällig eine zu treffen, relativ groß ist. Das ist aber bei üblichen Bewegungsplanungsaufgaben für Manipulatoren der Fall.

Kapitel 3

ZZ-Suchalgorithmus für n Freiheitsgrade

3.1 Zielsetzung und Vorgehensweise

Nach den theoretischen Komplexitätsbetrachtungen von Sharir [37] und anderen (Reif [34], Hopcroft, Joseph und Whitesides [26], Hopcroft, Schwartz und Sharir [27]) ist das allgemeine Bewegungsplanungsproblem mindestens NP-schwer. Doch wie bei anderen solchen Problemen, etwa beim Rucksack-Problem oder beim Problem des Handelsreisenden, heißt das nicht, daß alle konkreten Aufgabenstellungen dieses Problems praktisch unlösbar sind, sondern nur daß es für jeden Algorithmus, der ein solches Problem löst, konkrete Aufgaben gibt, die für diesen „schwer" sind, also nicht mit vernünftigem Speicher- und Rechenzeitverbrauch lösbar sind.

Es kann also durchaus ein Verfahren geben, das die normalen Fälle schnell löst, ohne daß ein Widerspruch zu der obigen Aussage entsteht.

Ziel bei der Entwicklung des hier beschriebenen Bewegungsplaners war, die Suche nach einer kollisionsfreien Bewegung dahin zu optimieren, daß die normalerweise im Zusammenhang mit Manipulatoren vorkommenden Aufgaben effizient bewältigt werden. Der Algorithmus sollte praktisch relevante Bewegungsaufgaben, also für alle sechs Freiheitsgrade eines Manipulators und in dreidimensionaler Umgebung, möglichst schnell lösen. Daß dadurch vielleicht seine Fähigkeit, Ausgänge aus einem Labyrinth zu finden, vermindert wird, war kein Kriterium, weil so etwas in üblichen Anwendungen nicht vorkommt. Die effiziente Bearbeitung der Problemklasse „Labyrinth" würde auch ein völlig anderes Vorgehen bedingen, da die Voraussetzungen und Randbedingungen wesentlich verschieden sind.

Die automatische Bewegungsplanung ist besonders wichtig bei Transportbewegungen (pick-and-place movement), und gerade bei diesen ist immer ein großes Maß an verschiedenen Lösungen möglich. Die Chance, mit dem hier entwickelten Verfahren zu einer Lösung zu gelangen, ist also gegeben.

NP-schwere Probleme haben die Eigenschaft, daß das Berechnen einer Lösung schwierig ist, daß aber das Feststellen, ob ein gegebener Vorschlag das

Problem löst oder nicht, sehr leicht durchgeführt werden kann [19]. Diese Eigenschaft macht sich die zufallsgesteuerte Komponente der ZZ-Suche zunutze: die Lösungsvorschläge (Zwischenziele) werden zufällig erzeugt und dann auf ihre Brauchbarkeit getestet. Beide Schritte, Erzeugung und Test, sind rechnerisch relativ einfach und hängen insbesondere nicht exponentiell von der Anzahl der Freiheitsgrade des Manipulators ab.

Natürlich wird nicht eine komplette Lösung einer Aufgabe durch Zufall erzeugt. Allein die Zwischenziele sind zufällig, aus ihnen wird dann deterministisch — durch zielgerichtete Suche — ein Weg zwischen Start und Ziel gewonnen.

Der Test, ob zwischen zwei Konfigurationen, die Start, Ziel oder Zwischenziele repräsentieren, ein kollisionsfreier Weg existiert, wird durch die andere, zielgerichtete Komponente des ZZ-Verfahrens ausgeführt.

Um eine exponentielle Komplexität des Verfahrens zu vermeiden, erforscht die zielgerichtete Suche unabhängig von der Anzahl der Dimensionen des Suchraums immer nur eindimensionale Unterräume. Diese untersuchten Teilmengen des gesamten Suchraums sind potentiell kollisionfreie Wege zwischen den betrachteten Konfigurationen. Als erstes wird immer der Weg direkt auf das momentane Ziel zu getestet. Wenn ein Hindernis auf dem Weg liegt, gleitet die Suche entlang der Hindernisoberfläche weiter in Richtung des Ziels, bis das Hindernis umgangen ist.

Sollte aber an einer bestimmten Stelle auf der Hindernisoberfläche ein weiteres Gleiten auf das Ziel zu nicht mehr möglich sein, weil man in einer Sackgasse steckt, so wird die zielgerichtete Suche einfach aufgegeben mit dem Ergebnis, daß eine kollisionsfreie Verbindung der beiden Konfigurationen ohne weiteres Zwischenziel nicht möglich ist.

Der überwiegende Rest des Suchraums wird nicht verwendet und braucht deshalb auch nicht explizit dargestellt zu werden. Wegen der Lokalität der zielgerichteten Suche werden natürlich manche möglichen Teilwege nicht gefunden; das macht aber nichts, wenn es genügend Alternativen gibt, was in praktischen Anwendungen der Fall ist.

Weil nur geringe Teile des Suchraums überhaupt betrachtet werden, gehen viele Aspekte globaler Optimierung verloren. Es kann also nicht garantiert werden, daß ein Weg in irgendeiner Hinsicht global optimal ist. Eine lokale Optimierung, wie etwa Verkürzung des gefundenen Wegs oder eine Glättung von scharfen Ecken, kann dagegen durchgeführt werden.

Da die Heuristik des ZZ-Verfahrens Zufallscharakter hat, kann man keine lösbare Aufgabe angeben, die prinzipiell von diesem nicht gelöst werden kann. Die ZZ-Suche ist vollständig: wenn ein Weg existiert, dann wird er auch gefunden. Leider kann man keine obere Schranke angeben für die Zeit, die das Verfahren dafür benötigt, deshalb nennt man diese Art von Vollständigkeit auch probabilistische Vollständigkeit [3]. Weiterhin kann das Verfahren bei unlösbaren Aufgaben prinzipiell nicht feststellen, daß keine Lösung existiert. Beide Eigenschaften sind aber für den praktischen Einsatz bedeutungslos, weil

dort andere Randbedingungen viel stärker wiegen, etwa Sicherheit oder Geschwindigkeit.

3.2 Zielgerichtete Suche

Da man in einem hochdimensionalen Suchraum $C \subset \mathbf{R}^n$ nicht alle möglichen Wege von einem Startpunkt S zu einem Zielpunkt G untersuchen kann — zumindest nicht, wenn man Wert auf Schnelligkeit der Berechnung legt —, muß man eine Wahl treffen. Am naheliegendsten ist wohl die direkte geradlinige Verbindung g,

$$g : [0,1] \to C, \quad x \mapsto g(x) = S + (G - S)x,$$

zwischen Start S und Ziel G.

Zur Bestätigung oder Widerlegung der Kollisionsicherheit dieser Bewegung könnte man nun das vom Manipulator dabei überstrichene Volumen mit dem Volumen der Umweltobjekte schneiden, oder die Weglinie auf Durchschnitt mit den Konfigurationsraum-Hindernissen testen. Beide Verfahren setzen voraus, daß man im allgemeinen Fall Volumina mit gekrümmten Oberflächen darstellen und bearbeiten muß. Das ist zwar möglich, aber sehr aufwendig (Donald [9], Lozano-Pérez [31]).

Eine wesentliche Voraussetzung für die Effizienz der ZZ-Suche ist der Verzicht auf explizite Darstellung von Konfigurationsraum-Hindernissen, so daß die oben genannten exakten Methoden zum Kollisionstest nicht in Frage kommen. Der Test auf Kollision oder Kollisionsfreiheit einer Linie wird zurückgeführt auf den Kollisionstest einer Menge von Punkten dieser Linie. Die Größe l ist die maximale Distanz im Objektraum, die ein Massenpunkt des Arms bei einer Bewegung zwischen zwei benachbarten, als kollisionsfrei getesteten Punkten zurücklegen kann. Sofern l eine gewisse Konstante $2s$ nicht übersteigt, ist auch die Kollisionssicherheit aller direkt dazwischenliegenden Stellungen gesichert, wenn alle Umwelthindernisse um eine kleine Schutzschicht mit der Dicke s „aufgebläht" werden (siehe Abbildung 3.1).

Da die grundlegende Annahme für das ZZ-Verfahren die Existenz sehr vieler möglicher Lösungen ist, ist der Verlust an Wegen, den diese Schutzschicht mit sich bringt, unbedeutend. Eine exakte Umweltbeschreibung wäre auch sinnlos, weil die meisten Feinheiten, etwa Schraublöcher, für die Transportbewegungen völlig bedeutungslos sind. Weiter ist eine Beschreibung unterhalb der Größe der Positionstoleranz des Manipulators unnötig; man kann sie aus Sicherheitsgründen sowieso nicht nutzen.

Die Schrittweite d, der maximale Abstand im Konfigurationsraum zwischen zwei Kollisionstests, der gerade noch die Kollisionsicherheit des dazwischenliegenden Geradenstücks sicherstellt, ist sowohl abhängig von der Dicke s der über die Umweltobjekte gezogenen Schutzschicht als auch von der Kinematik des Manipulators.

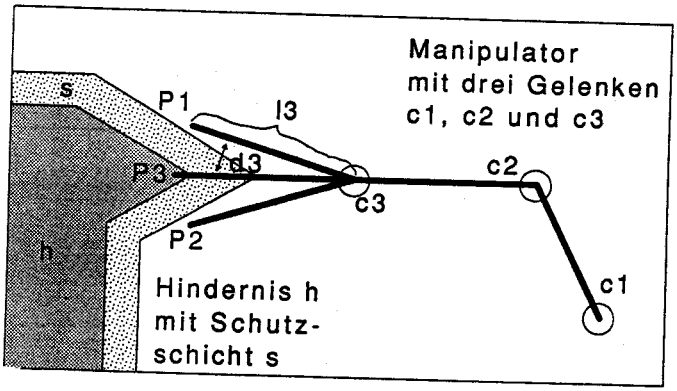

Abbildung 3.1: Schutzschicht und punktueller Kollisionstest. Der Test einer Bewegung auf Kollision wird nur in bestimmten Punkten (Stellungen) ausgeführt, die einen gewissen Höchstabstand nicht überschreiten dürfen, um die Kollsionssicherheit auch im dazwischenliegenden Bereich zu garantieren. Der Schritt von P_1 nach P_2 ist zu groß, die Sicherheitsbedingung $d_3 l_3 \leq 2s$ ist verletzt, und wie man sieht, liegt dazwischen auch wirklich eine Kollisionstellung P_3 vor. Rückt man P_2 näher an P_1 heran, so daß die Sicherheitsbedingung erfüllt ist, so liegt die Spitze des Arms im Sicherheitspuffer des Hindernisses und der Kollisionstest führt zur korrekten Ablehnung der Bewegung von P_1 nach P_2.

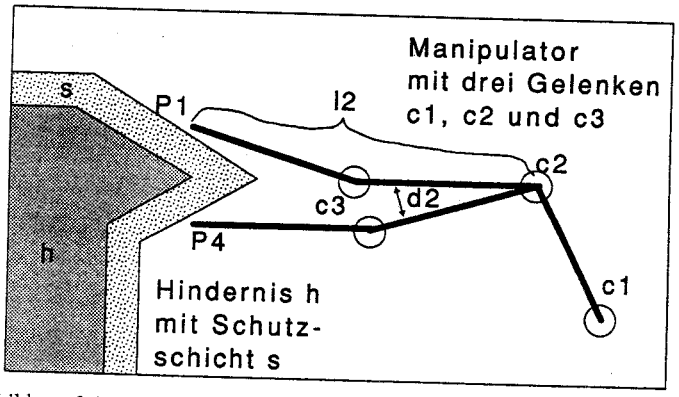

Abbildung 3.2: Wirkungsradien von Gelenken. Wegen der unterschiedlichen Wirkungsradien l_i der Gelenkvariablen c_i darf eine reine c_2-Bewegung nur rund halb so groß (in Winkelangaben) sein wie eine reine c_3-Bewegung (vergleiche Abbildung 3.1).

Die einzelnen Achsen des Manipulators haben verschieden starke Auswirkungen auf die absolute Bewegung des Effektors; die gleiche absolute Schrittweite d im Rumpfgelenk c_1 eines PUMA-Roboters etwa wirkt stärker als die im Handgelenk c_4. Bei Bestimmung der Schrittweite d zwischen zwei aufeinanderfolgenden Kollisionstests darf diese bei reinen c_4-Bewegungen also größer sein als bei reinen c_1-Bewegungen. Diesem Umstand wird durch eine geeignete Gewichtung der Freiheitsgrade c_i Rechnung getragen.

Mit $l_i(c_{i+1}, \ldots, c_n)$ sei der momentane Wirkungsradius der Gelenkvariablen c_i bezeichnet, das ist der Abstand des von der Achse i am weitesten entfernten Massenpunktes des Manipulatorteils, der von der Achse i bewegt werden kann; die Folgeachsen $i+1$ bis n befinden sich dabei in den Gelenkstellungen c_{i+1} bis c_n. Damit kann die maximal erlaubte (ungetestete) Distanz $d_i = |\Delta c_i|$ zwischen Testpunkten in Richtung der Gelenkvariablen c_i bestimmt werden durch $d_i l_i \leq 2s$, also $d_i \leq 2s/l_i$ (siehe auch Abbildungen 3.1 und 3.2).

Mit der Substitution $c'_i = c_i l_i$ und $d'_i = |\Delta c'_i|$ kann man auch schreiben $d'_i \leq 2s$, was den Vorteil hat, daß die Schrittweite nun für alle (substituierten) Gelenkvariablen c'_i gleich ist.

Jetzt muß noch geklärt werden, wie gleichzeitige Bewegungen mehrerer Gelenke zu behandeln sind. Ideal wäre ein Abschätzung wie $d' = |\Delta c'| \leq 2s$, wobei $|c| = d(0, c)$ die Euklidische Norm eines Vektors ist. Leider kann man den eindimensionalen Fall dieser Ungleichung nicht einfach auf den allgemeinen n-dimensionalen übertragen, weil sonst die Sicherheitsbedingung $l \leq 2s$ verletzt werden kann. Die maximale Verschiebung l eines Massenpunktes des Manipulatorarms, die durch eine Stellungänderung Δc hervorgerufen werden kann, läßt sich abschätzen als skalare Summe der Einzelverschiebungen $|\Delta c_i|\ l_i$ aller Gelenke mit

$$l \leq |\Delta c_1|\ l_1 + \ldots + |\Delta c_n|\ l_n.$$

Die Gleichheit von l mit dem Ausdruck auf der rechten Seite tritt dann ein, wenn alle Einzelverschiebungen der einzelnen Gelenkvariablen in die gleiche Richtung des Objektraums wirken. Das ist nur in speziellen Stellungen des Manipulators der Fall, muß aber berücksichtigt werden.

