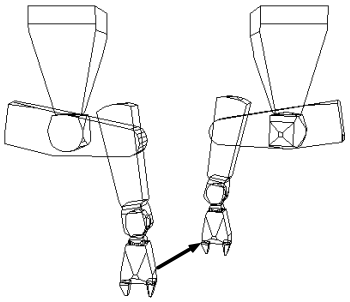
Universität Karlsruhe
Fakultät für Informatik
Institut für Prozeßrechentechnik und Robotik
Prof. Dr.-Ing. U. Rembold
Prof. Dr.-Ing. R. Dillmann

Diplomarbeit:

On-line Kollisionserkennung mit hierarchisch modellierten Hindernissen für ein Mehrarm-Robotersystem



Dominik Henrich

Dezember 1991

Verantwortlicher Betreuer: Prof. Dr.-Ing. U. Rembold

Betreuer am Institut: Dipl.-Inform. X. Cheng

Erklärung

Hiermit erkläre ich, die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Dominik Henrich

Danksagung

Für die Unterstützung bei dieser Diplomarbeit möchte ich mit herzlich bedanken bei:

E. Beer, für ihre konstruktive Kritik am Vortrag und der Ausarbeitung; U. Birkner, für mühsames Korrekturlesen; X. Cheng, für seine aufgeschlossene und engagierte Betreuung; Prof. E. Gilbert, der mir seine Fortran-Routine zur Benutzung überlassen hat; H. W. + I. Henrich, die mein Studium ermöglicht und unterstützt haben; Prof. U. Rembold, für die Genehmigung des Themas der Diplomarbeit.

Inhaltsverzeichnis

1 Einleitung

Die Einleitung umfaßt die gestellte Aufgabe dieser Diplomarbeit und den sich daraus ergebenden prinzipiellen Ansatz. Nach der Einführung einiger grundlegender Begriffe wird ein Überblick über die nächsten Kapitel gegeben.

1.1 Aufgabenstellung

Erst durch die Kooperation von zwei Armen lassen sich umfangreiche Klassen schwieriger Manipulationsaufgaben durchführen. Als Voraussetzung dafür muß man jedoch sicher stellen, daß keine Kollisionen der beiden Arme sowohl gegenseitig, als auch mit den in der Umgebung existierenden Objekten zustande kommt. Die meisten bisherigen Algorithmen zur Kollisionsvermeidung zeichnen sich durch extrem hohen Rechenaufwand aus, sodaß sie für eine on-line Anwendung - d. h. die Kollisionsvermeidung wird durchgeführt, während die Roboterarme sich bewegen - nicht geeignet sind.

Die Kollisionsvermeidung wird im allgemeinen in zwei Schritten durchgeführt. Im ersten Schritt wird der sogenannte Abstandsvektor berechnet, der den minimalen Abstand und dessen Orientierung zwischen einem Roboterarm und einem anderen, unter Umständen auch sich bewegenden Objekt darstellt. Falls dieser Abstandsvektor in einem kritischen Bereich liegt, dann wird im zweiten Schritt eine Ausweichtrajektorie für den Roboterarm geplant. Im Rahmen dieser Diplomarbeit soll versucht werden, nach einem hierarchischen Ansatz die Hindernisse, unter anderem auch dynamische Hindernisse (den zweiten Roboterarm z. B.), durch Hierarchien geometrischer Modelle mit unterschiedlichen Genauigkeiten zu modellieren und dadurch den Aufwand für die Berechnung des Kollisionsvektors im Durchschnitt zu reduzieren.

1.2 Prinzipieller Ansatz

In der Aufgabenstellung ist der prinzipielle Ansatz zur Kollisionserkennung vorgegeben. Es ist die Abstandsberechnung im kartesischen Raum. Es wird also in dem normalen Weltkoordinatensystem gerechnet. Daneben gibt es noch einen grundsätzlich anderen Ansatz.

Dieser arbeitet im sogenannten Konfigurationsraum des Roboters. Der Konfigurationsraum wird von unabhängigen Parametern, die die Stellung des Roboterarms beschreiben, aufgespannt. In diesem Raum ist der Roboter ein Punkt. Eine Bewegung des Roboterarms im kartesischen Raum entspricht einer Bewegung dieses Punktes im Konfigurationsraum.

Zur Kollisionserkennung werden die Hindernisse im Konfigurationsraum beschrieben. Dies geschieht durch zwei relativ aufwendige Schritte. Zuerst werden die Hindernisse in den Konfigurationsraum transformiert. Dann folgt die Konstruktion der Freiraumdarstellung. Der Freiraum ist das Komplement der Hindernisse im Konfigurationsraum. Er beschreibt also alle Konfigurationen, die der punktförmige Roboter ohne Kollision einnehmen kann. Die Kollisionserkennung beschränkt sich darauf zu detektieren, wann der Roboter den Freiraum verläßt. Bei Roboter mit wenigen Freiheitsgraden ist diese Methode der Kollisionserkennung erfolgreich durchgeführt worden.

In dieser Arbeit sollen nun die Kollisionen zwischen mehreren Roboterarmen erkannt werden. Dabei führt die Anwendung des Konfigurationsraums zu Problemen. Jeder Arm ist für den anderen Arm auch ein Hindernis. Er muß daher in dem Konfigurationsraum des anderen Arms beschrieben sein. Nun können sich die Arme auch gleichzeitig bewegen. D. h. nach jeder Bewegung des einen Arms muß er von Neuem in den Konfigurationsraum des anderen Arms transformiert werden. On-line ist dies viel zu zeitaufwendig.

Eine andere Möglichkeit ist die Anwendung eines Konfigurationsraums auf mehrere Arme gleichzeitig. Die Armstellungen beider Roboter werden zu einer Konfiguration zusammengefaßt. Dadurch erhöht sich die Dimension des Konfigurationsraums. In der Praxis wird schon für einen Arm mit sechs Freiheitsgraden der Konfigurationsraum nur für die ersten drei Gelenke aufgebaut. Bei mehr als drei Gelenken ist die kombinatorische Explosion der Konfigurationen nicht mehr handhabbar. Die Anwendung eines Konfigurationsraums für zwei Arme mit insgesamt zwölf Freiheitsgraden ist daher unmöglich.

Für die on-line Kollisionserkennung mit mehreren Armen ist der Ansatz des Konfigurationsraums ungeeignet. Daher wird nun die Kollisionserkennung im kartesischen Raum untersucht. Dabei sei eine **Kollision** die Berührung oder Durchdringung mehrerer realer Objekte, die sich nicht berühren sollen. Im kartesischen Raum gibt es verschiedene Möglichkeiten, eine Kollision zu erkennen. Diese werden im nächsten Unterkapitel vorgestellt.

1.3 Abstandsvektoren

Im kartesischen Raum gibt es drei Hilfsmittel, mit denen eine Kollision erkannt werden kann. Sie unterscheiden sich vor allem in dem Umfang der resultierenden Information. Alle werden zwischen jeweils zwei Objekte berechnet:

- Der **Kollisionstest** hat einen booleschen Wert als Ergebnis. Er gibt lediglich an, ob sich zwei Objekte berühren bzw. durchdringen oder nicht.
- Der **Abstandsvektor** ist die Differenz zweier Punkte mit minimalem Abstand auf den Objekten. Falls sich die Objekte berühren oder durchdringen ist der Abstandsvektor der Nullvektor. Es wird neben der booleschen Aussage, ob eine Kollision stattfindet, noch die Abstands- und Richtungsinformation generiert.
- Der **Kollisionsvektor** gibt zwischen zwei Objekten die Distanz und Richtung an, den eines der Objekte zurücklegen muß, um das andere zu berühren. Neben der Abstands- und Richtungsinformation wird daher auch der Grad einer Durchdringung angegeben. Er bezeichnet dann einen "negativen" Abstand zwischen den Objekten.

In der Abbildung 1.1 wird näher auf den Abstandsvektor eingegangen. Er kann über die Abstandspunkte berechnet werden. Dies sind Punkte mit minimalem Abstand auf zwei Objekten. Im allgemeinen sind sie nicht eindeutig, d. h. für zwei gegebene Objekte kann es mehrere Paare von Abstandspunkten geben. Bei den meisten Objektdarstellungen sind zunächst die Abstandspunkte das Ergebnis der Abstandsberechnung. Aus der Differenz der Abstandspunkte wird dann der Abstandsvektor berechnet. Bei konvexen Objekten ist er im Gegensatz zu den Abstandspunkten eindeutig, da er zwar die Richtung und den kleinsten Abstand, nicht aber die Lage im Raum angibt.

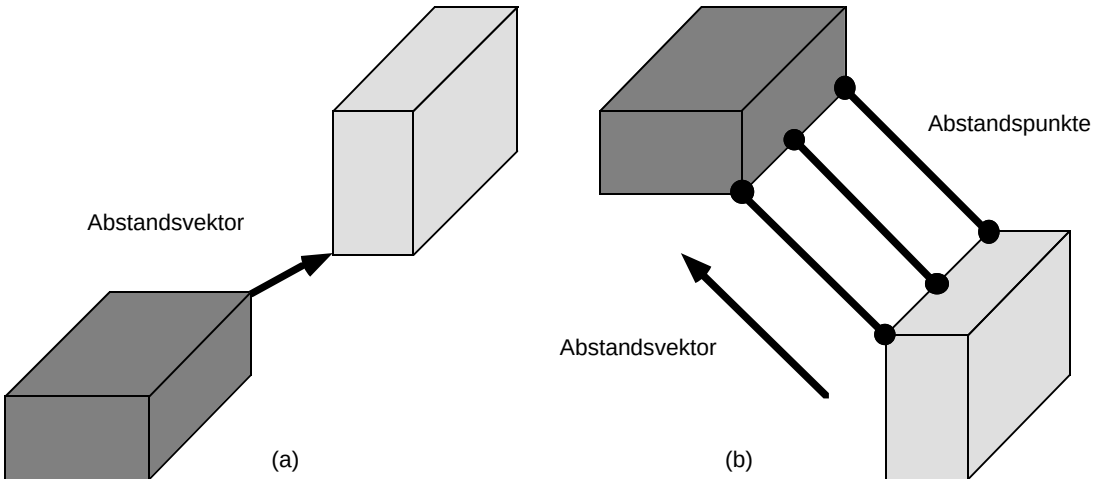


Abbildung 1.1: Unterschied zwischen Abstandsvektor und Abstandspunkte; (a): eindeutiger Abstandsvektor; (b): mehrere Paare von Abstandspunkten.

Es werden nun die drei Hilfsmittel zur Kollisionserkennung miteinander verglichen. Der größte Unterschied ist zwischen dem Kollisionstest und dem Abstandsvektor festzustellen. Der Abstandsvektor und Kollisionsvektor sind dagegen ähnlich. Bei der Verwendung des Abstandsvektors statt dem Kollisionstest ergeben sich folgende Vorteile:

- Eine auf dem Abstandsvektor basierende Kollisionsvermeidung kann durch die zusätzliche Abstands- und Richtungsinformation einer bevorstehende Kollision besser ausweichen, als nur mit einem Kollisionstest.
- Die Bewegungsgeschwindigkeit eines Roboterarms kann der Umgebung und der Situation angepaßt werden. Dazu wählt man die Geschwindigkeit reziprok proportional zu der Länge des Abstandsvektors.
- Andererseits kann ein Sicherheitsabstand durch den Abstandsvektor einfach angewendet werden. Er wird z. B. der (maximalen) Bewegungsgeschwindigkeit des Roboterarms angepaßt. Die Probleme bei dem sonst angewendeten Vergrößern von Objekten wie z. B. aus [Beingesser90] entfallen, wenn die Objekte um den Sicherheitsabstand radial vergrößert werden.
- Mögliche Diskretisierungsfehler bei einer punktweise geplanten Bahn können durch einen Sicherheitsabstand ausgeschlossen werden. Dazu muß der Sicherheitsabstand proportional zu der Abtastrate auf der Bahn gewählt werden. Die eventuelle Vergrößerung der Objekte zur Vermeidung der Diskretisierungsfehler kann durch die exakte Objektdarstellung ersetzt werden.

Wie aus der Auflistung hervorgeht ist die Verwendung des Abstandsvektors weit aus günstiger als die des Kollisionstests. Dem gegenüber steht nur der geringere Aufwand des Kollisionstests. In den nächsten Kapiteln wird jedoch gezeigt, daß die zusätzliche Abstandsinformation den Aufwand stark reduzieren kann. Der geringere Aufwand des Kollisionstests kommt nicht mehr zum Tragen.

Schließlich fehlt noch die Entscheidung, ob der Abstands- oder der Kollisionsvektor verwendet werden soll. In der Kollisionserkennung ist der Einsatz der beiden Vektoren austauschbar. Es ändern sich nur die Algorithmen zur Berechnung des entsprechenden Vektors. Es kommt viel mehr auf die Verwendung der Kollisionserkennung im Gesamtsystem an. Je nach dem, welche Strategie der darauf folgenden Kollisionsvermeidung benutzt wird, ist der eine oder der andere Vektor einzusetzen. Zur Untersuchung der on-line Kollisionserkennung für mehrere Arme genügt die Berechnung des Abstandsvektors.

1.4 Kapitelübersicht

In den letzten Unterkapiteln wurde die Problemstellung und der grundlegende Begriff des Abstandsvektors eingeführt. Die folgenden Kapitel beschäftigen sich mit dem Einsatz des Abstandsvektors in der Kollisionserkennung und seiner schnellen Berechnung.

In der Literatur gibt es keine Verfahren für verwandte Problemstellungen mit mehreren Armen. Darum wird in Kapitel 2 die Kollisionserkennung mit zunächst nur einem Arm beschrieben. Dafür gibt es eine Reihe von Beschleunigungsverfahren. Sie bauen alle auf einer Zweiteilung

der Szene in Roboter und Umwelt auf. Bei mehreren Armen kann diese Modellierung nicht mehr aufrecht gehalten werden. Es wird ein Konzept zur Modellierung der Szene bei mehreren Armen eingeführt.

In Kapitel 3 wird eine bestimmte Art von Beschleunigungs-Ansatz beschrieben. Er dient zur Verbesserung der einzelnen Berechnung des Abstandsvektors in der Kollisionserkennung. Dazu werden die Objekte in unterschiedlichen Ganauikeitsstufen dargestellt.

Eine zweite wichtige Gruppe von Beschleunigungs-Ansätzen sind die hierarchischen Modellierungen. Sie werden im Kapitel 4 beschrieben. Durch eine hierarchische Darstellung wird aus der Sicht der Kollisionserkennung die Anzahl der Objekte reduziert. Damit vermindert sich auch der Gesamtaufwand der Kollisionserkennung. Es wird vor allem auf die Anwendung der Hierarchien auf Roboterarme eingegangen.

Die Ansätze zur Kollisionserkennung lassen sich kaum analytisch bewerten. Darum werden in Kapitel 5 Experimente in einem Simulationssystem mit mehreren Armen durchgeführt. Anhand der Experimente können Aussagen über die Laufzeiten und die Qualität der implementierten Beschleunigungs-Ansätze gemacht werden.

Nach der Zusammenfassung dieser Arbeit in Kapitel 6 wird im Anhang auf die Implementierung der on-line Kollisionserkennung eingegangen. Sie dient zum besseren Verständnis, wie die Kollisionserkennung realisiert wurde.

Im Glossar sind die eingeführten und häufig verwendeten Stichworte aufgeführt. Es ist vor allem bei nur stückweiser Beschäftigumg mit dieser Arbeit nützlich.

2 Kollisionserkennung mit Abstandsvektoren

In diesem Kapitel werden Verfahren zur Kollisionserkennung erörtert. Sie bauen auf der Berechnung von Abstandsvektoren auf. In dem Unterkapitel 2.1 wird die Kollisionserkennung für Szenen mit einem Roboterarm beschrieben. Dieses "einfache" Verfahren kann durch verschiedene Ansätze beschleunigt werden. Dazu wird in 2.2 eine Klassifikation der Ansätze eingeführt und in 2.3 die bisherigen Ansätze dargestellt. Dann wird in 2.4 zu Szenen mit mehreren Roboterarmen übergegangen. Dies macht eine Modifikation des Weltmodells durch die Einführung von Kollisionsklassen erforderlich. Die Eigenschaften dieser Kollisionsklassen werden in 2.5 besprochen.

2.1 Kollisionserkennung für einen Arm

Die hier besprochene Kollisionserkennung baut auf der Berechnung von Abstandsvektoren auf. Dabei ist die Roboterumgebung vollständig bekannt. Die Abstandsvektoren liefern die Abstands- und Richtungsinformation von Objekten zueinander. Bewegt sich ein Objekt auf ein anderes zu, dann kann eine bevorstehende Kollision durch den Abstandsvektor erkannt werden: die Richtung des Abstandsvektors und die der Bewegung sind ähnlich. Zusätzlich nimmt die Länge des Abstandsvektors schrittweise ab.

Die Kollisionserkennung wird zu diskreten Zeitpunkten auf der Bewegungsbahn ausgeführt. Während sich der Roboterarm real oder simuliert von der Start- zu der Zielposition bewegt wird die Bahn in einzelne Schritte unterteilt. Nach jedem Bewegungsschritt wird das interne Weltmodell aktualisiert und die Kollisionserkennung durchgeführt. Je nach Anwendung kann die Kollisionserkennung auf einer Bewegungsbahn häufiger oder seltener berechnet werden. Die **Schrittweite** gibt an, nach welcher zurückgelegten Distanz des Roboterarms spätestens die Kollisionserkennung durchgeführt werden soll. Die Distanz ist hier als Norm in dem von den Gelenkwinkeln aufgespannten Konfigurationsraum angegeben. Sie ist ein Maß für das Ausmaß einer Bewegung des Roboterarms.

Für die Kollisionserkennung ist es aber nicht sinnvoll, zwischen allen Objektpaaren den Abstandsvektor zu berechnen. Zum Beispiel müssen zwischen unbewegten Objekten keine Kollisionen erkannt werden. In einer Szene interessiert vor allem der Abstand zwischen

Objekten, die miteinander kollidieren können. Dies ist in den meisten Fällen der Roboterarm mit den Hindernissen. Dabei wird die Szene wie in Abbildung 2.1 in zwei Gruppen unterteilt. Die erste Gruppe von Objekten enthält alle Teile, die zum Roboter gehören (a). Die zweite Gruppe wird von den Hindernissen gebildet (b). Es werden damit die festen und bewegten Objekte einer Szene voneinander getrennt. (Interne Kollisionen des Roboterarms werden bei dieser Unterteilung nicht berücksichtigt.)

Aber auch zwischen diesen beiden Gruppen sind nicht alle Abstandsvektoren der Objektpaare interessant. In einer Szene interessiert zu einem Augenblick zunächst nur, ob und wo die erste Kollision auftreten könnte. Die erste Kollision einer Szene kann nur zwischen den zwei Objekten entstehen, die den kürzesten aller Abstandsvektoren bilden. Für die Kollisionserkennung genügt es zunächst, zwischen den bewegten und festen Objekten den kürzesten Abstandsvektor zu finden.

Der *einfachste* Algorithmus zur Kollisionserkennung zählt alle Objektpaare auf. Dabei werden nur die Objektpaare betrachtet, die ein Armsegment und ein Hindernis enthalten. Es werden also nur Paare ausgewählt, zwischen denen überhaupt äußere Kollisionen stattfinden können. Für jedes Paar wird der Abstandsvektor berechnet. Während der Aufzählung und Berechnung kann die Länge der Vektoren untereinander verglichen werden. Der kürzeste Abstandsvektor unter allen Objektpaaren wird als Ergebnis der Kollisionserkennung ausgegeben.

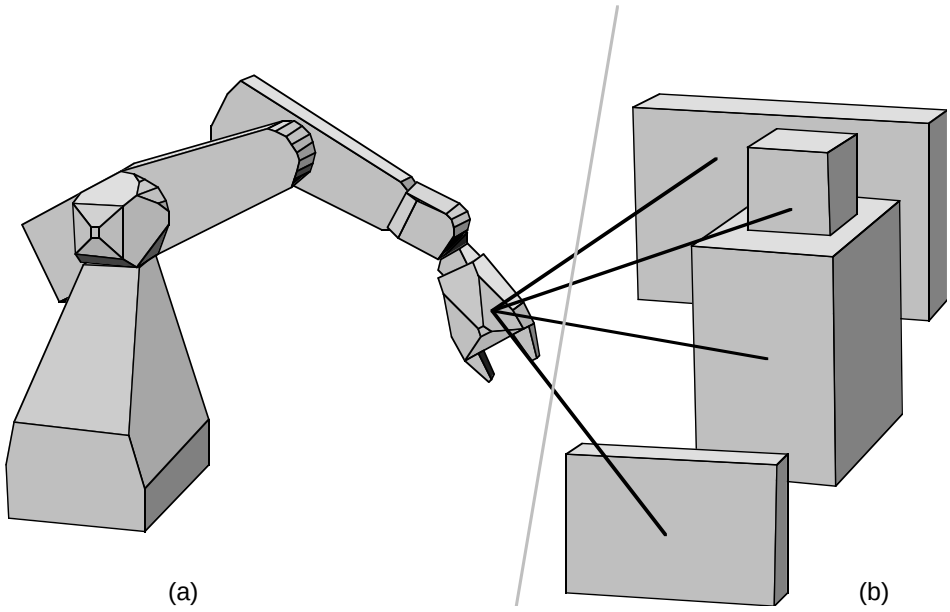


Abbildung 2.1: Unterteilung der Szene in Roboter (a) und Hindernisse (b) für die Kollisionserkennung mit einem Arm.

Der Aufwand des einfachsten Algorithmus zur Kollisionserkennung wird von mehreren Faktoren bestimmt. Der erste ist die Anzahl M der Armsegmente. Das sind die Objekte aus der ersten Gruppe in der Szene. Der zweite Faktor ist die Anzahl N der Hindernisse. Die Hindernisse bilden die zweite Gruppe in der Szene. Daraus ergeben sich also $M \cdot N$ Objektpaare,

zwischen denen der Abstand berechnet wird. Der letzte Faktor ist der Aufwand zur Abstandsberechnung selbst. Der Gesamtaufwand der Kollisionserkennung setzt sich also aus der Anzahl der Objektpaare und dem Aufwand der Abstandsberechnung zusammen. Wie dieser Aufwand reduziert werden kann wird in den nächsten Unterkapiteln beschrieben.

2.2 Klassifikation der Beschleunigungs-Ansätze

Die hier untersuchte Kollisionserkennung findet in einem on-line Planungssystem statt. Der Berechnungsaufwand für den kürzesten Abstandsvektor muß hinreichend gering sein. Eine darauf aufbauenden Kollisionsvermeidung kann sonst nicht rechtzeitig eine Ausweichbewegung planen. Für die on-line Kollisionserkennung ist aber der einfachste Algorithmus aus dem vorigen Kapitel 2.1 zu langsam. Durch Beschleunigungs-Verfahren kann der mittlere Aufwand erheblich reduziert werden.

Zu einem gegebenen Algorithmus gibt es eine Reihe von Methoden zur Beschleunigung. Um die verschiedenen Möglichkeiten der Beschleunigung zu analysieren wird eine Klassifikation der Verfahren eingeführt. Die Klassifikation unterscheidet bei den Ansätzen zunächst zwischen dem *Zeitpunkt* und der *Methode* der Aufwandsreduzierung. Es wird dadurch das "Wann" von dem "Wie" getrennt.

Der *Zeitpunkt*, in dem der Beschleunigungs-Ansatz wirkt, kann entweder off-line oder on-line sein. Diese Zeitangaben beziehen sich auf den Prozeß der Kollisionserkennung:

- off-line Reduzierung: Der Zeitpunkt, in dem der Beschleunigungs-Ansatz eingesetzt wird, liegt vor der Kollisionserkennung. Daten z. B. aus dem Weltmodell werden durch den Ansatz verändert und abgespeichert. Die Kollisionserkennung benutzt die veränderten Daten und kann dadurch schneller zu einem Ergebnis kommen. Zeitlich läßt sich der Prozeß des Beschleunigungs-Ansatzes von dem der Kollisionserkennung trennen.
- on-line Reduzierung: Der Zeitraum, in dem der Beschleunigungs-Ansatz eingesetzt wird, ist (quasi-) parallel zu der Kollisionserkennung. Während den einzelnen Schritten der Kollisionserkennung stellt der Ansatz Daten oder Methoden zur Verfügung, die schneller zu einem Ergebnis führen. Die Prozesse des Beschleunigungs-Ansatzes und der Kollisionserkennung sind zeitlich miteinander verwoben. Sie können nicht voneinander getrennt werden.

Der zweite Teil der Unterscheidung in der Klassifikation ist die *Methode*. Der Aufwand einer Berechnung, die oft wiederholt wird, ist durch zweierlei bestimmt: zum Einen durch die Anzahl der Berechnungen und zum Anderen durch die Berechnung selbst. Also sind die Reduzierung der Anzahl und die Reduzierung des Aufwands zwei Methoden der Beschleunigung:

- Reduzierung der Anzahl: Diese Methode der Beschleunigung vermindert die Anzahl der Abstandsberechnungen, die von der Kollisionserkennung durchgeführt werden müssen. Dies kann z. B. durch die Reduzierung der Objektpaare geschehen, die betrachtet werden.
- Reduzierung des Aufwands: Diese Methode der Beschleunigung der Kollisionserkennung vermindert den Aufwand einer einzelnen Abstandsberechnung. Dazu kann z. B. die Abstandsberechnung an sich verbessert werden oder aber die Darstellung der Objekte, zwischen denen der Abstand berechnet wird.

Bei der Klassifikation beschreibt der Begriff "Ansatz" eine einzelne Idee zur Beschleunigung. Jeder Ansatz beinhaltet ein für sich separiertes Verfahren. In der Anwendung sollten der Effizienz halber mehrere Ansätze kombiniert und gleichzeitig benutzt werden. Zur Untersuchung der verschiedenen Möglichkeiten, mit denen eine Beschleunigung bewirkt werden kann, sind die Ansätze hier getrennt.

Kombiniert man Methode und Zeitpunkt der Ansätze, dann entstehen die vier Klassen aus Tabelle 2.1. Jede Klasse bezeichnet eine Möglichkeit, mit der eine Beschleunigung erreicht werden kann. Zum Beispiel kann eine Beschleunigung des einfachen Algorithmus der Kollisionserkennung on-line durch eine Reduzierung der Anzahl von Berechnungen erreicht werden. Als ein Vertreter dieser Klasse steht die Abstands-Fortschreibung. Jeder dieser Vertreter wird im Folgenden kurz beschrieben.

Attribut der Klasse	**off-line**	**on-line**
Anzahl der Berechnungen	Statische Hierarchien Virtueller Manipulator	Abstands-Fortschreibung
Aufwand einer Berechnung	Primitiven-Approximation	Startwerte

Tabelle 2.1: Klasseneinteilung der Beschleunigungs-Ansätze in vier Klassen mit Beispielen.

2.3 Bisherige Beschleunigungs-Ansätze

Es werden nun die bisherigen Beschleunigungs-Ansätze zu der Kollisionserkennung aus der Literatur beschrieben. Sie sind zunächst für eine Szene mit einem Roboterarm und Hindernissen gedacht. Dabei sind alle Objekte der Umgebung dem System vollständig bekannt.

Zu den Ansätzen, die den mittleren Aufwand einer einzelnen Abstandsberechnungen reduzieren, gehören die Primitiven-Approximation und die Startwerte. Der erste Ansatz geht off-line und der letzte Ansatz on-line vor. Zu den Ansätzen, die die Anzahl der

Abstandsberechnungen reduzieren, gehören die statischen Hierarchien, der virtuelle Manipulator und die Abstands-Fortschreibung. Hierbei arbeiten die ersten beiden Ansätze off-line und der letzte Ansatz on-line. Am Schluß dieses Unterkapitels werden noch zwei die Implementierung betreffende Ansätze und ein mehr theoretischer Ansatz angesprochen.

Primitiven-Approximation

Die Primitiven-Approximation gehört zu den Ansätzen, die den mittleren Aufwand einer einzelnen Abstandsberechnung reduzieren. Die Hindernisse oder Armsegmente werden off-line durch einfachere geometrische Volumen (Primitive) approximiert. On-line wird der Abstand statt zwischen den realen Objekten zwischen den Primitiven berechnet. Als Primitive können z. B. Kugel, Zylinder oder Quader eingesetzt werden. Da die Primitive die realen Objekte annähern, ist der resultierende Abstandsvektor auch eine Annäherung an den realen Abstandsvektor. Dadurch, daß die Primitiven das Volumen der realen Objekte vollständig enthalten, bildet der angenäherte Abstand eine untere Schranke an den realen Abstand. Bei mehreren Primitiven-Approximationen pro Objekt ist das Objekt in unterschiedlichen Genauigkeitsstufen dargestellt. Ist der berechnete Abstand bei einer groben Darstellung zu klein, dann wird die Abstandsberechnung mit einer genaueren Darstellung wiederholt. Im Extremfall wird der Abstand mit der besten Darstellung der realen Objekte berechnet. (Siehe dazu z. B. [Adolphs]). Die Primitiven-Approximation wird im Kapitel 3 ausführlich beschrieben und untersucht.

Startwerte

Der Ansatz der Startwerte kann nur bei bestimmten Abstandsberechnungen zum Einsatz kommen. Dies sind vor allem die iterativen Verfahren. Die Startwerte sind im Gegensatz zu den anderen hier beschriebenen der einzige spezielle Ansatz. Er soll aber dennoch erwähnt werden.

Durch die Startwerte wird on-line der Aufwand der einzelnen Abstandsberechnungen reduziert. Betrachtet man die Kollisionserkennung für eine ganze Bewegungsbahn, dann kann eine einzelne Abstandsberechnung bei einem Bewegungsschritt beschleunigt werden. Dazu wird das Ergebnis aus dem letzten Bewegungsschritt für die aktuelle Abstandsberechnung benutzt. Die Ergebnisse müssen für jedes Objektpaar abgespeichert werden (siehe Anhang A.4) und dienen als Startwerte der neuen Berechnung. Die aktuelle Berechnung aktualisiert wiederum das Ergebnis für den nächsten Schritt.

Für Objekte, die durch konvexe Polyeder modelliert sind, wurde dies von Gilbert und Johnson mit einer iterativen Abstandsberechnung in [Gilbert88] durchgeführt. Eine besondere Darstellung des Abstandsvektors aus dem letzten Schritt dient als Startwert für die neue Iteration. Der Grad der Verbesserung hängt von der Veränderung der Objekte zueinander nach jedem Bewegungsschritt ab.

Abstands-Fortschreibung

Hier handelt es sich um eine on-line Reduzierung der Anzahl von Berechnungen. Der Ansatz geht von einer diskretisierten Bewegungsbahn aus. Nach jedem Bewegungsschritt wird die größte von einem beliebigen Punkt des Roboterarm zurückgelegte Distanz berechnet. Wird diese Distanz von der Länge des letzten Abstandsvektors subtrahiert, dann erhält man eine untere Abschätzung (Fortschreibung) für den realen kleinsten Abstand. Falls das Ergebnis kleiner als ein vorgegebener Sicherheitsabstand ist, wird der Abstandsvektor neu berechnet. Siehe dazu [Faverjon89]. Diese Fortschreibung des Abstands kann für jedes Objektpaar (lokal) oder nur für den kleinsten Vektor der gesamten Szene (global) geschehen. Eine Mischung der lokalen und globalen Fortschreibung sind auch denkbar. Allgemein wird mit Hilfe der Fortschreibung die Anzahl der Berechnungen auf einer Bewegungsbahn vermindert. Allerdings geht bei rotatorischen Bewegungen die Richtungsinformation des Abstandsvektors verloren.

globale Abstands-Fortschreibung: Hierbei wird der kürzeste Abstandsvektor der *gesamten* Szene fortgeschrieben. Es handelt sich nur um *einen* einzelnen Vektor. Zu Beginn der Bewegungsbahn wird zum ersten Mal der kürzeste Abstandsvektor berechnet. Nun bewegt sich der Manipulator schrittweise vorwärts. Nach jedem Schritt wird die größte zurückgelegte Distanz des Manipulators berechnet und der eine Abstandsvektor um diese Distanz verkürzt. Ist der Vektor kürzer als der Sicherheitsabstand, dann wird für die gesamte Szene der Abstandsvektor neu berechnet. Danach versucht man wieder für die nächsten Schritte den globalen kürzesten Abstand fortzuschreiben.

lokale Abstands-Fortschreibung: Es wird nicht nur der kürzeste Abstandsvektor der gesamten Szene, sondern es werden *alle* Abstandsvektoren zwischen *jedem* einzelnen Objektpaaren aktualisiert. Es handelt sich um mehrere Vektoren. Am Anfang einer Bewegungsbahn wird für jedes Objektpaar der Abstandsvektor berechnet und in einer Datenstruktur gespeichert. Nach einem Bewegungsschritt des Roboterarms wird jeder Vektor der Objektpaare um die größte zurückgelegte Distanz verkürzt. Der kleinste resultierende Abstand kann als untere Abschätzung des realen Abstands ausgeben werden. Hat einer der gespeicherten Abstandsvektoren nach der Fortschreibung eine Länge kleiner dem Sicherheitsabstand, dann muß dieser Vektor neu berechnet werden. Für die lokale Abstands-Fortschreibung ist eine Datenstruktur notwendig, die alle Abstandsvektoren speichert. Eine mögliche Implementierung dieser Datenstruktur ist im Anhang A.4 beschrieben.

Wie erfolgt die Berechnung der "größten zurückgelegten Distanz" eines Roboterarms? Sie muß vor jeder Kollisionserkennung mit Abstands-Fortschreibung berechnet werden. Für einen gegebenen Roboterarm kann man sie auf zwei Arten berechnen. Die Erste berechnet für jeden Bewegungsschritt die zurückgelegte Distanz exakt. Zwar ist die Berechnung relativ aufwendig, aber dafür wird der Abstandsvektor nicht mit einer fehlerbehafteten Distanz fortgeschrieben. Die zweite Art berechnet relativ zu der maximalen Bewegungsgeschwindigkeit die größte Distanz, die der Manipulator überhaupt zurücklegen kann. Es wird also nur ein Faktor der maximalen Geschwindigkeit angepaßt. Diese Berechnung ist eine obere Abschätzung der realen

Distanz und kann relativ schnell ermittelt werden. Aber es wird durch diese Abschätzung auch ein Fehler regelmäßig fortgeschrieben, d. h. der Ansatz muß öfter die Abstandsvektoren neu berechnen.

Statische Hierarchien

Die statischen Hierarchien bewirken off-line eine Reduzierung der Anzahl von Berechnungen. Dazu werden vor der Kollisionserkennung die Hindernisse sukzessive zusammengefaßt und durch ein neues Hindernis angenähert. Führt man das Zusammenfassen auch mit den neuen Hindernissen fort, dann entsteht eine hierarchische Anordnung von Hindernissen. Sie entspricht den sogenannten *assembly trees* von Faverjon in [Faverjon89]. Auf jeder Ebene der Hierarchie sind alle Hindernisse dargestellt. Je höher die Ebene, desto gröber ist die Darstellung. Auf der untersten Ebene befinden sich die ursprünglichen Hindernisse, auf der obersten Ebene werden sie durch ein einziges Hindernis approximiert. Durch die hierarchische Darstellung der Hindernisse kann die Objektanzahl für die Kollisionserkennung in den meisten Fällen logarithmisch reduziert werden.

Während der Kollisionserkennung wird z. B. zwischen einem Armsegment des Roboters und den hierarchisch dargestellten Hindernissen der Abstandsvektor berechnet. Dazu beginnt der Algorithmus mit der gröbsten Darstellung der Hindernisse. Ist der Abstand größer als ein vorgegebener Sicherheitsabstand, dann kann er ausgegeben werden. Ansonsten wird mit einer genaueren Darstellung der Hindernisse fortgefahren, solange bis entweder der Abstand groß genug ist oder man bei der genauesten Darstellung angelangt ist. Die statischen Hierarchien sind die Grundform der dynamischen Hierarchien. Beide Darstellungen werden im Kapitel 4 beschrieben.

Virtueller Manipulator

Dieser Ansatz wurde ursprünglich für den Aufbau des Konfigurationsraums konzipiert, aber er kann auf für die on-line Kollisionserkennung eingesetzt werden. Zur Reduzierung der Anzahl der Armsegmente werden off-line eine Reihe virtueller Roboterarme mit unterschiedlichen Freiheitsgraden berechnet. Nach Faverjon in [Faverjon89] ist ein *virtueller Manipulator* mit k Freiheitsgraden der Manipulator, dessen letzte Segmente durch ihr überstrichenes Volumen ersetzt wurden, wenn die Gelenke k+1 bis n frei bewegt werden. Dabei hat der ursprüngliche Arm n Freiheitsgrade. Zur Veranschaulichung siehe Abbildung 2.3.

Bei der Abstandsberechnung wird der Roboterarm durch eine Reihe von virtuellen Manipulatoren ersetzt. Jeder Manipulator ist eine gröbere Darstellung seines Vorgängers. Ist ein Hindernis weit genug vom Arm entfernt, dann wird mit dem gröbsten Manipulator gerechnet. Je näher ein Hindernis ist, desto feiner wird die Darstellung und desto mehr Freiheitsgrade hat der Roboterarm. Da die letzten Armsegmente durch ihre überstrichenen Volumen dargestellt

sind, ist die Approximation relativ grob. Zum Beispiel deckt der 0-te virtuelle Manipulator den gesamten Arbeitsraum des Roboterarms ab.

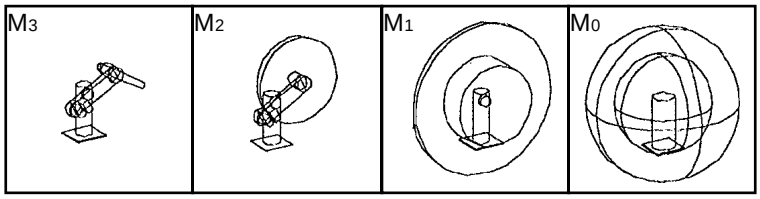


Abbildung 2.3: Beispiel der virtuellen Manipulatoren aus [Faverjon89].

Andere Ansätze

Hier sind die etwas ungewöhnlichen Ansätze gesammelt. Sie dienen der Vollständigkeit. Vor allem die ersten beiden Ansätze gehören mehr in den Bereich der Implementierung. Sie sind daher nur als Konzept dargestellt. Sie können jedoch auch einen Teil zur Beschleunigung beitragen und sollen darum nicht unerwähnt bleiben.

Abbruch bei Abstand Null: Erst bei der Eingliederung in das Gesamtsystem der Bahnplanung wird festgelegt, ob während der Kollisionserkennung Kollisionen auftreten können oder nicht. Sind Kollisionen ausgeschlossen, dann kann die Bewegung in Realität stattfinden. Sind Kollisionen erlaubt, dann muß die untersuchte Bewegung im System simuliert sein. Bei der simulierten Bewegung berechnet der einfache Algorithmus zur Kollisionserkennung immer den Abstand zwischen allen Objektpaaren. Tritt aber eine Kollision zwischen einem Paar auf, kann sofort die Berechnung abgebrochen und der Abstand Null ausgegeben werden. Nach einer Kollision können die weiteren Berechnungen den Abstandsvektor nicht mehr verändern, da er schon kleinstmöglich ist. Dieser Ansatz gehört zu der on-line Reduzierung der Anzahl der Berechnungen.

Kollisionsdaten-Verwaltung: Nach dem einfachsten Algorithmus zur Kollisionserkennung mit Abstandsvektoren werden alle Objektpaare einer Szene betrachtet. Die immense Anzahl der Objektpaare kann direkt eingeschränkt werden. Dazu werden nur die Paare betrachtet, die ein bewegtes Objekt enthalten. Führt man diese Einschränkung fort, dann entsteht eine ausgezeichnete Paarmenge. In dieser Menge sind nur Paare aufgeführt, die miteinander kollidieren können. Dadurch kann man Strukturwissen über die Szene einfließen lassen. Damit gehört dieser Ansatz zu der off-line Reduzierung der Anzahl der Berechnungen. Zusätzlich ist in der Paarmenge die Speicherung von Kollisionsdaten möglich. Zu jedem Objektpaar wird z. B. der Abstandsvektor zwischen den zwei Objekten abgelegt. Der Vektor kann bei der Abstandsberechnung zwischen Polyedern als Startwert des Iterations-Algorithums oder bei der lokalen Abstands-Fortschreibung verwendet werden. Die Implementierung der Kollisionsdaten-Verwaltung ist genauer im Anhang A.4 beschrieben.

Vorausberechnung: Ein eher theoretischer Ansatz ist die off-line Vorausberechnung. Dabei wird für jede Konfiguration zweier Roboterarme der Abstandsvektor off-line berechnet. Während

einer Bewegung wird zu einem Gelenkwinkelsatz beider Arme der gültige Abstandsvektor abgefragt. Dieser Ansatz benötigt im Normalfall enorm viel Speicherplatz. Er ist abhängig von der Diskretisierung der Gelenkwinkel und der (eventuell parametrisierten) Darstellung der Abstandsvektoren. Tastet man zum Beispiel bei zwei Puma260 die ersten drei Gelenke mit 1° ab, dann müssen $8{,}0 \cdot 10^{14}$ Abstandsvektoren gespeichert werden. Die restlichen drei Gelenke bleiben dabei in einer festen Stellung. Selbst wenn ein so großer Speicher zur Verfügung stünde, dann wäre es nicht machbar. Angenommen die Abstandsberechnung dauert eine Sekunde, dann würde es immer noch Jahrtausende dauern, bis alle Abstandsvektoren berechnet wären. Außerdem können bei diesem Ansatz andere Hindernisse nicht berücksichtigt werden.

2.4 Einführung von Kollisionsklassen

In dem vorigen Unterkapitel wurden eine Reihe von Ansätzen zur Beschleunigung der Kollisionserkennung beschrieben. Diese Ansätze bezogen sich auf Szenen mit nur einem Arm und Hindernissen. Der Mechanismus der Kollisionserkennung wird nun verallgemeinert. Es sind dann auch Szenen mit mehreren Armen erlaubt. Diese Verallgemeinerung erfordert Veränderungen sowohl von dem Weltmodell als auch von der darauf arbeitenden Kollisionserkennung.

Bisher wurde die Szene in zwei Gruppen unterteilt. Die eine enthielt die bewegten Armsegmente, die andere die festen Hindernisse. Diese Unterteilung ist bei mehreren Armen nicht aufrecht zu halten. Jeder Roboterarm ist für den anderen auch ein Hindernis. Er müßte also in die Gruppe der festen Objekte. Nun bewegt sich aber der andere Arm zur gleichen Zeit. Demnach müßte er zu der Gruppe der bewegten Objekte gehören. In beiden Gruppen kann ein Arm nicht gleichzeitig sein. Er würde ständig mit sich selbst kollidieren, da die Kollisionserkennung zwischen den beiden Gruppen erfolgt.

Dieser Widerspruch erfordert eine Modifikation der Szeneneinteilung. Statt die Szene nur in zwei Gruppen zu unterteilen wird sie in mehrere Gruppen untergliedert. Das führt zu einer Klasseneinteilung des Weltmodells. Jede Klasse enthält eine Menge von Objekten aus der Szene. Diese Objekte sind dadurch ausgezeichnet, daß zwischen ihnen keine Kollisionserkennung berechnet wird. Die Kollisionserkennung findet nur zwischen den sogenannten Kollisionsklassen statt.

Zum Beispiel würde man bei einer Szene mit zwei Armen und Hindernissen vier Kollisionsklassen aufstellen (siehe Abbildung 2.2). Die ersten beiden enthalten die Roboterarme (a), (b), die dritte enthält die Hindernisse (c) und die letzte wird durch die Befestigung der Arme gebildet (d). Die Kollisionserkennung berechnet zwischen den relevanten Kollisionsklassen paarweise den kürzesten Abstandsvektor. So können gleichzeitig Kollisionen zwischen den zwei Armen und zwischen Arm und Hindernissen erkannt werden.

Bei dieser Einteilung in Kollisionsklassen werden nicht alle möglichen Kollisionen erkannt. Die Kollisionserkennung berechnet nur die Abstände zwischen den Klassen, nicht innerhalb der Klassen. Dies hat vor allem zwei Vorteile. Der erste tritt bei der Zusammenfassung von Hindernissen auf. Da sich die Hindernisse nicht bewegen, können sie auch nicht miteinander kollidieren. Eine Kollisionserkennung zwischen den Hindernissen ist überflüssig. Faßt man die Hindernisse zu einer Kollisionsklasse zusammen, dann erspart man sich die überflüssigen Abstandsberechnungen. Der zweite Vorteil tritt bei der Zusammenfassung von Roboterarmen auf. Die Armsegmente können sehr wohl untereinander kollidieren. Diese internen Kollisionen können aber durch einfachere Mechanismen (z. B. durch Überprüfen der Gelenkwinkel) erkannt werden. D. h. durch die Zusammenfassung von Armsegmenten in einer Kollisionsklasse wird einer effizienteren Kollisionserkennung der Vortritt gelassen.

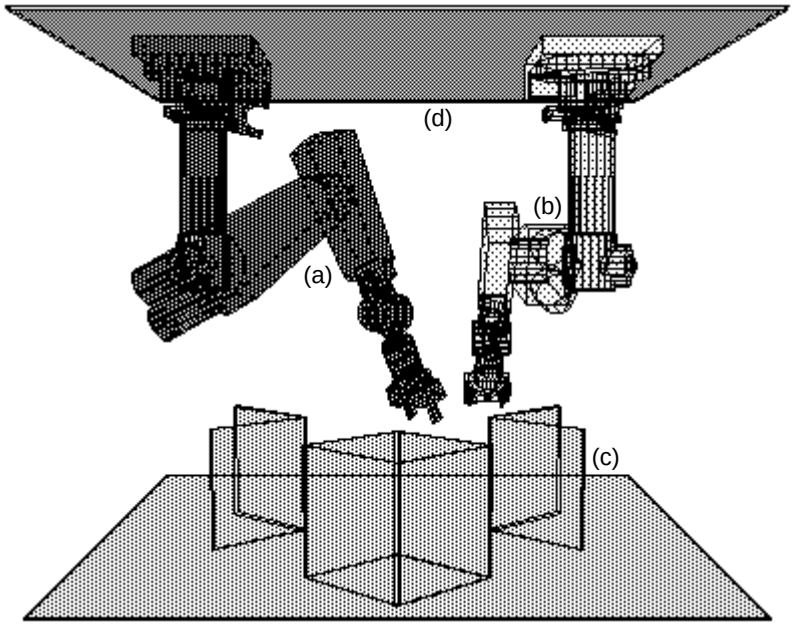


Abbildung 2.2: Bildung von vier Kollisionsklassen in einer Szene: zwei Arme (a),(b), Hindernisse (c) und Befestigung (d).

Formale Beschreibung der Klasseneinteilung

Die oben beschriebene Modifikation des Weltmodells soll nun formalisiert werden. Dabei ist die Klasseneinteilung konstant und disjunkt, d. h. die Zugehörigkeit eines Objektes zu seiner Klasse bleibt unverändert und jedes Objekt gehört genau zu einer Klasse. Natürlich kann sich dennoch die Position oder Geometrie der Objekte mit der Zeit verändern. Das Weltmodell enthält die zur Kollisionserkennung notwendige geometrische Information der Szene. Hier werden als Grundmodellierung der realen Objekte konvexe Polyeder verwendet. Die Art der Grundmodellierung ist austauschbar. Formal besteht das **Weltmodell *W*** aus einer Menge von Kollisionsklassen C^j:

$$W := \left\{ C^j \mid j = 1,\ldots,c \right\}.$$

Es ist definiert in einem gegebenen Welt-Koordinatensystem F_w. In einer **Kollisionsklasse** $\boldsymbol{C^j}$ sind die realen Objekte als eine Menge von konvexen Polyedern P_i^j modelliert:

$$C^j := \left\{ P_i^j \mid i = 1,\ldots,N^j \right\}.$$

Relativ zum Welt-Koordinatensystem F_w hat jede Kollisionsklasse C^j ihr eigenes lokales Koordinatensystem F^j in Frame-Notation. Das Frame gibt ein lokales Koordinatensystem bezüglich einem schon definierten Koordinatensystem an. Es wird durch eine Rotationsmatrix und einem Translationsvektor dargestellt. Üblicherweise werden **konvexe Polyeder** $\boldsymbol{P}$ durch die konvexe Hülle beschrieben, die von der Menge der Eckpunkte R des realen Objekts aufgespannt wird:

$$P := \mathrm{co}(R).$$

Zusätzlich habe jeder Polyeder P_i^j ein lokales Frame F_i^j, bezüglich F^j.

Spezifikation der Kollisionserkennung

In diesem formalen Weltmodell kann man die Aufgabe der Kollisionserkennung genauer spezifizieren. Die Kollisionserkennung berechnet zu einem Zeitpunkt t in einem gegebenen Weltmodell W die **Abstandsvektoren** $\boldsymbol{v^{j,l}}$ zwischen je zwei Kollisionsklassen

$$C^j := \left\{ P_i^j \mid i = 1,\ldots,N^j \right\},\ C^l := \left\{ P_k^l \mid k = 1,\ldots,N^l \right\},$$

mit $j,l \in \{1,...,c\}$, $j \neq l$ durch:

$$v^{j,l} := v_{i,k}^{j,l}, \text{ für } \left| v_{i,k}^{j,l} \right| = \min,$$

mit $i \in \{1,...,N^j\}$, $k \in \{1,...,N^l\}$ und $i \neq k$. Wobei $v_{i,k}^{j,l}$ allgemein als der Abstandsvektor zwischen zwei konvexen Polyedern $P_i^j \in C^j$ und $P_k^l \in C^l$ auf folgender Weise eindeutig definiert wird:

$$v_{i,k}^{j,l} := r - s, \text{ für } |r - s| = \min,$$

mit nicht unbedingt eindeutigen $r \in P_i^j$ und $s \in P_k^l$. $|\cdot|$ ist die Euklidsche Norm im 3-dimensionalen Raum.

Zusätzlich wird für das gesamte Weltmodell eine **Indexmenge** $\boldsymbol{I}$ definiert:

$$I := \left\{ \{j,l\} \mid j \neq l,\ j,l \in \{1,\ldots,c\} \right\}.$$

Sie gibt mit jedem ungeordnetem Paar $\{j,l\} \in I$ an, zwischen welchen Klassen C^j, C^l die Kollisionserkennung berechnet wird.

Die Nützlichkeit der Indexmenge zeigt sich bei folgender Konstellation: Zwei Arme sind in hängender Konfiguration an der Decke eines Raumes montiert. Zusätzlich existieren in der

Szene einige Hindernisse, wie in Abbildung 2.2. Die zwei Arme, die Befestigung und die Hindernisse bilden je eine von vier Klasse: $C^1,...,C^4$. In der Indexmenge sind alle Paare bis auf Arm/Befestigung {1,3},{2,3} und Befestigung/Hindernisse {3,4} aufgenommen. Die ersten zwei Paare würden sonst immer eine Kollision erzeugen. Zwischen dem letzten Paar ist die Kollisionserkennung unnötig. Die Indexmenge gibt nur die Klassen-Paare an zwischen denen eine Kollisionserkennung sinnvoll ist:

$$I = \{\{1,2\}, \{1,4\}, \{2,4\}, \{3,4\}\}.$$

Mit der Angabe von Kollisionsklassen in einem Weltmodell und einer zugehörigen Indexmenge ist die Aufgabe der Kollisionserkennung vollständig definiert.

2.5 Eigenschaften der Kollisionsklassen

Der formalen Definition folgend ist das Weltmodell in einige gleichartige Kollisionsklassen eingeteilt. Eine Kollisionsklasse ist eine abstrakte Zusammenfassung mehrerer Objekte. Das zeitliche Verhalten der Kollisionsklassen ist dabei konstant, d. h. die Aufteilung der Objekte in die Klassen verändert sich über die Zeit nicht. So bilden z. B. die Hindernisse eine und der Roboterarm eine andere Klasse. Aus der Sicht der Klasseneinteilung unterscheiden sie sich nicht. Nun sind aber dennoch sehr unterschiedliche Typen von Objekten zusammengefaßt. Diese Unterscheidung soll nun in Form von Eigenschaften der Kollisionsklassen festgehalten werden.

Trotz der konstanten Klasseneinteilung ist eine Bewegung der Objekte zugelassen. Neben der Geometrie und der Lage im Raum ist die Bewegung ein wesentliches Merkmal der Objekte. Hierbei wird angenommen, daß sich die einzelnen Objekte nicht deformieren können. Sie haben genau sechs Freiheitsgrade der Bewegung. Für ein Objekt sind nur Drehungen oder Veränderungen der Position erlaubt.

Betrachtet man nun eine Zusammenfassung von Objekten, z. B. eine Kollisionsklasse, dann ergeben sich weitere Bewegungsformen. Die Zusammenfassung kann sich auf zweierlei Art verändern. Zum Einen ist eine synchrone Bewegung aller Objekte möglich. Jedes Objekt verändert gleichzeitig mit den anderen seine Lage. Die Abstände und Orientierungen der Objekte untereinander verändern sich aber nicht. Zum Anderen können sich die Objekte relativ zueinander bewegen. Eine Veränderung der Abstände oder Orientierungen ist dort möglich.

Formal kann man diese zwei Bewegungsarten für die Kollisionsklassen festhalten. Es ergeben sich folgende vier Eigenschaften: Eine Kollisionsklasse C^j heißt **fest** über der Zeit t, falls das Klassen-Frame

$$F^j(t) = \text{const}$$

ist, ansonsten heißt C^j **bewegt**. Diese zwei Eigenschaften bezeichnen die Bewegung der Kollisionsklasse als Ganzes.

Bei einer über der Zeit *t* **statischen** Kollisionsklasse C^j sind die Polyeder-Frames

$$F_i^j(t) = \text{const für } i = 1,...,N^j,$$

ansonsten heißt C^j **dynamisch**. Hier sind die Bewegung der Objekte relativ zueinander als Eigenschaften der Kollisionsklasse definiert.

Eigenschaft der Kollisionsklasse	**statisch**	**dynamisch**
fest	Tisch	Roboterarm
bewegt	Fahrzeug	mobiler Roboter

Tabelle 2.2: Beispiele für die Kollisionsklassen

Es sind also zwei Bewegungsarten als Eigenschaften der Kollisionsklassen formuliert. Für jede Bewegungsart existieren wiederum zwei Eigenschaften. Diese Eigenschaften schließen sich gegenseitig aus und sind vollständig, d. h. eine Kollisionsklasse nimmt auf jeden Fall eine dieser Eigenschaften an. Kombiniert man die Bewegungsarten, dann entsteht eine Einteilung der Kollisionsklassen in vier Gruppen. In Tabelle 2.2 ist für jede Gruppe ein Beispiel einer Kollisionsklasse angeführt. Als Repräsentant der dynamischen bewegten Kollisionsklassen steht der mobile Roboter. Er kann durch sein Fahrzeug den Ort wechseln. Dabei ändern alle Elemente dieser Kollisionsklasse gleichzeitig die Position. Daher ist der mobile Roboter eine Kollisionsklasse mit der Eigenschaft "bewegt". Auf der anderen Seite kann ein mobiler Roboter seinen Arm bewegen. Die Armsegmente verändern ihre Lage relativ zueinander. Aus diesem Grund hat die Kollisionsklasse die Eigenschaft "dynamisch".

3 Approximation durch Primitive

Die Abstandsberechnung zwischen Objekten ist mit hohem Rechenaufwand verbunden, vor allem dann, wenn es sich um Objekte mit komplexer Geometrie handelt. Diesen Aufwand kann man durch eine einfachere Darstellung der Objekte verringern. Dazu werden die Volumen der Objekte durch Approximationen ersetzt.

In diesem Kapitel wird beschrieben, wie Objekte durch Primitive approximiert und die entsprechenden Primitive berechnet werden können. Es folgen die grundlegenden Abstandsberechnungen zwischen den Primitiven und die Messung der Laufzeiten. Darauf baut die Abstandsberechnung zwischen den durch Primitive approximierten Objekten auf.

3.1 Approximation der Objekte

Die realen Objekte einer Szene werden hier durch (konvexe) Polyeder dargestellt. Nun ist die Abstandsberechnung zwischen Polyedern ohne Einschränkung relativ schwierig. Sie kann aber erheblich vereinfacht werden. Dazu werden statt den Polyedern Approximationen verwendet. Eine **Approximation** ist eine angenäherte Darstellung des ursprünglichen Objekts. Da die Approximationen keine exakte Repräsentation der realen Objekte ist, kann auch der Abstand zwischen ihnen nicht exakt sein. Er weicht von dem Abstand der realen Objekten ab. Möchte man die Approximationen zur Abstandsberechnung einsetzten, dann müssen sie noch eine weitere Bedingung erfüllen. Man erhält eine worst-case-Abschätzung des Abstands, wenn die Approximationen konservativ sind. D. h. der berechnete Abstand ist eine untere Schranke für den realen Abstand. Dabei enthält eine **konservative** Approximation vollständig das Volumen des angenäherten Objekts. Es besteht also eine Teilmengen-Relation zwischen dem angenäherten Objekt O und einer der möglichen Approximationen A_i:

$$A_i \supseteq O.$$

Eine konservative Approximation eines allgemeinen Polyeders stellt z. B. seine konvexe Hülle dar. Sie wird durch die Hüllen-Operation aus den Polyeder-Eckpunkten berechnet. Ist die Darstellung eines Objekts durch konvexe Polyeder zu ungenau, dann können mehrere konvexe Polyeder zur Approximation eingesetzt werden. Prinzipiell ist jeder allgemeine Polyeder exakt

durch eine Vereinigung von konvexen Polyedern darstellbar. Jedoch kann unter Umständen die Anzahl der dazu benötigten Polyeder sehr groß werden.

Als weitere konservative Approximationen werden sogenannte Primitive eingesetzt. Ein **Primitiv** ist ein konvexes Volumen, das sich durch eine feste Anzahl skalarer Parameter beschreiben läßt. Diese Anzahl kann sich bei verschiedenen Primitiven unterscheiden, ist aber in der Regel geringer als die für das approximierte Objekt. Bei einer Grundmodellierung durch konvexe Polyeder wurden von Adolphs und Horsch in [Adolphs] Quader, Zylinder und Kugeln als Primitive verwendet. Neben diesen soll hier noch die Bounding-Box eingesetzt werden. In [Faverjon89] werden andere Primitive benutzt, auf die hier nicht eingegangen werden soll. Die Grundmodellierung (Polyeder) und die Primitiven sind am Beispiel eines Armsegments in Abbildung 3.1 angeführt.

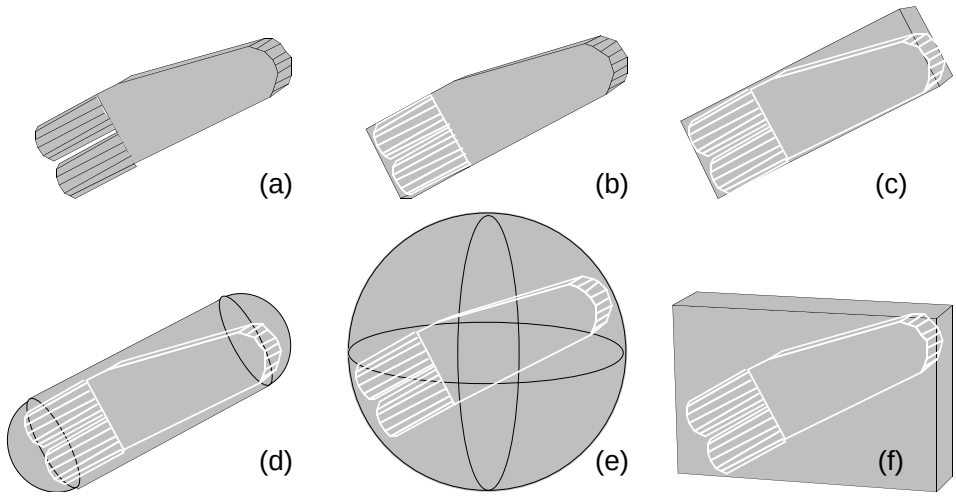


Abbildung 3.1: Primitiven-Approximation eines Armsegments; (a): reales Objekt; (b): konvexer Polyeder; (c): Quader; (d): Zylinder; (e): Kugel; (f): Bounding-Box

Im Folgenden werden die Primitive genauer beschrieben. Für jedes Primitiv ist als Komplexitätsmaß die Anzahl der skalaren Parameter angegeben, die dazu notwendig sind das Primitiv vollständig zu beschreiben. Geht man davon aus, daß beliebige Objekte in den meisten Fällen durch komplexere Primitive besser angenähert werden als durch weniger komplexe, dann ist das Original-Objekt durch die Primitiven-Approximation in unterschiedlichen Genauigkeits-Stufen dargestellt. Dabei approximiert ein Primitiv das Objekt *besser* als ein anderes Primitiv, wenn die Volumendifferenz zwischen dem ersten Primitiv und dem Objekt geringer ist. Da beide Primitive konservativ sind, ist dann auch die räumliche Ausdehnung des ersten Primitives kleiner als die des zweiten.

Polyeder: Ein Polyeder ist ein "Vielflächner", d. h. sein Volumen wird durch eine Vielzahl von Flächen begrenzt. Dabei sind die Flächen ebene Polygone im Raum. Rechner-intern können die Polyeder durch verzeigerte Listen von Eckpunkten, Kanten und Flächen beschrieben werden.

konvexer Polyeder: Er unterscheidet sich nur in einer zusätzlichen Bedingung zum Polyeder: Die Verbindung von zwei beliebigen Punkten im Innern des konvexen Polyeders muß wiederum innerhalb des konvexen Polyeders liegen. Neben der Listendarstellung kann ein konvexer Polyeder auch durch eine Menge von Hyperebenen oder von Eckpunkten beschrieben werden. Da ein Eckpunkt im 3-dimensionalen Raum durch drei Koordinatenwerte definiert wird, sind bei n Eckpunkten $3n$ skalarer Parameter zur Beschreibung des konvexen Polyeders nötig.

Quader: Das Volumen eines Quaders wird durch sechs rechteckige Flächen begrenzt. Die jeweils gegenüberliegenden Flächen sind parallel. Außerdem stehen benachbarte Flächen senkrecht aufeinander. Zur Beschreibung eines Quaders können die acht Eckpunkte oder die sechs Ebenengleichungen der Flächen benutzt werden. Beide Darstellungen sind aber redundant, da die Eigenschaft der räumlichen Beziehung zwischen den Flächen nicht ausgenutzt wird. Eine nicht-redundante Beschreibung ist ein Basiseckpunkt mit drei zueinander senkrecht stehenden Vektoren. Da die Richtung des dritten Vektors durch das Kreuzprodukt der ersten beiden Vektoren bestimmt wird, braucht man nur die Länge des dritten Vektors angeben. Daraus ergeben sich 10 skalarer Parameter.

Zylinder: Hier soll unter einem Zylinder die sphärische Erweiterung eines Geradenstücks im Raum um einen bestimmten Radius verstanden werden. Er ist also eine Röhre mit Halbkugeln als Enden. Der Zylinder ist einfach durch die zwei Endpunkte des Geradenstücks und dem Radius zu beschreiben. Die Anzahl der skalaren Parameter ist demnach 7.

Bounding-Box: Sie gibt die extreme Ausdehnung des zu approximierdenden Objekts in jede Koordinatenrichtung an. Im 3-dimensionalen Raum ergeben die Minima und Maxima 6 skalare Parameter. Eine andere Spezifikation der Bounding-Box sieht sie als achsenparallelen Quader an. Die Einschränkung an die Orientierung im Raum ergibt im Vergleich zum Quader eine Ersparnis von 4 Parametern.

Kugel: Die Kugel ist die sphärische Erweiterung eines Punktes um den Radius. Zur Beschreibung sind der Mittelpunkt und der Radius, also 4 skalare Parameter, nötig.

3.2 Berechnung der Primitive

Zu einem gegebenen Objekt wird ein Primitiv berechnet, das das Volumen des Objekts vollständig enthält. Es handelt sich also um eine konservative Approximation. Darüber hinaus soll der sogenannte **Approximations-Fehler** möglichst gering sein. Dieser Fehler wird hier durch die Volumendifferenz zwischen Primitiv und dem Objekt ermittelt. Je kleiner die Differenz, desto besser nähert das Primitiv das Objekt an. Da die Approximation konservativ ist, stimmt das Primitiv im Grenzfall (Approximations-Fehler = 0) mit dem angenäherten Objekt überein.