Zum Beispiel ($n = 2$): $\Delta c' = (\Delta c'_1, \Delta c'_2)$, mit $\Delta c'_1 = \Delta c'_2 = 2s\sqrt{2}/2$, erfüllt mit

$$d' = |\Delta c'| = \sqrt{(\Delta c'_1)^2 + (\Delta c'_1)^2} = 2s$$

die Bedingung $d' \leq 2s$ im Konfigurationsraum, aber die resultierende Bewegung l des Manipulatorarms im Objektraum kann nicht mehr mit Sicherheit kleiner als $2s$ gehalten werden:

$$l \leq |d_1|\ l_1 + |d_2|\ l_2 = (d'_1/l_1)l_1 + (d'_2/l_2)l_2 = (d'_1 + d'_2) = 2s\sqrt{2}.$$

l kann hier fast eineinhalb mal so groß werden wie erlaubt. Das liegt daran, daß die Komponenten d'_i im Konfigurationsraum vektoriell addiert werden, die maximale Verschiebung l im Objektraum aber als skalare Summe der Auswirkungen der d'_i gebildet werden muß.

Für den allgemeinen Fall mit n Dimensionen findet man, daß bei Erfüllung der modifizierten Ungleichung

$$d' = |\Delta c'| \leq 2s/\sqrt{n}$$

immer auch die Sicherheitsbedingung $l \leq 2s$ eingehalten wird. Eine ausführliche Ableitung ist im Anhang F enthalten.

Anschaulich ausgedrückt haben wir durch die Substitution $c_i' = c_i l_i$ den Konfigurationsraum gerade so verzerrt, daß ein Schritt von der Länge $d' = 2s/\sqrt{n}$ in jede beliebige Richtung eine Objektraumverschiebung des Arms von maximal $2s$ bewirkt.

3.3 Gleitschritt

Prinzipiell funktioniert die ZZ-Suche auch ohne eine spezielle Strategie zur Hindernisumgehung. Die Ergebnisse der praktischen Versuche zeigen aber, daß durch eine Kollisionsvermeidung eine enorme Leistungsverbesserung eintritt — es wird in den meisten Fällen viel schneller eine Lösung gefunden.

Zur Umgehung von Hindernissen sind sehr viele Ansätze veröffentlicht worden. Alle sind entweder sehr speziell auf bestimmte Manipulatoren und Bewegungen eingeschränkt (Brooks [5], Freund und Hoyer [15]) oder sie benötigen einen solchen Rechenaufwand, daß sie nicht zu unserer Zielsetzung, möglichst schnell einen Weg zu finden, passen.

Die Idee der hier verwendeten Methode ist folgende. Vom Start $S \in F$ ausgehend testet man in Abständen d' Punkte P_i auf der geraden Verbindungslinie g zum Ziel $G \in F$ hin. Bei Auftreffen auf ein Hindernis H, was man dadurch erfährt, daß der Kollisionstest für den momentanen Punkt P_i eine Kollision liefert ($P_i \in H$), macht man einen Schritt „zur Seite" und wählt den nächsten Punkt P_{i+1} nicht auf der Geraden g, sondern aus ihrem orthogonalen Komplement, nämlich aus der affinen Hyperebene $E = g^\perp + P_i$, die senkrecht auf der Gerade g im Punkt P_i steht. Im $(n-1)$-dimensionalen Unterraum $U = g^\perp$ wird nach der in Anhang E beschriebenen Formel eine Orthonormalbasis $B = \{B_1, \ldots, B_{n-1}\}$ gebildet, die diesen Unterraum aufspannt. Die Vektoren dieser Basis und deren Negative bilden eine Menge von Richtungen für mögliche Seitenschritte.

Die exakte Beschreibung der Oberfläche eines Konfigurationsraum-Hindernisses H ist durch die Nullstellenmenge $\{v \in C : f(v) = 0\}$ einer reellwertigen differenzierbaren Funktion $f : C \to \mathbf{R}$ auf dem Konfigurationsraum C gegeben. Lozano-Pérez [30] gibt an, wie die Menge H mit solchen Funktionen modelliert werden kann. Lokal läßt sich die Oberfläche von H in jedem Punkt somit durch eine lineare $(n-1)$-dimensionale Mannigfaltigkeit — also eine Hyperebene — im Konfigurationsraum approximieren [9, 8].

Der neue Punkt P_{i+1} soll natürlich kollisionsfrei sein. Unter der Annahme, daß die Oberfläche des Konfigurationsraum-Hindernisses H lokal durch eine

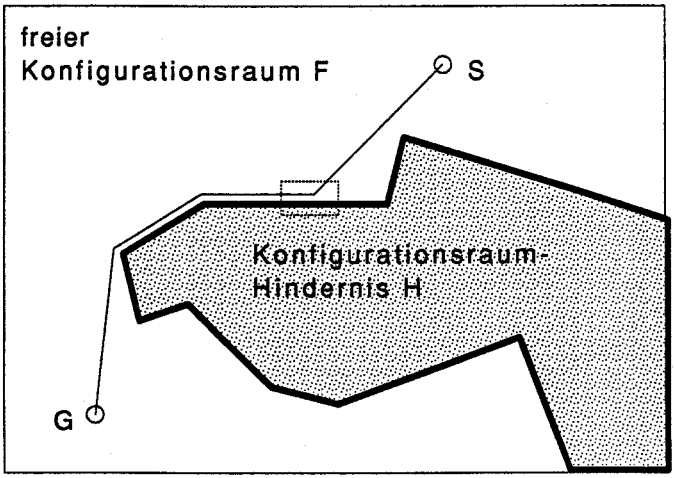

Abbildung 3.3: Zielgerichtete Suche mit Hindernisumgehung. Die zielgerichtete Suche geht vom Start S aus zunächst direkt auf das Ziel G zu, das Konfigurationsraum-Hindernis kann mit kollisionsvermeidenden Gleitschritten umgangen werden; in umgekehrter Richtung, von G nach S führte das Gleiten in eine Sackgasse. Diese Strategie beachtet nur lokale Eigenschaften des Suchraums; deshalb hängt ihre Komplexität nicht von der Größe des Suchraums ab, die exponentiell mit der Anzahl der Freiheitgrade wächst. Das markierte Rechteck in der Mitte des Bildes wird in Abbildung 3.4 vergrößert dargestellt und beschrieben.

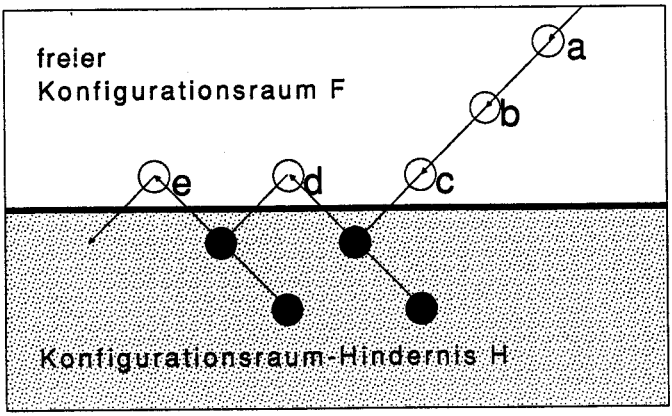

Abbildung 3.4: Hindernisumgehung durch Gleitschritte entlang der Hindernisoberfläche (Ausschnitt aus Abbildung 3.3). Alle Kreise sind Punkte, die auf Kollision getestet werden. Die hellen Kreise *a*, *b* und *c* sind kollisionsfreie Konfigurationen auf dem direkten Weg zum Ziel. Der dunkle Kreis nach *c* stellt eine kollisionsbehaftete Konfiguration dar, die den Ausgangspunkt für den Schritt zur Seite nach *d* bildet. Von *d* aus wieder auf des Ziel zugehend wird auf die selbe Art wieder ein Seitenschritt nach *e* erzeugt. Der gesuchte kollisionsfreie Weg führt durch die Punkte *a*, *b*, *c*, *d* und *e*.

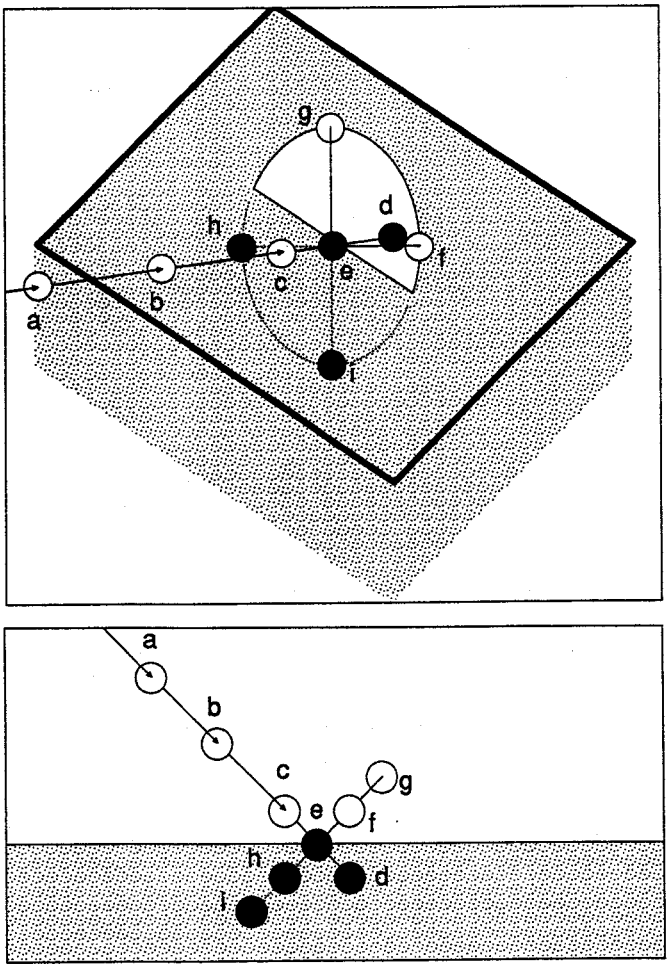

Abbildung 3.5: Veranschaulichung der $4 = 2 * (n - 1)$ möglichen Seitenschritte im Falle eines dreidimensionalen Konfigurationsraums. *Oben:* Dreidimensionale Ansicht. Den Start der Suche muß man sich links vorne oben vorstellen, das Ziel rechts hinten unten. Das dick umrandete Viereck ist ein Ausschnitt aus der Oberfläche eines Konfigurationsraum-Hindernisses, das sich unter dieser Fläche befindet. Die Punkte a bis d liegen auf dem direkten Weg zum Ziel, e ist der durch Bisektion gefundene Ausgangspunkt für die möglichen Seitenschritte nach f, g, h und i. *Unten:* Seitenansicht. Die Punkte f und i liegen vor der Bildebene, g und h dahinter.

$(n-1)$-dimensionale Hyperebene angenähert wird, kann man einen kollisionsfreien Punkt $P_{i+1} \in F$ in höchstens $2*(n-1)$ Versuchen finden: nämlich durch Austesten der Punkte der Menge

$$\{P_i + d'B_1, P_i - d'B_1, P_i + d'B_2, \ldots, P_i - d'B_{n-1}\},$$

wobei B eine Orthonormalbasis des Unterraums U des orthogonalen Komplements E der momentanen Zielgeraden g ist. Das Ausprobieren kann bei $n = 2$ als Ausweichen nach links oder rechts gedeutet werden, bei $n = 3$ hat man zusätzlich die Richtungen unten und oben (siehe auch Abbildung 3.5).

Hat man nach dem Durchtesten aller $2*(n-1)$ möglichen Ausweichpunkte keinen kollisionsfreien Punkt gefunden, so steckt man in einer „Sackgasse", aus der die hier verwendete zielgerichtete Suche keinen Ausweg finden kann. Sie wird beendet mit dem Ergebnis, daß kein direkter Weg zwischen dem momentanen Startpunkt S und dem momentanen Zielpunkt G gefunden wurde. Es ist dann Aufgabe der zufallsgesteuerten Komponente, Zwischenziele vorzugeben, mit denen die zielgerichtete Suche dann indirekt einen Weg finden kann; das wird im nächsten Abschnitt behandelt.

Gehen wir also davon aus, daß ein kollisionsfreier Punkt P_{i+1} gefunden wurde. Dieser Punkt hat nun vom letzten kollisionsfreien Punkt P_{i-1} die Entfernung

$$d(P_{i-1}, P_{i+1}) = \sqrt{d'^2 + d'^2} = \sqrt{2}d' > d'.$$

Es muß also zur Sicherheit noch ein Zwischenpunkt $P_i = (P_{i-1} + P_{i+1})/2$ eingeschoben und auf Kollision getestet werden; da das alte kollisionsbehaftete P_i nun nicht mehr gebraucht wird, können wir diese Bezeichnung hier neu belegen.

Wenn der Test dieses Zwischenpunkts P_i keine Kollision ergibt, ist ein Gleitschritt erfolgreich abgeschlossen. Jetzt kann das Ergebnis des Ausweichschritts, der Punkt P_{i+1}, als Start einer neuen direkt auf das Ziel G hin gerichteten Suche dienen. Die Situation ist ähnlich der ganz zu Beginn: man hat zwei kollisionsfreie Stellungen, Start und Ziel, und man soll von der einen direkt oder durch Gleiten zur anderen gelangen.

Es erfolgen also wieder ein oder mehrere Schritte der Länge d' direkt auf das Ziel hin, so lang, bis ein Hindernis oder das Ziel erreicht wird. Falls notwendig, werden weitere Gleitschritte unternommen. Führt gleich der erste versuchte direkte Schritt zu einer Kollision, so folgen zwei oder mehr Gleitschritte unmittelbar aufeinander, die dann in einem fortgesetzten Gleiten über die Oberfläche des Hindernisses resultieren, bis es auf diese Art umgangen ist, oder bis man in einem lokalen Minimum stecken bleibt.

Im Gegensatz zum Test einer direkten geraden Linie zwischen Start und Ziel ist es bei Verwendung der Gleitstrategie wesentlich, von welcher Seite aus man beginnt. Es besteht eine Asymmetrie zwischen Start und Ziel, eine Art „Widerhaken-Effekt": in eine Richtung gelingt das Gleiten gut, in der Gegenrichtung nicht. Deshalb wird bei Mißerfolg der Suche vom Start aus auch noch der Weg von Ziel zum Start hin untersucht.

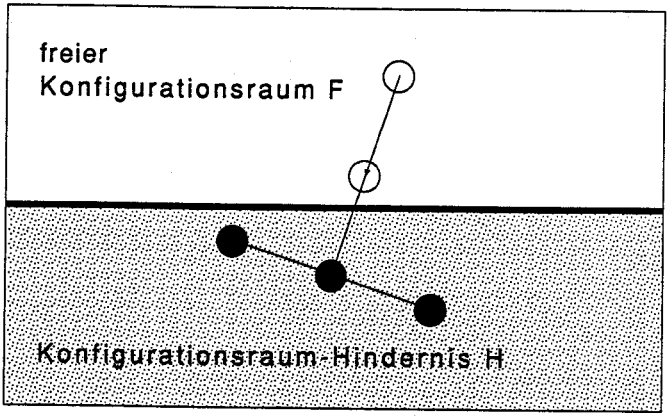

Abbildung 3.6: Scheinbare Sackgasse bei tiefem Eindringen. In diesem Fall führt keiner der möglichen Seitenschritte zu einer kollisionsfreien Ausweichstellung, weil der Ausgangspunkt für die Seitenschritte zu tief unter der Oberfläche steckt. Eine Abhilfe deutet Abbildung 3.7 an.

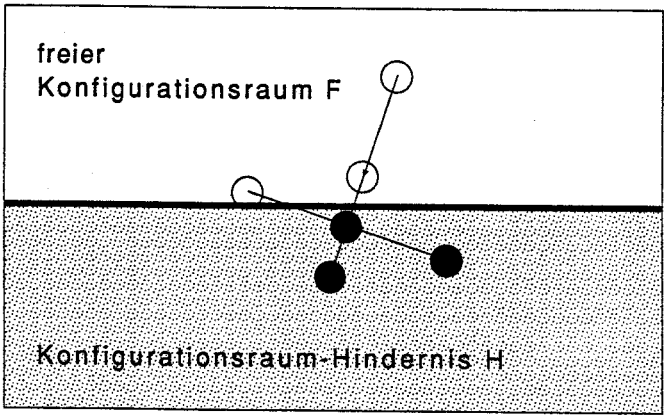

Abbildung 3.7: Vermeidung scheinbarer Sackgassen. Um die Gleitfähigkeit zu verbessern, wird das Ausgangszentrum für die Seitenschritte möglichst nahe an die Oberfläche des Konfigurationsraum-Hindernisses verlegt. Im Gegensatz zu der Situation in Abbildung 3.6 wird jetzt ein Ausweg gefunden.

Sind beim Durchtesten der $2*(n-1)$ Ausweichrichtungen alle Punkte mit Kollisionen behaftet, so findet diese einfache Methode zur Kollisionsvermeidung keinen Ausweg. Das kann drei verschiedene Gründe haben: man steckt in einem lokalen Minimum und es gibt keinen Ausweg, der die momentane Entfernung zum Ziel verringert — etwa senkrechtes Auftreffen auf eine flache Wand —, oder der Ausgangspunkt P_i für die Ausweichschritte ist zu tief unter der Oberfläche des Konfigurationsraum-Hindernisses angesetzt worden (siehe Abbildung 3.6), oder es liegt eine Oberfläche vor, die sich nicht lokal linear approximieren läßt („Rinne"). Die letzten beiden Gründe zum Abbruch können durch Verbesserung der Testpunkterzeugung weitgehend vermieden werden.

Um den Ausgangspunkt für Ausweichschritte möglichst wenig unter der Oberfläche anzusetzen, berechnet man durch Bisektion zwischen dem letzten kollisionsfreiem Punkt P_{i-1} und dem Kollisionspunkt P_i die ungefähre Einstichstelle des geradlinigen Wegs in die Hindernisoberfläche. Bereits vier Bisektionen genügen für eine wesentliche Erhöhung der Gleitfähigkeit, ab einem Einfallswinkel von etwa 7 Grad (gemessen gegen die Oberflächennormale) kann dann das Auffinden eines Auswegs garantiert werden. Zum Vergleich: ohne Bisektion findet man erst ab 45 Grad sicher einen existierenden Ausweg, bei exaktem Einstichpunkt bleibt die Suche nur bei Einfallswinkel = 0 stecken. Das gilt, wenn die Hindernisoberfläche lokal linear approximiert werden kann.

Gleitet der Manipulator gleichzeitig an zwei oder mehr verschiedenen Hindernissen, so ist die Oberfläche des entsprechenden Konfigurationsraum- Hindernisses, also die Vereinigung der einzelnen Konfigurationsraum-Hindernisse, an gewissen Stellen nicht mehr linear approximierbar. Anschaulich gesprochen liegen statt glatter Oberflächen nun Stellen mit kantigen Rillen vor, nämlich da, wo die Einzelhindernisse im Konfigurationsraum aneinanderstoßen. Der garantierte Ausweg aus einer solchen Situation ist nicht mehr mit einfachen Mitteln möglich. Er ist aber unter den Bedingungen einer Transportbewegung bei einem Manipulator auch gar nicht nötig, weil es technisch kaum möglich ist, daß der Arm gleichzeitig an mehreren Hindernissen entlanggleitet, und weil solche Bewegungen auch gar nicht wünschenswert sind. Es ist also kein Schaden, wenn man diese Einschränkung macht.

Der Gleitschritt nach der oben beschriebenen Methode benötigt für jede Dimension des Unterraums (Hyperebene) einen Basisvektor und dessen Negativ, so daß der Aufwand für den Test linear von der Anzahl n der Dimension des Suchraums abhängt. Die Erzeugung der orthogonalen Basis nach der Methode von Schmidt [14] hat quadratischen Aufwand. Damit ist der Gesamtaufwand für das Gleiten quadratisch abhängig von n, im Gegensatz zur exponentiellen Abhängigkeit bei exakten Methoden.

3.4 Zwischenzielerzeugung

Bei der zielgerichteten Komponente der ZZ-Suche wurde bewußt zugunsten ihrer Effizienz auf eine Vollständigkeit verzichtet, manche vorhandenen Wege

können also nicht direkt gefunden werden. Das gleicht die zufallgesteuerte Komponente der ZZ-Suche aus, sie stellt die Vollständigkeit der Gesamtsuche sicher.

Wenn die zielgerichtete Suche keinen Weg zwischen Startpunkt S und Zielpunkt G finden kann, so wird zufallsgesteuert ein neues Zwischenziel $Z_1 \in F$ erzeugt. Dieses Zwischenziel zerlegt die ursprüngliche, nicht direkt lösbare Aufgabe in zwei neue Teilaufgaben. Es wird dann zielgerichtet ein Weg zwischen S und Z_1, und ein Weg zwischen Z_1 und G gesucht. Sind die beiden Teilaufgaben erfolgreich gelöst, so läßt sich daraus leicht die Gesamtlösung zusammensetzen.

Die Erzeugung des Zwischenziels erfolgt durch zufällige Auswahl seiner n skalaren Komponenten aus den erlaubten Wertebereichen $C_1, \ldots, C_n$ der Gelenkvariablen. Danach wird diese Stellung auf Kollision getestet. Falls nötig, wird die Erzeugung zufälliger Werte so lang wiederholt, bis man eine kollisionsfreie Stellung erhält.

Die Idee, Zwischenziele zufällig zu erzeugen, ist naheliegend und einfach; nach den Ergebnissen der Experimente mit sechsachsigen Manipulatoren in realistischen Umgebungen ist sie auch genügend wirkungsvoll. Dennoch sei hier noch auf weitere Möglichkeiten hingewiesen, die sich vielleicht in speziellen Fällen bewähren könnten. Die Struktur des Arbeitsraums ist nicht homogen, Stellungen in bestimmten Bereichen eignen sich unter Umständen viel besser als andere zur Auswahl von Zwischenzielen. Hier könnte eventuell eine gewichtete zufällige Erzeugung oder gar eine gezielt aus den Umweltdaten berechnete Erzeugung von Zwischenzielen Verbesserungen bringen. In Kombination mit einem Programmiersystem für Roboter ist auch die interaktive Vorgabe von einzelnen kritischen Zwischenzielen durch den Bediener denkbar; sozusagen ein grobes Teach-In mit nur einem oder wenigen Zwischenpunkten.

3.5 Stellungsgraph

In komplexeren Umgebungen reicht es manchmal nicht aus, die ursprüngliche Bewegungsplanungsaufgabe in nur zwei Teilaufgaben zu zerlegen. Es sind dann zwei oder mehr Zwischenziele nötig, um die zielgerichtete Suche um die Hindernisse herumzuleiten. Deswegen werden alle erzeugten Zwischenziele Z_i und Start S und Ziel G als Knoten des Stellungsgraphs gespeichert. Für jedes neu erzeugte Zwischenziel wird mit Hilfe der zielgerichteten Suche getestet, ob es von allen schon im Graph vorhandenen Stellungen aus erreichbar ist. Die von der zielgerichteten Suche erhaltenen Informationen über die Erreichbarkeit zwischen einzelnen Stellungen werden als Kanten zwischen diesen Knoten eingetragen.

Der Stellungsgraph repräsentiert eine Relation R, die die Erreichbarkeit zweier Punkte im Konfigurationsraum mithilfe der zielgerichteten Suche beschreibt.

$$R \subset C \times C, \quad (a, b) \in R \Leftrightarrow$$

b ist von a aus durch zielgerichtete Suche erreichbar.

Anfänglich besteht der Stellungsgraph nur aus dem Startknoten und dem Zielknoten und aus einer leeren Menge von Kanten. Im Laufe der ZZ-Suche erzeugte Zwischenziele und erhaltene Teilwege werden schritthaltend hinzugefügt. Sobald der Startknoten und der Zielknoten in der gleichen Zusammenhangskomponente des Stellungsgraphs liegen — oder anders ausgedrückt: sobald Start S und Ziel G in der symmetrischen und transitiven Hülle R^*_{sym} von R liegen, $(S, G) \in R^*_{sym}$ —, ist ein Weg für die ursprüngliche Gesamtlösung gefunden und die ZZ-Suche wird abgebrochen.

Mit zunehmender Knotenanzahl bildet der Stellungsgraph eine immer besser werdende Annäherung an die Topologie des Freiraums. Im Gegensatz zu der komplexen geometrischen Struktur des freien Konfigurationsraums F ist die Topologie auch hochdimensionaler Konfigurationsräume sehr einfach, sie ist im wesentlichen nur von der Anzahl der Zusammenhangskomponenten des Raums abhängig. Die Ergebnisse der Experimente zeigen, daß auch in sechsdimensionalen Suchräumen einige wenige zufällige Zwischenziele ausreichen, um die Aufgaben zu lösen. Die geringe Anzahl von Zwischenzielen führt zu einem entsprechend kleinen Graph, so daß mit Standardalgorithmen aus diesem leicht der Gesamtweg vom Start S zum Ziel G extrahiert werden kann.

Da nur irgendein Weg gesucht wird, kann man sich darauf beschränken, nur diejenigen Kanten des Graphen auf Kollision zu testen, die einzelne noch unzusammenhängende Komponenten des Graphen verbinden könnten. Ein vollständig getesteter Graph ergibt im allgemeinen kürzere Wege, aber unter dem Nachteil einer mehrfach längeren Rechenzeit. Im Normalfall genügt ein Zwischenziel zur Lösung, nur ist eben nicht jedes beliebige Zwischenziel dafür brauchbar, so daß mehrere erzeugt und getestet werden müssen. In den Fällen, wo die Lösungsbewegung aus einem einzigen Zwischenziel konstruiert werden kann, bekommt man den kürzesten Weg auch ohne Austesten aller möglichen Verbindungen, wenn man beim Test der Erreichbarkeit des jeweiligen neuen Zwischenziels mit Start und Ziel beginnt.

Wie oben bereits angedeutet, ist die ZZ-Suche im Prinzip vollständig, wenn man ihr unbeschränkt viel Zeit gibt. Sie ist probabilistisch vollständig, aber nicht deterministisch vollständig [3]. Das ist im praktischen Einsatz belanglos, dort wird nach einer gewissen Anzahl erfolgloser Versuche einfach abgebrochen mit der Vermutung, es gebe keinen Weg. Das kann in kritischen Fällen, bei sehr schwierigen Problemen, dazu führen, daß vom Programm manchmal eine Lösung gefunden wird und manchmal die gleiche Aufgabe unlösbar erscheint.

Kapitel 4

Kollisionstest

4.1 Anforderungen

Wie wir gesehen haben, muß sowohl beim Test eines direkten, geraden Wegs als auch beim Gleiten entlang einer Hindernisoberfläche in ausreichend kleinen Abständen d' die Kollisionsfreiheit der momentanen Konfiguration überprüft werden, damit die Kollisionsfreiheit der ganzen Bewegung sichergestellt werden kann. Der Kollisionstest wird also häufig verwendet und macht somit einen großen Teil der Gesamtrechenzeit aus. Eine effiziente Kollisionsprüfung ist damit wesentlich für die Effizienz des Bewegungsplaners.

Eine unmittelbare Prüfung von ganzen Bewegungen auf Kollision ist nicht erforderlich. Bei der ZZ-Suche im Konfigurationsraum C des Manipulators ist nur von Bedeutung, zu wissen, ob der Manipulator in einer bestimmten Konfiguration $c \in C$ mit Objekten seiner Umwelt kollidiert ($c \in H$) oder nicht ($c \in F$). Zur Erinnerung: H ist die Menge der Konfigurationsraum-Hindernisse, $F = C \setminus H$ ist der freie Konfigurationsraum. Als Ergebnis des Kollisionstests genügt also eine einfache Anwort „kollidiert" oder „kollidiert nicht". Weitere Informationen, etwa über die Position oder „Stärke" einer Kollision bzw. den kleinsten Abstand zu den Hindernissen bei Kollisionsfreiheit werden nicht benötigt.

Da nur ein Ja-Nein-Ergebnis vom Kollisionstest erbracht werden soll, können in der Berechnung viele Vereinfachungen verwendet werden, die den Ablauf eines Tests signifikant beschleunigen. So wird etwa nach der ersten gefundenen Kollision sofort abgebrochen, und der ganze Rest der Umwelt gar nicht mehr betrachtet.

4.2 Umweltrepräsentation

Prinzipiell sind alle aus dem CAD-Bereich bekannten Methoden zur Beschreibung von dreidimensionalen Objekten verwendbar, wie zum Beispiel das Polyeder-Flächen-Modell, das Volumenmodell, CSG-Bäume, usw. [10, 35]. Am geeignetsten für die Zusammenarbeit mit dem ZZ-Verfahren erscheint das Polyeder-Flächen-Modell, weil es bei der erforderlichen, relativ geringen Genauigkeit am einfachsten zu speichern und zu verarbeiten ist. Nach diesem Mo-

dell besteht ein voluminöses Objekt aus der Menge seiner Begrenzungsflächen, die jeweils durch Polygone (Facetten) angenähert werden. Die Facetten sind als geordnete Folge der räumlichen Koordinaten ihrer Eckpunkte dargestellt.

Die in Kapitel 3 geforderte Schutzschicht der Dicke s, die zur Sicherstellung einer kollisionsfreien Bewegung aus dem Test von einzelnen Punkten nötig ist, wird entweder bereits bei der Modellierung der Umwelt berücksichtigt, oder sie kann nachträglich automatisch durch Versetzen der Facetteneckpunkte nach außen (in Richtung der Facettennormalen) erfolgen.

Zusätzlich zur Beschreibung der eigentlichen Objekte werden einfache Hüllkörper, wie Hüllquader oder Hüllkugeln, von Objektteilen (Facetten), von Objekten und von Objektgruppen als sogenannte Pseudo-Objekte gebildet. Die Hüllen von Facetten und Einzelobjekten werden automatisch erzeugt, bei weiteren Zusammenfassungen sind zusätzliche Angaben nötig. Zum Beispiel: alle Teile der Maschine M1 sollen zusammengefaßt werden, oder: die obere Hälfte des Fließbandes gibt ein Pseudo-Objekt [11].

Alle Objektflächen, Objekte und Pseudo-Objekte werden in einem Baum zusammengefaßt. Die Blätter des Baums enthalten die ursprüngliche Information über die Geometrie der wirklichen Objekte, die internen Knoten des Baums sind jeweils die geometrischen Hüllen ihrer Unterbäume. Jede Schicht des Baums stellt eine Umweltbeschreibung auf einer bestimmten Abstraktionsebene dar, je näher diese Schicht im Baum an den Blättern liegt, desto größer wird die Genauigkeit; die Wurzel des Baums ist ein Quader, der den gesamten Arbeitsraum des Roboters umfaßt.