Die Berechnung der Primitiven-Approximationen erfolgt off-line, also vor der eigentlichen Kollisionserkennung. Sie hat damit keine wesentliche Zeitbeschränkung. Es kann immer auf die genaueste zur Verfügung stehende Darstellung zurückgegriffen werden. Damit steigt zwar der Rechenaufwand, aber der Approximationsfehler bleibt gering. Verwendet man schon berechnete Primitive als Grundlage, dann ist der Aufwand geringer, aber der Approximationsfehler wächst. Hier sollten die (nicht unbedingt konvexen) Polyeder als Grundlage zur Berechnung der Approximation verwendet werden.

Im Folgenden wird für jedes der hier betrachteten Primitive der Algorithmus zur Berechnung als Approximation beschrieben. Zur Verdeutlichung der Komplexität der einzelnen Verfahren wird der asymptotische Aufwand im O-Kalkül abgeschätzt.

Approximation durch Bounding-Boxes

Um von einem Polyeder eine Approximation durch eine Bounding-Box zu berechnen, werden die Extreme unter den Polyeder-Eckpunkten gesucht. Für jede Koordinatenachse wird unter den Koordinatenwerten der Eckpunkte das Minimum und das Maximum ermittelt. Bei drei Koordinatenachsen ergeben sich sechs Extremwerte. Durch Kombination dieser sechs Extremwerte ergeben sich die Koordinaten der Eckpunkte der Bounding-Box.

Die resultierende Bounding-Box ist optimal bezüglich des Approximations-Fehlers, da sie die kleinste konservative Bounding-Box zu dem Polyeder ist. Der Rechenaufwand, gemessen im O-Kalkül, steigt linear mit der Anzahl der Polyeder-Eckpunkte. Da die Bounding-Box abhängig ist von der Lage des anzunähernden Polyeders im Koordinatensystem, muß sie bei einer Drehung des Polyeders neu berechnet werden. Die Bounding-Box kann im Allgemeinen nicht wie andere Primitive mit der selben Transformation aktualisiert werden, die der Polyeder erfahren hat.

Approximation durch Kugeln

Die Approximation eines Polyeders durch eine Kugel erfolgt ähnlich dem Algorithmus in [Preparata85]. Dort wird für eine Punktmenge in der Ebene die kleinste Kreisscheibe, die alle Punkte enthält, durch eine vollständige Suche ermittelt. Hier wird diese Vorgehensweise auf die Eckpunktmenge des Polyeders angewendet. Dazu wird die Bedingung ausgenutzt, daß auf der Oberfläche der gesuchten Kugel eine Reihe von Eckpunkten liegen müssen, ansonsten ist sie nicht die kleinste. In den folgenden drei Fällen werden Kugeln von Punkten eindeutig aufgespannt:

- Zwei Punkte bestimmen die Lage der Pole der Kugel.
- Drei Punkte bestimmen die Lage des Äquators der Kugel.
- Vier Punkte liegen beliebig auf der Kugeloberfläche.

Bei mehr als vier Punkten ist die Beschreibung der Kugel überbestimmt. Die kleinste Kugel wird dann von mehreren Punkte-Tupeln aufgespannt. Es genügen aber immer zwei, drei oder vier Punkte, um die kleinste Kugel zu bestimmen. Nun werden nacheinander Kugeln erzeugt, die von allen Punkte-Paaren, -Tripeln und -Quartupeln der Eckpunktmenge aufgespannt werden. Für jede erzeugt Kugel wird geprüft, ob sie alle Eckpunkte des Polyeder enthält. Unter denjenigen Kugeln, die alle Eckpunkte enthalten, wird die mit dem kleinsten Volumen als die gesuchte ausgegeben.

Bezüglich des Approximations-Fehlers liefert dieser Algorithmus zu jeder Eckpunktmenge von Polyedern eine optimale Lösung. Der Aufwand steigt aber um $O(n^4)$ mit der Anzahl n der Polyeder-Eckpunkte.

Approximation durch Zylinder

Zur Berechnung des Zylinders als Approximation eines Polyeders wird heuristisch vorgegangen. Bei diesem Verfahren kann bezüglich des oben definierten Approximations-Fehlers keine optimale Lösung garantiert werden. Aber der Aufwand steigt nur quadratisch mit der Anzahl der Polyeder-Eckpunkte.

Zunächst wird die Achse des Zylinders durch die zwei Eckpunkte des Polyeders mit maximalem Abstand zueinander gelegt. Der Radius des Zylinders bestimmt sich dann aus der Distanz des entferntesten Eckpunkts zur Zylinderachse. Der berechnete Zylinder enthält nun den gesamten Polyeder, aber der Zylinder kann unter Umständen noch verkleinert werden. Dazu werden die Achsen-Endpunkte um den Radius entlang der Achse in Richtung der Zylindermitte verschoben. Nun liegen die zwei entferntesten Eckpunkte des Polyeders und der den Radius bestimmende Punkt innerhalb des Zylinders. Die jetzt außerhalb liegenden Eckpunkte haben nun einen kleineren, senkrechten Abstand zur Achse als der Radius. Um diese Punkte einzufangen wird für jeden dieser Eckpunkte der jeweils günstigere Endpunkt der Zylinderachse nach außen verschoben, bis der Eckpunkt im Innern liegt.

Approximation durch Quader

Auch die Approximation eines Polyeders durch Quader entsteht mit Hilfe eines heuristischen Verfahrens. Es baut auf der Bounding-Box-Approximation auf. Der Aufwand steigt dabei linear mit der Anzahl der Eckpunkte des Polyeders. Zusätzlich ist der Aufwand abhängig von der gewünschten Genauigkeit der Approximations-Berechnung. Diese Abhängigkeit bewirkt aber nur eine Veränderung des Aufwands um einen konstanten Faktor und ändert an dem linearen Verhalten nichts.

Der Unterschied von einem Quader zu einer Bounding-Box ist, daß der Quader in seiner Geometrie zusätzlich drei rotatorische Freiheitsgrade besitzt. Ein Quader ist also eine verdrehte Bounding-Box. Um zu einem gegebenen Polyeder die kleinste konservative Quader-Approximation zu berechnen, wird der Bounding-Box-Algorithmus verwendet. Statt nun einen

unter Umständen verdrehten Quader zu berechnen, wird der Polyeder gedreht und durch eine Bounding-Box angenähert. Leider ist die Drehung des Polyeders um die Koordinatenachsen, die den kleinsten achsenparallelen Quader erzeugt, nicht bekannt. Also wird der Polyeder iterativ um eine Koordinatenachse für kleine Winkelbeträge gedreht und die Polyederlage mit der kleinsten Bounding-Box ausgegeben. Aufbauend auf der Suche für die erste Achse, wird der verdrehte Polyeder um die anderen zwei Achsen gedreht und die günstigste Lage gesucht. Da die Bounding-Box um eine Koordinatenachse rotations-symmetrisch ist, muß der Polyeder insgesamt bei einer Achse nur um 90° gedreht werden.

Für eine Achse kann die Drehung um andere Achsen die optimale Lage verändern, d. h. die Suche muß für die erste Achse erneut durchgeführt werden. Das Drehen des Polyeders um die drei Koordinatenachsen muß solange wiederholt werden, bis das Volumen der berechneten Bounding-Boxes nicht mehr verkleinert werden kann. Dann hat man die Polyeder-Lage gefunden, die die kleinste Bounding-Box liefert. Die gesuchte Quader-Approximation des ursprünglichen Polyeders erhält man dadurch, daß diese Bounding-Box in einen Quader transformiert und in die entgegengesetzte Richtung gedreht wird. Jetzt hat man den kleinsten Quader gefunden, der den Polyeder approximiert.

Diese Quader-Approximation hat eine Genauigkeit entsprechend der Diskretisierung der Drehungen durch die kleinen Winkelbeträge. In jedem Fall ist die gewonnene Approximation konservativ. Gegebenenfalls könnte durch Verdrehen des Quaders um einen Winkel kleiner der Diskretisierung das Volumen um einen kleinen Betrag verringert werden.

Ergebnisse der Approximations-Berechungen

Die oben beschriebenen Algorithmen wurden implementiert und die Primitiven-Approximationen von den Armsegmenten eines Puma260 berechnet. Als Eingabe dienten die konvexen Polyedern der Armsegmente. Sie haben durchschnittlich 30 Eckpunkte. Für die diskretisierte Drehung bei der Quader-Approximation wurde ein Winkel-Abstand von 2° gewählt. Bis auf die Berechnung der Bounding-Box benötigen die Algorithmen auf einer Sun-Workstation mehrere Sekunden. Zur Bewertung der Approximationen sind die Volumen der berechneten Primitive für die Armsegmente in Abbildung 3.2 in logarithmischem Maßstab aufgeführt.

Durch die unterschiedlichen Volumen wird die relative Qualität der Approximationen deutlich. Hierbei fällt auf, daß der Approximations-Fehler bei der Bounding-Box immer geringer ist als bei der Kugel und dem Zylinder. Der Approximations-Fehler ist sogar dem des Quaders ähnlich. Nun ist die Bounding-Box ein achsenparalleler Quader und damit von der Drehlage des Polyeders abhängig. Die dem Quader ähnlichen Volumen-Werte sind also durch eine konstruktionsbedingte günstige Lage der zu approximierenden Polyeder zu erklären.

In zwei Fällen brachte der Algorithmus für den Zylinder ein besseres Ergebnis als die Kugel. Obwohl hier der Zylinder eine allgemeinere Kugel ist und er dadurch auch genauer

approximieren kann, konnten in den anderen Fällen die optimalen Kugel-Approximationen nicht durch die Zylinder verbessert werden. Die Ursache für diesen Mißstand muß in dem Algorithmus zur Berechnung des Zylinders gesucht werden.

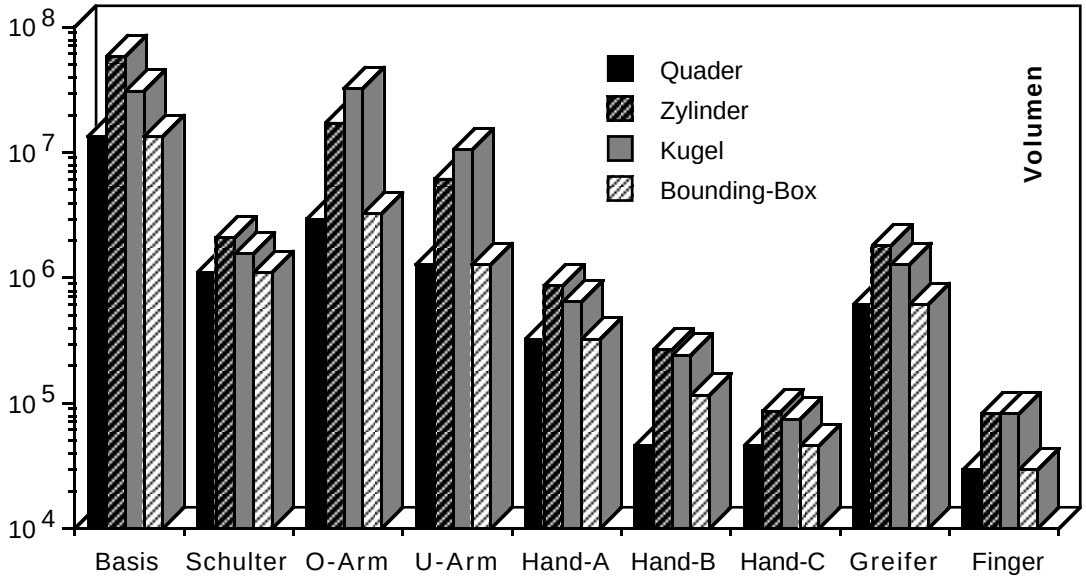


Abbildung 3.2: Volumen der Primitiven-Approximationen des Roboterarms Puma260.

3.3 Abstandsberechnung zwischen Primitiven

Sind die Objekte einer Szene durch Primitive approximiert, dann kann man deren Eigenschaften zur Abstandsberechnung nutzen. Statt die Abstände zwischen den komplizierten Objektdarstellungen zu betrachten, werden die Abstände zwischen den einfacheren Primitiven berechnet. Für die Berechnung des Abstandsvektors zwischen zwei gleichen Primitiven können Verfahren aus der Literatur angewendet werden. Bei zwei unterschiedlichen Primitiven kommt eine Kombination der Algorithmen zum tragen. Da zwischen den Objekten nur Kollisionen erkannt und keine Durchdringungen betrachtet werden sollen, werden negative Abstände nicht berücksichtigt. Tritt dennoch eine Durchdringung auf, dann wird der Abstand Null ausgegeben.

Abstand zwischen Bounding-Boxes

Zu zwei gegebenen Bounding-Boxes soll der Abstandsvektor berechnet werden. Da die Bounding-Boxes definiert sind durch Extremwerte in jeder Koordinaten-Richtung, kann der Abstandsvektor koordinatenweise berechnet werden. Dazu werden die Bounding-Boxes entlang einer Koordinatenachse betrachtet, quasi auf die Achse projiziert. Sind die Achsenabschnitte der Bounding-Boxes nicht überlappend, dann bilden die beiden nächsten Extremwerte zwischen den Abschnitten die Koordinatenwerte der Endpunkte des

Abstandsvektors. Überlappen sich die Achsenabschnitte, so sind die Koordinatenwerte des Abstandsvektors in dieser Richtung nicht eindeutig und es kann *ein* beliebiger Wert innerhalb der Überlappung für *beide* Endpunkte gewählt werden. Diese Schritte werden für jeden der drei Koordinatenachsen durchgeführt. Daraus ergeben sich die einzelnen Koordinatenwerte der Endpunkte des Abstandsvektors.

Abstand zwischen Kugeln

Der Abstandsvektor zwischen zwei Kugeln wird aus ihren Mittelpunkten berechnet. Ist der Abstand größer als die Summe der Radien, dann wird der Vektor zwischen den Mittelpunkten um die Länge der Radien verkürzt. Ansonsten durchdringen sich die Kugeln, und es wird der Abstand Null ausgegeben. Der verkürzte Vektor bildet den Abstandsvektor zwischen den beiden Kugeln.

Abstand zwischen Zylindern

Ein Zylinder ist das Volumen um eine Strecke mit einem bestimmten Radius. Bei der Abstandsberechnung zwischen zwei Zylinder braucht zunächst nicht das gesamte Volumen betrachtet werden, sondern nur die Achsen der Zylinder. Für den Abstand zwischen zwei Strecken wurde von Lumelsky ein effizientes Verfahren im *n*-dimensionalen Raum angegeben [Lumelsky85]:

Zuerst wird eine Fallunterscheidung bzgl. der Länge der zwei Strecken vorgenommen. Sind beide Längen > 0, dann wird zusätzlich untersucht, ob die beiden Strecken parallel zueinander sind. Für jeden dieser fünf Fälle werden die zwei Geraden-Parameter λ_1, λ_2 berechnet. Eingesetzt in die jeweilige Geradengleichung $\boldsymbol{x} = \boldsymbol{p} + \lambda(\boldsymbol{q} - \boldsymbol{p})$ geben sie den Punkt mit minimalem Abstand zu der anderen Geraden an. Dabei sind $\boldsymbol{p}$ und $\boldsymbol{q}$ der Anfangs- und Endpunkte der Strecke. Die Geraden-Parameter werden auf die Strecke ($\lambda \in [0,1]$) projiziert. Erst am Schluß erfolgt dann auf Grund der Strecken-Parameter die eigentliche Abstandsberechnung.

Aus dem Abstand der Zylinderachsen wird, analog zur Kugel, der Zylinderabstand berechnet. Falls der Achsabstand größer ist als die Summe der beiden Radien (die Zylinder berühren sich also nicht), dann wird der Abstandsvektor zwischen den Achsen um die Radien der Zylinder verkürzt. So kommen die Endpunkte des Abstandsvektors auf die Zylinderoberfläche zu liegen.

Abstand zwischen Quadern

Zur Berechnung des Abstandsvektors zwischen zwei Quadern *A* und *B* wurde von Meyer in [Meyer86] ein Algorithmus vorgeschlagen, der durch die Klassifikation von Kanten bezüglich des anderen Quaders effizient wird.

In dem Grundalgorithmus wird getestet, ob der Quader *A* in Quader *B* liegt. Dazu wird für jeden Eckpunkt des Quaders *A* geprüft, ob er auf der Innenseite der sechs Flächen des Quaders *B* liegt. Dies geschieht durch Einsetzen des Eckpunkts in die Ebenengleichung der Fläche. Falls ein Eckpunkt innerhalb aller Flächen liegt, dann schneiden sich die Quader und es kann der Abstand Null ausgegeben werden. Ansonsten schneiden sich die Quader nicht oder es liegt eine Durchdringung vor. In beiden Fallen wird dann der kleinste Abstand aus zwei Gruppen von Abständen gesucht:

(1) Ein Eckpunkt des Quaders *A* bildet den kleinsten Abstand zu einer Fläche des Quaders *B.*

(2) Eine Kante des Quaders *B* bildet den kürzesten Abstand zu dem Quader *A.*

Zu der ersten Gruppe wird in [Meyer86] keine besondere Beschleunigung der Berechnung vorgeschlagen. Hier wird für jeden Eckpunkt die senkrechte Projektion auf die Außenseite jeder Fläche berechnet. Diese muß nicht unbedingt existieren, denn die Projektion kann entweder außerhalb der Quaderfläche liegen oder der Eckpunkt ist auf der Innenseite der Ebene, induziert durch die Quaderfläche.

Für die zweite Gruppe wird in [Meyer86] eine Klassifizierung der Kanten des Quaders *B* bezüglich den Flächen des Quaders *A* eingeführt. Erweitert man die Quaderflächen von *A* ins Unendliche, dann unterteilen sie den kartesischen Raum in 27 Regionen. Zu jedem der acht Eckpunkte, zu jeder der zwölf Kanten und zu jeder der sechs Flächen kann jeweils eine angrenzende Region außerhalb des Quaders *A* zugeordnet werden. Die letzte Region liegt im Innern von *A* und wird von den Quaderflächen begrenzt.

Die Kanten des Quaders *B* werden durch ihre Lage in den Regionen in Klassen aufgeteilt. Die Zugehörigkeit zu einer Klasse kann durch Einsetzen der Endpunkte in die Ebenengleichungen der Flächen festgestellt werden. Liegt eine Kante in mehreren Klassen, so wird sie zerschnitten, sodaß jedes ihrer Teile in einer Klasse liegt. Je nachdem in welcher Klasse eine Kante liegt, kann der kleinste Abstand der zu der Klasse zugeordneten Ecke/Kante/Fläche berechnet werden. Durch die Klasseneinteilung ist garantiert, das keine andere Ecke, Kante oder Fläche als die zugeordnete näher an der Kante liegt. Es kann also eine Vielzahl von Abstandsberechnungen weggelassen werden.

Der Abstandsvektor zwischen den Quadern *A* und *B* ist der kürzeste Abstand aus den zwei oben genannten Gruppen.

Abstand zwischen konvexen Polyedern

Für die Abstandsberechnung im 3-dimensionalen Raum zwischen zwei konvexen Polyedern sind in der Literatur mehrere Verfahren zu finden. Hier sollen einige davon ausgeschieden und zwei andere skizzenhaft beschrieben werden.

Zwei Autoren entwickelten Abstandsalgorithmen mit beschränktem asymptotischem Aufwand. Hierbei wird der Aufwand im O-Kalkül gemessen, abhängig von der Summe $M = M_1 + M_2$ der

Eckenanzahl beider Polyeder, M_1 und M_2. In [Orlowski85] wird ein Verfahren mit einem asymptotischen Aufwand von O(M log M) beschrieben. Dobkin und Kirkpatrick geben in [Dobkin85] einen Algorithmus mit einem Aufwand von $O(M)$ an. Nachteilig bei diesen Algorithmen ist, daß die Betonung auf der Reduzierung des asymptotischen Aufwands liegt. Es ist daher nicht klar, ob diese Verfahren für praktische Probleme (z. B. M ist groß, aber nicht riesen-groß) schnell genug sind. Bei beiden sind wenige oder keine Experimente durchgeführt worden.

In [Cameron86] wird neben der normalen Abstandsberechung auch die Durchdringung in Form von negativen Abständen zwischen zwei Objekten betrachtet. Ansonsten sind die Algorithmen in [Dobkin85] und [Orlowski85] in dem asymptotischen Aufwand schneller. Da hier nur Kollisionen erkannt werden sollen, ist die Berechnung von negativen Abständen nicht notwendig.

Von Gilbert und Johnson et al. werden in [Gilbert88] konvexe Mengen im n-dimensionalen Raum als Grundlage genommen. Die konvexen Mengen sind Polyeder, die durch ihre Eckpunkte definiert sind. Die Abstandsberechnung zwischen zwei Polyedern wird reduziert auf die Berechnung zwischen dem Ursprung und einer konvexen Menge. Dabei werden die sogenannten "support"-Eigenschaften der konvexen Mengen verwendet. Ein iterativer Algorithmus, der nach endlich vielen Schritten terminiert, bildet eine Abstiegs-Prozedur für die Abstände zwischen elementaren Polyedern, die in den konvexen Mengen enthalten sind. Als Startwerte der Iteration können die Schwerpunkte der Polyeder verwendet werden. Ein Unteralgorithmus berechnet die Abstände zwischen dem Ursprung und den elementaren Polyedern. Der Aufwand für die Abstandsberechnung ist annähernd linear in der Gesamtanzahl M von Eckpunkten, die nötig sind, um die konvexen Polyeder zu definieren. Eine günstige Wahl der Startwerte (z. B. die Ergebnisse aus der letzten Berechnung) ermöglicht eine Reduzierung des mittleren Aufwands. Durch ausgiebige Experimente wird gezeigt, daß der Algorithmus numerisch stabil ist.

Bobrow benutzt in [Bobrow89] die begrenzenden Flächen der Polyeder, darstellbar durch die Basispunkte und Normalenvektoren von Halbräumen. Bei diesem Algorithmus ist keine explizite Berechnung der umrandenden Kanten und Eckpunkte der Flächen nötig. Um den Abstand zu berechnen, wird eine nicht-lineare Abstands-Funktion durch Generation einer Reihe von Suchrichtungen entlang der Oberfläche der Objekte minimiert.

Zusammenfassend kann man sagen, daß nur die Algorithmen in [Gilbert88] und [Bobrow89] für die hier gestellte Aufgabe in Frage kommen. Ein Aufwandsvergleich der zwei Verfahren nach der Anzahl von Flächen bzw. Ecken ist prinzipiell möglich, denn nach Euler verhalten sich in einem Polyeder die Anzahl der Flächen proportional zu der Anzahl der Eckpunkte. Die beiden Algorithmen wurden von den Autoren auf einer Harris 800 (entspricht VAX 780) bzw. Apollo DN3000 mit 12,5 MHz MC68020/68881 implementiert. Davon ausgegangen, daß die verwendeten Rechner einen direkten Vergleich zulassen, ist nach den Angaben der Autoren das Verfahren in [Gilbert88] etwa um den Faktor 10 schneller als das in [Bobrow89].

Abstand zwischen unterschiedlichen Primitiven

Die Algorithmen für die Abstandsberechnungen zwischen zwei unterschiedlichen Primitiven ergeben sich aus Kombinationen der oben beschriebenen Verfahren. Entweder werden die Algorithmen den Primitiven angepaßt (bei Bounding-Box/Kugel, Kugel/Zylinder, Kugel/Quader und Zylinder/Quader), oder die Primitiven werden den Algorithmen angepaßt (bei Bounding-Box/Zylinder, Bounding-Box/Quader, Bounding-Box/Polyeder, Kugel/Polyeder, Zylinder/Polyeder, Quader/Polyeder).

Um einen Algorithmus dem gemischten Primitiven-Paar anzupassen, wird ein bekannter Algorithmus für komplexere Primitive vereinfacht. Zum Beispiel kann bei einem Zylinder/Kugel-Paar der Abstands-Algorithmus für zwei Zylinder modifiziert werden. Dieser Algorithmus enthält nämlich auch den Spezialfall, daß ein Zylinder zu einer Kugel degeneriert. Also können für Zylinder/Kugel-Paare alle Teile des Verfahrens weggelassen werden, die über diesen Spezialfall hinaus gehen.

Bei der Anpassung der Primitive wird in einem gemischten Paar ein Primitiv in die Darstellung des anderen umgewandelt, sodaß es sich um ein gleichartiges Paar handelt und einer der oben beschriebenen Algorithmen angewendet werden kann. Zum Beispiel kann bei einem Quader/Polyeder-Paar der Quader in die Darstellung eines Polyeders umgewandelt werden, sodaß die Berechnung zwischen zwei Polyeder erfolgt.

3.4 Laufzeiten der Primitiven-Abstandsberechnung

Als ein Zwischenergebnis sollen hier die Laufzeiten der Abstandsberechnungen zwischen zwei Primitiven beschrieben werden. Sie haben erheblichen Einfluß auf den Aufwand der darauf aufbauenden Kollisionserkennung.

Zur Messung der Laufzeiten werden in einer Testumgebung (beschrieben im Anhang A.1) von einem "Generator" Szenen erzeugt. Jede Szene enthält zwei Primitive in zufälliger Lage und Orientierung zueinander. Die Szenen können also auch zwei kollidierende Primitive enthalten. Da während der Kollisionserkennung aber vorwiegend Szenen vorkommen die keine sich berührende Objekte beinhalten, werden die Szenen durch einen "Selektor" gefiltert. Der "Selektor" scheidet alle Szenen mit auftretender Kollision aus. Danach werden die ausgewählten Szenen von einem "Tester" übernommen und die Laufzeiten gemessen. Da die Meßgenauigkeit der internen Uhr des Testsystem nur 16,667 ms beträgt, muß die Genauigkeit durch Wiederholungen erhöht werden. Der "Tester" wiederholt die Abstandsberechnung innerhalb einer Zeitmessung z. B. 1000 mal, sodaß die von der Systemuhr erhaltene Zeit durch den Faktor 1000 geteilt werden kann. Dadurch wird auch die Meßungenauigkeit um diesen Faktor verringert. Die Laufzeiten können so mit einem Meßfehler von nur ±0,0166 ms

angegeben werden. Von dem "Generator" werden wiederum pro Primitiven-Paar mehr als 7500 verschiedene Szenen erzeugt, sodaß eine gemittelte Laufzeit berechnet werden kann. (Bei Primitiven, die Polyeder enthalten wurden weniger Szenen generiert; siehe unten.) Diese Messungen wurden auf einer SPARC 2 mit der höchsten Optimierungsstufe des C-Kompilers (-O4) durchgeführt.

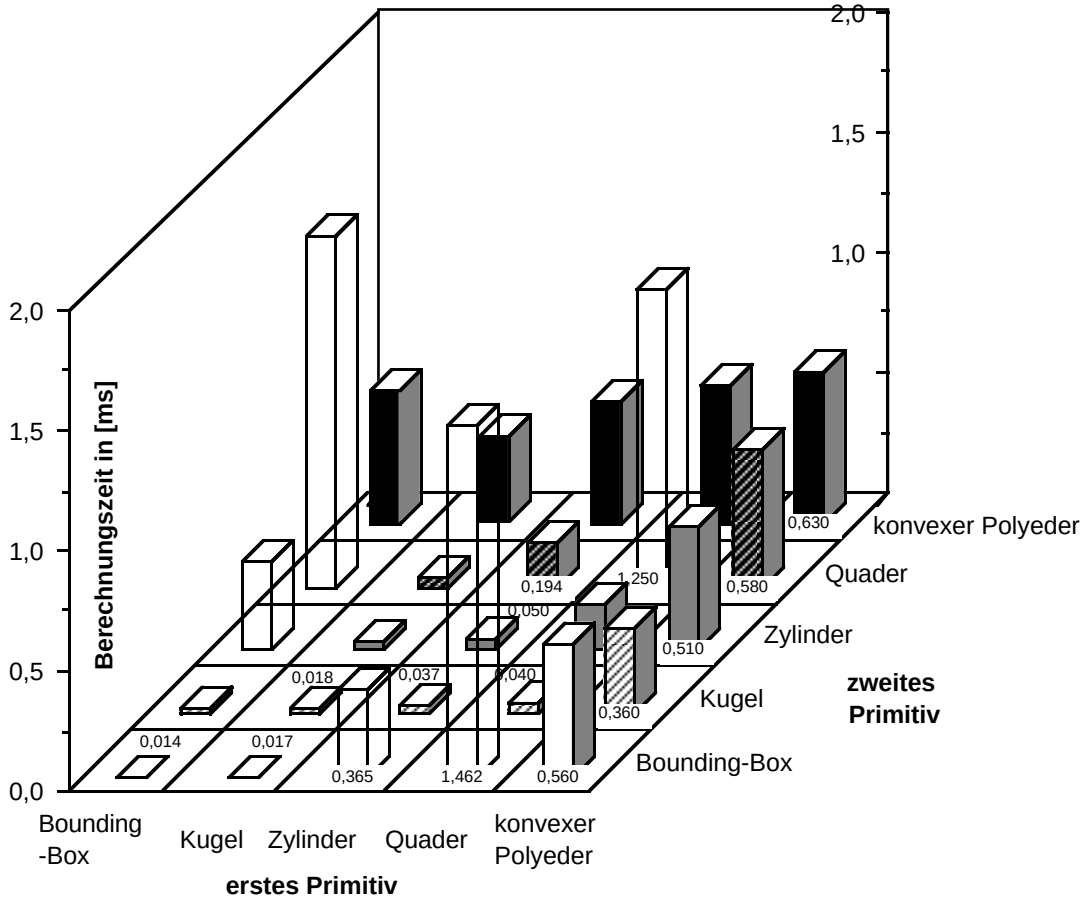


Abbildung 3.3: Mittlere Berechnungszeiten für die Abstandsvektoren zwischen je zwei Primitive

Enthalten die Primitiven-Paare konvexe Polyeder, dann muß bei der Laufzeitmessung die Komplexität der Polyeder mitberücksichtigt werden. Die Komplexität eines Polyeders hängt von der Anzahl der Eckpunkte ab, die den Polyeder beschreiben. Da von Gilbert und Johnson et al. in [Gilbert88] schon genaue Messungen durchgeführt wurden, werden hier die Laufzeiten nur für die benötigten Armsegmente gemessen. Die konvexen Polyeder der Armsegmente bestehen im Mittel aus 30 Eckpunkten. Für die Messungen wird bei dem iterativen Algorithmus als Abbruchbedingung eine Abstandsgenauigkeit von 10^{-9} gewählt. Auch können die Laufzeiten bei Polyedern durch eine Initialisierung mit den Ergebnissen aus der letzten Berechnung verbessert werden. D. h. die hier erhaltenen Meßwerte für Polyeder-Abstände, gemittelt über ca. 80 Szenen, dienen nur als Anhaltspunkt bzw. als Bestätigung der Messungen in [Gilbert88].