4.3 Algorithmus

Wenn zwei Körper (Polyeder) miteinander kollidieren, also nicht-leeren Durchschnitt haben, so gibt es an jedem der beiden Polyeder eine Fläche, die mit einer Fläche des anderen Polyeders kollidiert — oder aber das eine Polyeder ist vollständig im anderen enthalten. Nachdem aber weder Teile der Umwelt noch des Roboters frei in der Luft schweben können, muß in letztem Fall das innere, völlig umschlossene Polyeder an einem oder mehreren anderen Objekten festgemacht sein, und dann schneiden eben Flächen dieser Befestigungsobjekte Flächen des umfassenden Polyeders. Man kann also in unserem Zusammenhang den Schnitt von Polyedern auf den Schnitt von Polyederflächen (Polygonen, Facettten) zurückführen.

Der Baum, der aus der hierachischen Beschreibung der Umwelt und den Objektzerlegungen in Facetten resultiert, ist völlig einheitlich von der Wurzel bis zu den Blättern. Für den Kollisionstest liegt einfach ein Baum mit Flächen an den Blättern und Hüllkörpern an den Knoten vor. Es wird nur so tief im Baum nach unten gegangen wie nötig ist, um zu entscheiden, ob eine Kollision vorliegt oder nicht. Wenn sich beispielsweise bereits die Hüllquader des Manipulators und einer komplexeren Maschine nicht überlappen, so ist das Betrachten sämtlicher „inneren" Objekte überflüssig, weil sie unter diesen Bedingungen gar nicht kollidieren können.

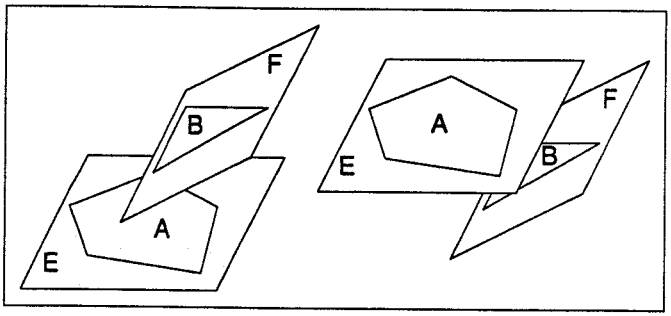

Abbildung 4.1: Durchdringungstest bei Flächen, erste Stufe. Hier liegen einfache Bedingungen zum Ausschluß von Kollisionen zwischen den Flächen (Facetten) A und B vor. *Links:* Fläche B liegt ganz über der Trägerebene E der Fläche A. *Rechts:* Fläche A liegt ganz über der Trägerebene F der Fläche B.

Das Verfahren erlaubt in beiden möglichen Fällen — Kollision oder Kollisionsfreiheit — eine schnelle Entscheidung ohne den ganzen Baum zu durchsuchen: bei Vorliegen einer Kollision wird der Durchlauf durch den Baum sofort abgebrochen, sobald die erste Durchdringung von wirklichen Objekten entdeckt ist; und bei Umweltteilen, die großen oder mittleren Abstand vom Manipulator in seiner momentanen Stellung haben, wird der Baum nur auf den oberen bzw. mittleren Schichten traversiert unter Vermeidung des Großteils der unteren Schichten. Es werden also genau diejenigen Konfigurationen ausführlich behandelt, bei denen sich der Manipulator sehr nahe an einem Objekt befindet. Da der Manipulator aus praktischen Gründen gleichzeitig aber nur sehr wenigen Objekten sehr nahe kommen kann, ist der Rechenzeitaufwand nur unwesentlich abhängig von der Gesamtanzahl der Objekte in der Umwelt.

Bei geeigneter hierachischer Strukturierung der Umweltbeschreibung ist eine Zunahme der für einen Kollisionstest benötigten Rechenzeit mit dem Logarithmus der Anzahl der Flächen zu erwarten.

Der Test zweier achsenparalleler Hüllquader oder zweier Hüllkugeln auf Vorliegen einer Durchdringung ist trivial — und damit schnell ausführbar. Die einzige aufwendige Operation beim Kollisionstest ist das Schneiden von zwei planen, polygonalen Facetten, die beliebig im Raum liegen. Aber auch hier lassen sich aufgrund der Forderung, daß nur ein Ja-Nein-Ergebnis gewünscht wird, starke Vereinfachungen realisieren (siehe Abbildung 4.1).

Zunächst wird die Lage der Facette A relativ zur Trägerebene F der Facette B untersucht: liegt sie vollständig darüber oder vollständig darunter, so ist der Durchschnitt $A \cap B$ leer, und das Ergebnis ist „keine Kollision". Falls dieser Schritt keine Entscheidung bringt, probiert man den umgekehrten Test aus: liegt die Facette B relativ zur Trägerebene E von A ganz darunter oder ganz darüber? Eine Facette liegt vollständig über oder unter einer Ebene, wenn

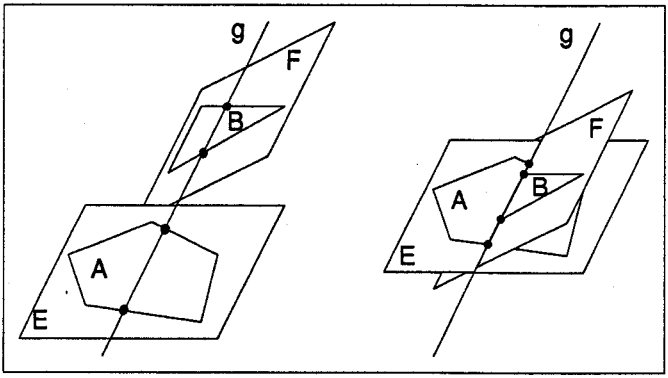

Abbildung 4.2: Durchdringungstest bei Flächen, zweite Stufe. In diesen beiden Fällen liegt weder die Fläche A über B noch liegt die Fläche B über A. Es gibt also eine Gerade $g = E \cap F$, die sowohl A als auch B schneidet. Durch Untersuchen der Reihenfolge der markierten Punkte $P_A \in g \cap \partial A$ und $P_B \in g \cap \partial B$ auf der Geraden g kann das Vorliegen einer Durchdringung (Kollision) bestätigt oder widerlegt werden.

die gerichteten Abstände aller ihrer Punkte im Bezug auf diese Ebene gleiches Vorzeichen haben (Formel in Anhang C).

Wenn in den beiden Schritten keine Entscheidung fallen sollte, weiß man, daß es eine Gerade $g = E \cap F$ gibt, die sowohl die Facette A als auch die Facette B schneidet. Man berechnet nun die Menge S,

$$S = S_A \cup S_B, \quad S_A = g \cap \partial A, \quad S_B = g \cap \partial B,$$

der Schnittpunkte zwischen der Geraden g und den Randlinien der Facetten A und B. Aus der Anordnung der Punkte von S auf der Geraden g ergibt sich eindeutig, ob sich die Facetten schneiden oder nicht (siehe Abbildung 4.2).

Sonderfälle beim Schnitt von einer Facette mit der gemeinsamen Geraden werden für die Programmierung wie folgt behandelt. Fällt die Gerade g mit einer Facettenkante zusammen, so werden nur die Eckpunkte dieser Kante als Schnittpunkte gezählt; fällt die Gerade g mit einem Eckpunkt, aber nicht mit angrenzenden Kanten zusammen, so wird der Eckpunkt doppelt gezählt. Die Mächtigkeit der Mengen S_A und S_B ist deshalb immer ein Vielfaches von 2.

Programmiertechnisch ist es günstiger, die Schnittpunkte der Menge S direkt durch Schneiden der Facettenkanten mit der Trägerebene der anderen Facette zu berechnen (siehe Anhang D). Dabei kann man sich auf die Kanten beschränken, die jeweils einen Endpunkt unter und einen Endpunkt über der Trägerebene haben. Diese Information hat man ja bereits bei den vorigen Tests berechnet.

Die Schnittpunkte der Menge S sind durch ihre Lage auf der Geraden g

Bedeutung	Zustand	Punkt	Folgezustand
„außerhalb A und außerhalb B“	– –	$\in S_A$	A –
„außerhalb A und außerhalb B“	– –	$\in S_B$	– B
„außerhalb A und innerhalb B“	– B	$\in S_A$	A B
„außerhalb A und innerhalb B“	– B	$\in S_B$	– –
„innerhalb A und außerhalb B“	A –	$\in S_A$	– –
„innerhalb A und außerhalb B“	A –	$\in S_B$	A B
„innerhalb A und innerhalb B“	A B	$\in S_A$	– B
„innerhalb A und innerhalb B“	A B	$\in S_B$	A –

Tabelle 4.1: Zustandsübergangsmatrix zur Feststellung von Durchschnitten zweier Facetten A und B.

geordnet. Ordnungskriterium eines Punkts p ist das Skalarprodukt

$$\langle p, n_E \times n_F \rangle,$$

wobei n_E und n_F die Normalenvektoren der beiden betrachteten Facetten sind.

Jeder Punkt bezeichnet einen Übergang zwischen den vier Zuständen „außerhalb A und außerhalb B“, „außerhalb A und innerhalb B“, „innerhalb A und außerhalb B“ und „innerhalb A und innerhalb B“. Der letzte Zustand kennzeichnet eine Durchdringung der beiden Facetten A und B. Die vollständige Zustandsübergangsmatrix ist in Tabelle 4.1 wiedergegeben.

Beginnend mit dem Startzustand „außerhalb A und außerhalb B“ arbeitet man die Schnittpunktmenge S von einem Ende der Geraden g aus ab. Kommt man während der Abarbeitung aller Punkte in den Zustand „innerhalb A und innerhalb B“, so liegt eine Durchdringung der Facetten A und B vor, andernfalls nicht.

Eine andere Optimierung („Gedächtnis“) nutzt die „Lokalität“ der Anfragen. Da meistens aufeinanderfolgende Kollisionstests sich auf benachbarte (ähnliche) Konfigurationen beziehen, ist ein weiterer Geschwindigkeitsgewinn dadurch möglich, daß man sich die Stelle im Objektbaum merkt, an welcher in der letzten Anfrage eine Kollision gefunden wurde — falls eine Kollision vorlag. Diese gemerkte Stelle kann dann bei der nächsten Kollisionsanfrage die erste zu testende sein. Im Fall erneuter Kollision, die ja wahrscheinlich ist, wenn die vorige Nachbar-Konfiguration eine Kollision lieferte, hat man dann eine minimale Rechenzeit, weil ja sofort da abgebrochen wird, wo man begonnen hat.

Kapitel 5

Ergebnisse

5.1 Implementierung

Der ZZ-Algorithmus zur Bewegungsplanung in dreidimensionaler Umgebung wurde für Manipulatoren mit drei und sechs Freiheitsgraden realisiert. Beide Moduln wie auch der Modul für den Kollisionstest wurden in VAX-Pascal unter dem Betriebssystem VMS implementiert.

Das Verfahren zum Feststellen von Kollisionen arbeitet zur Zeit nur mit einem dreischichtigen Baum: Objekthüllen als oberste abstrakte Ebene, dann Facettenhüllen und schließlich die eigentlichen Facetten mit der geometrischen Information an den Blättern des Baums. Eine weitere Hierarchisierung läßt bei den Beispielumwelten mit relativ wenigen, einfach strukturierten Objekten keine signifikante Geschwindigkeitsverbesserung erwarten. Insbesondere die Zusammenfassung von mehreren benachbarten Objekten zu einem gemeinsamen übergeordneten Hüllobjekt ist nicht realisiert, weil sie bisher nicht nötig war und nur den Programmieraufwand des Prototyps erhöht hätte. Je höher der Detaillierungsgrad einer Umwelt ist, desto mehr Vorteile werden weitere Hierarchisierungsebenen bei der Umweltbeschreibung erbringen.

Die in folgenden Beispielläufen angegebenen Rechenzeiten wurden auf einer VAXstation 3100 gemessen. Diese Maschine hat etwa 2.5 VAX-MIPS und ist in ihrer Leistungsfähigkeit vergleichbar mit einer SUN 3. Die Zeitmessung erfolgte mit einer Funktion, die die vom Benutzerprozeß aufgenommene Rechenzeit liefert.

Bei der ZZ-Suche ist keine Vorverarbeitung nötig wie bei vielen anderen Verfahren, etwa Berechnung von Konfigurationsraum-Hindernissen [13, 22, 32, 38], Darstellung des freien Konfigurationsraums [5, 41], Erzeugung künstlicher Potentialfunktionen [3, 39] oder Aufbau anderer Hilfsstrukturen. Die Gesamtrechenzeit beim ZZ-Verfahren enthält also alle notwendigen Verarbeitungsschritte.

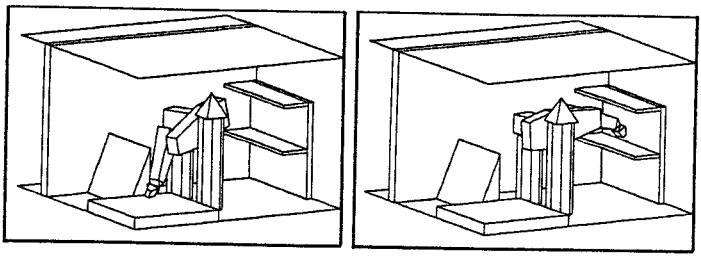

Abbildung 5.1: Aufgabe REGAL6. Beispiel einer Transportaufgabe für einen PUMA-Roboter mit sechs Freiheitsgraden. Die Orientierung der Hand wird bei der Suche einer Bewegung voll berücksichtigt. Für die Erzeugung einer kollisionsfreien Bewegung von der Startstellung (links) in die Zielstellung (rechts) benötigt das ZZ-Verfahren 70 Sekunden. Diese Aufgabe wird ohne zufälliges Zwischenziel deterministisch gelöst, und zwar beim zweiten Versuch vom Ziel zum Start hin.

5.2 Beispielläufe mit repräsentativen Aufgaben

Um die Leistungsfähigkeit des ZZ-Verfahrens in der Praxis zu untersuchen, wurden vier Umwelten modelliert. Sie wurden so ausgewählt, daß sie repräsentativ sind für jeweils eine Klasse von Aufgaben. Im folgenden werden diese Umwelten vorgestellt und die Ergebnisse diskutiert.

In den Abbildungen 5.1 und 5.2 ist eine Umgebung mit einem Regallager dargestellt. Der PUMA-Roboter soll ein Teil von dem Podest im Vordergrund in das Regelfach transportieren. Dabei wird er sowohl von der niedrigen Decke als auch von dem Turm im Vordergrund und der Wand im Hintergrund in seiner Bewegungsfähigkeit stark eingeschränkt.

Bei der Aufgabe REGAL6 in Abbildung 5.1 wurde der PUMA-Roboter mit allen sechs Gelenken und Freiheitsgraden modelliert. Zur Planung einer kollisionsfreien Bewegung werden vom ZZ-Verfahren 70 Sekunden benötigt. Die Lösung wird in diesem Fall sogar direkt von der zielgerichteten Suche allein erbracht; ein zusätzliches Zwischenziel muß nicht erzeugt werden. Deshalb ist das Ergebnis und auch die Rechenzeit hier deterministisch und ohne Zufallsvariation.