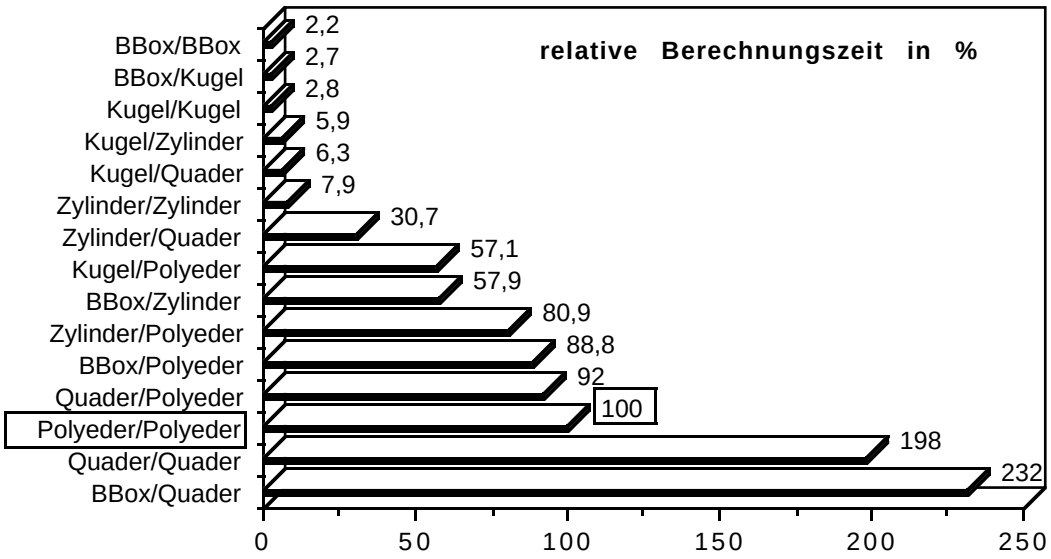


Abbildung 3.4: Relative Laufzeiten der Abstandsberechnungen zwischen Primitiven bezogen auf Polyeder-Berechnungen

In Abbildung 3.3 ist zu jedem Primitiven-Paar die Laufzeit der Abstandsberechnung unter den oben genannten Meßbedingungen angegeben. Es ist dabei irrelevant, welches Primitiv an erster Stelle steht; die Abbildung ist achsensymmetrisch zu der Diagonalen. Im Wesentlichen bilden die Laufzeiten ein schiefe Ebene, die ihren niedrigsten Punkt bei dem Bounding-Box-Paar und ihren höchsten Punkt bei dem Polyeder-Paar hat. Dazu kommen drei Ausreißer, die sich durch übermäßig hohe Zeiten auszeichnen. Sieht man von diesen überhöhten Werten ab, so entsprechen die Laufzeiten den Erwartungen: sie verhalten sich proportional zu der Komplexität beider Primitiven. Die Ausreißer entstehen durch die uneffizente Abstandsberechnung zwischen Quadern aus [Meyer86]. Da bei einem Paar mit Bounding-Box und Quader die Bounding-Box in die Darstellung eines Quaders gebracht wird, entstehen so die anderen zwei Ausreißer. Sie können gemildert werden, indem man die Quader als konvexe Polyeder darstellt, denn die Abstandsberechnung zwischen Polyeder ist, trotz der komplexeren Primitive, schneller als die zwischen den einfachen Quadern.

In Abbildung 3.4 sind die Laufzeiten für die Primitiven-Paare nochmals einzeln angegeben. Sie sind als relative Zeiten zu der komplexesten Abstandsberechnung zwischen zwei Polyeder sortiert dargestellt. Hieraus ersieht man die Zeitersparnis, die die Wahl eines speziellen Primitiven-Paars zu der aufwendigsten Darstellung erbringt. So wird deutlich, daß die ersten fünf Primitiven-Paare unter 10 % liegen und damit eine erhebliche Reduzierung des Rechenaufwandes bedeuten.

3.5 Abstandsberechnung zwischen Objekten

Die Abstandsberechnung bei der Kollisionserkennung zwischen Objekten (dargestellt durch konvexe Polyeder) kann durch die off-line berechnete Primitiven-Approximation beschleunigt werden. Dazu wird der Abstandsvektor statt zwischen den Polyedern zwischen den Primitiven berechnet. Da der Abstand zwischen zwei Primitiven nie größer als der zwischen den approximierten Polyedern ist, stellt dieser eine konservative Annäherung an den realen Abstand dar. D. h. die Approximation des Abstandsvektors ist immer kürzer als der reale Vektor. Auch stehen für ein Objekt mehrere Primitive als Approximation zu Verfügung. Die Abstandsberechnung kann also in unterschiedlichen Genauigkeitsstufen erfolgen.

Bei der Berechnung des Abstandsvektors zwischen zwei Objekten wird in dem ersten Schritt nach einer bestimmten Strategie ein Primitiven-Paar ausgewählt. Zwischen diesem Paar wird der Abstandsvektor berechnet. Ist der Abstand größer als ein vorgegebener Schwellwert, dann kann der Vektor als Ergebnis ausgegeben werden. Ist er kleiner und es existiert noch ein unbenutztes Primitiven-Paar, wird die Berechnung damit wiederholt. Führt keine Berechnung mit den Primitiven zu einem Abstand der größer als der Schwellwert ist, muß der Vektor zwischen den konvexen Polyedern berechnet werden. In Abbildung 3.5 ist in (a) ein Beispiel für die erfolgreiche Abstandsberechnung zwischen den Primitiven gezeigt. In Abbildung 3.5 (b) muß auf eine genauere Primitiven-Approximation zurückgegriffen werden, um einen Abstandsvektor mit einer Länge größer als den Schwellwert zu erhalten.

Bei der Auswahl der Primitiven-Paare stehen mehrere Möglichkeiten zur Verfügung. Die Strategie zur Wahl der Paare beeinflußt stark den Aufwand, der benötigt wird, um einen zufriedenstellenden Abstandsvektor zu erhalten. Ein Vergleich der Strategien beim Einsatz in der Kollisionserkennung wird in dem Kapitel "Experimentelle Ergebnisse" durchgeführt. Hier einige Beispiele für Auswahl-Strategien:

- gleichmäßig: Es wird für jedes der beiden Objekte nach einer festen Reihenfolge das nächst komplexere Primitiv gewählt. Z. B. könnte man für beide Objekte nacheinander die Bounding-Box, die Kugel, den Zylinder, den Quader und am Schluß den konvexen Polyeder als Primitiv wählen. Es werden also keine gemischten Primitiven-Paare benutzt. Hierbei kann keine Rücksicht auf den Approximations-Fehler genommen werden.
- in Laufzeitreihenfolge: Hier werden die Paare nach dem Aufwand der Abstandsberechnung sortiert. Die Reihenfolge entspricht dann der in Abbildung 3.4 angegebenen Reihe der relativen Laufzeiten. Nachteilig an dieser Strategie ist, daß Primitive, die eine schlechtere Approximation darstellen dennoch benutzt werden.
- Reduzierung des Approximations-Fehlers: Bei dieser Strategie wird bei jeder Wahl des nächsten Primitiven-Paares die Summe der Approximation-Fehler reduziert. Die Paare sind dabei in der Laufzeitreihenfolge aus der Abbildung 3.4 sortiert. Verschlechtert ein Primitiven-Paar den Approximations-Fehler des letzten Paares, so wird dieses Paar übersprungen und mit dem nächsten weiter gerechnet.

- nur Polyeder und Bounding-Box: Statt zur Abstandsberechnung alle Primitiven zu verwenden, werden hier nur zwei Darstellungsformen benutzt. Die Bounding-Box als grobe Approximation mit der schnellsten Abstandsberechnung, und der konvexe Polyeder mit einer langsamen Abstandsberechnung als genaueste Darstellung. Scheitert die Abstandsberechnung mit den Bounding-Boxes, dann wird sofort der Abstand mit den konvexen Polyedern berechnet.

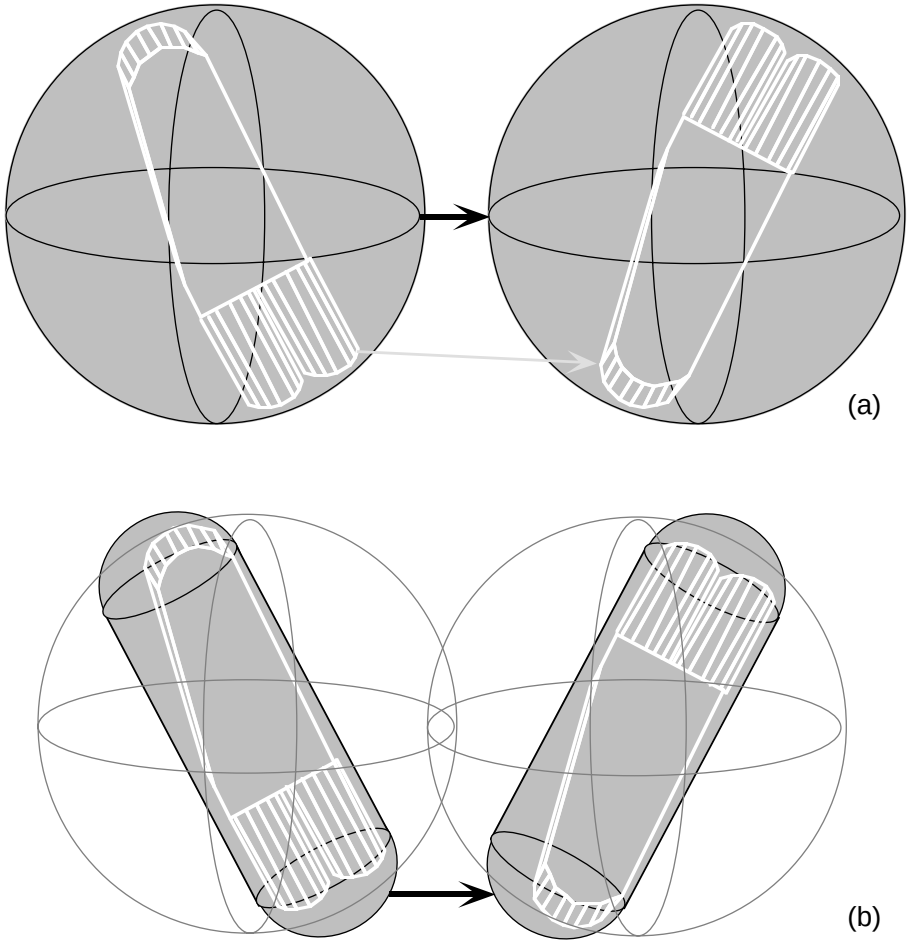


Abbildung 3.5: Abstandsberechnung bei einer Primitiven-Approximation; (a): Kugel-Abstand > Schwellwert; (b): Kugel-Abstand ≤ Schwellwert.

4 Dynamische Hierarchien

Der Ansatz der Primitiven-Approximation aus dem vorhergegangenen Kapitel diente zur Beschleunigung einer einzelnen Abstandsberechnung in der Kollisionserkennung. Im Gegensatz dazu dienen die hier eingeführten Hierarchien zur Reduzierung der Objektanzahl. Dabei werden durch sukzessives Zusammenfassen mehrere Objekte durch einzelne dargestellt. Mit einer hierarchischen Darstellung wird zusätzlich eine schnelle Selektion der relevanten Objekte bei der Kollisionserkennung erreicht.

In diesem Kapitel wird basierend auf Kompositionen eine hierarchische Darstellung von Teilszenen beschrieben. Ein Gütekriterium dient zur Auswahl optimaler Hierarchien. Probleme bei der Anwendung der Hierarchien auf Roboterarme werden diskutiert und gelöst. Die Zusammenfassung ergibt eine Modellierung in dynamischen Hierarchien.

4.1 Hierarchische Modellierung

Bei der Primitiven-Approximation wird ein Objekt durch Primitive dargestellt, um die Abstandsberechnung zu beschleunigen. Hier soll nicht eine einzelne Berechnung beschleunigt, sondern die Anzahl der Berechnungen reduziert werden. Diese wird durch die Anzahl der Objekt-Paare einer Szene bestimmt. Faßt man nun mehrere Objekte zu einem zusammen, dann reduziert sich damit die Anzahl der Paare in der Kollisionserkennung. Eine Zusammenfassung mehrerer Objekte zu logisch einem Objekt soll **Komposition** heißen. Wieviele Objekte auf einmal zusammengefaßt werden und wie eine Komposition dargestellt wird, ist hierbei nicht festgelegt. Aber um eine brauchbare Abstandsberechnung durchführen zu können, muß die Darstellung einer Komposition konservativ sein. D. h. die Darstellung enthält vollständig das Volumen der zusammengefaßten Objekte. Analog zur Primitiven-Approximation berechnet man statt dem genauen Abstand zwischen allen Objekt-Paaren, einen angenäherten Abstand zwischen den Kompositionen. Da die Kompositionen konservativ approximiert sind, ist dieser Abstand immer kleiner als der reale Abstand. Die konservative Approximation einer Komposition kann genauso wie ein Primitiv off-line berechnet werden.

In der Abbildung 4.1 sind die Objekte A_1, A_2 und A_3, A_4 zusammengefaßt und durch die Objekte B_1 und B_2 approximiert. A_1, A_2 und A_3, A_4 bilden also zwei Kompositionen. Dadurch

wird die Anzahl der Objekte in dieser Szene von vier auf zwei reduziert. Um die Abstände zwischen den Kompositionen berechnen zu können werden sie durch die Quader B_1 und B_2 konservativ approximiert. Zur Abstandsberechnung werden also die zusammengefaßten Objekte durch die Approximation der Komposition ersetzt. (Aus diesem Grund wird im Folgenden bei dem Begriff "Komposition" nicht ausdrücklich zwischen der Zusammenfassung von Objekten und der Approximation der Zusammenfassung unterschieden.)

Wiederholt man das Zusammenfassen und Ersetzen der Objekte für andere Objekte, aber auch für die Kompositionen selbst, dann baut sich eine sogenannte **Kompositions-Hierarchie** auf. Auf der untersten Ebene der Hierarchie befinden sich die Original-Objekte. Jede Ebene darüber besteht aus Kompositionen. Je höher die Ebene, desto gröber wird die Darstellung der Objekte. Auf der obersten Hierarchie-Ebene befindet sich nur noch eine Komposition, die alle Original-Objekte zusammenfaßt. Diese hierarchischen Darstellungen sind den *assembly-trees* von Faverjon in [Faverjon89] ähnlich.

In Abbildung 4.1 (a) sind die Kompositionen A_1, A_2 und A_3, A_4 durch B_1 und B_2 angenähert. Diese wiederum sind durch den Quader C_1 approximiert. Dadurch wurde eine Hierarchie mit drei Ebenen aufgebaut. In der Abbildung 4.1 (b) ist die zur Hierarchie gehörenden Baumstruktur mit den Approximationen abgebildet. Auf der untersten Ebene befinden sich die Original-Objekte A_1, A_2, A_3, und A_4, darüber die Approximation B_1 und B_2 der zwei Kompositionen und oben deren Kompositions-Approximation C_1. An der Baumstruktur ist zu erkennen, daß jeder Schnitt von links nach rechts, also z. B. B_1, B_2 oder A_1, A_2 und B_2, die Szene konservativ approximiert.

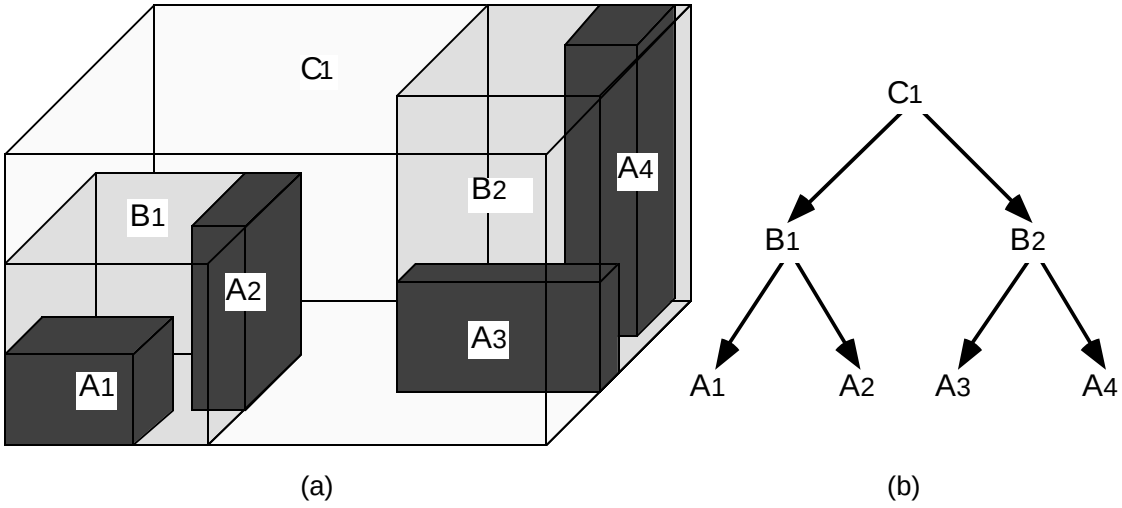


Abbildung 4.1: Bildung von Kompositions-Hierarchien; (a): Beispiel-Szene mit Kompositionen; (b): zugehörige Baumstruktur.

Berechnung der Kompositionen

Wie berechnet man nun eine Approximation zusammengefaßter Objekte? Das Problem soll hier nur für die Komposition von zwei Objekten untersucht werden. Falls mehr als zwei zusammengefaßt werden sollen, dann kann man die Berechnung auf die Komposition von zwei Objekten zurückführen. Als Grundmodellierung der Objekte werden wie bei der Primitiven-Approximation konvexe Polyeder angenommen.

Als Kompositionen können wiederum die unterschiedlichen Primitive oder die Polyeder selbst verwendet werden. Der Ausgangspunkt der Kompositionsberechnung sollte immer die genaueste zur Verfügung stehende Darstellung der Objekte sein. Ansonsten ist die Approximation auf einer höheren Hierarchie-Ebene nicht so kompakt, wie sie eigentlich sein könnte. Der Approximationsfehler aus der unteren Ebene wird mit in die höhere Ebene übernommen.

Eine Ausnahme dazu bildet die Bounding-Box. Per Definition gibt die Bounding-Box die extreme Ausdehnung des Objekts in jeder Koordinaten-Richtung an. Nun ist es gleichgültig, ob die zu komponierenden Objekte durch Polyeder oder durch Bounding-Boxes dargestellt wurden. Die Komposition durch die Bounding-Box bleibt gleich. Also ist eine Komposition berechnet aus zwei Polyedern immer gleich der Komposition berechnet aus ihren Bounding-Box-Approximationen. Diese Eigenschaft ist nicht auf z. B. die Kugel übertragbar. Die Kugel-Komposition zweier Polyeder ist im Allgemeinen nicht gleich der Kugel-Komposition zweier Kugeln, die die Polyeder approximieren. Eine Kompositions-Hierarchie aus Bounding-Boxes braucht beim Aufbau nur für die zweite Hierarchie-Ebene (erste Ebene mit Kompositionen) die Polyeder zu verwenden. Bei jeder höheren Ebene genügt die Verwendung der Bounding-Boxes aus der nächst-tieferen Ebene. Diese besondere Eigenschaft beschleunigt erheblich den Aufbau von Kompositions-Hierarchien aus Bounding-Boxes.

Es sind Algorithmen gesucht, die aus zwei durch Polyeder dargestellten Objekten die Kompositionen berechnen. Die Algorithmen entsprechen im Prinzip der Berechnung von Primitiven zu einem gegebenen Polyeder. Anstatt die *zwei* Polyeder einzeln zu verwenden, werden die Polyeder-Eckpunkte der zu komponierenden Objekte zusammengefaßt und als *ein* neuer konvexer Polyeder betrachtet. Mit diesem konvexen Polyeder können die Primitiven als Kompositionen nach den Algorithmen der Primitiven-Approximation berechnet werden.

4.2 Optimale Hierarchien

Durch die hierarchische Modellierung hat man nun aus der Sicht der Kollisionserkennung eine starke Reduzierung der Objektanzahl erreicht. Neben dieser Reduzierung kann ein weiterer Vorteil dieser Modellierung ausgenutzt werden: die Hierarchien reflektieren die geometrische Anordnung der Objekte im Raum. Dadurch ist ein schnelles Auffinden der für die Kollisionserkennung wichtigen Objekte möglich. Bei jeder Zusammenfassung von Objekten zu

Kompositionen bleibt also die Frage, *welche* Objekte durch ein neues approximiert werden sollen.

Ohne Einschränkung kann man aus ein und derselben Szene sehr viele unterschiedliche Kompositions-Hierarchien aufbauen. Um diesen Vorgang zu steuern, wird hier ein Gütekriterium eingesetzt. Das Gütekriterium bewertet jede der möglichen Kompositionen einer Szene. So kann man es bei jedem Schritt im Aufbau einer Kompositions-Hierarchie einsetzen. Wird auf jeder Ebene einer Hierarchie der Wert des Gütekriteriums minimiert, dann soll es sich um eine **optimale Kompositions-Hierarchie** handeln. Wohlgemerkt: der Einsatz des Gütekriteriums ändert nichts an der Geometrie einer einzelnen Komposition. Es beeinflußt nur *welche* Objekte komponiert werden sollen.

Das Gütekriterium sollte so gewählt werden, daß die damit aufgebaute Hierarchie eine möglichst schnelle Kollisionserkennung erlaubt. In Abbildung 4.1 würde man eine Komposition von A_1 und A_2 als günstiger erachten, als die Zusammenfassung von A_1 und A_3. Jede Komposition sollte für sich die kompakteste Zusammenfassung aller zu diesem Zeitpunkt möglichen Kompositionen sein. Als Maß für die Kompaktheit können z. B. folgen Gütekriterien dienen:

- Abstand der komponierten Objekte: Aus einer Szene wählt man die zwei Objekte deren Schwerpunkte den kürzesten Abstand zueinander haben. Somit fallen weit entfernte Objekte für eine Zusammenfassung weg. Leider läßt dieser Abstand keine Aussage über das Volumen zu.
- Volumen der Kompositionen: Von allen möglichen Kompositionen einer Szene wählt man diejenige, deren Volumen am geringsten ist. Nachteilig bei diesem Gütekriterium ist, daß lange dünne Kompositionen möglich werden. Solche Zusammenfassungen sind aber für die Kollisionserkennung sehr ungünstig.
- maximale Durchmesser der Kompositionen: Von allen möglichen Kompositionen einer Szene wählt man diejenige, deren maximaler Durchmesser am geringsten ist. Dieses Gütekriterium ist ein gutes Maß für die Kompaktheit einer Komposition-Hierarchie. Es werden damit Hierarchien aufgebaut, die denjenigen, die ein Mensch aufbauen würde, sehr nahe kommen.
- Oberfläche der Kompositionen: Von allen möglichen Kompositionen einer Szene wählt man diejenige, deren Oberfläche am geringsten ist. Auch dies ist ein ähnlich gutes Gütekriterium wie der Durchmesser einer Komposition. Aber die Berechnung der Oberfläche ist aufwendiger als die des maximalen Durchmessers einer Komposition.

Natürlich lassen sich die verschiedenen Gütekriterien beliebig kombinieren. Neben diesen Gütekriterien, die mehr auf die geometrische Anordnung der Objekte eingehen, sind sicher auch andere denkbar. Eine weitere Sorte von Gütekriterien könnte die Wahrscheinlichkeit für Kollisionen mit anderen Objekten einbauen. Bei einem Roboterarm treten mit dem Greifer sicherlich mehr Kollisionen auf, als mit der Basis oder der Schulter. Also sollte ein solches

Gütekriterium versuchen die Objekte mit großer Kollisions-Wahrscheinlichkeit (z. B. den Greifer) möglichst weit oben in einer Hierarchie zu platzieren, damit sie bei der Kollisionserkennung möglichst früh bearbeitet werden.

Für die Kollisionserkennung sind nicht nur die Roboterarme, sondern allgemein dynamische Hindernisse zu modellieren. Die Wahrscheinlichkeit für Kollisionen eines Objekts kann für den allgemeinen Fall nicht ermittelt werden. Deshalb wird hier das Gütekriterium verwendet, das die geometrische Anordnung der Objekte am besten zum Ausdruck bringt.

Aufbau optimaler Hierarchien

Zu einer gegebenen Szene mit Original-Objekten und einem Gütekriterium kann eine optimale Kompositions-Hierarchie ähnlich dem Nächste-Nachbar-Algorithmus berechnet werden. Dieser Algorithmus liefert für ein abstraktes Abstandsmaß (hier das Gütekriterium) einen minimal spannenden Baum, d. h. die Summe der abstrakten Abstände ist minimal (siehe [Duda72]). Diese Berechnung erfolgt hier off-line und führt zu einer Hierarchie mit binärer Baumstruktur. Dabei ist nicht festgelegt, durch welches geometrische Primitiv die Kompositionen dargestellt werden.

Zuerst werden alle Objekte der Szene in eine Liste eingefügt. Dann wird das Objektpaar aus der Liste gesucht, welches das Gütekriterium minimiert. Für dieses Paar wird eine Komposition berechnet und statt den zwei ursprünglichen Objekten in die Liste eingefügt. Die Liste ist jetzt um ein Element kürzer. Diese Schritte werden solange wiederholt, bis die Liste einelementig ist. Das Ergebnis ist ein bzgl. dem Gütekriterium optimaler Binärbaum.

Bei den meisten Szenen handelt es sich um eine logarithmische Reduzierung der Objekte mit ansteigender Höhe des Baums. Allgemein können pathologische Objekt-Anordnungen in einer Szene auch eine nur lineare Reduzierung bewirken. Diese hat aber nicht ihre Ursache im Algorithmus, sondern in der geometrischen Anordnung der Objekte oder in dem Gütekriterium.

4.3 Probleme der Dynamik

Bei den oben beschriebenen optimalen Kompositions-Hierarchien werden die Objekte einer Szene mit Hilfe eines Gütekriteriums schrittweise zusammengefaßt. Diese Hierarchien können direkt eingesetzt werden, um statische Objekte (wie z. B. Hindernisse) darzustellen. Die off-line berechnete Hierarchie bleibt dann über den gesamten Zeitraum der Kollisionserkennung unverändert. Leider kann man diese Darstellung nicht direkt auf eine Zusammenfassung dynamischer Objekte (wie z. B. Roboterarme) anwenden. Die einzelnen Armsegmente eines Roboterarms verändern beispielsweise bei einer Bewegung ihre Lage relativ zueinander. Durch diese Veränderung können für die Kompositions-Hierarchien die folgenden zwei Probleme entstehen:

1. Die einzelnen off-line berechneten Kompositionen stimmen nicht mehr mit den Objekten überein, die sie approximieren sollen, d. h. die Kompositionen sind nach der Bewegung nicht mehr konservativ. In Abbildung 4.2 enthält z. B. die berechnete Komposition von Ober- und Unterarm nach der Abwärtsbewegung des 2. Segments nicht mehr vollständig den Unterarm. Um beide Armteile konservativ zu approximieren, muß die Komposition nach der Bewegung angepaßt werden.

2. Die Baumstruktur der Hierarchie ist nicht mehr optimal bezüglich dem vorgegebenen Gütekriterium. D. h. nach einer Bewegung ist nicht mehr garantiert, daß die Objekte in der gleichen Reihenfolge zusammengefaßt werden, wie vor der Bewegung. In Abbildung 4.3 (a) wird z. B. dem Gütekriterium folgend der Greifer mit dem Unterarm komponiert. Nach der Bewegung ist in Abbildung 4.3 (b) nach dem Gütekriterium eine Zusammenfassung von Greifer und Basis günstiger. Hier ist nach der Bewegung eine Aktualisierung der Baumstruktur notwendig.

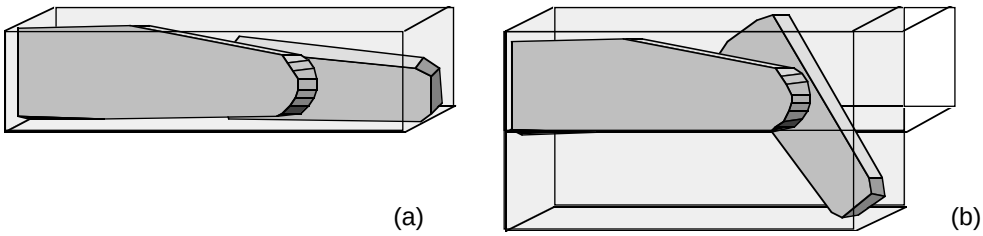


Abbildung 4.2: Notwendige Aktualisierung der Komposition nach einer Bewegung der Armsegmente; (a): korrekte Komposition vor der Bewegung; (b): nicht mehr konservative Komposition nach der Bewegung

Für eine Zusammenfassung von dynamischen Objekten in einer solchen hierarchischen Darstellung muß also eine off-line berechnete optimale Kompositions-Hierarchie während der Kollisionserkennung (on-line) den möglichen Veränderungen der Objekte angepaßt werden, um konservativ und optimal zu bleiben. Dazu kann man zum Einen die Kompositionen aktualisieren und zum Anderen abhängig von der aktuellen Armkonfiguration die optimale Baumstruktur auswählen. Diese Aktualisierungen werden im Folgenden beschrieben.

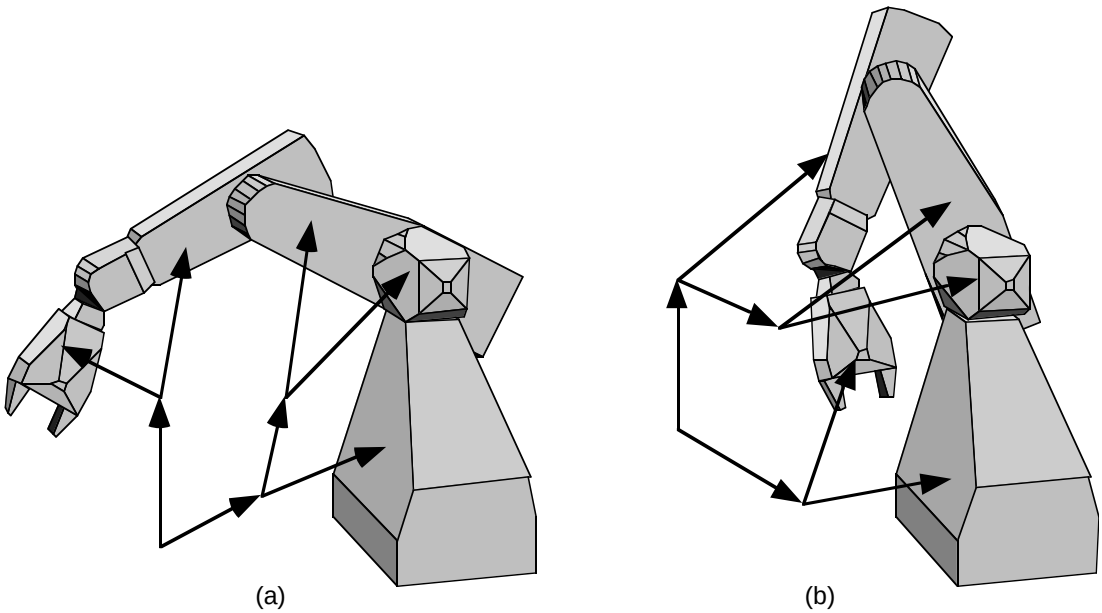


Abbildung 4.3: Unterschiedliche optimale Baumstrukturen bzgl. eines Gütekriteriums in verschiedenen Armstellungen; (a): Zusammenfassung von Greifer und Unterarm; (b): Zusammenfassung von Greifer und Basis.

4.4 Aktualisierung der Kompositionen

Bei einer Komposition werden mehrere Objekte zu logisch einem zusammengefaßt und durch eine Approximation dargestellt. Möchte man dynamische Objekte (z. B. Armsegmente) durch Kompositionen darstellen, dann entstehen Probleme. Eines der Probleme ist die notwendige Aktualisierung der Kompositionen nach einer Bewegung der Objekte.

Bei der Beschreibung der Kompositions-Hierarchien wurde die Wahl des Primitivs, mit dem die Komposition dargestellt wird, offen gelassen. Man hat beim Aufbau einer Hierarchie die Freiheit z. B. Quader, Zylinder oder andere Primitive zu verwenden. Für die Kompositions-Hierarchie wird hier die Bounding-Box als Primitiv gewählt. Sie stellt bezüglich der Berechnung des Abstandsvektors bei der Primitiven-Approximation das günstigste Primitiv dar. Zudem kann eine Approximation durch eine Bounding-Box on-line sehr schnell berechnet werden (siehe Kapitel 3.4).

Die notwendige Aktualisierung der Kompositionen nach einer Bewegung der dynamischen Objekte kann mit der Bounding-Box schnell ausgeführt werden. On-line werden nach jeder Veränderung der Objekte zueinander in der Hierarchie alle Kompositionen neu berechnet. Beugt z. B. der Roboter in Abbildung 4.2 (a) seinen Unterarm, dann wird nach dieser Bewegung die Bounding-Box neu erzeugt. In (b) enthält die neue Bounding-Box wieder konservativ beide Armsegmente. In einer Kompositions-Hierarchie sind nach einer Bewegung mehrere Kompositionen zu aktualisieren. Dies kann z. B. top-down oder bottom-up geschehen.

Bei der bottom-up-Berechnung werden für jede Hierarchie-Ebene alle Kompositionen neu berechnet. Man beginnt bei der untersten Ebene und endet mit der obersten. Normalerweise würde man zur Berechnung der Kompositionen immer die genaueste Darstellung der Objekte wählen. Dies erfordert bei konvexen Polyeder als Modellierung der Objekte mit insgesamt n Eckpunkten und einer ausgeglichenen Hierarchie mit h Ebenen einen Aufwand von $O(nh)$ Berechnungen. Bei der Bounding-Box als Darstellung der Kompositionen ist dieses Vorgehen nicht nötig. Die Bounding-Box gibt in jeder Koordinatenrichtung die extreme Ausdehnung des zu approximierenden Objekts an. Um die Bounding-Box von mehreren Kompositionen zu berechnen, genügt es die extremen Ausdehnungen der Kompositionen zu betrachten. Da nun die Kompositionen an sich wiederum Bounding-Boxes sind, muß man nicht auf die konvexen Polyeder zurückgreifen. Damit reduziert sich die Anzahl der Berechnungen bei einem bottom-up-Vorgehen mit Bounding-Boxes in ausgeglichenen Hierarchien auf $O(n + ld(2) + \ldots + ld(h))$.

Bei der top-down-Aktualisierung der Hierarchie werden nicht alle Kompositionen neu berechnet. Das Verfahren beginnt an der obersten Ebene und berechnet für jede Ebene nur die Kompositionen, die zur Kollisionserkennung auch wirklich gebraucht werden. Da die Kompositionen auf den Ebenen zwischen der aktuellen und der untersten noch nicht aktualisiert sind, können sie auch nicht zur Berechnung herangezogen werden. Bei jeder Berechnung muß also auf die Polyeder zurückgegriffen werden. Auch erfordert das Verfahren einen nicht unerheblichen Verwaltungsaufwand bei Suche der betroffenen Polyeder. Ein dritter Nachteil ist, daß die Aktualisierung nicht vor jedem Bewegungsschritt komplett ausgeführt werden kann, da nicht bekannt ist, welche Kompositionen aktualisiert werden müssen. Dies ergibt sich erst bei der Abstandsberechnungen mit den anderen Objekten. D. h. es muß während der Abstandsberechnung das Aktualisierungs-Verfahren stückweise angestoßen werden. Aus diesen Abschätzungen heraus, wird im Folgenden nur die bottom-up-Aktualisierung verwendet.

4.5 Auswahl-Funktion für Baumstrukturen

Dynamische Objekte (z. B. Armsegmente) verändern ihre Lage relativ zueinander. Sind diese Objekte in einer hierarchischen Darstellung zusammengefaßt, dann muß sie nach einer Bewegung der Objekte aktualisiert werden. Im Kapitel 4.4 wurde eine Anpassung der einzelnen Zusammenfassungen (Kompositionen) an die veränderte Situation eingeführt. Hier wird die notwendige Aktualisierung der optimalen Baumstruktur der Hierarchie nach einer Bewegung beschrieben.

Für die meisten Kompositions-Hierarchien sind die möglichen Bewegungen der Objekte im Voraus bekannt. Bei diesen kann man für jede mögliche Konfiguration der Objekte die optimale Baumstruktur off-line berechnen. Dabei definiert eine Baumstruktur die Art und Reihenfolge der Zusammenfassung der Objekte. On-line wird dann über eine Auswahl-Funktion für eine gegebene Konfiguration die optimale Baumstruktur ausgewählt. Die Auswahl-Funktion ist eine

Abbildung von der Menge der möglichen Konfigurationen auf die Menge der optimalen Baumstrukturen.

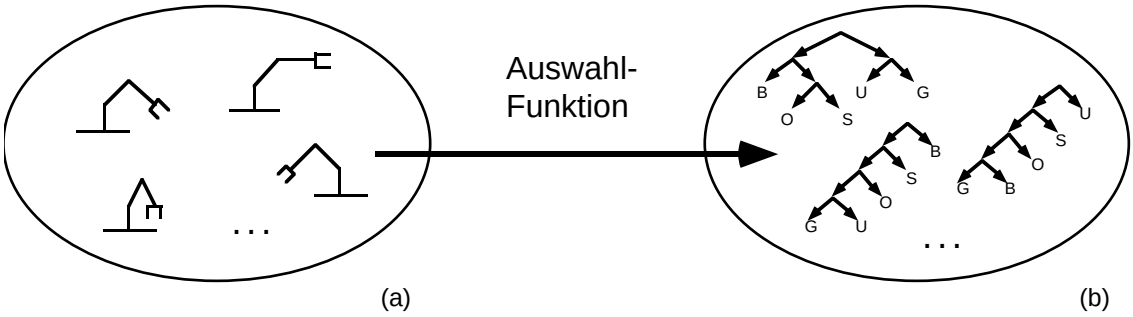


Abbildung 4.4: Schema der Auswahl-Funktion von der Menge der Konfiguratioen bzw. Gelenkwinkelsätze (a) auf die Menge der optimalen Baumstrukturen (b).

Zum Beispiel lassen bei einem Roboterarm die Gelenke in den Verbindungen der Armsegmente nur bestimmte Lageveränderungen der einzelnen Segmente zu. Off-line kann man die mehrdimensionalen Gelenkwinkel-Intervalle bestimmen, in denen jeweils eine Baumstruktur der Kompositions-Hierarchie optimal bleibt. On-line sucht dann eine roboterabhängige Auswahl-Funktion zu einem gegebenen Gelenkwinkelsatz aus einer Tabelle die zu dieser Armstellung optimale Baumstruktur heraus (siehe Abbildung 4.4). Durch diese Aktualisierung steht für die Kollisionserkennung in jedem Moment bezüglich eines vorgegebenen Gütekriteriums die günstigste Struktur der Kompositions-Hierarchie zur Verfügung.

Kombinatorische Abschätzung

Je nachdem, wieviele Objekte in einer Kompositions-Hierarchie zusammengefaßt werden sollen, können im Allgemeinen sehr viele unterschiedliche Baumstrukturen für die Hierarchie aufgebaut werden. Zwar wird die Anzahl optimaler Baumstrukturen durch das Gütekriterium vermindert, aber bei einem ungünstigen Gütekriterium können nach unterschiedlichen Bewegung der Objekte dennoch viele Baumstrukturen optimal sein. Die Anzahl der möglichen Baumstrukturen wird nun abhängig von den beteiligten Objekte abgeschätzt.

In Tabelle 4.1 ist die Anzahl der Objekte durch die Anzahl N der Blätter eines Hierarchie-Baumes gegeben. Daraus läßt sich die Anzahl M der ungeordneten, binären Bäume mit N Blättern ableiten. Hierbei sind Binärbäume, die sich durch Vertauschen von einem linken und einem rechten Ast ineinander überführen lassen als *ein* Baum gezählt ("ungeordnete Bäume", denn für die Kollisionserkennung ist eine Reihenfolge der Kompositionen auf gleicher Ebene unerheblich). An jedem Binärbaum können nun N verschiedene Objekte an den Blättern unterschiedlich angeordnet werden. Wäre keine Vertauschung der Äste möglich, dann ergäbe dies $N!$ Kombinationen der Objekte. Aber für jeden Knoten im Binärbaum, dessen Unterbaum symmetrisch aufgebaut ist, reduziert sich diese Anzahl um den Faktor 2. Zum Beispiel ergeben sich für den einen Binärbaum mit drei Blättern $3! = 6$ Kombinationen. Da nun die untersten

zwei Blätter mit dem Knoten darüber einen symmetrischen Unterbaum bilden ist die Anzahl der gesuchten Baumstrukturen 6 / 2 = 3.

Anzahl *N* der **Blätter** des Binärbaums	Anzahl *M* der ungeordneten **Binärbäume** mit *N* Blättern	Anzahl der **Anordnungen** von *N* Blättern an die *M* Binärbäume
1	1	1
2	1 	1
3	1 	3
4	2 	15
5	3 	105
6	6	945
7	11	10.395

Tabelle 4.1: Anzahl der möglichen Binärbäume ohne Vertauschung der Äste und die möglichen Baumstrukturen, abhängig von der Anzahl der Blätter.

Auswahl-Funktion für Puma260

Eine Fortführung der Tabelle 4.1 läßt die Anzahl der möglichen Baumstrukturen ins Unermeßliche steigen. Aber in der Praxis werden diese theoretischen Werte nicht angenommen. Von allen möglichen Baumstrukturen wird nur ein kleiner Bruchteil durch das Gütekriterium ausgewählt, wie folgende Untersuchung zeigt:

Für einen Puma260, modelliert aus 10 einzelnen Objekten, wurde die Abbildung der Gelenkwinkelsätze auf die optimale Baumstrukturen untersucht. Dazu wurden alle Konfigurationen des Arms mit einer Abtastrate von 30° bzw. 15° in den ersten drei Gelenken erzeugt. Die restlichen Gelenke nahmen eine feste Stellung ein. Für jede Arm-Konfiguration wurde die Kompositions-Hierarchie aus Bounding-Boxes mit optimaler Baumstruktur bzgl. dem Gütekriterium berechnet. Als Gütekriterium für die Kompaktheit diente die Raumdiagonale der Bounding-Box-Kompositionen, also eine Art Durchmesser.

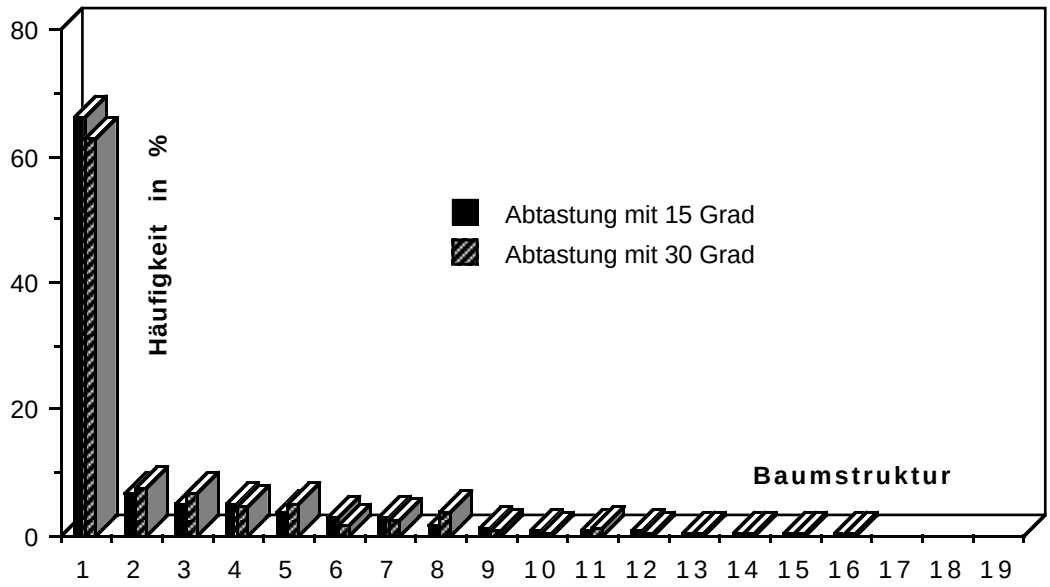


Abbildung 4.5: Histogramm über die optimalen Baumstrukturen des Puma260.

In Abbildung 4.5 sind die Ergebnisse der Untersuchung aufgeführt. Zu jeder der berechneten Baumstrukturen ist die Häufigkeit ihres Auftretens in fallender Reihenfolge angegeben. Bei den Werten wird zwischen den zwei Abtastraten unterschieden, um den Einfluß der Diskretisierung deutlich zu machen. Ausgegangen von fünf Armsegmenten kommen von den 105 möglichen Baumstrukturen aus Tabelle 4.1 nur 19 wirklich vor. Zusätzlich fällt auf, daß für zwei Drittel aller Konfigurationen nur *eine* Baumstruktur optimal ist. Die restlichen Baumstrukturen treten in extremen Armstellungen auf, wenn z. B. die Hand sehr nahe der Basis kommt und mit ihr zusammengefaßt wird, statt wie üblich mit dem Unterarm.

Für den Puma260 läßt dies den Schluß zu, daß die berechnete Baumstruktur im Wesentlichen von dem Gütekriterium und der Geometrie der Armsegmente und nicht wie erwartet von der Arm-Konfiguration abhängt. In der Praxis wird man für diesen Arm eine konstante Auswahl-Funktion, die zu jeder Konfiguration dieselbe Baumstruktur auswählt, anwenden können. Zwar ist für die Kollisionserkennung die am häufigsten auftretende Baumstruktur nicht in jeder Konfiguration optimal, aber für die schnelle Berechnung des Abstandsvektors ausreichend.

Zur Vollständigkeit sind in der Tabelle 4.2 die in der Untersuchung aufgetretenen Baumstrukturen in der Infix-Form angegeben.

Nummer der Baumstruktur	**Baumstruktur** in Infix-Notation	**Nummer** der Baumstruktur	**Baumstruktur** in Infix-Notation
1	(((O, S), B), (G, U))	11	(((G, S), O), (U, B))
2	((((U, S), O), B), G)	12	(((U, B), G), (O, S))
3	((G, B), ((U, S), O))	13	((G, O), ((U, S), B))
4	(((U, O), (G, S)), B)	14	(((O, S), G), (U, B))
5	(((G, S), B), (U, O))	15	(((U, O), B), (G, S))
6	((((O, S), U), B), G)	16	(((O, S), (U, B)), G)
7	(((U, S), O), (G, B))	17	((U, O), ((G, S), B))
8	(((S, B), O), (G, U))	18	((((G, S), O), U), B)
9	((G, B), ((O, S), U))	19	(((G, U), B), (O, S))
10	(((O, S), U), (G, B))		

Tabelle 4.2: Optimale Baumstrukturen für Puma260; B: Basis; S: Schulter; O: Oberarm; U: Unterarm; G: Greifer.

Allgemeine Auswahl-Funktion

Die obige Untersuchung zeigt, daß sich bei dem speziellen Roboterarm Puma260 die optimale hierarchische Darstellung nach einer Bewegung selten verändert. Die Baumstruktur einer solchen Kompositions-Hierarchie bleibt in vielen Armstellungen optimal. Die Funktion zur Auswahl der optimalen Baumstruktur kann konstant gewählt werden. Wie muß jedoch die Auswahl-Funktion für einen allgemeinen Roboterarm beschaffen sein?

Für einen allgemeinen Arm muß man annehmen, daß mehrere optimale Baumstrukturen verwendet werden müssen. Die Auswahl-Funktion ist dann nicht mehr konstant. Sie selektiert aufgrund des gegebenen Gelenkwinkel eine von mehreren Baumstrukturen. Diese Funktion kann auf verschiedene Weisen realisiert werden. Drei Beispiele werden dafür nun aufgezählt und bewertet:

- Neuberechnung: Die Neuberechnung ist die einfachste Auswahl-Funktion. Nach jeder Bewegung des Arms wird nach dem Nächste-Nachbar-Algorithmus aus Kapitel 4.2 eine optimale Baumstruktur erzeugt. Damit garantiert der Algorithmus zu jedem Zeitpunkt das Optimum. Leider benötigt er für jede einzelne Zusammenfassung viele Iterationen. Er verlangsamt daher die Verwendung der Kompositions-Hierachien mehr, als der Ansatz zu beschleunigen vermag.
- Hash-Funktion: Bei einer Hash-Funktion wird aus den Gelenkwinkeln der Index der optimalen Baumstruktur berechnet. D. h. es ist eine Formel gegeben, die als freie Parameter die Gelenkwinkel enthält und (ohne Iteration) schnell zu berechnen ist. Die Schwierigkeit liegt nun darin für einen Arm diese Formel allgemein festzulegen.

- Prüfung: Diese Auswahl-Funktion geht den umgekehrten Weg. Die zwei anderen Funktionen versuchen von einem Gelenkwinkelsatz auf die optimale Baumstruktur zu schließen. Stattdessen werden hier die off-line berechneten Baumstrukturen geprüft, ob sie für einen gegebenen Gelenkwinkelsatz optimal sind. Diese Vorgehensweise reduziert den Aufwand erheblich und scheint die einzige praktikable allgemeine Auswahl-Funktion zu sein. Sie soll im Folgenden genauer beschrieben werden.

Das Problem aller Auswahl-Funktionen liegt in dem Umfang der möglichen Gelenkwinkelsätze (Urmenge) und der möglichen optimalen Baumstrukturen (Bildmenge). Die Menge der Gelenkwinkelsätze kann nicht reduziert werden. Aber durch eine geeignete Wahl des Gütekriteriums wird die Anzahl der optimalen Baumstrukturen stark reduziert. Es ist daher viel schwerer eine Funktion für alle Gelenkwinkelsätze auf Baumstrukturen zu finden als für alle optimalen Baumstrukturen auf einen speziellen Gelenkwinkelsatz. (Es ist nur ein spezieller Gelenkwinkelsatz, da die Auswahl-Funktion immer nur für eine bestimmte Konfiguration aufgerufen wird.) Die Auswahl durch Prüfen sucht aus den off-line berechneten möglichen Baumstrukturen diejenige heraus, die für eine bestimmte Konfiguration optimal ist.

Das Prüfen erfolgt durch Vergleich der off-line berechneten Baumstrukturen mit Hilfe des Gütekriteriums. Dazu werden die Baumstrukturen bewertet. Der Wert einer Baumstruktur berechnet sich aus der Summe der Kompositons-Werte. Eine Komposition wird durch das Gütekriterium bewertet. Die Baumstruktur mit der kleinsten Summe ist für die aktuelle Komposition optimal.

Diese Auswahl durch Prüfen kann noch beschleunigt werden. Statt alle Baumstrukturen komplett zu bewerten, wird ebenenweise vorgegangen. Dazu beginnt man an der untersten Ebene der Kompositions-Hierarchien. Diese Ebene wird von der ersten Zusammenfassung zweier Objekte einer Baumstruktur gebildet. Für jede Baumstruktur wird das Gütekriterium der ersten Komposition berechnet. Nur eine von den Baumstrukturen mit dem kleinsten Wert kann die optimale sein. Die restlichen werden ausgeschieden. Bleiben mehrere Baumstrukturen übrig, dann wird die Bewertung auf der nächst höheren Ebene wiederholt. Ansonsten ist die Baumstruktur mit dem kleinsten Wert des Gütekriteriums die optimale für die aktuelle Konfiguration.

Als Beispiel werden die ersten drei Baumstrukturen aus der Tabelle 4.2 verwendet. Sie lauten in der Infix-Notation: (((O, S), B), (G, U)); ((((U, S), O), B), G) und ((G, B), ((U, S), O)), mit Basis, Schulter, Oberarm, Unterarm und Greifer. Nun soll für eine bestimmte Konfiguration des Roboterarms die günstigste unter den drei Baumstrukturen gefunden werden. Die Auswahl-Funktion fügt alle drei in eine Liste ein. Bei jeder Baumstruktur wird das Gütekriterium für die erste Komposition berechnet - also für (O, S), (U, S) und (U, S). Die letzten beiden Kompositionen fassen die gleichen Objekte zusammen, d. h. die Werte des Gütekriteriums sind identisch. In unserem Beispiel sei das Gütekriterium für die letzten beiden Kompositionen kleiner als für die erste. Somit wird die erste Komposition aus der Liste gelöscht. Nun wird der Vorgang für die nächste Hierarchie-Ebene mit ((U, S), O) und (U, S),

O) wiederholt. Auch hier hat das Gütekriterium den gleichen Wert. Erst auf der folgenden Ebene mit (((U, S), O), B) und (G, B) entscheidet sich, welche Baumstruktur optimal ist.

4.6 Zusammenfassung der Modellierung

Hier soll nun die hierarchische Modellierung von Objekten zusammengefaßt werden. Dies erfolgt am Beispiel eines Roboterarms. Ein Manipulator gehört zu den dynamischen Objekten. Da die Modellierung statischer Objekte nur ein Spezialfall der dynamischen ist, kann diese Modellierung direkt auch auf statische Objekte angewendet werden.

Die **dynamischen Hierarchien** sind eine Kombination aus Primitiven-Approximation und Kompositions-Hierarchien. Es wird versucht, die Vorteile und Ergebnisse aus jedem Ansatz in dieser Modellierung zu vereinen. Darüber hinaus erfolgt eine Anpassung der Ansätze an ein möglicherweise dynamisches Verhalten der Objekte. Die Modellierung geschieht in den folgenden drei Stufen:

Als Grundmodellierung der realen Objekte wird eine Darstellung aus konvexen Polyedern gewählt (siehe Abbildung 4.6 (a)). Die konvexen Polyeder lassen, vor allem wenn man dazu mehrere einzelne Polyeder verwendet, eine sehr genaue Repräsentation der Objekte zu. Für die Kollisionserkennung ist eine feine Darstellung besonders bei kleinen Freiräumen der Roboter wichtig.

Darauf aufbauend folgt eine Approximation durch Primitive (siehe Abbildung 4.6 (b)). Sie dient zur Reduzierung des Aufwands einer einzelnen Abstandsberechnung während der Kollisionserkennung. Aus den Erfahrungen der Primitiven-Approximation (siehe Kapitel 3.4 und 5.5), zeigt sich, daß die Bounding-Box als einzelnes Primitiv bei der Approximation am wirkungsvollsten ist.

In der letzten Stufe wird sukzessive die Anzahl der dargestellten Objekte reduziert. Dies geschieht durch eine Kompositions-Hierarchie. Dazu wird eine der off-line berechneten Baumstrukturen verwendet. Als Kompositionen werden Bounding-Boxes eingesetzt, um den dynamischen Veränderungen gerecht zu werden (siehe Abbildung 4.6 (b)-(e)). On-line erfolgt gegebenenfalls eine dynamische Anpassung der Baumstruktur und der Geometrie der Kompositionen.

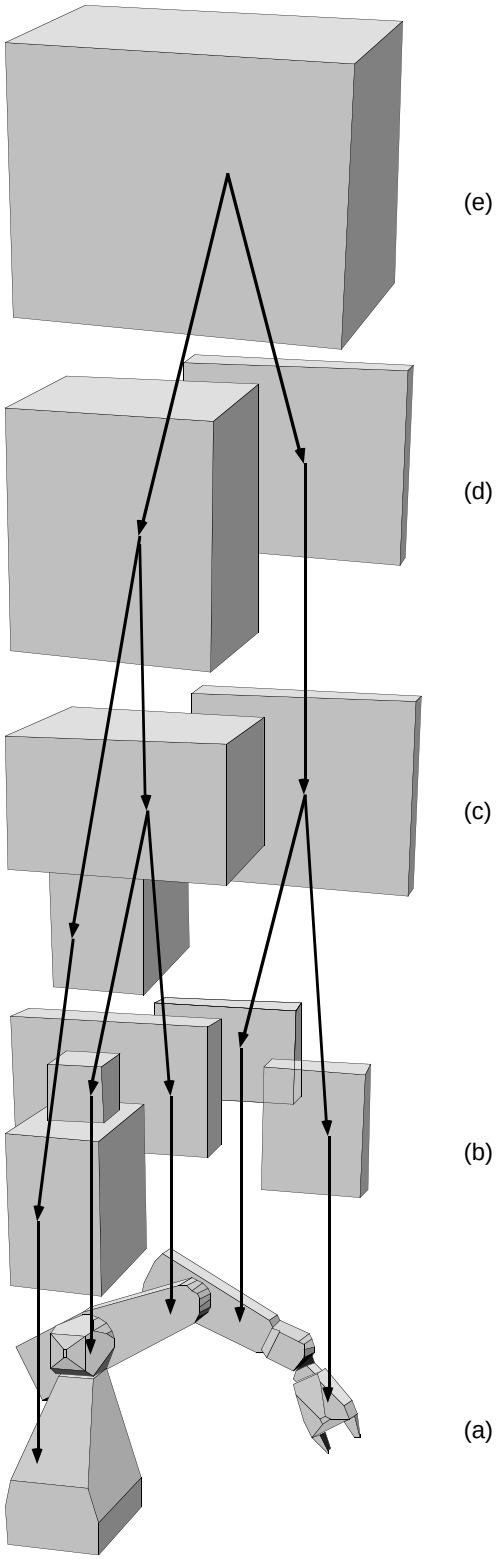


Abbildung 4.6: Modellierung eines Roboterarms als dynamische Hierarchie; (a): konvexe Polyeder; (b): Bounding-Box-Approximation; (b)-(e): Bounding-Box-Hierarchie.

On-line Aktualisierung dynamischer Hierarchien

Falls es sich bei der oben beschriebenen Modellierung um eine hierarchische Darstellung von dynamischen Objekten handelt, dann ist eine on-line Aktualisierung notwendig. Prinzipiell ist die Kollisionserkennung nur nach einer Bewegung erforderlich, denn ohne Bewegung verändern sich die Abstände zwischen den Objekten nicht. Die Modellierung muß also nach einer Bewegung und vor der nächsten Kollisionserkennung aktualisiert werden. Eine Bewegung wird dabei durch eine rechner-interne Transformation der konvexen Polyeder dargestellt. Die Aktualisierung der dynamischen Hierarchien besteht aus drei Schritten:

(1) Für jeden konvexen Polyeder wird die kleinste, konservative Bounding-Box-Approximation berechnet. Dies geschieht mit dem im Kapitel 3.2 beschriebenen Verfahren.

(2) Zu einer gegebenen Konfiguration der Objekte (Gelenkwinkelsatz, falls es sich um einen Manipulator handelt) wird mit Hilfe der Auswahl-Funktion aus Kapitel 4.5 die optimale Baumstruktur herausgesucht. (Bei den verwendeten Roboterarmen Puma260 ist die Auswahl-Funktion konstant.)

(3) An jedem Knoten dieser Baumstruktur wird, bei der untersten Ebene beginnend, nach dem Verfahren aus Kapitel 4.4 die Bounding-Box-Komposition berechnet. Dazu werden nur die zusammengefaßten Bounding-Boxes aus der nächst-tieferen Ebene benötigt. Das Verfahren kann rekursiv von der obersten Ebene absteigen und sukzessiv von unten nach oben die Kompositionen berechnen.

4.7 Abstandsberechnung mit Hierarchien

Ein Roboterarm oder eine Reihe von Hindernissen liegen in einer hierarchischen Darstellung vor. Mit Hilfe der Hierarchien soll nun effizient der Abstandsvektor ermittelt werden. Für die Berechnung ist es unwesentlich, ob es sich um statische Hierarchien (bei unbewegten Hindernisse) oder um dynamische Hierarchien (bei Roboterarmen) handelt. Der Algorithmus bleibt auch bei gemischten Hierarchien gleich. Es ist aber entscheidend, ob es sich bei dem resultierenden Abstandsvektor um eine Abschätzung oder um den exakten Vektor handeln soll. Für die Kollisionserkennung ist eine Abschätzung ausreichend, solange sie größer einem vorgegebenen Sicherheitsabstand ist.

Bei der Berechnung eines angenäherten Abstandsvektors für die Kollisionserkennung wird mit den beiden obersten Hierarchie-Ebenen, die nur eine Komposition enthalten, begonnen. Der rekursive Algorithmus versucht zu zeigen, daß der Abstand zwischen zwei Kompositionen größer dem Sicherheitsabstand ist. Hierbei besteht ein Paar aus Kompositionen von jeweils einer Hierarchie. Falls der Abstand eines Paares kleiner dem vorgegebenen Sicherheitsabstand ist, dann werden die Partner nach einer festen Strategie durch ihre genaueren Darstellungen auf der darunterliegenden Ebene ersetzt. Zwischen diesen neuen Kompositionen werden Paare

gebildet und die Abstandsberechnung fortgesetzt. Der rekursive Algorithmus terminiert, wenn entweder der Abstand aller betrachteten Paare größer dem Sicherheitsabstand ist oder ein Paar auf der untersten (genauesten) Darstellungsebene beider Klassen kollidiert.

Bei der Auswahl, welcher Partner eines Paares mit zu geringem Abstand ersetzt werden soll, können unterschiedliche Strategien verwendet werden, z. B.:

- Ersetze nur *einen* Partner, und zwar den mit dem größeren Approximationsfehler.
- Ersetze *beide* Partner soweit möglich durch ihre genauere Darstellung.

Hier wurde letztere Strategie verwendet, weil sie keine zusätzlichen Berechnungen erfordert und am schnellsten in der Hierarchie absteigt.

Das Resultat des oben beschriebenen Abstiegs-Algorithmus ist eine untere Abschätzung des Abstandsvektors. Da die Kompositionen in den Hierarchien konservativ sind, ist der reale Vektor ist auf jeden Fall länger als die berechnete Approximation. In anderen Anwendungen neben der Kollisionserkennung kann es wünschenswert sein den exakten Abstandsvektor zu erhalten. Dies ist mit den Kompositions-Hierarchien ebenso möglich.

Für die exakte Abstandsberechnung zwischen hierarchisch dargestellten Objekten kann der obige Abstiegs-Mechanismus modifiziert werden. Er ist dann der A*-Suche ähnlich. Dazu wird der Sicherheitsabstand auf den Wert "unendlich" gesetzt. D. h. es wird auf jeden Fall bis zur untersten Ebene (den Polyedern) abgestiegen. Durch die genaueste Darstellung ergibt sich auch der exakte Abstand. Aber auf Grund dieses Sicherheitsabstands werden auch immer alle Paare ersetzt. Deshalb wird der Algorithmus so modifiziert, daß nur sukzessive das Paar ersetzt wird, welches den kleinsten Abstandsvektor aller aktuellen Paare hat. Verglichen mit der A*-Suche ist der angenäherte Abstandsvektor zwischen den Kompositionen die untere Abschätzung des noch zu bewältigenden Wegs. Der modifizierte Algorithmus terminiert, wenn ein Paar auf der untersten Ebene (mit Polyedern) den kleinsten Abstand aller aktuellen Paare hat.

5 Experimentelle Ergebnisse

In den vorigen Kapiteln wurde eine Reihe von Ansätzen zur Beschleunigung der Kollisionserkennung beschrieben. Nun wäre es wünschenswert, eine objektive Bewertung der einzelnen Ansätze zu haben, um den jeweils günstigsten in einem Gesamtsystem anwenden zu können. Neben dem Grad der Beschleunigung sind noch andere Faktoren wie z. B. die Qualität des Abstandsvektors wichtig. All diese Werte und Einflußgrößen können analytisch kaum oder gar nicht bestimmt werden. Daher wurden die Ansätze implementiert und in Experimenten verglichen. Die Experimente und ihre Ergebnisse werden in diesem Kapitel dargestellt.

Zuerst werden die Meßbedingungen und die Beispiele, mit denen die Experimente durchgeführt wurden, beschrieben. Das wichtigste Kriterium bei der Bewertung ist die Laufzeit der Ansätze. Sie wird in dem zweiten Unterkapitel angegeben. Dann folgt eine Diskussion der Genauigkeit des Abstandsvektors. Neben diesen zwei Faktoren werden noch andere Einflußgrößen bei der Berechnung des Abstandsvektors betrachtet. Zum Schluß werden die Ergebnisse der Experimente zusammengefaßt.

5.1 Meßumgebung

Die verschiedenen Ansätze zur on-line Kollisionserkennung wurden implementiert und in ein bestehendes Simulationssystem eingebaut. Das System ist in der Lage, zu gegebener Anfangs- und Ziel-Konfiguration einer Szene eine Bahn zu planen und die Ausführung dieser Bahn darzustellen. Dazu werden die Armstellungen an diskreten Stellen der Bewegungsbahn auf dem Bildschirm gezeichnet. Die Szene kann ein oder zwei Arme und weitere Hindernisse enthalten. Als Grundmodellierung werden alle Objekte durch Polyeder dargestellt und die konvexen Hüllen vor der Simulation berechnet. Mit diesen konvexen Polyedern wird die Planung durchgeführt. Das Simulationssystem ist in der Programmiersprache C implementiert und läuft auf einer SPARC-2 Workstation unter UNIX. Siehe dazu [Hanke91]. Der ausführbare Programmcode wurde für die Messungen mit der höchsten Optimierungsstufe des C-Kompilers (-O4) übersetzt.