Die Aufgabe REGAL3 in Abbildung 5.2 stellt die gleichen Voraussetzungen wie die Aufgabe REGAL6, nur mit dem Unterschied, daß statt sechs Freiheitgraden nur noch die ersten drei berücksichtigt werden. Die Hand des PUMA-Roboters wurde zusammen mit der kleinen Last von einem Hüllkörper umschrieben, der alle möglichen Orientierungen des Handgelenks umfaßt. Die durchschnittliche Rechenzeit beträgt 207 Sekunden, wobei im Schnitt 3,4 Zwischenziele für die Lösung erzeugt werden müssen. Die Verteilungsfunktionen der Rechenzeiten und der Anzahl der Zwischenziele für 1000 Läufe sind in

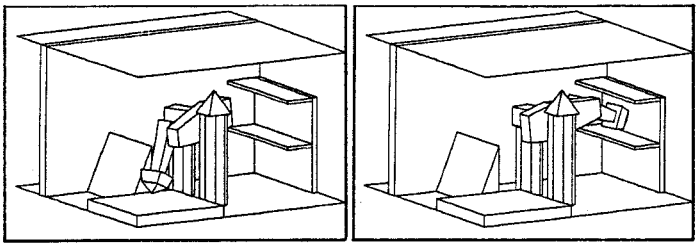

Abbildung 5.2: Aufgabe REGAL3. Das ist die gleiche Aufgabe wie REGAL6 (Abbildung 5.2) mit dem Unterschied, daß nur die ersten drei Gelenke (Freiheitsgrade) berücksichtigt werden. Die Hand und eine kleine Last finden unabhängig von der Orientierung des Handgelenks im Hüllkörper am Ende des Unterarms Platz. Die durchschnittliche Rechenzeit beträgt 207 Sekunden, was überrascht, weil es länger dauert als im Fall mit sechs Freiheitsgraden.

Abbildung 5.3 zu sehen.

Andere, klassische Konfigurationsraum-Methoden lösen solche Aufgaben mit nur drei Freiheitsgraden in 30 Sekunden [32] bis mehreren Minuten [22], sind aber wegen ihrer exponentiellen Komplexität für mehr als vier Freiheitsgrade nicht mehr einsetzbar. Im Gegensatz dazu braucht die ZZ-Suche für den sechsdimensionalen Fall nur etwa ein Drittel der durchschnittlichen Zeit des dreidimensionalen Falls.

Die Behauptung, daß die Behandlung von mehr Freiheitsgraden bei der Bewegungsplanung (in praktisch relevanten Umgebungen) die Aufgabe viel schwerer lösbar macht [9], wird durch das gezeigte Beispiel nicht gestützt. Wenn man den voluminösen Hüllkörper am Ende des Arms betrachtet, ist auch intuitiv klar, daß im Gegenteil zur Behauptung der angeblich leichtere dreidimensionale Fall durchaus schwieriger sein kann, weil zum Beispiel Engstellen für den vergrößerten Arm nun noch enger erscheinen. Der freie Konfigurationsraum F wird kleiner, und die Anzahl möglicher Lösungen wird stark reduziert.

Die zweite Beispielumgebung (Abbildung 5.4) soll eine Simulation repräsentativer Aufgaben für Roboteranwendungen in Verbindung mit numerisch gesteuerten Werkzeugmaschinen und Bearbeitungszentren ermöglichen. Wesentlich an diesem Einsatz ist das Hineingreifen des Roboters in Maschinenöffnungen.

Es wurden zwei Bewegungsplanungen mit dieser Umgebung durchgeführt: das Beladen der Maschine aus einer nahe gelegenen Kiste (Aufgabe MASCHINE1) und das Entladen in eine weiter entfernt stehende Kiste (Aufgabe MASCHINE2).

Die Lösung der Belade-Aufgabe MASCHINE1 dauert durchschnittlich 60 Sekunden, wobei im Schnitt 1,9 Zwischenziele zufallsgesteuert erzeugt werden müssen. Die genauen Verteilungsfunktionen der Laufzeiten und der Anzahl der Zwischenziele für 1000 Läufe sind in Abbildung 5.5 gezeigt.

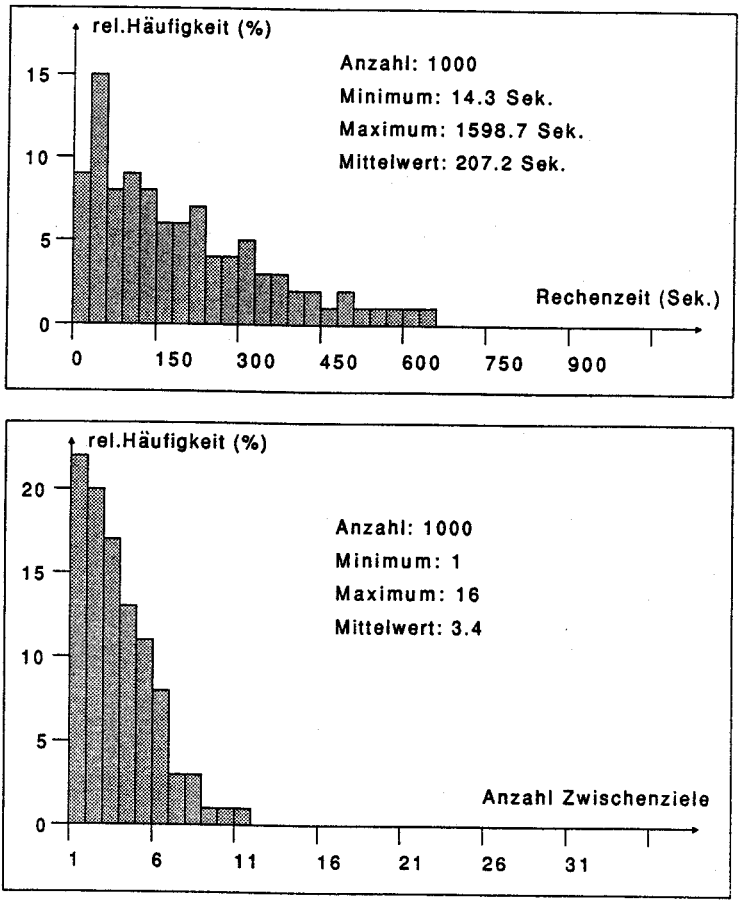

Abbildung 5.3: Verteilung der Rechenzeiten und der Anzahl der Zwischenziele für die Aufgabe REGAL3 (Abbildung 5.2).

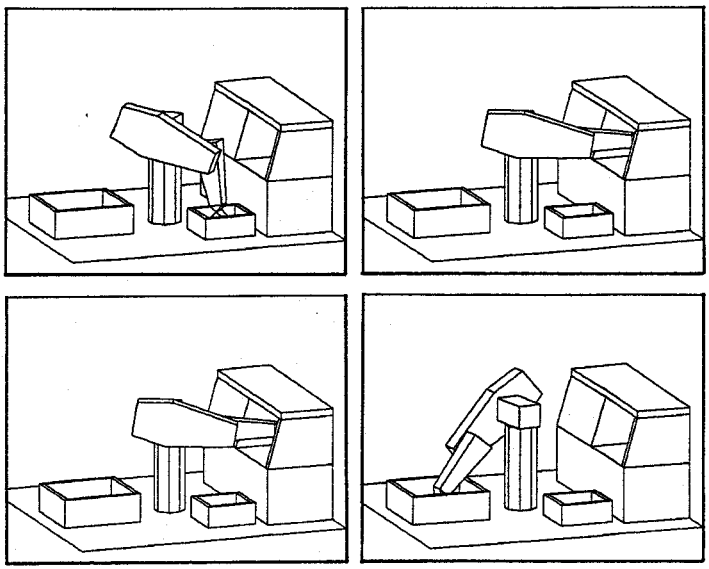

Abbildung 5.4: Aufgaben MASCHINE1 und MASCHINE2. Dies sind zwei Beispiele für Bewegungsplanungsaufgaben in einer Fabrikumgebung. Der PUMA-Roboter wird mit allen sechs Freiheitsgraden behandelt. Die erste Aufgabe MASCHINE1 (oben) besteht im Beladen der Maschine mit einem Rohling oder einem Werkzeug aus der kleinen Kiste; in der zweiten Aufgabe MASCHINE2 (unten) soll das bearbeitete Teil aus der Maschine genommen und in der großen Kiste abgelegt werden. Die Rechenzeit für die obere Aufgabe MASCHINE1 beträgt im Durchschnitt 60 Sekunden. Für die untere Aufgabe MASCHINE2 werden 8 Sekunden benötigt; hier wird die Bewegung ohne weiteres Zwischenziel gefunden.

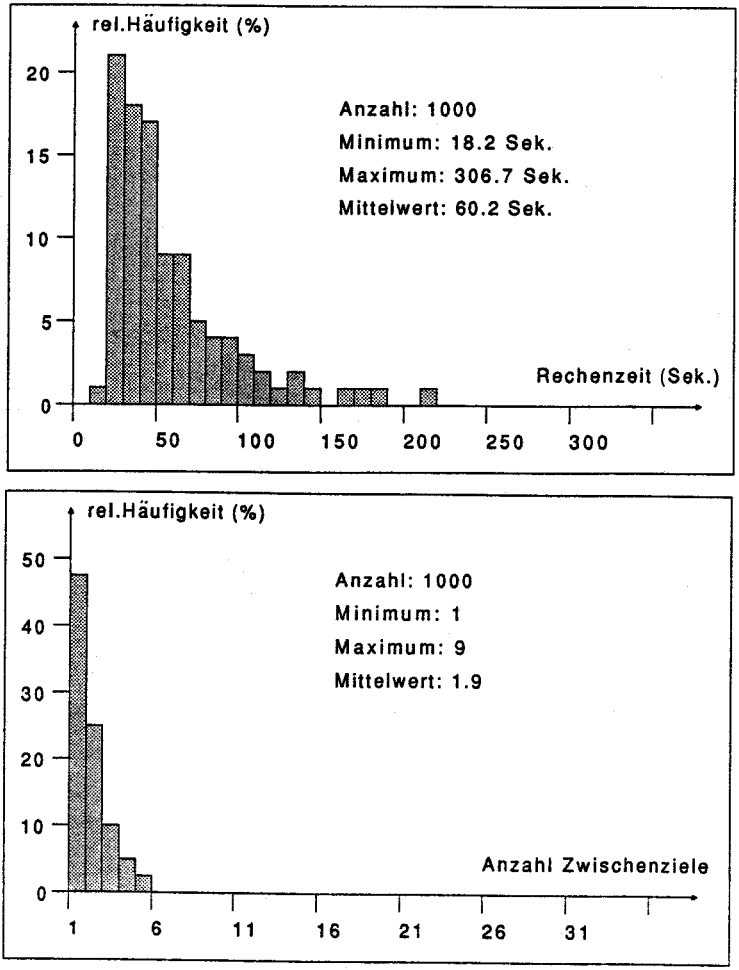

Abbildung 5.5: Verteilung der Rechenzeiten und der Anzahl der Zwischenziele für die Aufgabe MASCHINE1 (Abbildung 5.4 oben).

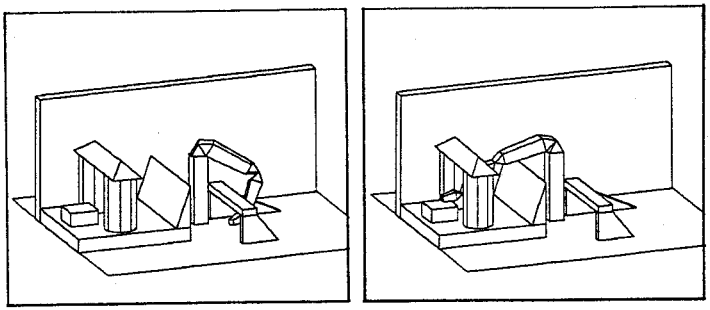

Abbildung 5.6: Aufgabe SCHLANGE. Das ist ein Beispiel einer Bewegungsplanungsaufgabe für einen schlangenartigen Manipulator mit sechs Freiheitsgraden. Die durchschnittliche Rechenzeit beträgt 241 Sekunden. Dieser Roboter kann nicht sinnvoll in einen dreigelenkigen Arm und eine dreigelenkige Hand entkoppelt werden; deshalb scheitern hier alle Verfahren, die auf einer Entkopplung basieren und nur mit drei oder vier Gelenken arbeiten.

Das Entladen gemäß Aufgabe MASCHINE2 benötigt nur 8 Sekunden. Hier kann die Planung einer kollisionsfreien Bewegung direkt ohne Verwendung eines mit Hilfe des Zufalls erzeugten Zwischenziels erfolgen. In diesem Fall läuft die Bewegungsplanung deterministisch ab, und die Rechenzeit ist immer gleich lang.

Die üblichen Manipulatoren, zum Beispiel die PUMA-Typen, mit sechs Freiheitsgraden sind so konstruiert, daß die ersten drei Freiheitsgrade, also Gelenke, hauptsächlich zum Einstellen der Position des Effektors dienen. Die restlichen drei Freiheitsgrade sind im Handgelenk zusammengefaßt und dienen dann nur noch zum Einstellen der Orientierung des Effektors; sie haben nur geringe Wirkung auf die Position des Effektors.

Diese Entkopplung von Positionierung und Orientierung des Effektors in jeweils drei Freiheitsgrade ist bei anderen Konstruktionen nicht immer möglich. Bei redundanten Manipulatoren sind bereits bei der Einstellung der Position mehr als drei Gelenke beteiligt. Die dreidimensionale Bewegungsplanung für solche Manipulatoren ist noch völlig unerforscht, bei zweidimensionalen Umgebungen existieren Ansätze [7].

Ein Beispiel dafür ist der schlangenähnliche Roboter in Abbildung 5.6. Er verwendet sechs Gelenke zur Einstellung einer bestimmten Position. Hier muß man in jedem Fall mit allen sechs Freiheitsgraden rechnen, weil sonst die große Beweglichkeit des Arms drastisch reduziert wird.

Deswegen sind alle klassischen Methoden, die eine Reduktion der Anzahl Freiheitsgrade auf drei oder vier voraussetzen, zur Bewegungsplanung dieser Roboter nicht einsetzbar. Das ZZ-Verfahren braucht zur Lösung der abgebildeten Bewegungsplanungsaufgabe für den schlangenartigen Roboter durchschnittlich 241 Sekunden Rechenzeit. Dabei werden im Mittel 6,1 Zwischenziele per Zufall erzeugt. Die genauen Verteilungskurven der Rechenzeiten und

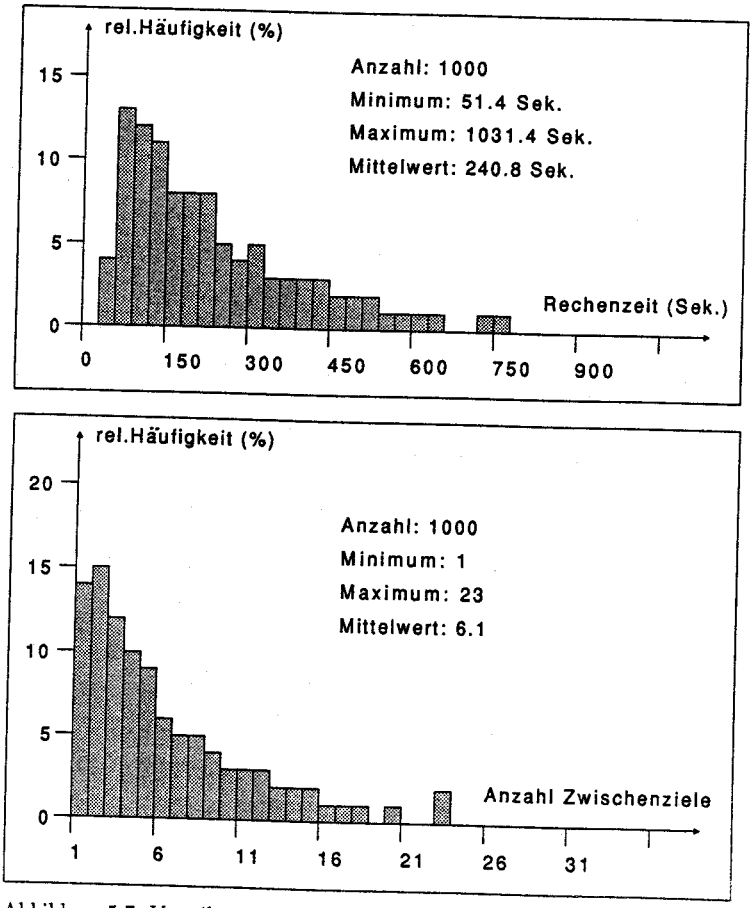

Abbildung 5.7: Verteilung der Rechenzeiten und der Anzahl der Zwischenziele für die Aufgabe SCHLANGE (Abbildung 5.6).