Zum Vergleich der Ansätze wurden in dem Simulationssystem zwei typische Beispiel-Szenen generiert. Die erste Szene enthält einen Puma260, modelliert aus 10 Polyedern mit

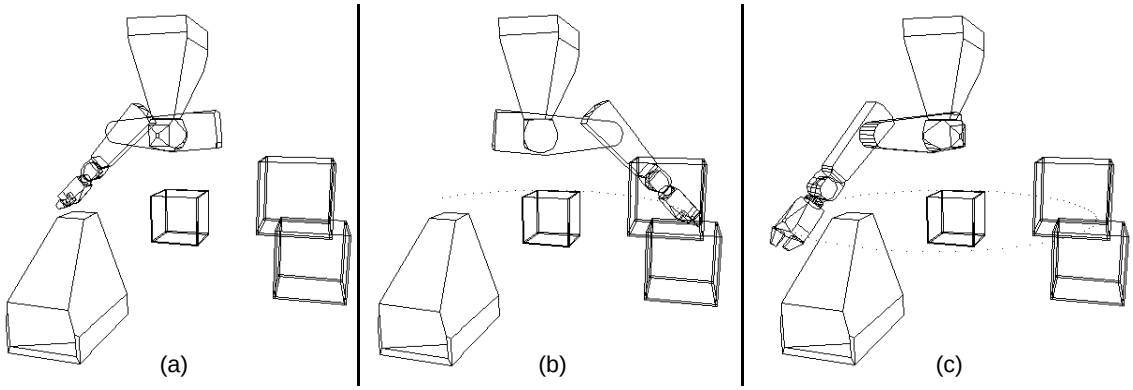


Abbildung 5.1: Beispiel-Szene mit Arm und Hindernissen, dargestellt durch konvexe Polyeder; (a): Anfangsposition; (b): Zwischenposition; (c): Endposition.

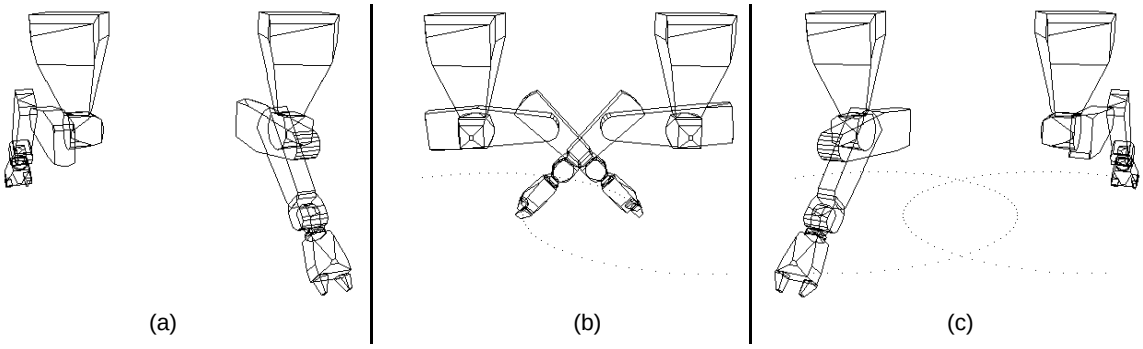


Abbildung 5.2: Beispiel-Szene mit zwei Armen, dargestellt durch konvexe Polyeder; (a): Anfangspositionen; (b): Zwischenpositionen; (c): Endpositionen.

durchschnittlich 30 Eckpunkten, und 16 weitere Hindernisse (siehe Abbildung 5.1). Anhand dieser Szene wurden die möglichen Kollisionen zwischen Arm und Hindernissen untersucht. Sie bilden je eine Kollisionsklasse aus dem Kapitel 2. Die zweite Szene enthält zwei Puma260-Arme, eingeteilt in je eine Kollisionsklasse (siehe Abbildung 5.2). An dieser Szene wurde das Verhalten der Ansätze bei möglicher Kollision von zwei bewegten Objekten untersucht.

Zu jeder Szene mit gegebener Anfangs- und Ziel-Konfiguration erzeugte das Simulationssystem eine diskretisierte Bahn. Vor der Ausführung dieser Bahn wurde der Prozeß der Kollisionserkennung initialisiert und die off-line generierten Primitiven-Approximationen eingelesen. Dann stellte das System die Bahn in einzelnen Bewegungsschritten dar. Zwischen jedem Bewegungsschritt wurde der Prozeß der Kollisionserkennung aufgerufen. Dieser berechnete zwischen den Kollisionsklassen der Szene die Abstandsvektoren.

Bei der Planung der Bahn blieben die möglichen Kollisionen zunächst unberücksichtigt. So wurde für jeden Arm der direkte Weg von der Startposition zu seiner Zielposition generiert. Die auftretenden Kollisionen wurden nun dazu genutzt, das Verhalten der on-line

Kollisionserkennung in verschiedenen Phasen zu untersuchen. Es lassen sich hierbei drei Phasen unterscheiden:

- Der Roboterarm bewegt sich weit entfernt von den Hindernissen.
- Der Roboterarm ist in unmittelbarer Nähe der Hindernissen.
- Der Roboterarm kollidiert mit einem oder mehreren Hindernissen.

Diese Phasen helfen die Stärken und Schwächen der verschiedenen Ansätze zur Kollisionserkennung zu ermitteln. Je nachdem wie die Kollisionsvermeidung in dem Gesamtsystem vorgeht, also welche der obigen Situationen während der Planung häufiger auftritt, kann der eine oder andere Ansatz in das Gesamtsystem eingebaut werden.

5.2 Laufzeiten der Abstandsberechnung

In dieser Arbeit sind mehrere Ansätze zur Beschleunigung der Abstandsberechnung bei der Kollisionserkennung beschrieben (Kapitel 2 bis 4). Es ist nicht sinnvoll, nur einen dieser Ansätze in seiner Reinform anzuwenden. Erst durch eine Kombination der Ansätze können die verschiedenen Arten der Zeitgewinnung ausgenutzt werden. Die Ansätze werden prinzipiell unterschieden in die "Reduzierung des Aufwands einer Berechnung" und die "Reduzierung der Anzahl von Berechnungen". So sollte zumindest ein Ansatz aus jeweils einer dieser zwei Gruppen verwendet werden. Auf der anderen Seite läßt eine Untersuchung von kombinierten Ansätzen kaum eine Aussage über die Wirkung eines einzelnen Ansatzes zu. Um diesem Zwiespalt zu entkommen wurden zwar Kombinationen untersucht, aber die einzelnen Ansätze sukzessive ausgetauscht. Durch den Vergleich von Kombinationen, die sich nur um einen Ansatz unterscheiden, kann man zu einer relativen Aussage über einen Ansatz kommen. Im Folgenden werden zunächst die Kombinationen der Ansätze vorgestellt und danach die Laufzeiten der Kombinationen angegeben.

Kombination der Ansätze

In der Tabelle 5.1 ist eine Liste der untersuchten Kombinationen aufgeführt. Sie kann in vier Gruppen gegliedert werden:

- Darstellung nur durch Polyeder (a)-(d),
- zusätzliche Primitiven-Approximation (e)-(h),
- Abstands-Fortschreibung (i)-(j) und
- hierarchischen Darstellung (k)-(l).

Im Folgenden wird jede Kombination der Ansätze beschrieben. Die eigentliche Beschreibung der Ansätze findet sich in den angegebenen Kapiteln:

Kombination (a): Als Grundlage aller Ansätze dient die Abstandsberechnung zwischen konvexen Polyedern (Kapitel 3.3, [Gilbert88]). Daher ist sie in der Tabelle 5.1 nicht extra

aufgeführt. Diese Abstandsberechnung wird zwischen allen Objektpaaren mit mindestens einem bewegten Objekt durchgeführt. Ein Objektpaar enthält also auf jeden Fall ein Armsegment, denn nur diese können sich in der Simulation bewegen und mit anderen Objekten kollidieren (Kapitel 2.1).

Kombination (b): Zusätzlich zur Abstandsberechnung mit Polyedern (a), wird sobald ein Objektpaar mit dem Abstand Null vorliegt, die Kollisionserkennung für diesen Bewegungsschritt abgebrochen (Kapitel 2.3). Dies ist ein relativ selbstverständlicher Ansatz. Dennoch ist der Zeitpunkt eines Abbruchs nicht vorhersehbar. Die Wirkungsweise anderer Ansätze wird durch den Abbruch verwischt. Daher wird der Abbruch hier einzeln untersucht und ansonsten in den anderen Kombinationen weggelassen.

Kombination (c): Statt wie in (a) die Objektpaare zu generieren, werden hier alle Objektpaare explizit in einer Datenstruktur abgelegt. Für jedes Objektpaar wird der Abstand zwischen den konvexen Polyedern berechnet. Für sich erbringt die Datenstruktur keine Verbesserung, aber es kann der Verwaltungsaufwand gemessen werden (Kapitel 2.3 und Anhang A).

Kombination (d): Die Verwendung der Kollisionsdaten (c) ermöglicht zu jedem Paar den Abstandsvektor aus dem letzten Bewegungsschritt zu speichern. Er dient hier als Startwert der iterativen Abstandsberechnung zwischen Polyedern (Kapitel 2.3, [Gilbert88]).

Soweit die Kombinationen der Ansätze, die nur mit Polyedern arbeiten. Nun folgen die Kombinationen der Primitiven-Approximation (nach z. B. [Adolphs]) mit unterschiedlichen Abstiegs-Strategien. Dabei wird die günstigste Kombination der vorigen Untersuchung beibehalten. Dies ist die Verwendung der Startwerte mit der Kollisionsdaten-Verwaltung.

Kombination (e): Neben den Startwerten und den Kollisionsdaten (d) wird nun die Primitiven-Approximation eingesetzt. Als Strategie dient der gleichmäßige Abstieg zwischen den unterschiedlichen Primitiven bei jedem Objektpaar. Nacheinander werden also der Abstand zwischen Bounding-Box, Kugel, Zylinder, Quader und konvexen Polyeder berechnet, bis der Abstand größer dem Sicherheitsabstand ist (Kapitel 3.5).

Kombination (f): Um bei der gleichmäßigen Abstiegs-Strategie (e) den schnellen Berechnungen den Vortritt zu lassen, sind die Primitiven-Paare nach der Laufzeit der Abstandsberechnungen sortiert. Dabei sind auch gemischte Primitiven-Paare zugelassen (Kapitel 3.5).

Kombination (g): Bei der Laufzeitreihenfolge (f) ist nicht garantiert, daß sich der Approximations-Fehler der Primitiven verringert. Neben der Laufzeitreihenfolge sind also hier die Primitiven-Paare zusätzlich nach abnehmenden Approximations-Fehler sortiert (Kapitel 3.5).

Kombination (h): Als einfachste Primitiven-Approximation wird nur die Bounding-Box als schneller Vortest verwendet. Ist der Abstand mit der Bounding-Box kleiner dem Sicherheitsabstand, dann wird mit den konvexen Polyedern weiter gerechnet. Es handelt sich um eine noch einfachere Strategie als der gleichmäßige Abstieg (e) (Kapitel 3.5).

Reduzierung von:	Beschleunigungs-Ansätze:	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)	(k)	(l)
Aufwand einer Berechnung	Startwerte				•	•	•	•	•	•	•		
	gleichmäßiger Primitiven-Abstieg					•							
	in Laufzeitreihenfolge						•	•					
	ohne schlechte Approximation.							•					
	nur Bounding-Box								•		•	•	•
Anzahl der Berechnungen	Abbruch		•										
	dynamische Hierarchien												•
	statische Hierarchien											•	
	globale Abstands-Fortschreibung									•	•		
	Kollisionsdaten-Verwaltung			•	•	•	•	•	•	•	•		

Tabelle 5.1: Kombinationen der Ansätze für die Laufzeit-Messungen.

Insgesamt stellen die Kombinationen mit Primitiven-Approximationen (e)-(h) und den Startwerten (d) die verschiedenen Möglichkeiten zur Reduzierung des Aufwands einer einzelnen Berechnung dar. Im Gegensatz dazu wird in den nächsten Kombinationen die Anzahl der Berechnungen reduziert:

Kombination (i): Hier wird global der kürzeste Abstand um die größte zurückgelegte Distanz in einem Bewegungsschritt fortgeschrieben. So ist erst nach einigen Schritten eine neue Abstandsberechnung nötig (Kapitel 2.3, [Faverjon89]). Dies erfolgt nur mit den Startwerten und den Kollisionsdaten, aber ohne die Primitiven-Approximation. Die größte zurückgelegte Distanz wird als Abschätzung relativ zur Geschwindigkeit berechnet.

Kombination (j): Zusätzlich zur globalen Abstands-Fortschreibung (i) wird die beste Abstiegs-Strategie (h) der Primitiven-Approximation eingesetzt.

Ein prinzipiell anderer Ansatz ist die hierarchische Modellierung in den folgenden Kombinationen. Auch sie dient zur Reduzierung der Anzahl von Abstandsberechnungen:

Kombination (k): Alle unbewegten Hindernisse sind durch die statischen Kompositions-Hierarchien aus Bounding-Boxes modelliert. Dabei werden die Hindernisse sukzessive zusammengefaßt und durch Bounding-Boxes approximiert. Es entsteht eine Hierarchie, die die Hindernisse auf unterschiedlichen Genauigkeitsstufen darstellt (Kapitel 2.3 und 4.1, [Faverjon89]). Da auf der untersten Hierarchieebene die konvexen Polyeder durch Bounding-Boxes approximiert sind, kommt auch die einfachste Primitiven-Approximation (h) ins Spiel.

Kombination (l): Neben den statischen Hierarchien (k) werden dynamische Hierarchien für die Roboterarme eingesetzt. Sie sind Kompositions-Hierarchien mit Bounding-Boxes. Nach jedem Bewegungsschritt wird eine für die aktuelle Konfiguration optimale Baumstruktur ausgewählt. Diese ist für den Puma260 konstant. Dann werden die Bounding-Boxes an den Hierarchie-Knoten der aktuellen Geometrie angepaßt. Damit erhält man zu jeder Armstellung die optimale hierarchische Darstellung des Roboterarms (Kapitel 4.2 bis 4.7). Es wird automatisch die einfachste Primitiven-Approximation (h) angewendet.

Laufzeiten der Kombinationen

Jede der in Tabelle 5.1 aufgelisteten Kombinationen der Beschleunigungsansätze wurde in den zwei Beispiel-Szenen angewendet. Dabei wurde die Laufzeit der Abstandsvektor-Berechnung für jeden Bewegungsschritt gemessen und in Abbildung 5.3 dargestellt. Bei den schnelleren Kombinationen tritt das Problem der Meßungenauigkeit auf. Dies wurde, analog zu den Messungen bei der Primitiven-Approximation, durch wiederholtes Ausführen der Berechnung innerhalb einer Zeitmessung gelöst. Bei den Kombinationen (a)-(j) wurde eine Reduzierung des Meßfehlers auf ±0,1666 ms und bei den Kombinationen (k)-(l) auf ±0,0166 ms erzielt. Dieser Meßfehler bezieht sich auf die Laufzeit der Abstandsberechnung zu einem Bewegungsschritt. Zusätzlich wurden diese Zeiten über die gesamte Bewegungsbahn der Manipulatoren gemittelt. Damit sind die Ergebnisse relativ unabhängig von der jeweiligen Armstellung und Umgebung. Um das Verhalten der Ansätze vor, während und nach einer Kollision beurteilen zu können, wurden die Zeiten für Fälle mit und ohne auftretender Kollision getrennt gemessen.

Bei der Implementierung wurde angenommen, daß das geometrische Weltmodell unabhängig von der Kollisionserkennung im Gesamtsystem existiert. Nach jedem Bewegungsschritt das muß diese Weltmodell aktualisiert werden um mit der Realität konsistent zu bleiben. Dazu wird jeder Polyeder-Eckpunkt eines bewegten Objekts in seine neue Lage transformiert. Dies geschieht mit einer einfachen Matrix-Multiplikation. Da die Abstandsberechnung in dem selben Koordinatensystem stattfindet, sind alle Objekte im Weltkoordinatensystem definiert. Zusätzlich müssen nach jeder Polyeder-Transformation bei gegebener Primitiven-Approximation auch die zugehörigen Primitiven transformiert werden. Für den Puma260 mit zehn Armsegmenten beträgt die Transformationszeit der konvexen Polyeder mit Berechnung der Bounding-Boxes konstant ca. 25 ms. Werden die restlichen Primitiven Kugel, Zylinder und Quader auch transformiert, dann ergibt sich eine Zeit von ca. 27 ms.

Diese Transformationen werden vom Gesamtsystem vorgenommen. Daher wurde die dazu benötigte Zeit nicht mitgemessen. Aber die Aktualisierung des Weltmodells ist nicht für alle Beschleunigungs-Ansätze nach jeder Bewegung notwendig. Zum Beispiel greift die Abstands-Fortschreibung nur bei einer Neuberechnung des Abstandsvektors auf das Weltmodell zu. (Vorausgesetzt der Betrag um den fortgeschrieben wird hängt nicht von Weltmodell ab.) In den anderen Fällen kann das Weltmodell in einem unveränderten (inkonsistenten) Zustand bleiben.

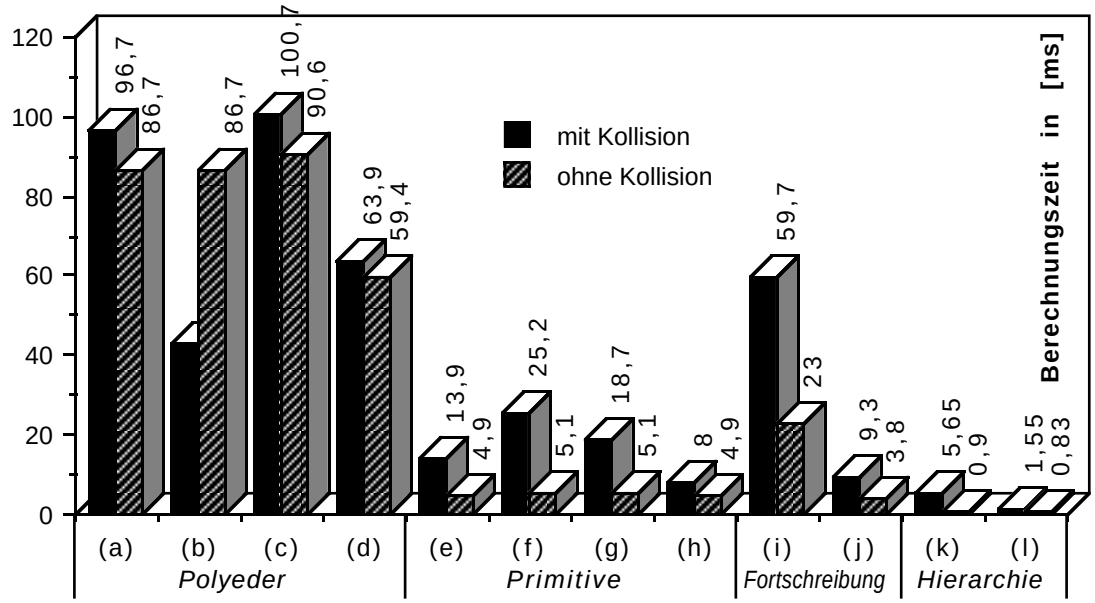


Abbildung 5.3: Mittlere Laufzeiten der Abstandsvektor-Berechnung bei Kombinationen der Beschleunigungsansätze.

D. h. je nach Beschleunigungs-Ansatz und Art des Gesamtsystems muß die Transformationszeit mitberücksichtigt werden.

Die Ergebnisse aus Abbildung 5.3 lassen eine klare Bewertung der einzelnen Beschleunigungs-Ansätze zu: Der Abbruch bei Abstand Null (b) bewirkt gegenüber der einfachen Kollisionserkennung (a) eine Beschleunigung von ca. 50 % im Falle einer Kollision. Treten keine Kollisionen in der Szene auf, dann kann dieser Ansatz auch keine Verbesserung einbringen. Der Aufwand für eine Kollisionsdaten-Verwaltung (c) beträgt für die Beispiel-Szenen nur ca. 4 ms, sodaß sich die Verwendung von Startwerten (d) bei einer Beschleunigung um ca. 30 % dennoch lohnt. Bei den Strategien der Primitiven-Approximation (e)-(h) zeigt sich, daß sie nochmals eine Beschleunigung um ca. 60 % erbringen. Speziell die Bounding-Box allein erweist sich als die günstigste Approximation. Insgesamt kann mit den Ansätzen zur Beschleunigung einer einzelnen Abstandsberechnung eine Reduzierung der Laufzeit um über 90 % erreicht werden.

Nun zu den Ansätzen zur Reduzierung der Objektanzahl: Die globale Abstands-Fortschreibung (i) bildet quasi das Komplement zu dem Abbruch (b): sie beschleunigt vor allem die Bewegungsschritte ohne Kollision. Verglichen mit den Startwerten (d) bewirkt sie eine Verbesserung um ca. 50 %. Im direkten Vergleich mit der Primitiven-Approximation (h) kann die zusätzliche Abstands-Fortschreibung (j) keine signifikante Verbesserung erbringen! Eine deutliche Beschleunigung wird durch die hierarchische Darstellung (k),(l) erzielt. Sowohl in Fällen mit als auch in Fällen ohne Kollision erzielen die hierarchischen Ansätze die besten Ergebnisse. Trotz der notwendigen Aktualisierung der Baumstruktur und Kompositionen lohnt

sich der Einsatz der dynamischen Hierarchien. Sie erbringen im Vergleich mit der einfachen Kollisionserkennung (a) eine Beschleunigung um den Faktor von ca. 100.

Zum Vergleich der Ansätze wurde bewußt eine "minimale" Szene verwendet: Die minimale Szene beinhaltet möglichst wenige Objekte. Dies sind bei den dynamischen Hierarchien nur die zwei Roboterarme. Im Vergleich mit den nicht-hierarchischen Ansätzen ist dies der ungünstigste Fall, da der Vorteil der hierarchischen Ansätze, aus der Sicht der Kollisionserkennung viele Objekte auf wenige zu reduzieren, nicht ausgenutzt werden kann. Im durchschnittlichen Fall wachsen die Laufzeiten der hierarchischen Ansätze nur logarithmisch, die Laufzeiten der nicht-hierarchischen Ansätze quadratisch mit der Objektanzahl. Durch die minimale Szene wird also die Beschleunigung der hierarchischen Darstellung nach unten abgeschätzt. Die üblichen Szenen beinhalten weit mehr Objekte als die minimale Szene. Die Ansätze mit hierarchischer Darstellung werden in den üblichen Szenen eine noch größere relative Beschleunigung erzielen.

Nun folgt eine genauere Betrachtung der Abstiegs-Strategien bei der Primitiven-Approximation (e)-(h). Schon bei den Ergebnissen aus der Abbildung 5.3 fällt eine Besonderheit auf: die Strategien unterscheiden sich vor allem in den Zeiten mit auftretender Kollision. In den Zeiten ohne Kollisionen sind sie erstaunlich ähnlich. Dieses Phänomen soll in der Abbildung 5.4 genauer analysiert werden. Dort sind für die einzelnen Bewegungsschritte die Laufzeiten der Strategien aufgetragen. Die horizontale Achse kann man auch mit einer diskretisierten Zeitachse gleichsetzten. (Es handelt sich also nicht um gemittelte Zeiten.) Von der Bewegungsbahn ist ein Stück mit auftretender Kollision ausgewählt. An diesem Stück sind drei unterschiedliche Phasen zu erkennen. Die erste Phase liegt in dem Zeitraum weit vor und weit nach der Kollision (bis zum 6. und ab dem 20. Schritt). Die Laufzeiten der Abstiegs-Strategien sind dort fast identisch. Die zweite Phase liegt um den Zeitpunkt, an dem eine Kollision beginnt oder endet (6. bis 11. und 19. bis 20. Schritt). Die Laufzeiten steigen oder fallen alle mehr oder weniger stark. Die letzte Phase liegt im Zeitraum, in dem eine Kollision stattfindet, die kollidierenden Objekte sich also voll durchdringen (11. bis 19. Schritt). In dieser Phase bleiben die Laufzeiten etwa auf einer Höhe. Insgesamt sind die drei Phasen symmetrisch zu dem Bereich der Kollision.

Was passiert nun während einem solchen Stück der Bewegungsbahn? Die unterschiedlichen Abstiegs-Strategien verhalten sich im Prinzip gleich. Nur der Anstieg in der zweiten und das Niveau in der dritten Phase sind unterschiedlich. In der ersten Phase treten die niedrigsten Laufzeiten auf. Die Strategien können alle nach dem ersten Primitiven-Paar (zwei Bounding-Boxes) den Abstieg abbrechen, da der Abstand größer dem Schwellwert ist. In der zweiten Phase werden immer mehr Objekte in die Kollision verwickelt. Die Strategien müssen für immer mehr Objekte bis zu der genauesten Darstellung (konvexe Polyeder) absteigen. In der dritten Phase bleibt die Anzahl der kollidierenden Objekte gleich. Die Strategien unterscheiden sich vor allem in der Anzahl der Abstandsberechnungen. Bei vier Primitiven-Approximationen pro Objekt sind es für den gleichmäßigen Abstieg vier Stück bis zur genauesten Darstellung.

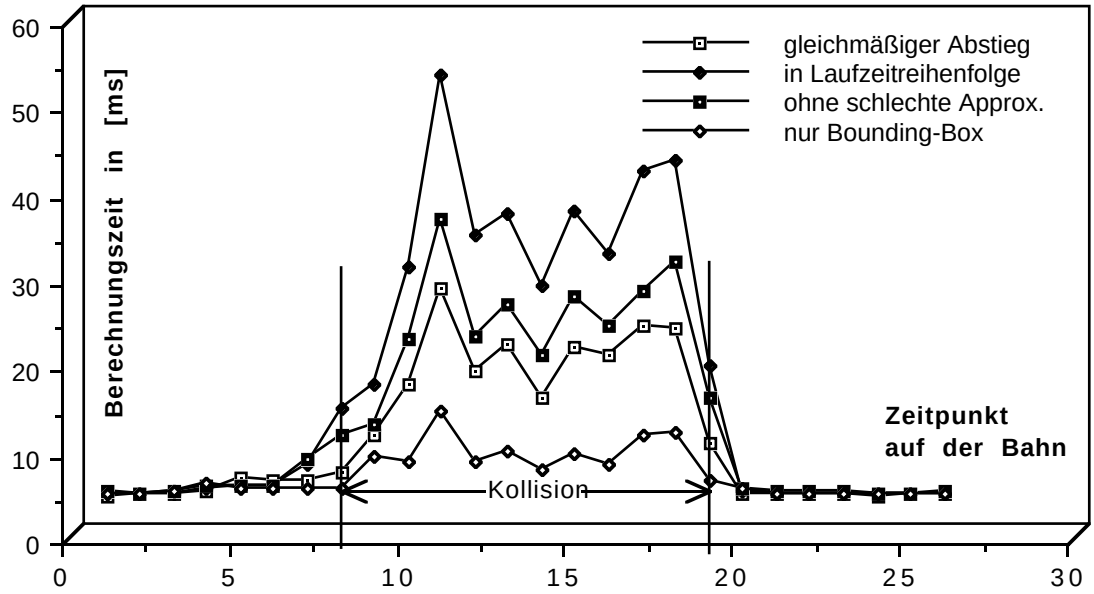


Abbildung 5.4: Berechnungszeiten des Abstandsvektors bei unterschiedlichen Abstiegs-Strategien mit der Primitiven-Approximation bei auftretender Kollision.

Dagegen werden in der Laufzeitreihenfolge 13 Berechnungen durchgeführt. Wird nur die Bounding-Box verwendet, so sind es lediglich zwei Berechnungen. Insgesamt überschneidet sich keine Strategie in der Laufzeit mit einer anderen. Die zeitliche Rangfolge bleibt in jeder Phase gewahrt. Hier zeigt sich deutlich, daß sich in keiner Phase der Kollisionserkennung der Einsatz von anderen Primitiven als der Bounding-Box lohnt.

5.3 Qualität des Abstandsvektors

Bei den meisten Beschleunigungs-Ansätzen in der Kollisionserkennung wird das gleiche Grundprinzip zur Beschleunigung angewendet. Statt dem realen Abstandsvektor wird eine Approximation berechnet und so die Zeitersparnis erreicht. Natürlich geht das auf Kosten der Genauigkeit des Abstandsvektors. Im Wesentlichen gilt: je wirkungsvoller ein Beschleunigungs-Ansatz, desto ungenauer der Abstandsvektor. Die Ungenauigkeit des Abstandsvektors betrifft Abweichungen in der Richtung und in der Länge des Vektors. Hier wird nur die Längengenauigkeit betrachtet, da sie für die Kollisionserkennung zunächst wichtiger ist. Zuerst wird nun die Qualität des Abstandsvektors bei den Strategien der Primitiven-Approximation diskutiert. Dann folgt ein Vergleich mit den hierarchischen Darstellungen.

In der Abbildung 5.5 ist das gleiche Stück der Beispiel-Bewegung ausgewählt wie in Abbildung 5.4. Statt den Laufzeiten sind hier die Längen der Abstandsvektoren bei unterschiedlichen Abstiegs-Strategien der Primitiven-Approximation aufgetragen. Zusätzlich ist noch der "wahre" Abstand hinzugefügt. Er wurde mit Hilfe der konvexen Polyeder ermittelt. Bei

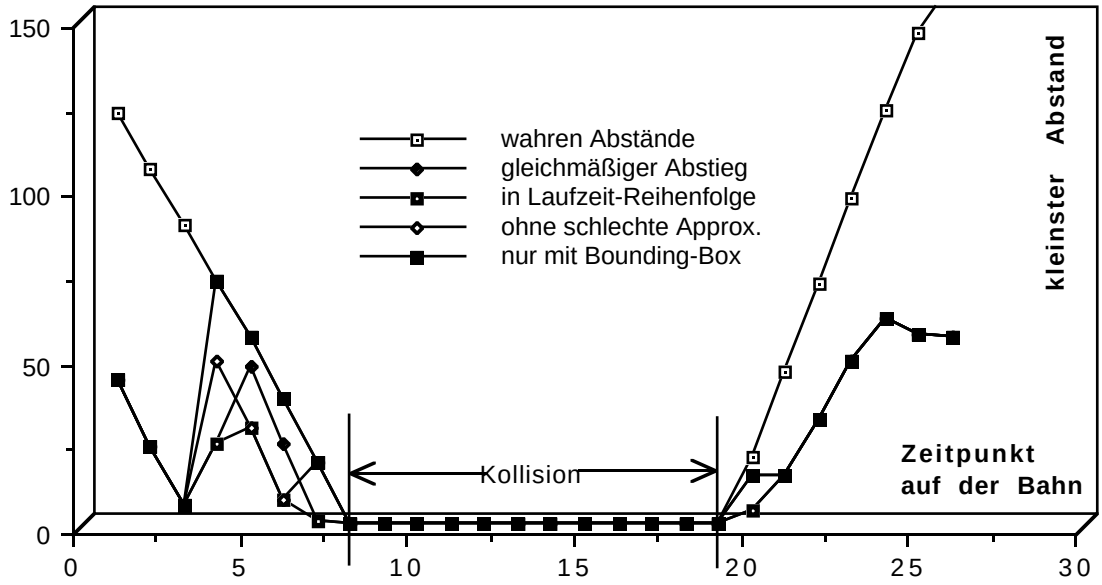


Abbildung 5.5: Längengenauigkeit des Abstandsvektors bei der Primitiven-Approximation um eine auftretende Kollision

dem Verlauf der Abstände kann man eine ähnliche Einteilung in drei Phasen feststellen wie in der Abbildung 5.4. Die erste Phase beinhaltet den Zeitraum weit vor und weit nach einer Kollision (bis zum 3. und ab dem 21. Schritt). Die Abstände der Abstiegs-Strategien differieren alle gleich von dem wahren Abstand. Die zweite Phase spielt sich kurz vor und kurz nach einer Kollision ab (3. - 8. und 19. - 21. Schritt). Die meisten Strategien ergeben nun unterschiedliche Abstände. Nur bei der reinen Bounding-Box-Strategie stimmen die Abstände mit den wahren Abständen überein. In der dritten Phase (8. - 19. Schritt) findet die Kollision statt und die Abstände aller Strategien müssen sich mit dem wahren Abstand Null überdecken.