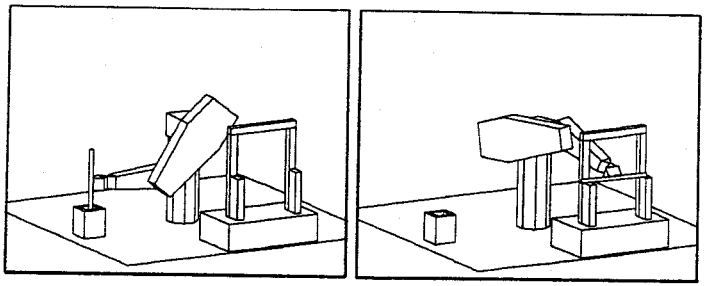

Abbildung 5.8: Aufgabe TOR. Diese Bewegungsplanungsaufgabe ist sehr schwierig, weil die Form und Größe der Last — ein langer dünner Stab — und die Enge des Tors eine geeignete Umorientierung des Effektors erfordern. Eine Lösung ist deshalb nur möglich, wenn man bei der Planung alle sechs Freiheitsgrade des PUMA-Roboters berücksichtigt. Die durchschnittliche Rechenzeit beträgt 224 Sekunden.

der Anzahl der Zwischenziele sind in Abbildung 5.7 angegeben.

Wie beim vorigen Beispiel mit dem schlangenartigen Roboter ist auch bei der Aufgabe TOR in Abbildung 5.8 eine Behandlung aller sechs Freiheitsgrade wesentlich für die Lösung. War im vorigen Beispiel die Enkopplung von Arm und Hand nicht möglich, so ist hier die Vernachlässigung der Handorientierung nicht erlaubt, weil die Größe der Last nicht mehr als unbedeutend angesehen werden darf.

Bei kleinen Greifern und kleinen Lasten wird bei reinen Orientierungsänderungen die Raumbelegung des Manipulators gleich bleiben, wenn man das Handgelenk, den Greifer und die Last durch einen Hüllkörper modelliert, der diese drei Komponenten bei allen erlaubten Orientierungen umfaßt (vergleiche Abbildung 5.2). Der Genauigkeitsverlust ist in diesem Fall — mit einer kleinen Last — bei bestimmten großräumigen Anwendungen erträglich. Dafür spart man die Planung der Bewegung der letzten drei Achsen ein, was für viele der bekannten Methoden eine absolute Voraussetzung ist.

Hat man dagegen eine sperrige Last, etwa einen relativ langen, dünnen Stab wie bei der Aufgabe TOR in Abbildung 5.8, so ist die Orientierung des Effektors während der Bewegung wesentlich, wenn man einen kollisionsfreien Weg sucht. Man muß dann unbedingt alle sechs Freiheitsgrade in der Bewegungsplanung berücksichtigen. Eine zur Vernachlässigung der Orientierung geeignete Umhüllung der Last wäre in diesem Fall so groß, daß kein kollisionsfreier Weg mehr existierte.

Die Bewegungsplanung mit der ZZ-Suche findet eine Lösung für die Aufgabe TOR in durchschnittlich 224 Sekunden. Es werden dabei im Mittel 5,3 zufallsgesteuert erzeugte Zwischenziele benötigt. Die genaue Verteilung der Rechenzeiten und der Anzahl der Zwischenziele für 1000 Läufe ist der Abbildung 5.9 zu entnehmen.

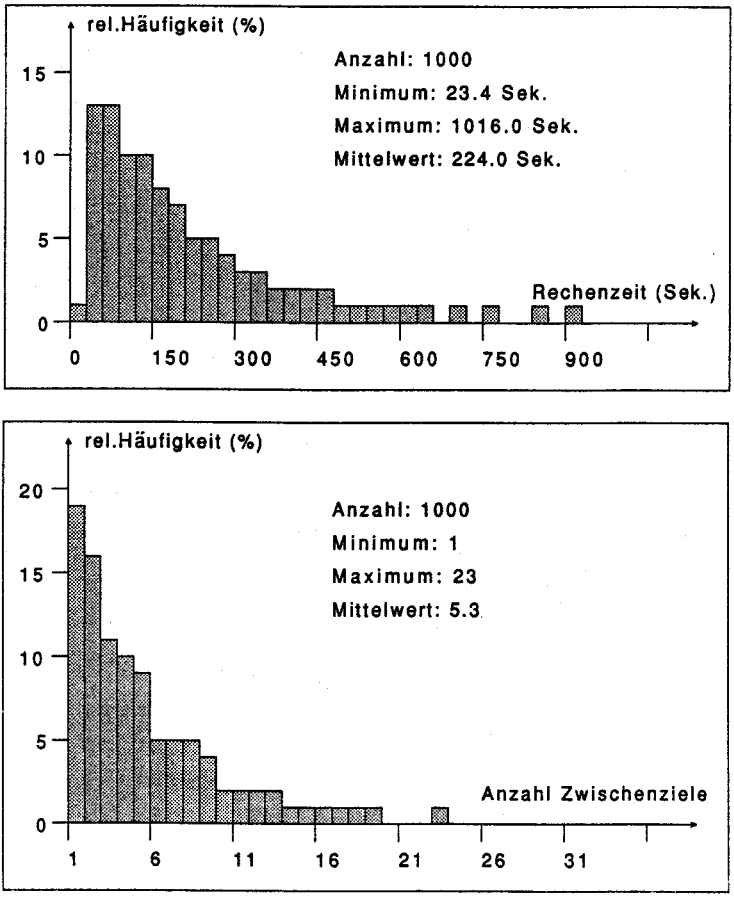

Abbildung 5.9: Verteilung der Rechenzeiten und der Anzahl der Zwischenziele für die Aufgabe TOR (Abbildung 5.8).

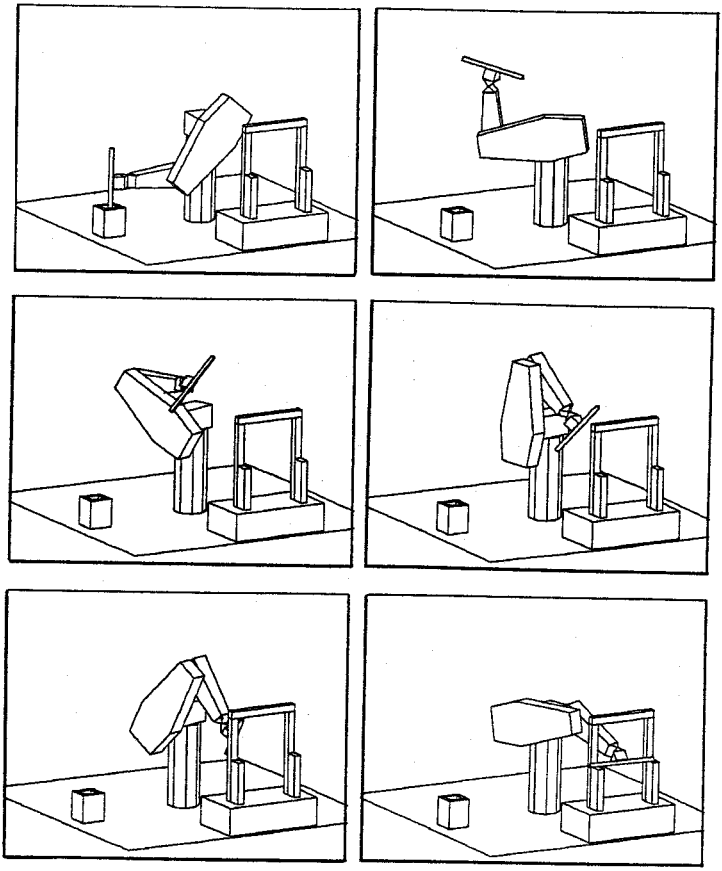

Abbildung 5.10: Eine Folge von Ausschnitten aus einer typischen, mit dem ZZ-Verfahren erzeugten Bewegung für die Aufgabe TOR (Abbildung 5.8). Diese Lösung wurde in 226 Sekunden berechnet, wobei sechs zufällige Zwischenziele erzeugt wurden. Die Gesamtbewegung besteht aus 234 Einzelschritten.

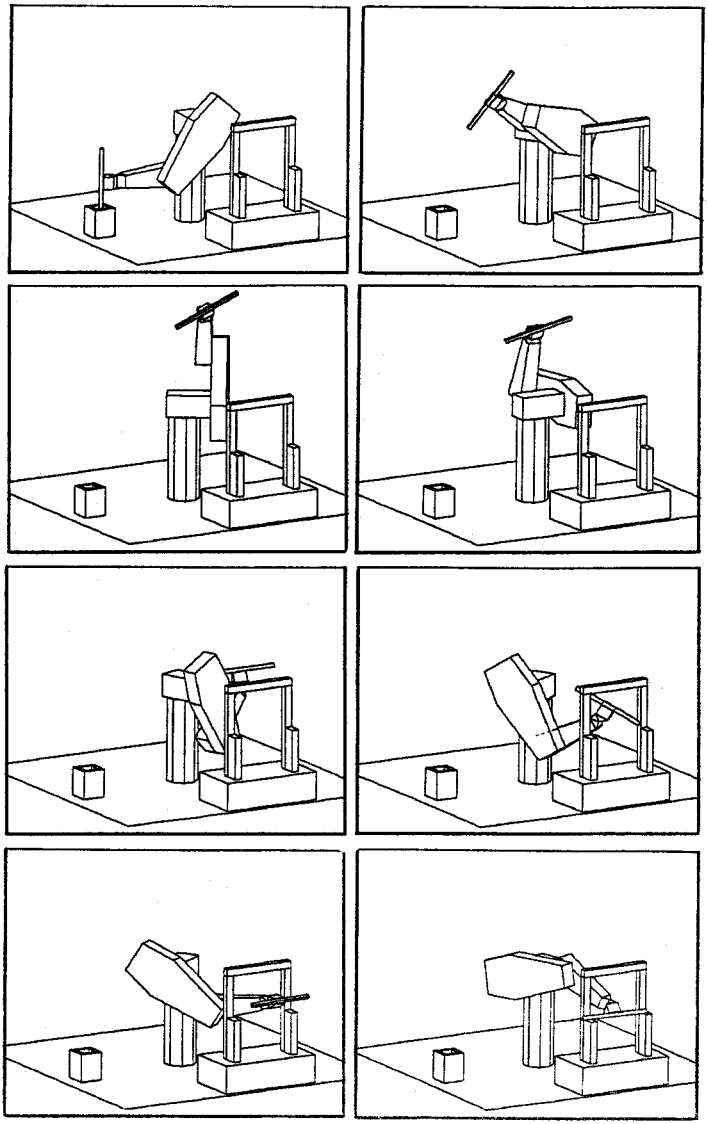

Abbildung 5.11: Eine Folge von Ausschnitten aus einer anderen Lösung für die Aufgabe TOR (Abbildung 5.8). Diese Bewegung wurde in 154 Sekunden berechnet, wobei drei zufällige Zwischenziele erzeugt wurden. Die Gesamtbewegung besteht aus 433 Einzelschritten (vergleiche mit Abbildung 5.10).

Zwei beliebig aus den 1000 Testläufen herausgegriffene Lösungen für diese Aufgabe TOR sind in den Abbildungen 5.10 und 5.11 veranschaulicht. Der Vergleich dieser beiden Bewegungen vermittelt einen Eindruck von der Variationsbreite möglicher Lösungen für eine Aufgabe. Die erste Lösung (Abbildung 5.10) entspricht in Rechenzeit und Anzahl der Zwischenziele etwa dem Durchschnitt. Die zweite, in Abbildung 5.11 gezeigte Bewegung wurde schneller und mit weniger Zwischenzielen erzeugt; bei ihr legt der Effektor im Objektraum einen fast doppelt so langen Weg zurück wie in der ersten Lösung.

5.3 Vergleich mit Ergebnissen anderer Autoren

Wie schon erwähnt, behandelten bisher nur wenige Autoren die Bewegungsplanung von Manipulatoren unter dem Gesichtspunkt der praktischen Einsetzbarkeit. Viele Implementierungen sind auf zweidimensionale Umwelten beschränkt, und etliche dreidimensionale Ansätze sind ohne Angaben zu ihrer praktischen Leistungsfähigkeit [20, 25]. So bleiben nur sehr wenige Implementierungen für dreidimensionalen Umgebungen, die in ihrer Leistung mit dem ZZ-Verfahren verglichen werden können.

In der Tabelle 5.1 sind die Rechenzeiten von Bewegungsplanern für dreidimensionale Umwelten zusammengefaßt. Ein direkter Vergleich der Angaben ist sehr schwierig, da sich sowohl die erlaubten Umwelteigenschaften wie auch die Rechner-Geschwindigkeit stark unterscheiden.

Um wenigstens einen gewissen Anhaltswert für die Leistungsfähigkeit der einzelnen Methoden zu erhalten, wurden die originalen Rechenzeiten in der letzten Spalte normiert durch Multiplikation mit Faktoren, die den Unterschied zwischen der jeweils verwendeten Maschine und einer VAX 11/780 (mit ca. 1 MIPS) ausgleichen. Diese Faktoren wurden aus den beiden Benchmark-Programmen Dhrystone 2.0 [40] und Xfroot [28] gewonnen. Obwohl die Ausgleichsfaktoren je nach verwendetem Übersetzer und Anweisungsmix der Programme in gewissen Bereichen schwanken können, geben sie doch die Größenordnung der Leistungsdifferenzen deutlich wieder.

Alle fremden aufgeführten Verfahren, außer dem von Barraquand und Latombe, basieren wesentlich auf einer Entkopplung der die Hand in Position bringenden Freiheitsgrade von den im wesentlichen nur orientierungsändernden Freiheitsgraden. Weiter dürfen wegen der exponentiellen Abhängigkeit der Rechenzeit und des Speicherbedarfs für die explizite Darstellung des freien Konfigurationsraums F von der Anzahl Freiheitsgrade höchstens drei bis vier Gelenke zur Positionsveränderung beitragen. Eine eventuelle Berücksichtigung der verbleibenden Orientierungsfreiheitsgrade erfolgt unter der heuristischen Annahme eines verhältnismäßig kleinen Hand-Last-Volumens.

So sind die von den Autoren veröffentlichten Beispielaufgaben denn auch so konstruiert, daß sie den Bedingungen genügen. Die Schwierigkeit der meisten fremden Aufgaben ist mit den Aufgaben REGAL6 oder MASCHINE1 und

Autor(en)	Aufgabe, Umwelt, Roboter, Anzahl Freiheitgrade (FG)	Rechenzeit	
		Gesamt, Rechner	normiert VAX 11/780
Barraquand, Latombe [3]	128 * 128 * 128 Voxelraster als Umwelt, Strichroboter mit 31 FG.	15 Min., DECstation 3100	3 Std.
Brooks [5]	einige senkrechte prismat. Stäbe, PUMA mit 3+1 FG.	< 1 Min. Lisp Machine	(30 Sek.)
Donald [9]	5 Quader und 1 Prisma, fliegender Hammer mit 3+3 FG.	einige Std. Symbolics 3600	(2 Std.)
Faverjon, Tournassoud [13]	weiträumige Transportbewegung, Knickarmroboter mit 3 FG.	5 Min.+10 Sek. (SUN 3)	(13 Min.)
Faverjon, Tournassoud [13]	sperrige Last (Stab) durch Tor, Knickarmroboter mit 3+3 FG.	(einige Std.) +einige Min. (SUN 3)	(10 Std.)
Hasegawa [21]	7 Quader, ETA-2 mit 3 FG.	2.5 Min.+10 Sek. VAX 11/780	2.6 Min.
Hasegawa, Terasaki [22]	Gehäuse mit Öffnung zum Reingreifen, ETA-2 mit 3+3 FG.	7 Min.+2 Min. VAX 11/780	9 Min.
Lozano-Pérez und andere [32]	großräumige Transportbewegung mit wenigen Hindern., PUMA, 3 FG.	30 Sek., (SUN 3)	(1.3 Min.)
Siméon [38]	Wand mit Loch zum Durchgreifen, Knickarmroboter mit 4 FG.	50 Min.+10 Sek. SUN 3	2 Std.
Glavina	REGAL6, PUMA mit 6 FG.	70 Sek. VAXstation	2.8 Min.
Glavina	MASCHINE1, PUMA mit 6 FG.	60 Sek. VAXstation	2.4 Min.
Glavina	MASCHINE2, PUMA mit 6 FG.	8 Sek. VAXstation	19 Sek.
Glavina	SCHLANGE, schlangenartiger Roboter, 6 FG.	241 Sek. VAXstation	9.6 Min.
Glavina	TOR, PUMA mit 6 FG. Stab durch Tor beweg.	224 Sek. VAXstation	8.9 Min.

Tabelle 5.1: Zusammenfassung von Rechenzeiten bekannter Bewegungsplaner für dreidimensionale Umwelten. Bei Verfahren mit Vorverarbeitungsschritten ist die Gesamtrechenzeit als Summe von Vorverarbeitungszeit und Zeit für die eigentliche Wegsuche angegeben. Klammern kennzeichnen eigene Schätzungen für Angaben, die in den Artikeln nicht oder nicht explizit enthalten sind.

MASCHINE2 vergleichbar. Die dafür benötigten normierten Rechenzeiten liegen bei den fremden Verfahren zwischen 30 Sekunden und 13 Minuten, und beim ZZ-Verfahren zwischen 19 Sekunden und 2,8 Minuten. Die Leistung des Bewegungsplaners mit der ZZ-Suche ist also bei diesen einfacheren Aufgaben leicht besser als die der fremden Verfahren.

Dagegen sind die beiden letzten Aufgaben aus dem vorigen Abschnitt — der schlangenartige, nicht entkoppelbare Roboter (Aufgabe SCHLANGE) und die Bewegung eines langen Stabes durch ein Tor (Aufgabe TOR) — für die meisten Verfahren prinzipiell unlösbar. Das ZZ-Verfahren liefert auch in diesen schwierigeren Fällen eine kollisionsfreie Bewegung in wenigen Minuten, was einer normierten Rechenzeit von unter 10 Minuten entspricht.

Einzig Faverjon und Tournassoud versuchen, durch eine zusätzliche Heuristik den Fall einer sperrigen Last zu lösen; dabei geben sie selbst zu, daß ihr Algorithmus nicht immer eine Lösung findet, aber „viele praktische Fälle löst" [13]. Dieses Verfahren scheint zudem Laufzeiten zu benötigen, die nicht akzeptabel sind. Das läßt sich aus der Tatsache erschließen, daß bereits die Wegesuche, die gewöhnlich bei diesen Techniken nur einen geringen Anteil an der Gesamtrechenzeit hat, bereits einige Minuten dauert. Der Zeitbedarf für die aufwendigere Vorverarbeitung wird von den Autoren folgerichtig auch lieber nicht erwähnt. Das Verhältnis von Wegesuchzeit zu Vorverarbeitungszeit beträgt bei dem anderen Beispiel von Faverjon und Tournassoud 1 zu 30, so daß eine Gesamtrechenzeit für die komplexe Aufgabe in der Größenordnung von einigen Stunden auf dem Originalrechner erwartet werden darf, was einer normierten Rechenzeit von etwa 10 Stunden entspricht.

Der Algorithmus von Barraquand und Latombe hat diese Einschränkung nicht, er behandelt alle Freiheitsgrade gleichberechtigt. Sein Nachteil scheint einerseits die Darstellung der Umwelt als grob diskretisiertes, dreidimensionales Feld aus Volumenelementen, und andererseits die Darstellung des Manipulators als bloßes Strich-Gebilde ohne Volumen. Obwohl der Planer die bisher unerreichte Anzahl von 31 Freiheitsgraden berücksichtigen kann, ist er im jetzigen Zustand nicht für einen praktischen Einsatz brauchbar, weil wirkliche Roboter nicht aus dünnen Linien bestehen. Eine Behebung dieser Nachteile ist sicher möglich, schlägt sich aber wohl deutlich in den benötigten Rechenzeiten nieder. Hier muß man zukünftige Weiterentwicklungen abwarten, um weitere Vergleiche ziehen zu können.

5.4 Zusammenfassende Bewertung

Es wurden auch Testläufe ohne Hindernisumgehung durch Gleitschritte durchgeführt. Da ein Gleitschritt etwa fünf- bis zehnmal soviel Rechenzeit benötigt wie ein normaler Schritt, lag die Vermutung nahe, daß man in dieser eingesparten Zeit viel mehr Zwischenziele ausprobieren könne. Das ist zwar richtig, aber außer in sehr einfachen Umgebungen mit wenig Hindernissen und viel Freiraum um Start und Ziel herum kann diese simple Suchmethode ohne Gleitschritt kaum kollisionsfreie Bewegungen in vernünftiger Zeit finden.

Auch die direkte Suche mit Gleitschritten zur Hindernisumgehung allein erbringt keine befriedigende Leistung, weil sie durch Sackgassen gefährdet ist.

Wie die Ergebnisse der Testläufe zeigen, ist die Kombination von zielgerichteter Suche und zufallsgesteuerter Zwischenzielerzeugung in der ZZ- Suche ein sehr leistungsfähiges Verfahren zur Planung von Bewegungen für Manipulatoren mit sechs Freiheitsgraden in dreidimensionalen Arbeitsräumen.

Der Vergleich mit den bekannten Implementierungen anderer Autoren ergibt, daß das ZZ-Verfahren in einfachen Fällen, wie Transportbewegungen in weitgehend freien Räumen (Abbildungen 5.1 und 5.4) mit den schnellen Verfahren für drei Freiheitsgrade konkurrieren kann. Eine deutliche Überlegenheit der ZZ-Suche kann bei den komplexeren Aufgaben festgestellt werden, wo die einschränkenden Voraussetzungen — wie Entkopplung von Arm und Hand, Zulassen von nur kleinen Lasten, Behandlung von nur drei oder vier Freiheitsgraden — für diese Verfahren nicht mehr gemacht werden können (Abbildungen 5.6 und 5.8). Hier erzeugt das ZZ-Verfahren Bewegungen in wenigen Minuten, wogegen die anderen Ansätze Lösungen nur noch in bestimmten Fällen („many practical cases“ [13]) oder gar nicht liefern können.

Alle Angaben zu Rechenzeiten der ZZ-Suche waren auf eine VAXstation 3100 bezogen. Mit der Geschwindigkeit moderner RISC-Prozessoren, die mehr als fünf mal so schnell sind wie eine VAXstation, würde der Bewegungsplaner Laufzeiten erreichen, die für die hier diskutierten Beispiele deutlich unter einer Minute liegen. Damit erreicht man einen zeitlichen Rahmen, der einen sinnvollen und nützlichen Einsatz in Roboterprogrammiersystemen ermöglicht.

Kapitel 6

Ausblick

Der beschriebene Suchalgorithmus arbeitet ohne Berücksichtigung globaler Aspekte. Deswegen kann die gefundene Bewegung prinzipiell nicht optimal sein. Falls gewünscht oder benötigt, können eine Verkürzung der Gesamtbewegung, eine Glättung von abrupten Richtungsänderungen oder andere Optimierungen, etwa kürzeste Bewegungszeit, geringster Energieverbrauch usw., durch einen Nachbearbeitungsschritt erfolgen. Die bisherige Implementierung arbeitet ohne zusätzliche Optimierung.

Alle diese nachträglichen Verbesserungen der Bewegung müssen sinnvollerweise in ihrer Zeitkomplexität mit dem eigentlichen Bewegungsplaner vergleichbar sein, damit die Gesamteffizienz nicht verschlechtert wird. Deswegen bieten sich hier nur Verfahren an, die durch Abwandlung der gefundenen Bahn lokale Suboptima erreichen.

Eine gewisse Optimierung kann man auch dadurch erreichen, daß man die Suche nach dem ZZ-Verfahren so lang weiterlaufen läßt, bis eine gewisse Anzahl verschiedener Lösungen gefunden ist. Daraus sucht man dann je nach Kriterium die beste aus.

Für Einsätze, in denen eine bestimmte Bewegung sehr oft ausgeführt werden muß, ist ein höherer Aufwand an Planungszeit akzeptabel. Hier kann man mehrere verschiedene gefundene Bewegungen einzeln optimieren, und dann aus dieser Menge durch einfachen Vergleich die beste Lösung für die Steuerung des Manipulators verwenden. Die Endlösung wird im allgemeinen sehr nahe am globalen Optimum liegen.

Die jetzige Implementierung verwirft die Teilwege von mißglückten Suchvorgängen, die in Sackgassen endeten. Die Endpunkte dieser Teilwege könnten auch als neue Zwischenziele Verwendung finden. Problematisch dabei ist, daß unter Umständen viel mehr solche Zwischenziele entstehen als man auf Erreichbarkeit testen kann. Im Extremfall entsteht beim Test eines neuen Punktes für jedes bereits vorhandene Zwischenziel ein neues Sackgassenzwischenziel. Dann ist für ein gutes Funktionieren der Bewegungsplanung eine geeignete Auswahlstrategie wesentlich. Denkbar wäre zum Beispiel die Verwendung der Sackgassenpunkte als rein passive Ziele, zu denen von echten Zwischenzielen aus ein Weg gesucht wird, die aber nicht direkt untereinander verknüpft werden.

Den größten Anteil an der Gesamtrechenzeit hat der Kollisionstest mit Operationen auf Vektoren und Matrizen. Diese Operationen werden in der momentanen Implementierung durch Prozeduren in Hochsprache realisiert, eignen sich aber wegen ihrer einfachen und systematischen Ablaufstruktur ausgezeichnet für eine Kodierung in Mikroprogrammen. Ähnlich wie in der graphischen Datenverarbeitung sind durch dezidierte Koprozessoren („geometry engine") deutliche Geschwindigkeitsverbesserungen zu erwarten.

Eine andere Möglichkeit, die Kollisionsprüfung zu beschleunigen, besteht darin, die Glieder des Manipulators gleichzeitig parallel und unabhängig voneinander auf Durchschnitt mit den Umwelthindernissen zu testen. Je nach der verfügbaren Anzahl parallel arbeitender Prozessoren kann man einem Prozessor auch nur Teile eines Glieds zuordnen, im Extremfall wird jede Fläche des Arms von je einem Prozessor auf Kollision geprüft.

Da die Zwischenzielerzeugung der ZZ-Suche zufällig ist, variiert die zur Lösung einer Aufgabe nötige Rechenzeit in einem gewissen Bereich. In manchen Beispielaufgaben unterscheiden sich die kurzen von den langen Zeiten um den Faktor 10. Durch mehrfachen, parallelen Ablauf des Algorithmus mit gleichen Suchaufgaben kann man die mittlere Suchzeit verringern. Im Mittel bekommt man statt der durchschnittlichen Rechenzeiten dann eher die kurzen Zeiten aus dem Streuungsbereich.

Zum Beispiel die parallele Bewegungsplanung auf zwei Rechnern oder Koprozessoren für die Aufgabe „Stab durch Tor" (TOR) ergibt einen Erwartungswert der Rechenzeit von 129 Sekunden — statt 224 Sekunden bei nicht paralleler Bearbeitung. Stärkere Parallelisierung reduziert die durchschnittliche Rechenzeit bis zur ersten gefundenen Lösung weiter: vier Prozessoren brauchen durchschnittlich 80 Sekunden und acht Prozessoren 54 Sekunden.

Nimmt man acht RISC-Prozessoren — die heute bereits mehr als fünf mal so schnell rechnen wie die für die Beispielmessungen verwendete VAXstation 3100 —, so scheinen leistungsfähige Echtzeitbewegungsplaner für Manipulatoren in der näheren Zukunft realisierbar.

Anhang

Anhang A: Drehung eines Punkts $p \in \mathbf{R}^3$ um eine Ursprungsachse a mit Winkel ϕ

Die Achse ist durch ihren Richtungvektor $a = (a_1, a_2, a_3)$ mit $\mid a \mid = 1$ gegeben, die Drehrichtung bildet bei positivem Winkel ϕ eine Rechtsschraube mit der Achsenrichtung. Der gedrehte Punkt p' ergibt als Produkt zu

$$p' := R(a, \phi)p,$$

wobei

$$R(a,\phi) = \begin{pmatrix} a_1^2(1-c)+c & a_2a_1(1-c)-a_3s & a_3a_1(1-c)+a_2s \\ a_1a_2(1-c)+a_3s & a_2^2(1-c)+c & a_3a_2(1-c)-a_1s \\ a_1a_3(1-c)-a_2s & a_2a_3(1-c)+a_1s & a_3^2(1-c)+c \end{pmatrix}$$

und $c = \cos\phi$ und $s = \sin\phi$.