Die Qualität des Abstandsvektors ist in der ersten und dritten Phase für alle Strategien gleich. In der ersten Phase weichen jedoch die Abstände von dem wahren Abstand ab und in der dritten Phase stimmen sie mit dem wahren Abstand überein. In der ersten Phase ist für jede Strategie der Kollisionserkennung wieder die Bounding-Box voll ausreichend. Der Abstand berechnet mit Bounding-Boxes ist hier immer größer als der Schwellwert, sodaß keine anderen Primitiven zum Zug kommen. Das ist daran zu erkennen, daß alle Abstände mit dem der "nur Bounding-Box"-Strategie übereinstimmen. In der dritten Phase sind alle Abstände gleich dem wahren Abstand Null. Ansonsten wäre eine Strategie inkorrekt, denn die Approximation des Abstandsvektors muß immer kürzer sein als der reale Abstand. Nur in der zweiten Phase weichen die Abstände voneinander ab. Dies ist der Zeitraum, in dem die Primitiven-Approximation in ihrer gedachten Form zum tragen kommt. In der Umgebung eines Hindernisses werden je nach Distanz unterschiedlich genaue Primitive gewählt. Dem entsprechend sind auch die berechneten Abstandsvektoren unterschiedlich. Leider ist die zweite Phase auch der Zeitraum, in dem die Approximation mit mehreren Primitiven die ungenausten Ergebnisse liefert. Wird im Gegensatz dazu nur mit der Bounding-Box approximiert, dann ist

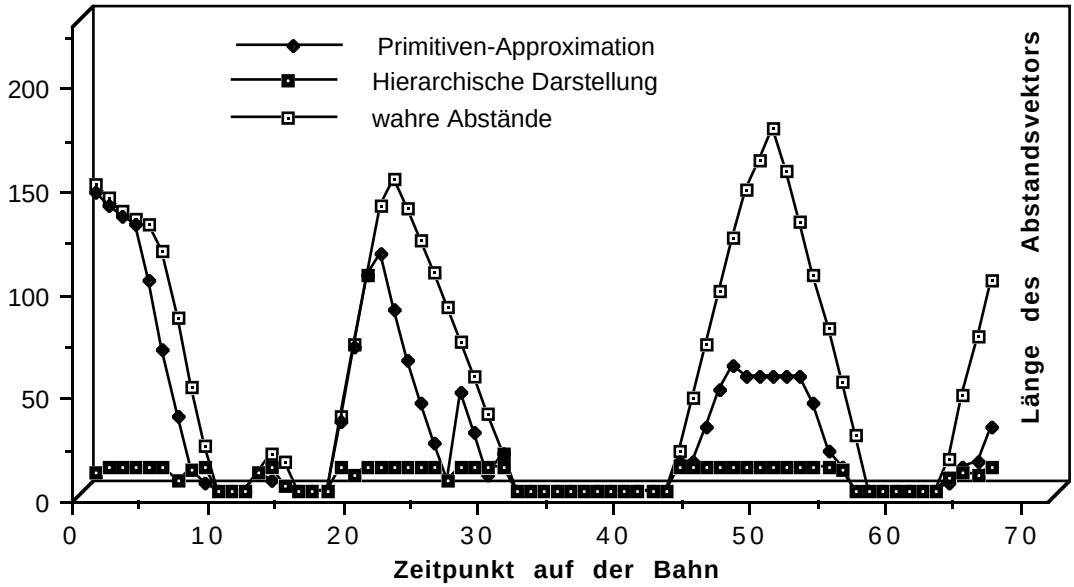


Abbildung 5.6: Längengenauigkeit des Abstandsvektors bei verschiedenen Beschleunigungs-Ansätzen.

diese Strategie in der zweiten Phase gezwungen, auf die Polyeder-Abstände zurückzugreifen. Dort liefert diese Strategie als einzige immer den wahren Abstand.In Abbildung 5.6 ist die Länge des Abstandsvektors für unterschiedliche Beschleunigungsansätze der Kollisionserkennung gegeben. Diesmal ist der Zeitraum der ganzen Bewegungsbahn der ersten Beispiels-Szene aus Abbildung 5.1 abgebildet. Es wird die Primitiven-Approximation mit der hierarchischen Darstellung bei gegebenen "wahren" Abständen verglichen. (Dabei ist es unwesentlich, welche der Abstiegs-Strategien bzw. welche der Hierarchien genau dargestellt ist. In dieser Abbildung würden sie sich kaum unterscheiden.) Während der Bewegung treten vier Kollisionen mit Hindernissen auf. Diese finden zwischen dem 10. - 12., 16. - 17., 32. - 43. und 58. - 63. Bewegungsschritt statt. Zwischen den Kollisionen zeigt der wahre Abstand das Entfernen und die Annäherung des Roboterarms an die Hindernisse an. Dieses Verhalten wird von dem Abstand der Primitiven-Approximation nur abgeschwächt und von der hierarchischen Darstellung kaum wiedergegeben. Die Länge des Abstandsvektors bei den Hierarchien ist in diesem Fall also nur eingeschränkt zu gebrauchen. Dabei ist zu bemerken, daß allgemein der Abstiegs-Algorithmus, sowohl bei den Primitiven als auch bei den Hierarchien, durch den Sicherheitsabstand gesteuert wird. Dieser ist hier auf Null gesetzt. D. h. sobald bei der Kollisionserkennung für ein Objekt (oder eine Komposition) die Abstandsabschätzung größer Null ist, dann wird die Berechnung abgebrochen. Bei einem Sicherheitsabstand größer Null wird auch die Längengenauigkeit des Abstandsvektors verbessert. Näheres dazu ist in Unterkapitel 5.4 beschrieben.

Eine solche Verbesserung der Qualität des Abstandsvektor ist bei der globalen Abstands-Fortschreibung nicht möglich. Dort wird der kleinste Abstandsvektor bei jedem Bewegungsschritt um die größte zurückgelegte Distanz verkürzt. Statt den Vektor neu zu

berechnen, wird der alte aus dem vorhergegangenen Schritt in der Länge aktualisiert. Aber eine Aktualisierung der Richtung kann auf diese Art und Weise nicht erfolgen. Es existiert keine explizite Information darüber, in welche Richtung der Abstandsvektor korrigiert werden sollte. Also bleibt bis zur nächsten Neuberechnung die Richtung des Vektors unverändert. Die von der globalen Abstands-Fortschreibung berechnete Annäherung des Abstandsvektors läßt keine Aussage über die Richtung des realen Abstandsvektors zu. Die Annäherung gibt nur eine untere Schranke für die Länge des realen Vektors an.

5.4 Einflußgrößen der Kollisionserkennung

Der Prozeß der Kollisionserkennung wird von mehreren Parametern beeinträchtigt. Es ist wichtig, diese Parameter und deren Wirkungsweise zu kennen. Die Ausprägung der Parameter kann zum Beispiel die Wahl der Ansätze zur Beschleunigung beeinflussen. Die hier besprochenen Parameter der Kollisionserkennung sind der Sicherheitsabstand und die Schrittweite. Diese zwei Parameter können von außen verändert werden und sind von einer bestimmten Szene unabhängig. Umgebungsabhängige Parameter, wie z. B. die Geometrie der Objekte (Hindernisse, Roboterarme) oder die Lage der Hindernisse, sind kaum zu steuern. Auf sie wird hier nicht weiter eingegangen.

Sicherheitsabstand

Der Sicherheitsabstand ist ein Schwellwert. Er gibt an, bis zu welcher Distanz zwei Objekte für die Kollisionserkennung interessant sind. Ist der Abstand zwischen zwei Objekten größer als der Sicherheitsabstand, so müssen die Objekte nicht weiter betrachtet werden. Mit diesem Schwellwert wird die Abstandsberechnung der Beschleunigungsansätze gesteuert, die mit mehreren Darstellungen arbeiten. Dies ist sowohl die Primitiven-Approximation als auch die hierarchische Darstellung. In beiden Ansätzen gibt es unterschiedlich genaue Repräsentationen der Objekte. Meistens wird bei der Abstandsberechnung mit der ungenausten Darstellung begonnen. Ist der Abstand kleiner dem Sicherheitsabstand, dann wird zu einer genaueren Darstellung abgestiegen. Ansonsten ist die Distanz größer dem Schwellwert und kann als Ergebnis ausgegeben werden. Der Sicherheitsabstand gibt also den Einflußbereich eines Objektes auf seine Umgebung an.

Der Sicherheitsabstand hat auch Einfluß auf die Qualität des Abstandsvektors. Je größer der Schwellwert, desto genauer sind die Abstände. Man kann zum Beispiel den Sicherheitsabstand auf den Wert Null setzen. Dann stoppt der Abstieg, sobald für ein Objektpaar der Abstand echt größer Null berechnet wird. Die resultierenden Abstände zweier Objekte sind also meistens nur so groß, wie die gröbste Darstellung der Objekte voneinander entfernt ist. Wird dagegen der Sicherheitsabstand auf einen Wert deutlich größer Null gesetzt, dann sind alle berechneten Abstände kleiner der Schwelle exakt. Der Abstieg in den Darstellungen versucht immer einen Abstand größer der Schwelle zu erhalten. Ist dies nicht möglich, dann steigt der Algorithmus

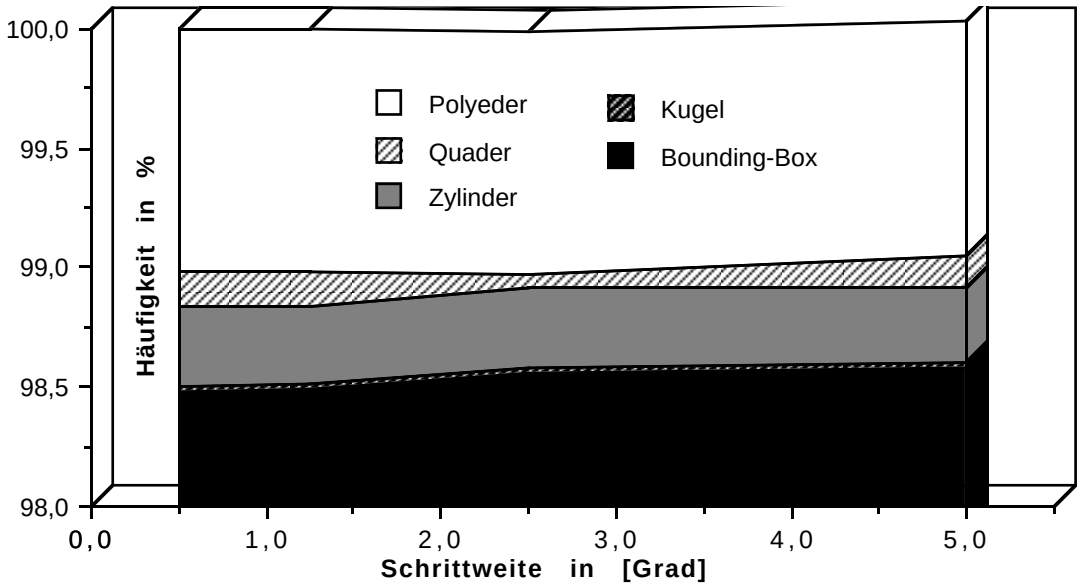


Abbildung 5.7: Anteile der verwendeten Primitiven bei unterschiedlichen Schrittweiten.

bis zur genauesten Darstellung hinab. Im Extremfall - einer unendlich großen Schwelle - entsprechen alle resultierenden Abstände den realen Distanzen. In diesem Fall sind die Beschleunigungs-Ansätze ziemlich unnütz, da sie die eigentliche Berechnung verlangsamen. Statt ungenaue Approximationen zu verwenden, könnte dann gleich die genaueste Darstellung (konvexe Polyeder) eingesetzt werden.

Schrittweite

Die Schrittweite ist der zweite wichtige Parameter der Kollisionserkennung. Sie gibt an, nach welcher zurückgelegten Distanz des Roboterarms spätestens die Kollisionserkennung aufgerufen werden soll. Die Distanz ist hier als Norm in dem von den Gelenkwinkeln aufgespannten Konfigurationsraum angegeben. Sie ist ein Maß für das Ausmaß einer Bewegung des Roboterarms. Je nach Beschleunigungs-Ansatz kann es günstiger sein, eher öfter oder eher seltener die Abstandsberechnungen durchzuführen. Wird die Schrittweite groß gewählt, dann kann der Roboterarm weite Distanzen zurücklegen und die Abstandsberechnungen werden seltener durchgeführt. Bei einer kleinen Schrittweite wird die Kollisionserkennung öfter aufgerufen. Aber die Schrittweite ist nur ein indirektes Maß für die größte zurückgelegte Distanz im kartesischen Raum. Die kartesische Distanz muß extra berechnet oder abgeschätzt werden.

Betrachtet man das Verhalten der Primitiven-Approximation bei unterschiedlichen Schrittweiten der Kollisionserkennung, so ergibt sich Abbildung 5.7. Dort ist für verschiedene Schrittweiten die Häufigkeit der "terminierenden Primitiven" angegeben. Dabei ist ein terminierendes Primitiv die letzte Darstellung, die bei dem Abstieg in der Abstandsberechnung verwendet wurde. Zum

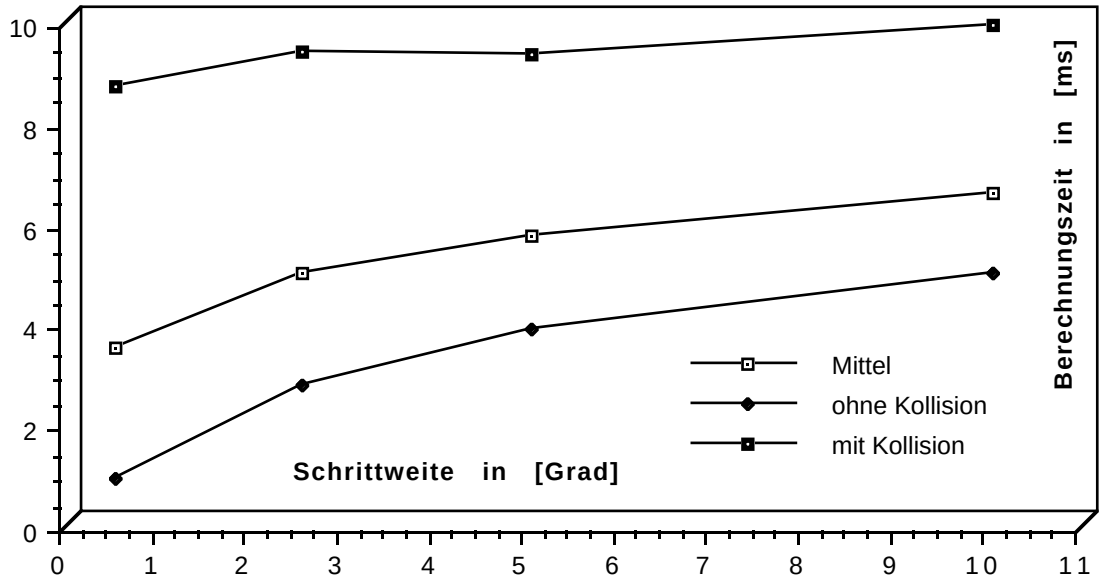


Abbildung 5.8: Berechnungszeiten bei unterschiedlichen Schrittweiten der globalen Abstands-Fortschreibung.

Beispiel ist der Zylinder dann terminierend, wenn er das erste Primitiv mit Abstand größer dem Sicherheitsabstand ist.

In der Abbildung ist nun für jedes der eingesetzten Primitive die Häufigkeit angegeben, mit der dieses Primitiv die Abstandsberechnung zum Terminieren geführt hat. Von dem gesamten Ergebnis ist zur Verdeutlichung nur ein Ausschnitt von 2 % dargestellt. Man erkennt, daß die Primitiven zwischen der Bounding-Box und den konvexen Polyedern sehr selten zum Einsatz kommen. Die Kugel, der Zylinder und der Quader werden von allen beobachteten Bewegungsschritten nur in 0,5 % verwendet. Außerdem ist der Einsatz der einzelnen Primitiven in dem Interval der Schrittweiten ziehmlich ausgeglichen. Es gibt kaum Schwankungen bei veränderter Schrittweite.

In Abbildung 5.8 sind für vier Schrittweiten der Kollisionserkennung die Laufzeiten der Abstands-Fortschreibung angegeben. Die Zeiten sind unterschieden zwischen den Schritten mit und ohne aufgetretener Kollision. Zusätzlich ist das gewichtete Mittel aus diesen zwei Fällen aufgeführt.

5.5 Zusammenfassung der Ergebnisse

Zusammenfassend sollen die wichtigsten Ergebnisse aus den Experimenten wiederholt werden.

- Bei der Primitiven-Approximation führt der ausschließliche Einsatz der Bounding-Box zu den kürzesten Berechnungszeiten. D. h. bei mehreren Primitiven als Approximation wird die Abstandsberechnung entgegen den Erwartungen aus [Adolphs] verlangsamt.

- Von allen verglichenen Beschleunigungs-Ansätzen sind die dynamischen Hierarchien mit Bounding-Boxes als Kompositionen trotz der notwendigen Aktualisierung am schnellsten.
- Bei der Primitiven-Approximation führt der ausschließliche Einsatz der Bounding-Box zu den genauesten Abstandsvektoren. Bei mehreren Primitiven wird in bestimmten Situationen ein ungenauerer Vektor berechnet.
- Im Allgemeinen gilt: Je größer die Beschleunigung eines Ansatzes, desto ungenauer ist der berechnete Abstandsvektor. (Eine Ausnahme bildet die Bounding-Box als Primitiven-Approximation.)
- Der Sicherheitsabstand beeinflußt bei allen Abstiegs-Strategien die Genauigkeit des Abstandsvektors: Die berechneten Abstandsvektoren kleiner als der Sicherheitsabstand entsprechen dem exakten Vektor.

Bewertung der Ansätze

Aus den Ergebnissen der Experimente werden nun die einzelnen Beschleunigungs-Ansätze für den Einsatz in der on-line Kollisionserkennung bewertet. Auf Grund dieser Bewertung soll eine sinnvolle Anwendung in einem Gesamtsystem ermöglicht werden.

- Aus den vorgestellten Beschleunigungs-Ansätzen sollten mindestens zwei gewählt werden. Einer sollte aus der Gruppe "Reduzierung der Anzahl von Berechnungen" und einer aus der Gruppe "Reduzierung des Aufwands einer Berechnung" sein.
- Von der Gruppe "Reduzierung des Aufwands einer Berechnung" sollte eine Primitiven-Approximation mit nur der Bounding-Box eingesetzt werden.
- Von der Gruppe "Reduzierung der Anzahl der Berechnungen" sollten die hierarchischen Darstellungen verwendet werden. Insbesondere zeigen sich die dynamischen Hierarchien als wirkungsvolle Approximation der Roboterarme.
- Je nach Aufwand zur Berechnung der größten zurückgelegten Distanz kann man die hierarchische Darstellung mit einer Abstands-Fortschreibung kombinieren.
- Der Abbruch der Berechnung bei einem Abstand Null kann mit den meisten Ansätzen ohne Einschränkungen kombiniert werden. (Eine Ausnahme bildet z. B. die lokale Abstands-Fortschreibung.) Der Abbruch beschleunigt nur die Kollisionserkennung bei auftretender Kollision.
- Die Verwaltung der Kollisionsdaten zum Einsatz der Startwerte bei der Verwendung von hierarchischen Darstellungen ist so aufwendig, daß sie die gewonnene Beschleunigung der Hierarchien zunichte machen würde.

6 Zusammenfassung

Dieses Kapitel gliedert sich in drei Teile. Zuerst wird die Vorgehensweise in dieser Diplomarbeit zusammengefaßt. Dann folgen die gewonnenen Schlußfolgerungen mit einer Bewertung der verglichenen Ansätze. Der letzte Teil ist ein Ausblick auf die möglichen Anwendungen der Ergebnisse dieser Arbeit.

Methodik

Bei der Aufgabe, eine on-line Kollisionserkennung mit hierarchisch modellierten Hindernissen für ein Mehrarm-Robotersystem zu untersuchen, wurden folgende Schritte vorgenommen:

- Klassifizierung der bisherigen Ansätze zur Beschleunigung der Kollisionserkennung mit einem Arm. Dabei wurde unterschieden zwischen dem Einsatz-Zeitpunkt und der Methode der Ansätze.
- Modifikation des Weltmodells der Kollisionserkennung für den Einsatz von mehreren Roboterarmen. Kollisionsklassen wurden in einer formalen Darstellung eingeführt und ihre Eigenschaften betrachtet.
- Untersuchung der Approximation von Objekten (z. B. Armsegmente und Hindernisse) durch Primitive. Dabei wurden Algorithmen zur Berechnung der Approximationen entworfen und implementiert. Unterschiedliche Strategien zur Abstandsberechnung mit Primitiven-Approximationen wurden entwickelt.
- Untersuchung und Erweiterung der hierarchischen Modellierung für den Einsatz bei bewegten Objekten (wie z. B. Roboterarmen). Dazu wurden eine on-line Aktualisierung der Geometrie und eine Auswahl der optimalen Baumstruktur eingesetzt.
- Implementierung der bisherigen und eigenen Beschleunigungs-Ansätze. Für die Durchführung von Experimenten wurden die Ansätze in ein Simulationssystem eingebunden und das Simulationssystem erweitert.
- Vergleich und Bewertung der Beschleunigungs-Ansätze durch Messung von Laufzeiten und Qualität der resultierenden Abstandsvektoren. Untersuchung anderer einflußnehmender Parameter, wie z. B. der Sicherheitsabstand oder die Frequenz (Schrittweite) der Kollisionserkennung.

Schlußfolgerungen

Die Untersuchung der implementierten Ansätze zur on-line Kollisionserkennung erlaubt folgende Bewertung und Folgerungen:

- Die on-line Kollisionserkennung mit Abstandsvektoren für mehrere Arme ist bei der Verwendung der entsprechenden Beschleunigungs-Ansätze möglich. Die Berechnungszeit pro Bewegungsschritt liegt im Bereich von wenigen Millisekunden.
- Die eingeführte Unterteilung der Umwelt in Kollisionsklassen schafft einen einfachen Mechanismus zur Kollisionserkennung in Szenen mit mehreren bewegten Objekten. Die Kollisionsklassen ermöglichen eine systematische Kollisionserkennung für ein Mehrarm-Robotersystem.
- Der Vergleich der Primitiven-Approximationen zeigt, daß der ausschließliche Einsatz der Bounding-Box als Primitiv zu besseren Ergebnissen führt als der Einsatz von mehreren Primitiven wie z. B. in [Adolphs]. Die Verbesserungen betreffen den Aufwand der Abstandsberechnung und die Qualität des Abstandsvektors.
- Schon bei wenigen Objekten empfiehlt sich eine hierarchische Darstellung zur Beschleunigung der Kollisionserkennung, da sie zu sehr schnellen Abstandsberechnungen führt. Vor allem bei Szenen mit vielen Objekten ist eine hierarchische Darstellung unverzichtbar.
- Durch das neue Konzept der dynamischen Hierarchien ist eine hierarchische Modellierung auch für bewegte Objekte möglich. Die dynamischen Hierarchien garantieren eine optimale Darstellung und ermöglichen eine relativ genaue Modellierung des Roboterarms.

Bei der Kollisionserkennung für mehrere Roboterarme ist durch den Einsatz von dynamischen Hierarchien ein Beschleunigungsfaktor von 100 gegenüber dem einfachen Verfahren erreicht worden. Damit ist eine on-line Anwendung möglich.

Ausblick

Aufbauend auf den hier erzielten Ergebnissen sind folgende Anwendungen denkbar:

- Eine on-line Bahnplanung basierend auf den berechneten Abstandsvektoren ist möglich. Von Punkt zu Punkt geplante Bahnen können mit Hilfe der Abstandsvektoren modifiziert und Ausweichtrajektorien generiert werden.
- Die Potential-Feld-Methode zur lokalen Bahnplanung kann aufgrund den on-line berechneten Abstandsvektoren angewendet werden.
- Auch exakte Abstände können schnell berechnet werden. Diese Abstandsberechnung wird effizient durch die Kombination der dynamischen Hierarchien mit einer A*-Suche.
- Die schnelle Abstandsberechnung kann auch für andere Gebiete eingesetzt werden. Beispiele dafür sind der Aufbau von Konfigurations-Räumen oder die Layout-Planung.

- Bei zu handhabenden Objekten, wie z. B. Werkstücke, kann das Konzept der Kollisionsklassen einfach um einen dynamischen Wechsel erweitert werden. Dabei wechselt z. B. ein Werkstück, wenn es von dem Arm gegriffen wird, die Klasse.

Anhang A: Implementierung

Hier wird auf die Implementierung der Kollisionserkennung eingegangen. Dadurch kann ein Überblick über den Umfang und die Wirkungsweise des Systems gewonnen werden. Außerdem dient die Bschreibung der Implementierung als Hilfe für einen späteren Benutzer.

Zunächst wird die Testumgebung zur Verifizierung der Grundalgorithmen beschrieben. Durch sie ist es möglich geworden, viele einzelne Funktionen in relativ kurzer Zeit auszutesten. Dann folgt ein Überblick über das Gesamtsystem mit der Beschreibung der einzelnen Module. Ein spezielles Modul ist dabei herausgegriffen und genauer dargestellt. Es ist die schnelle Verwaltung einer komplexen Datenstruktur. Sie dient zur Speicherung zusätzlicher Information zu jedem relevanten Objektpaar. Schließlich sollen einige Erfahrungen zur Programmiertechnik und ein Anwendungsbeispiel beschrieben werden.

A.1 Testumgebungen

Bei der Realisierung der Kollisionserkennung sind sehr viele einzelne Algorithmen zu implementieren. Vor allem die Grundalgorithmen sollen eine relativ große Fehlerfreiheit aufweisen. Auf der anderen Seite sollen die Algorithmen leicht zu testen sein. Um diesen beiden Gesichtspunkten zu entsprechen wurde eine Testumgebung implementiert. Sie besteht aus zwei Modulen, deren Schnittstelle im Kapitel A.3 genauer beschrieben wird. Im ersten Modul "textuell" werden Funktionen zur textuellen Ein- und Ausgabe von Daten zur Verfügung gestellt. Da alle Algorithmen mit derselben globalen Datenstruktur arbeiten, können die Funktionen bei den meisten Algorithmen als Testausgaben verwendet werden. Das zweite Modul "ddd" dient zur Darstellung dreidimensionaler Objekte. Durch eine visuelle Repräsentation von Daten können Algorithmen und deren Ergebnisse schnell überprüft werden.

In der Abbildung A.1 ist eine solche Darstellung dreidimensionaler Objekte gezeigt. Die Objekte werden gleichzeitig von drei Ansichten aus gezeichnet. Dies geschieht in einem SunView-Fenster mit drei Unterfenster. Die Sichtweisen der Ansichten wurden parallel zu den Koordinatenachsen gewählt. Die Objekte können verschiedene Primitive, wie Polyeder, Quader, Zylinder, Kugel oder Bounding-Box, sein. Die Darstellung der Objekte erfolgt in der Parallelprojektion.

Eingesetzt wurde das ddd-Fenster bei der Implementierung von Grundalgorithmen. Die korrekte Berechnung der Primitiven-Approximation könnte damit überprüft werden. Der Polyeder wurde dazu mit seinen Approximationen in dem Fenster dargestellt. Auch die Korrektheit des Abstandsvektors zwischen zwei Primitiven läßt sich dadurch feststellen.

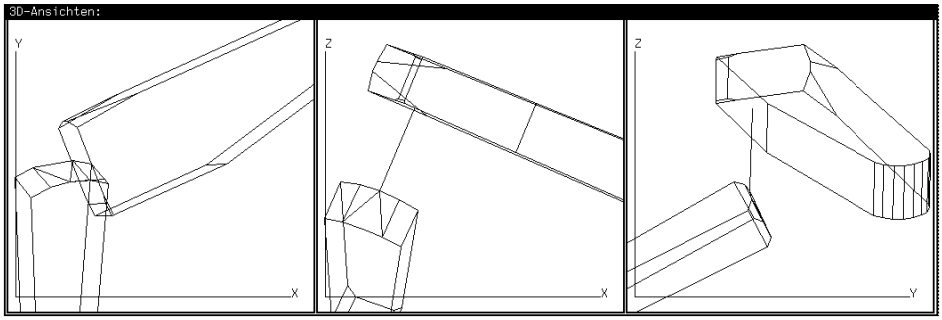


Abbildung A.1: SunView-Fenster zur Darstellung der drei Parallel-Projektionen entlang den Koordinatenachsen.

Ein anderes Hilfsmittel zum Testen der Algorithmen wurde in das Simulationssystem eingebaut. Damit konnten vor allem die Abstiegs-Strategien der Primitiven-Approximation überprüft werden. In dem vorhandenen Simulationssystem konnten verschiedene Darstellungsmodi für den Roboter per Menü ausgewählt werden. Die Darstellungen wurden um die der Primitiven-Approximation erweitert. So kann man sich nun die Objekte in ihren verschiedenen Annäherungen anzeigen lassen.

In Abbildung A.2 sind diese Darstellungsmodi für einen Roboterarm aufgeführt. Als Primitive sind Bounding-Box, Kugel, Zylinder, Quader, konvexe Polyeder und Polyeder verwendet worden. Hier sieht man deutlich den Unterschied zwischen dem Quader und der Bounding-Box. Der Quader ist zum Beispiel bei dem Unterarm in der Lage, die Orientierung des Armsegments genau zu modellieren. Die Bounding-Box hingegen muß mit ihren Flächen parallel zu den Koordinatenachsen bleiben. Es ist auch zu erkennen, daß für den Puma-Manipulator die sphärischen Primitiven (Kugel, Zylinder) nicht so geeignet sind.

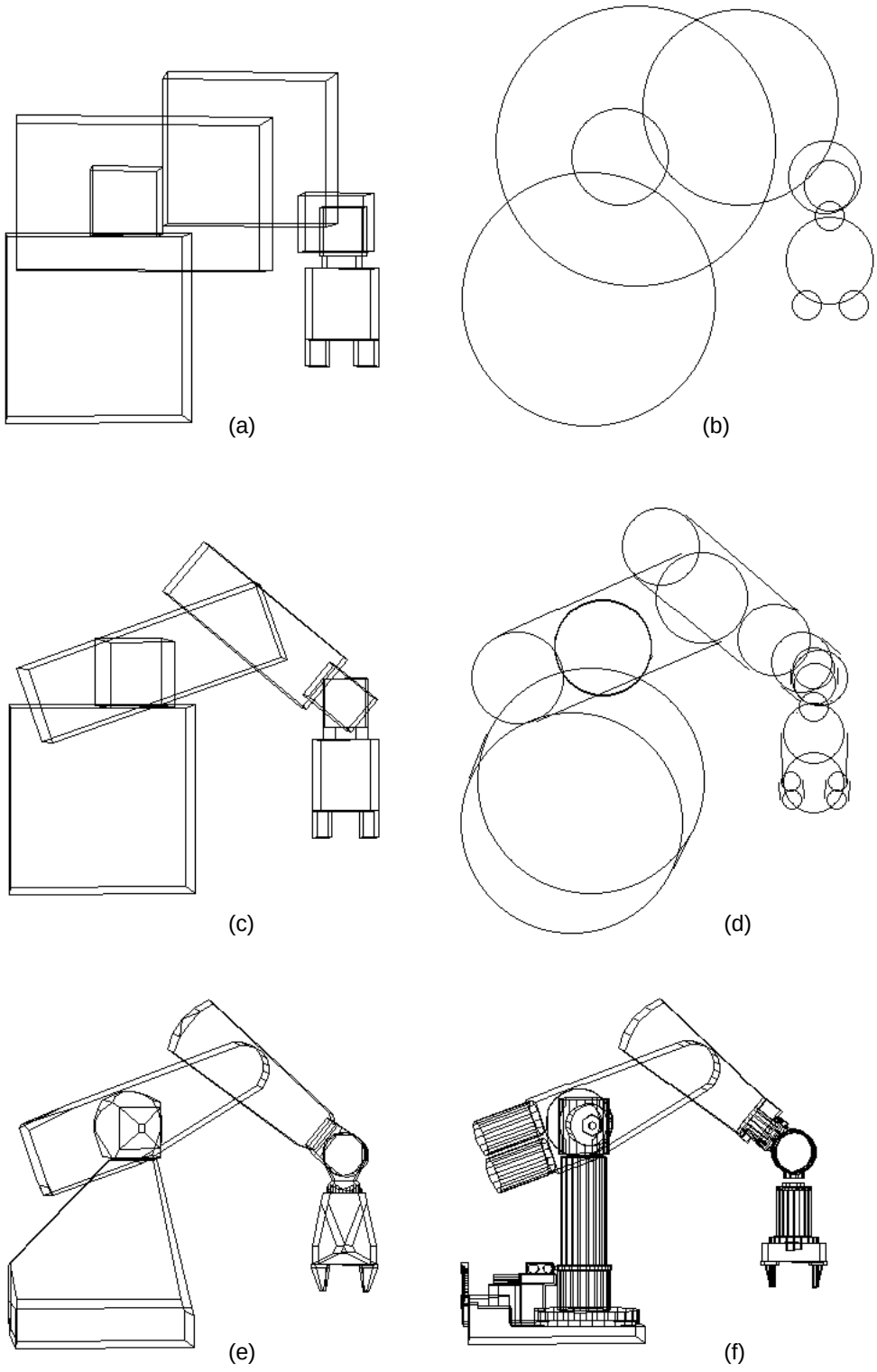


Abbildung A.2: Darstellungsformen des Puma260 in Primitiven-Approximation; (a): Bounding-Boxes; (b): Kugeln; (d): Zylinder; (c): Quader; (e): konvexe Polyeder; (f): Polyeder.