Anhang B: Abstand zweier Punkte $a, b \in \mathbf{R}^n$

Der Abstand der zwei Punkte a und b nach der Euklidischen Metrik ergibt sich zu

$$d(a,b) := \sqrt{\langle (a-b), (a-b) \rangle} = \sqrt{(a_1-b_1)^2 + \ldots + (a_n-b_n)^2}.$$

Anhang C: Abstand eines Punkts $a \in \mathbf{R}^n$ von einer Hyperebene E

Die Ebene $E = \{x \in \mathbf{R}^n : \langle n_E, x \rangle = d_E\}$ sei durch ihre Normale $n_E \in \mathbf{R}^n$ mit $\mid n_E \mid = 1$ und ihren gerichteten Ursprungsabstand d_E gegeben. Dann berechnet sich der gerichtete Abstand zwischen dem Punkt a und der Ebene E zu

$$d(a, E) := \langle n_E, a \rangle - d_E.$$

Anhang D: Schnitt einer Geraden g mit einer Hyperebene E

Die Gerade g,

$$g = \{x \in \mathbf{R}^n : x = a + \lambda(b - a), \lambda \in \mathbf{R}\},$$

sei durch zwei Punkte a und b gegeben mit $a, b \in g$ und $a \neq b$. Die Ebene E,

$$E = \{x \in \mathbf{R}^n : \langle n_E, x\rangle = d_E\},$$

sei durch ihre Normale $n_E \in \mathbf{R}^n$ mit $\mid n_E \mid = 1$ und ihren gerichteten Ursprungsabstand d_E gegeben. Dann berechnet sich der Schnittpunkt $s \in \mathbf{R}^n$ von Gerade g und Ebene E zu

$$s = g \cap E := a + (a - b)\frac{d_E - \langle n_E, a\rangle}{\langle n_E, (a - b)\rangle},$$

sofern $\langle n_E, (a - b)\rangle \neq 0$ gilt. Andernfalls gibt es keinen gemeinsamen Schnittpunkt.

Anhang E: Bildung einer Orthonormalbasis für eine Hyperebene E

Die Hyperebene E soll das orthogonale Komplement $g^\perp$ der Geraden g im Punkt p sein. Die Richtung der Geraden g ist durch den Einheitsvektor b_1 gegeben.

$$E = \{x \in \mathbf{R}^n : \langle b_1, x\rangle = \langle b1, p\rangle\}$$

Gesucht ist eine Orthonormalbasis $B = \{b_1, b_2, \ldots, b_n\}$ so, daß

$$E = p + \langle\{b_2, \ldots, b_n\}\rangle.$$

Es sei $m \in \{1, \ldots, n\}$ der Index der betragsmäßig größten Komponente des Einheitsvektors b_1, die Vektoren e_1 bis e_n seien die kanonischen Einheitsvektoren in $\mathbf{R}^n$. Dann ist B',

$$B' = \{b'_1, \ldots, b'_n\} := \{b_1, e_2, \ldots, e_{m-a}, e_1, e_{m+1}, \ldots, e_n\},$$

eine Basis von $\mathbf{R}^n$.

Sukzessive müssen nun die Basisvektoren b'_2 bis b'_n in das orthogonale Komplement $g^\perp$ der Geraden g projiziert werden, und zwar so, daß der jeweils betrachtete Basisvektor zu allen schon bearbeiteten orthogonal ist. Da die Projektion die Längen verändert, muß man den Vektor anschließend wieder normieren. Die Formel nach dem Verfahren von Schmidt [14] zur Berechnung des Vektors b_i aus den Vektoren b_1 bis b_{i-1} und dem Vektor b'_i lautet:

$$b''_i := b'_i - \sum_{j=1}^{i-1} b_j\langle b'_i, b_j\rangle, \quad b_i := \frac{b''_i}{\mid b''_i \mid}$$

Dann ist $B = \{b_2, \ldots, b_n\}$ eine Orthonormalbasis der Hyperebene E.

Anhang F: Ableitung der Bedingung $d' \leq 2s/\sqrt{n}$

Es wird gezeigt, daß bei einer Schrittweite $d' \leq 2s/\sqrt{n}$ im n-dimensionalen Konfigurationsraum immer auch die Sicherheitsbedingung $l \leq 2s$ im Objektraum eingehalten wird.

Die maximale Verschiebung l eines Massepunkts des Manipulatorarms ist kleiner als die skalare Summe der Einzelverschiebungen $d_i l_i$ aller Gelenke $c_1, \ldots, c_n$:

$$l \leq d_1 l_1 + \ldots + d_n l_n = d'_1 + \ldots + d'_n.$$

Dabei steht $d_i = |\, \Delta c_i \,|$ für den Absolutbetrag der Bewegungänderung der Gelenkvariablen c_i, und l_i für den maximalen Wirkungsradius des Gelenks c_i, die Größe d'_i ist die mit l_i gewichtete, normierte Bewegungsänderung $d'_i = d_i l_i$.

Gesucht ist nun eine Abschätzung

$$d' = |\, \Delta c' \,| \leq y$$

so, daß

$$l \leq \sum_{i=1}^{n} |\, \Delta c'_i \,| = \sum_{i=1}^{n} d'_i$$

gilt. Der Vektor $\Delta c' = (\Delta c'_1, \ldots, \Delta c'_n)$ ist ein Schritt im normierten Konfigurationsraum, wo gleich lange Schritte in die verschiedenen Achsenrichtungen gleich große maximale kartesische Verschiebungen im Objektraum nach sich ziehen. Die wirkliche Bewegung Δc ergibt sich daraus zu $\Delta c = (\Delta c'_1/l_1, \ldots, \Delta c'_n/l_n)$.

Wir nehmen zunächst y als fest vorgegeben an, und suchen im Bereich $|\, x \,| \leq y$ die Stelle m der Funktion

$$f : \mathbf{R}^n \to \mathbf{R}, \quad f(x) = \sum_{i=1}^{n} |\, x_i \,|,$$

bei der f maximal wird. Wegen der Monotonie- und Symmetrieeigenschaften von f können wir uns auf den Bereich $|\, x \,| = y$, $x_i \geq 0$ beschränken.

$$|\, x \,| = y \quad \Rightarrow \quad \sum_{i=1}^{n-1} x_i^2 + x_n^2 = y \quad \Rightarrow \quad x_n = \sqrt{y - \sum_{i=1}^{n-1} x_i^2}$$

Damit können wir $f(x)$ schreiben als

$$f(x) = \sum_{i=1}^{n} |\, x_i \,| = \sum_{i=1}^{n} x_i = \sum_{i=1}^{n-1} x_i + \sqrt{y - \sum_{i=1}^{n-1} x_i^2}.$$

Zur Bestimmung der Extrema von f berechnen wir die Nullstellen der partiellen Ableitungen von f:

$$\frac{\partial}{\partial x_j} f(x) = 1 - \frac{x_j}{\sqrt{y - \sum_{i=1}^{n-1} x_i^2}} = 0 \quad \Rightarrow$$

$$d_j = \sqrt{y - \sum_{i=1}^{n-1} x_i^2} = d_n \quad \Rightarrow$$
$$d_i = d_j, \quad i,j = 1,\ldots,n$$

Es sei $M = (1,\ldots,1) \in \mathbf{R}^n$, dann gibt der Vektor m,

$$m = M\frac{y}{|M|},$$

die gesuchte Stelle an, und das Maximum hat dort den Wert

$$f(m) = \sum_{i=1}^{n} m_i = \sum_{i=1}^{n} M_i \frac{y}{|M|} = \sum_{i=1}^{n} 1 \frac{y}{\sqrt{\sum 1^2}} = n\frac{y}{\sqrt{n}} = y\sqrt{n}.$$

Damit haben wir gefunden:

$$|x| \le y \quad \Rightarrow \quad f(x) \le y\sqrt{n}.$$

Gewünscht war aber $f(x) \le 2s$, also muß die Schranke y auf den Wert

$$y = 2s/\sqrt{n}$$

gesetzt werden. Dann ist die Sicherheitsbedingung

$$l = f(\Delta c') = \sum_{i=1}^{n} |\Delta c_i'| \le 2s$$

erfüllt für alle $\Delta c'$ mit $|\Delta c'| = d' \le 2s/\sqrt{n}$.

Literaturverzeichnis

[1] Barraquand, J.; Latombe, J.-C.: *Robot Motion Planning: A Distributed Representation Approach.* Department of Computer Science, Stanford University, Stanford, California, Report STAN-CS-89-1257 (May 1989)

[2] Akman, V.: *Obstructed Shortest Paths in Polyhedral Environments.* Springer, 1987 (Lecture Notes in Computer Science; 251)

[3] Barraquand, J.; Latombe, J.-C.: *A Monte-Carlo Algorithm for Path Planning With Many Degrees of Freedom.* Proc. IEEE Int. Conf. on Robotics and Automation, Cincinnati, USA, May 1990, pp. 1712–1717

[4] Boyse, J.W.: *Interference Detection Among Solids and Surfaces.* Comm. ACM 22, 3–9 (January 1979)

[5] Brooks, R.A.: *Planning Collision-Free Motions for Pick-and-Place Operations.* Int. J. Robotics Research 2, No. 4, 19–44 (Winter 1983)

[6] Canny, J.: *The Complexity of Robot Motion Planning.* Cambridge, Massachusetts: The MIT Press, 1988

[7] Chirikjian, G.S.; Burdick, J.W.: *An Obstacle Avoidance Algorithm for Hyper-Redundant Manipulators.* Proc. IEEE Int. Conf. on Robotics and Automation, Cincinnati, USA, May 1990, pp. 625–631

[8] Donald, B.R.: *Motion Planning with Six Degrees of Freedom.* Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Report AI-TR-791 (May 1984)

[9] Donald, B.R.: *A Search Algorithm for Motion Planning with Six Degrees of Freedom.* Artificial Intelligence 31, 295–353 (1987)

[10] Eastman, C.M.: *Representations for Space Planning.* Comm. ACM 13, 242–250 (April 1970)

[11] Faverjon, B.: *Hierarchical Object Models for Efficient Anti-Collision Algorithms.* Proc. IEEE Int. Conf. on Robotics and Automation, Scottsdale, USA, May 1989, pp. 333–340

[12] Faverjon, B.; Tournassoud, P.: *A Local Based Approach for Path Planning of Manipulators with a High Number of Degrees of Freedom.* Proc. IEEE Int. Conf. on Robotics and Automation, Raleigh, USA, April 1987, pp. 1152–1159

[13] Faverjon, B.; Tournassoud, P.: *Motion Planning for Manipulators in Complex Environments.* Proc. Workshop on Geometry and Robotics, Toulouse, France, May 1988, pp. 87–115

[14] Fischer, G.: *Lineare Algebra.* Braunschweig: Vieweg, 1980

[15] Freund, E.; Hoyer, H.: *Real-Time Pathfinding in Multirobot Systems Including Obstacle Avoidance.* Int. J. Robotics Research 7, No. 1, 42–70 (February 1988)

[16] Glavina, B.: *Trajektorienplanung bei Handhabungssystemen.* Institut für Informatik, Technische Universität München, Report TUM-I8818 (1988)

[17] Glavina, B.: *Wegesuche in der Ebene: eine Lösung durch Synthese von zielgerichteter und zufallsgesteuerter Suche.* Beiträge 5. Fachgespräch über Autonome Mobile Systeme, München, November 1989, S. 41–47

[18] Glavina, B.: *Solving Findpath by Combination of Goal-Directed and Randomized Search.* Proc. IEEE Int. Conf. on Robotics and Automation, Cincinnati, USA, May 1990, pp. 1718–1723

[19] Goldschlager, L.; Lister, A.: *Informatik: eine moderne Einführung.* München: Hanser, 1984

[20] Gupta, K.K.: *Fast Collision Avoidance for Manipulator Arms: A Sequential Search Strategy.* Proc. IEEE Int. Conf. on Robotics and Automation, Cincinnati, USA, May 1990, pp. 1724–1729

[21] Hasegawa, T.: *Collision Avoidance Using Characterized Description of Free Space.* Proc. Int. Conf. on Advanced Robotics, Tokyo, Japan, 1985, pp. 69–76

[22] Hasegawa, T.; Terasaki, H.: *Collision Avoidance: Divide-and-Conquer Approach by Determining Intermediate Goals.* Proc. Int. Conf. on Advanced Robotics, Versailles, France, October 1987, pp. 295–306

[23] Heinhold, J.; Behringer, F.: *Einführung in die höhere Mathematik, Teil 2: Infinitesimalrechnung.* München: Hanser, 1976

[24] Hörmann, K.: *A Cartesian Approach to Findpath for Industrial Robots.* in Rembold, U.; Hörmann, K. (eds.): *Languages for Sensor-Based Control in Robotics.* NATO ASI series, Vol. F 29, 1987

[25] Hörmann, K.: *Kollisionsfreie Bahnen für Industrieroboter. Ein Planungsverfahren.* Berlin: Springer, 1988 (Informatik Fachberichte; 166)

[26] Hopcroft, J.E.; Joseph,D.A.; Whitesides, S.H.: *Movement Problems for 2-Dimensional Linkages.* SIAM J. Computing 13, 610–629 (1984)

[27] Hopcroft, J.E.; Schwartz, J.T.; Sharir, M.: *On the Complexity of Motion Planning for Multiple Independent Objects: PSPACE Hardness of the Warehouseman's Problem.* Int. J. Robotics Research 3, No. 4, 76–88 (1984)

[28] Kubaitis, E.: *Xfroot Timings. 14th Update, 18 March 1990.* Verteilung über Rechnernetz von ejk@uxh.cso.uiuc.edu, Computing Service Office, University of Illinois, Urbana, USA

[29] Laugier, C.; Germain, F.: *An Adaptive Collision-Free Trajectory Planner.* Proc. Int. Conf. on Advanced Robotics, Tokyo, Japan, 1985, pp. 33–41

[30] Lozano-Pérez, T.: *Spatial Planning: A Configuration Space Approach.* IEEE Trans. Computers 32, 108–120 (1983)

[31] Lozano-Pérez, T.: *A Simple Motion-Planning Algorithm for General Robot Manipulators.* IEEE J. Robotics and Automation 3, 224–238 (June 1987)

[32] Lozano-Pérez, T.; Jones, J.L.; Mazer, E.; O'Donnell, P.A.: *Task-Level Planning of Pick-and-Place Robot Motions.* Computer 22, No. 3, 21–29 (March 1989)

[33] Preparata, F.P.; Shamos, M.I.: *Computational Geometry. An Introduction.* New York: Springer, 1985

[34] Reif, J.: *Complexity of the Mover's Problem and Generalizations.* Proc. 20th IEEE Symp. on Foundations of Computer Science, CS Press, Los Alamitos, CA, USA, 1979, pp. 421–427

[35] Requicha, A.A.G.: *Representations for Rigid Solids: Theory, Methods, and Systems.* Computing Surveys 12, 437–464 (December 1980)

[36] Schwartz, J.T.; Sharir, M.: *On the Piano Movers Problem. II: Genaral Techniques for Computing Topological Properties of Real Algebraic Manifolds.* New York University, Department of Computer Science, Courant Institute of Mathematical Sciences, Report 41 (1982)

[37] Sharir, M.: *Algorithmic Motion Planning in Robotics.* Computer 22, No. 3, 9–20 (March 1989)

[38] Siméon, T.: *Planning Collision-Free Trajectories by a Configuration Space Approach.* Proc. Workshop on Geometry and Robotics, Toulouse, France, May 1988, pp. 116–132

[39] Warren, C.W.: *Global Path Planning Using Artificial Potential Fields.* Proc. IEEE Int. Conf. on Robotics and Automation, Scottsdale, USA, May 1989, pp. 316–321

[40] Weicker, R.P: *Dhrystone Benchmark Program.* Comm. ACM 27, 1013–1030 (October 1984)

[41] Wendel, H.: *Neue Ansätze zum Problem der Kollisionsvermeidung.* Institut für Informatik, Technische Universität München, Report TUM-I8704 (1987)

[42] Whitesides, S.H.: *Computational Geometry and Motion Planning.* in Toussaint, G.T. (ed.): *Computational Geometry.* North-Holland, 1985, pp. 377–427