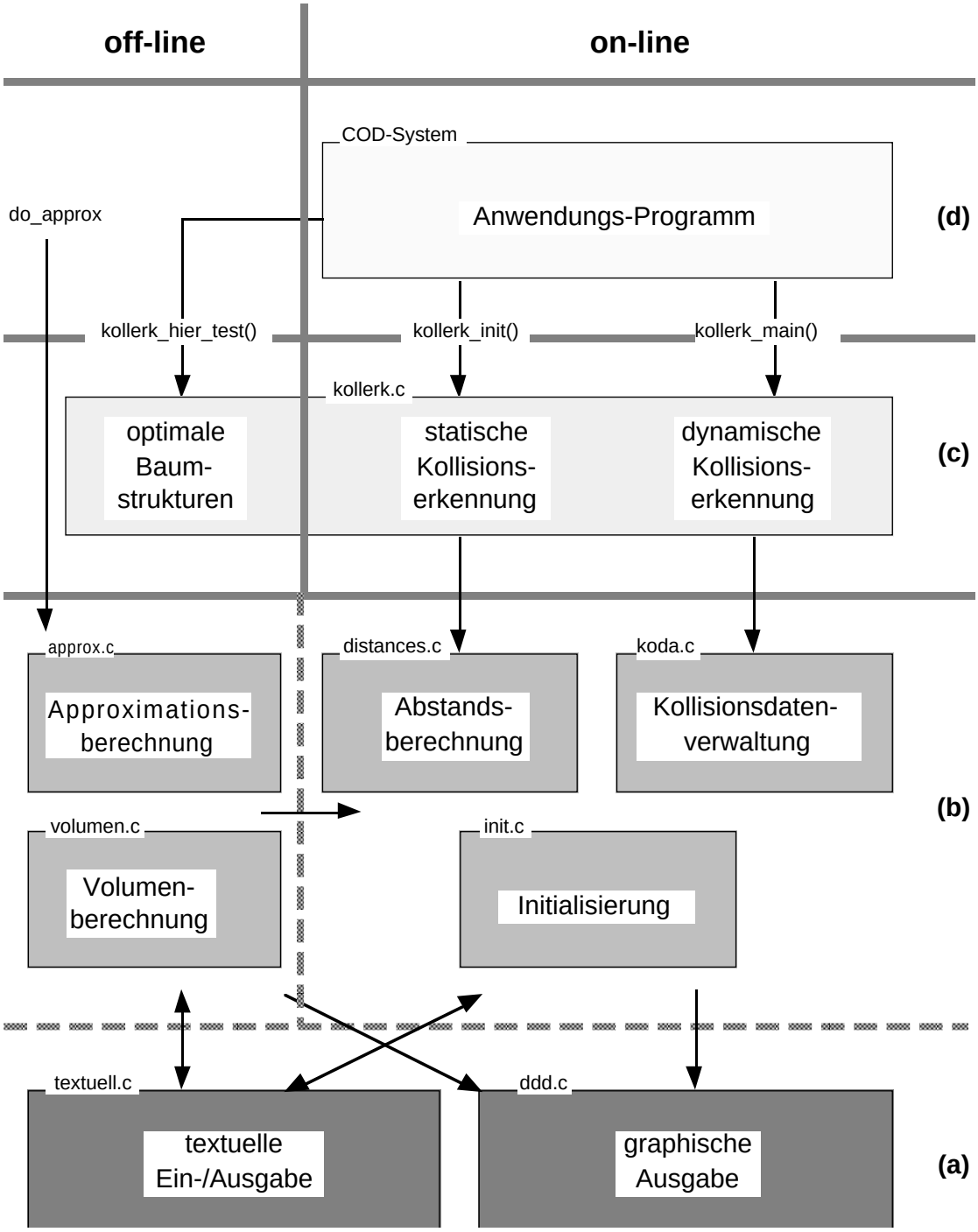


Abbildung A.3: Modulstruktur für die Kollisionserkennung in vier Ebenen: (a): Testumgebung,(b): Grundalgorithmen, (c): Kollisionserkennung, (d): Anwendung.

A.2 Gesamtsystem

Das Gesamtsystem der Kollisionserkennung wird durch einen Überblick der Module beschrieben. Die Modulstruktur ist in Abbildung A.3 gezeigt. Dabei wurde eine Trennung in off-line und on-line Berechnungen vorgenommen. Sie spiegelt sich auch in der Aufteilung der Module wieder. Dies schließt nicht aus, daß einige Module in beiden Bereichen verwendet werden können. Zusätzlich läßt sich der Algorithmus der Kollisionserkennung weiter in statische und dynamische Berechnungen unterteilen. In dem statischen Teil werden einmalig Vorbereitungen getroffen. Dies ist z. B. für den Aufbau von hierarchischen Darstellungen oder von den Kollisionsdaten notwendig. Die dynamische Kollisionserkennung wird zwischen den Bewegungsschritten der Roboterarme aufgerufen. Sie führt dann die eigentliche Abstandsberechnungen durch.

Eine weitere Untergliederung der Modulstruktur findet in horizontalen Ebenen statt. Sie entsprechen verschiedenen Abstraktionsniveaus. Es lassen sich vier Ebenen unterscheiden: Testumgebung, Grundalgorithmen, Kollisionserkennung und Anwendung. Die Testumgebung dient zur Entwicklung und Verifizierung der Grundalgorithmen. Die nächste Ebene enthält Algorithmen, die sich von der Kollisionserkennung abtrennen lassen und isoliert für sich stehen können. Der Einsatz in anderen Anwendungsgebieten ist denkbar. Die Ebene der Kollisionserkennung enthält die unterschiedlichen Beschleunigungsansätze. Zu ihr gehören auch die entsprechenden off-line Berechnungen. Die oberste Ebene ist die Anwendung. Hier ist die Trennung am stärksten. Der Zugriff erfolgt im wesentlichen über nur drei Funktionen.

Globale Datenstrukturen

In der globalen Datenstruktur werden Typen und Varialben declariert, die von allen Modulen aus zugreifbar sind. So sind alle Arten von Primitiven samt den Polyedern dort beschrieben. Die Primitiven-Deklarationen sind sehr einfach und sollen hier nicht aufgeführt werden.

Als wichtigstes Beispiel soll die Deklaration der Hierarchie angegeben werden. Sie dient in der Kollisionserkennung zur Reduzierung der Objektanzahl. Es handelt sich um Kompositions-Hierarchien mit Bounding-Boxes, d. h. an jedem Knoten existiert eine Bounding-Box, die die Unterbäume konservativ approximiert. Die Blätter der Hierarchie zeigen auf die genaueste Darstellung: die konvexen Polyeder. Die Marke in der Deklaration wird bei dem Aufbau durch den Nächste-Nachbar-Algorithmus benutzt. Genauso die mögliche Verzeigerung der Elemente als Liste.

```
struct Hierarchie_struct {
     struct Hierarchie_struct *left;       /*linker Unterbaum*/
     struct Hierarchie_struct *right;      /*rechter Unterbaum*/
     struct bbox bb;                       /*Bounding-Box*/
     codbody_poi body;                     /*Zeiger auf body */
     bool marke;                           /*privat*/
     struct Hierarchie_struct *next;       /*fuer Liste*/
};
typedef struct Hierarchie_struct *Hierarchie;
```

A.3 Modulbeschreibungen

Aufbauend auf der Beschreibung des Gesamtsystems aus Kapitel A.2 werden nun die einzelnen Module näher erläutert. Dabei wird vor Allem auf die Schnittstelle der Module nach außen eingegangen. Sie wird durch die aufrufbaren Funktionen und Datenstrukturen definiert. Insgesamt sind knapp 100 einzelne Funktionen (ohne modul-interne Funktionen) in 8 Modulen implementiert worden. Die hier beschriebenen Module sind "textuell", "ddd", "volumen", "approx", "init", "distances" und "kollerk". Das Modul "koda" wird ausführlich in dem Kapitel A.4 besprochen.

Modul "textuell"

Das Modul "textuell" befindet sich auf der Ebene der Testumgebung. Es wird sowohl off-line als auch on-line von anderen Modulen benutzt. Es hat keine eigene Datenstruktur. Das Modul dient vor allem zum Austesten der Algorithmen. Dazu werden an den gewünschten Stellen Funktionen mit Kontrollausgaben eingefügt. In dem Gesamtsystem wird es nur zum Einlesen der Primitiven verwendet. Die Funktionen gliedern sich in drei Gruppen: Lese-Funktionen (`read_*`) für Benutzer/Datei-Eingaben, Schreib-Funktionen für Ausgaben an den Benutzer (`print_*`) und Schreib-Funktionen für Ausgaben auf eine Datei (`write_*`). Dem entsprechend sind auch die Parameter der Funktionen. Die Lese-Funktionen haben als Eingabe einen Zeiger auf die Eingabe- und Ausgabedatei. Sie geben in einem dritten Parameter das eingelesene Primitiv zurück. Die Schreib-Funktionen benötigen nur einen Zeiger auf die Ausgabedatei und das Primitiv als Parameter. Alle Funktionen aus diesem Modul liefern als Funktionswert einen Fehlercode zurück. Er ist in der globalen Datenstruktur (`cod.h`) definiert.

```
int  read_punkt();       int  print_punkt();      int  write_punkt();
int  read_ebene();       int  print_ebene();
int  read_linie();       int  print_linie();      int  write_linie();
int  read_bbox();        int  print_bbox();       int  write_bbox();
int  read_kugel();       int  print_kugel();      int  write_kugel();
int  read_rzylinder();   int  print_rzylinder();  int  write_rzylinder();
int  read_quader();      int  print_quader();     int  write_quader();
int  read_quader2();
int  read_kpoly();       int  print_kpoly();      int  write_kpoly();
kPoly  *read_body();
int  free_kpoly();
int  read_mindist();     int  print_mindist();
```

Die Funktionen, die auf Dateien lesen oder schreiben, arbeiten mit dem selben Format. Es ist der Eingabe des Benutzers angepaßt. Daher sind keine extra Funktionen für das Einlesen von Dateien nötig. Dateien mit dem hier verwendeten Format tragen das Postfix "`.ddd`". Das Format ist sehr einfach: Zuerst kommt eine Kennung, die das Primitiv bestimmt (b = Bounding-Box, k = Kugel, r = runder Zylinder, q = Quader, p = konvexer Polyeder). Dann folgen reelle Zahlenwerte, die das ausgewählte Primitiv parametrisieren. Alle Eingaben werden mit dem Zeichen <Return> abgeschlossen.

Modul "ddd"

Das ist das zweite Modul in der Testumgebung. Es wird von anderen Modulen off-line und on-line benutzt. Es dient zur Darstellung dreidimensionaler Objekte. Die Objekte werden von drei Ansichten aus gleichzeitig gezeichnet. Die Sichtweisen der Ansichten wurden parallel zu den Koordinatenachsen gewählt und sind in einem SunView-Fenster zusammengefaßt (siehe Abbildung A.1). Die dreidimensionalen Objekte (meist Primitive) werden in der Parallelprojektion dargestellt. Dazu gibt es zwei Sorten von Funktionen: die erste verwaltet das Fenster mit den drei Ansichten, die zweite zeichnet ein Primitiv in das Fenster. Bei dem Testen der Grundalgorithmen ist immer nur ein Fenster nötig. Daher können die meisten Parameter auch implizit übergeben werden. Im Wesentlichen sind die Verwaltungsfunktionen parameterfrei und die Zeichenfunktionen haben das Primitiv als Parameter.

```
void pw_circle();                 void ddd_draw_linie();
                                  void ddd_draw_bbox();
void ddd_koord_achsen();          void ddd_draw_kugel();
void ddd_open_window();           void ddd_draw_rzylinder();
void ddd_close_window();          void ddd_draw_quader();
void ddd_clear_window();          void ddd_draw_kpoly();
```

Die SunView-Bibliothek für Graphik-Operationen unterstützt vor allem Linienzeichnungen und komplex Objekte. Eine einfache Routine zum Zeichnen von Kreisen fehlt jedoch. Kreise sind zur Darstellung der Primitiven Kugel und Zylinder notwendig. Daher wurde eine zusätzliche Funktion `pw_circle()` implementiert. Diese Funktion zeichnet effizient einen Kreis nach einem Algorithmus von Bresenham aus [Foly83]. Dabei wird die 8-Symmetrie des Kreises bezüglich des Koordinatensystems ausgenutzt. Da die Funktion auch zur Roboterdarstellung

wie in Abbildung A.2 vom Gesamtsystem verwendet werden kann, wurde sie mit in die Schnittstelle aufgenommen.

Modul "volumen"

Dieses Modul ist eine Sammlung sehr einfacher Berechnungen. Es befindet sich auf der Ebene der Grundalgorithmen. Diese Modul kommt nur off-line bei der Berechnung von Primitiven-Approximationen zum Einsatz. Zu jedem Primitiven-Typ existiert eine entsprechende Funktion. Sie berechnet einem konkreten Primitiv das Volumen, welches von dem Primitiv eingenommen wird. Die Parameter der Funktionen sind dem entsprechend ein Primitiv als Eingabe und das Volumen als Ausgabe.

```
int  volumen_bbox();
int  volumen_kugel();
int  volumen_rzylinder();
int  volumen_quader();
int  volumen_kpoly();
```

Modul "approx"

Das Modul "approx" gehört auf die Ebene der Grundalgorithmen. Da es sehr aufwendige Funktionen enthält, kann es nur off-line eingesetzt werden. Die Funktionen dienen zur Berechnung von Approximationen. Es wird jeweils ein Primitiv durch ein einfacheres ersetzt. Die Approximationen sind, wenn auch nicht immer optimal, konservativ. Sie enthalten also auf jeden Fall das zu approximierende Primitiv. Bei der Namensgebung der Funktionen steht das angenäherte Primitiv an erster Stelle und das annähernde an zweiter. Für die Kollisionserkennung sind nur die Funktionen wichtig, die konvexe Polyeder approximieren. Die anderen Funktionen führen nicht unbedingt zu der optimalen Approximation des realen Objekts, da sie ihrerseits auf Annäherungen aufbauen.

```
void approx_kugel_bbox();
void approx_rzylinder_bbox();
void approx_rzylinder_kugel();
void approx_quader_bbox();
void approx_quader_kugel();
void approx_quader_rzylinder();
void approx_kpoly_bbox();
void approx_kpoly_kugel();
void approx_kpoly_rzylinder();
void approx_kpoly_quader();
```

Modul "init"

Das Modul "init" ist keine Initialisierung im strengen Sinn. Es enthält Funktionen, die von mehreren anderen Modulen zu Beginn des Ablaufs verwendet werden. Der Einsatz ist sowohl off-line als auch on-line möglich.

```
void make_quader_ebenen();
void bbox_to_quader();
```

Modul "distances"

Eines der wichtigsten Module der Kollisionserkennung ist das Modul "distances". Die enthaltenen Funktionen berechnen zwischen zwei Primitiven den kürzesten Abstandsvektor. Das Modul zählt damit zu der Ebene der Grundalgorithmen. Hauptsächlich werden die Funktionen on-line aufgerufen. Zur off-line Approximation von Objekten werden die Abstandsberechnungen jedoch auch eingesetzt. Von dem Modul "koda", welches die Kollisionsdaten verwaltet, wird die Datenstruktur `struct kPoly_Init` verwendet. Diese Daten können als Startwerte bei den Abstandsberechnungen mit konvexen Polyedern dienen.

Das Modul ist in zwei Dateien aufgeteilt. Die erste Datei enthält die Abstandsberechnungen ohne konvexe Polyeder (linke Spalte) und die zweite Datei die mit konvexen Polyedern (rechte Spalte). Zusätzlich unterscheiden sich die Funktionen darin, ob sie den quadratischen Abstand (`dist2_*`) oder den echten Abstand (`dist_*`)berechnen. Dabei ist immer die Variante implementiert worden, deren Berechnung einfacher und damit der Aufwand geringer war.

Die Parameter der Funktionen sind die zwei Primitive als Eingabe und der kürzeste Abstand als Ausgabe. Dabei ist der kürzeste Abstand eine Datenstruktur aus zwei Punkten und dem Abstand der Punkte. Der Abstandsvektor berechnet sich aus der Differenz der Punkte. Bei den Funktionen für konvexe Polyeder kommt zusätzlich der Startwert der Iteration und eine boolesche Variable, ob er verwendet werden soll, als Eingabe hinzu.

```
void dist_punkt_punkt();
void dist2_punkt_punkt();
void dist2_punkt_linie();
void dist2_linie_linie();
void dist2_bbox_bbox();
void dist_bbox_kugel();
void dist_bbox_rzylinder();
void dist2_bbox_quader();
void dist_kugel_kugel();
void dist_kugel_rzylinder();
void dist_kugel_quader();
void dist_rzylinder_rzylinder();
void dist_rzylinder_quader();
void dist2_quader_quader();
```

```
int  dist2_bbox_kpoly();
int  dist_kugel_kpoly();
int  dist_rzylinder_kpoly();
int  dist2_quader_kpoly();
int  dist2_kpoly_kpoly();
```

Modul "kollerk"

Das Hauptmodul der hier implementierten Kollisionserkennung ist "kollerk". Neben den globalen Datenstrukturen benutzt es die Szenenbeschreibung aus dem Simulationssystem. Die Schnittstelle des Moduls konnte sehr klein gehalten werden. Sie enthält nur zwei Funktionen. Eine dritte Funktion `kollerk_hier_test()` dient zum Austesten der möglichen optimalen Baumstrukturen eines Manipulators. Sie wird nur off-line verwendet und ist damit von den anderen separiert. Die zwei eigentlichen Funktionen des Moduls werden on-line aufgerufen. Sie führen die statische und dynamische Kollisionserkennung aus. Für die statische Kollisionserkennung wird `kollerk_init()` einmalig vor dem Prüfen einer Bahn auf

Kollisionen ausgeführt. Während der Ausführung einer Bahn wird zwischen den Bewegungschritten `kollerk_main_timer()` aufgerufen. Zum Experimentieren ist in diese Funktion zusätzlich eine Zeitmessung eingebaut.

```
int  kollerk_init();
int  kollerk_main_timer();

int  kollerk_hier_test();
```

Die Parameter der Funktionen sind so gewählt, daß jeder der implementierten Ansätze damit ausgeführt werden kann. `kollerk_init()` erhält als Eingabe einen Zeiger auf die Szenenbeschreibung. Daraus können die Kollisionsdaten oder hierarchische Darstellungen berechnet werden. Die Ausgabe ist ein Zeiger auf die Kollisionsdaten und eine eventuell veränderte Szenenbeschreibung. `kollerk_main_timer()` erhält als Eingabe eine Zeiger auf die Szenenbeschreibung, zwei aufeinander folgende aktuelle Roboterpositionen und einen Genauigkeitsfaktor für die Zeitmessung. Die Ausgabe ist der kürzeste Abstandsvektor der Szene, die verbrauchte Zeit für seine Berechnung und eine Statistik über die verwendeten Algorithmen. (Von dieser Funktion wird zunächst nur ein Abstandsvektor berechnet, da das Simulationssystem nur zwei Kollisionsklassen vorsieht. Eine Anpassung auf mehrere Klassen ist trivial.) Nun die Parameter im Detail:

`int` **`kollerk_init`**`(data, koda)`

Eingabe:

`data`	Zeiger auf Szenenbeschreibung

Ausgabe:

`data`	Zeiger auf Szenenbeschreibung, ev. um hierarchische Darstellung erweitert.
`koda`	initialisierte Kollisionsdaten für die Szene
`kollerk_init`	= `SUCC` bei Erfolg, `ERROR` sonst

`int` **`kollerk_main_timer`**`(data, koda, pos1, pos2, md, faktor, time, stat)`

Eingabe:

`data`	Zeiger auf aktuelle Szenenbeschreibung
`koda`	Zeiger auf Kollisionsdaten, mit `kollerk_init()` initialisiert
`pos1, pos2`	Zeiger auf 2 aufeinander folgende Positionen der Roboter
`faktor`	Genauigkeits-Faktor der Zeitmessung

Ausgabe:

`md`	Zeiger auf kürzesten Abstandsvektor der Szene
`time`	Zeiger auf verbrauchte Zeit in [ms]; falls `faktor` <= 0: keine Zeitmessung; bei faktor > 0: Zeitmessung mit Genauigkeit von `±(CLK_DELTA*CLK_MS)/faktor`
`stat`	Histogramm über die Aufrufe der Abstandsalgorithmen, falls `STATISTIK` definiert ist.
`kollerk_main_timer`	= `SUCC` bei Erfolg, `ERROR` sonst

Die Besonderheit an diesem Modul ist, daß die unterschiedlichen Ansätze zur Beschleunigung der Kollisionserkennung eingestellt werden können ohne daß sich die Schnittstelle ändert. Dies geschieht durch die Präprozessor-Variable `ALGO`. Ihr können Werte zwischen -1 und 10 zugewiesen werden. Damit wird zur Übersetzungszeit der entsprechende Quellcode ausgewählt. Die Beschleunigungs-Ansätze entsprechen den Kombinationen aus dem Kapitel 5:

```
 -1: KEIN Koda; KEINE Startwerte; nur BBox- und
     kPoly-Abstände
  0: keine Kollionsvektor-Berechnung
  1: alle Paare, ohne Kollisionsdaten-Verwaltung     (exakte  Abstände)
     (Koda)
  2: mit Koda                                        (exakte  Abstände)
  3: Koda; mit Startwerten für kPolys                (exakte  Absätnde)
  4: Koda; Startwerte; gleichmäsiger Primitiven-
     Abstieg
  5: Koda; Startwerte; Primitiven-Abstieg in
     Laufzeitreihenfolge
  6: Koda; Startwerte; Laufzeit; schlechte
     Approximationen überspringen
  7: Koda; Startwerte; nur BBox- und kPoly-Abstände
  8: Koda; Startwerte; nur kPoly; globale Abstands-
     Fortschreibung
  9: BBox/kPoly; hierarchische Hindernisse
 10: BBox/kPoly; hierarchische Hindernisse und       (schnellste
     Roboter                                         Berechnung)
```

Zusätzlich stehen noch drei weitere Variablen zur Verfügung. Es ist das `KRITERIUM`, mit dem die optimale hierarchische Darstellung aufgebaut werden soll. Mit dieser Variablen kann entweder das Volumen oder die Diagonale als Kriterienfunktion ausgewählt werden. Durch die Definition von `STATISTIK` wird ein Histogramm über die Anzahl der Aufrufe der Abstandsalgorithmen in der Primitiven-Approximation erstellt. Das Histogramm wird über die Funktion `kollerk_main_timer()` ausgegeben. Die letzte Variable stellt den `ABBRUCH` bei einem Abstand kleiner dem Sicherheitsabstand ein.

Hinter der Schnittstelle befindet sich eine Vielzahl weiterer Funktionen. Sie sollen hier nur die Komplexität des Moduls veranschaulichen. Die Funktionen gliedern sich in mehrere Gruppen. In `kollerk_bodies_*()` stecken die Varianten für die Beschleunigung einer einzelnen Abstands-Berechnungen. Die `kollerk_algo_*()`-Funktionen sind Varianten für die Reduzierung der Anzahl der Berechnungen. Die Funktionen `kollerk_init_*()` liefern die dazugehörigen Initialisierungen. Dazu kommt noch eine Reihe Hilfsfunktionen, die z. B. die hierarchischen Darstellungen berechnen.

```
int  kollerk_init_1()     int  kollerk_bodies_1()  int  kollerk_algo_1()
int  kollerk_init_2()     int  kollerk_bodies_2()  int  kollerk_algo_2()
int  kollerk_init_3()     int  kollerk_bodies_3()  int  kollerk_algo_3()
                          int  kollerk_bodies_4()  int  kollerk_algo_4()
int  kollerk_bbox_hier()  int  kollerk_bodies_5()  int  kollerk_algo_5()
void kollerk_free_hier()                           int  kollerk_algo_6()
void sprint_hier()        int  kollerk_body_hier() int  kollerk_algo_7()
                          int  kollerk_hier_hier()
                                                   int  kollerk_update_hier()
                                                   int  kollerk_dist()
                                                   int  kollerk_update_koda()
```

A.4 Kollisionsdaten-Verwaltung

Um Kollisionen zu erkennen wird der Abstand zwischen den Objekten betrachtet. Dazu werden die Objekte paarweise aufgezählt und jeweils der Abstandsvektor zwischen dem Paar berechnet. In einer Szene können unter Umständen sehr viele solche Paare gebildet werden. Es ist aber nicht nötig alle möglichen Objektpaare zu betrachten. Im Prinzip sind nur diejenigen Paare entscheidend, zwischen denen auch Kollisionen auftreten können. Z. B. enthalte eine Szene einen Roboterarm und Hindernisse. Die Armsegmente sind die einzigen Objekte, die sich bewegen. Also können nur Kollisionen mit den Armsegmenten auftreten. Bei der Kollisionserkennung müssen ausschließlich die Paare betrachtet werden, die ein Armsegment enthalten. Damit reduziert sich die Anzahl der zu betrachtenden Paare beträchtlich.

Für die Kollisionserkennung ist es nützlich, wenn diese Objektpaare explizit in einer Datenstruktur gespeichert werden. Zum Einen werden die aufwendigen Abstandsberechnungen nur zwischen solchen Paaren durchgeführt, die auch miteinander kollidieren können. Zum Anderen kann zusätzlich zu jedem Paar ein sogenannter Startwert gespeichert werden. Der Startwert dient bei der iterativen Abstandsberechnung aus [Gilbert] als Initialisierung (siehe Kapitel 2.3). Diese Speicherung ist ohne explizite Darstellung der Objektpaare nicht möglich. Eine weiter Anwendung der Datenstruktur ist die lokale Abstands-Fortschreibung. Die zusätzlichen Informationen zu jedem Objektpaar sollen **Kollisionsdaten** heißen.

Der Zugriff auf die Datenstruktur der Objektpaare muß hinreichend schnell sein. Anderenfalls wird die Anwendung in der on-line Kollisionserkennung unbrauchbar. Daraus ergeben sich folgende Anforderungen an die Datenstruktur und deren Verwaltung:

- Zu einem gegebenen Objekt sollten alle Paare zugreifbar sein, die dieses Objekt enthalten. Das ist ein *indizierter* Zugriff auf eine Liste von Paaren. Dies ist nützlich, falls sich nur relativ wenige Objekte bewegt haben. Es werden dann nur die Paare betrachtet, die auch ein bewegtes Objekt enthalten. Dadurch wird der Aufwand erheblich reduziert.
- Andererseits sollte ein *sequentieller* Zugriff auf die einzelnen Objektpaare möglich sein. Dies ist von Vorteil, falls sich relativ viel Objekte bewegt haben. Es wird dann jedes Objektpaar nur einmal betrachtet. Im Gegensatz dazu würden bei dem indizierten Zugriff viele Paare mehrmals untersucht werden.
- Der indiziert Zugriff von Objekten in dieser Datenstruktur sollte über die Objekt-Adresse möglich sein. Damit fällt eine weitere Datenverwaltung für die Objekt-Indizes weg.
- Es sollten keine Paare mehrfach gespeichert werden. Ansonsten wächst unnötigerweise der Speicheraufwand.

Für die gesuchte Datenstruktur können die bekannten abstrakten Datentypen "Baum", "Liste" oder "Menge" nicht angewendet werden. Sie lassen vor allem keinen gleichzeitigen indizierten und sequentiellen Zugriff auf die Paarmenge zu. Daher wurde ein eigenes Konzept zur

Verwaltung der Kollisionsdaten entwickelt. Es ist schematisch in der Abbildung A.4 gezeigt und soll im folgenden beschrieben werden.

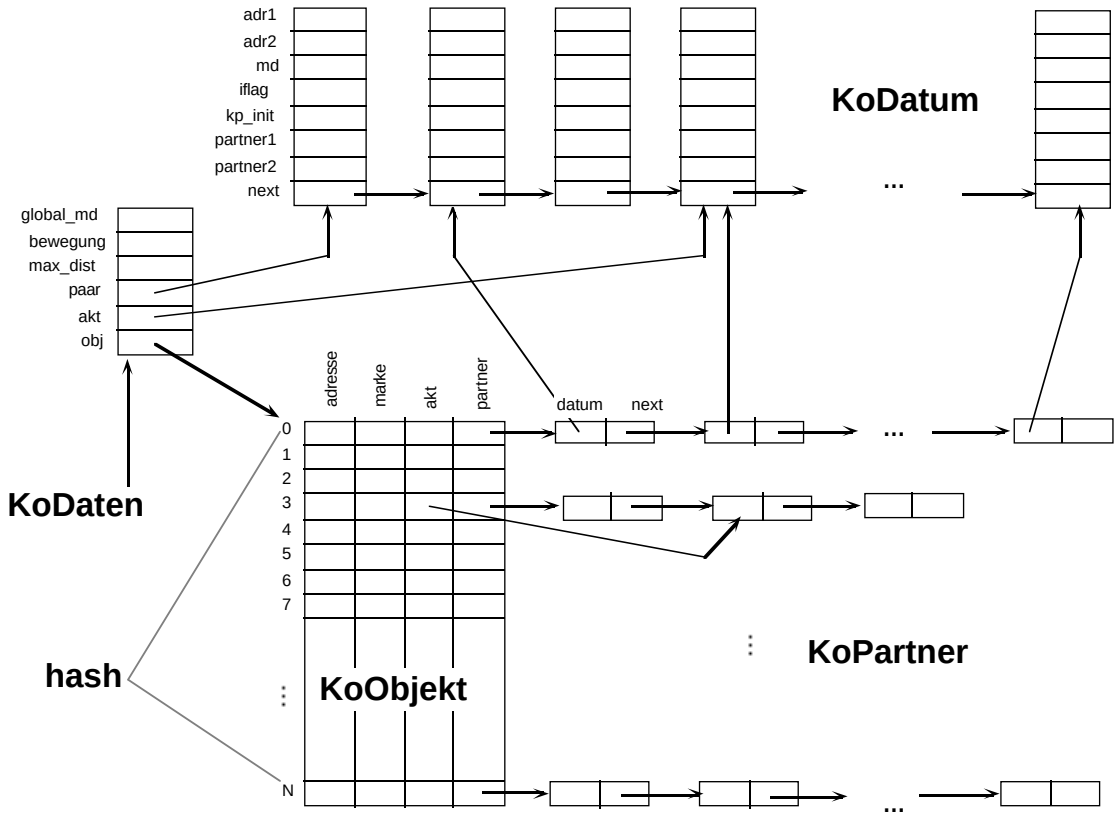


Abbildung A.4: Struktur der Kollisionsdaten mit Hash-Funktion

Die Paare (bzw. deren Kollisionsdaten) sind in einer linear verzeigerten Liste gespeichert. Dadurch ist der schnelle sequentielle Zugriff gewährleistet. Für den indizierten Zugriff wird eine Hashtabelle eingeführt. Zu jedem Index eines Objekts beinhaltet sie eine Liste mit Referenzen auf die zugehörigen Paare. Jedes dieser referenzierten Paare enthält das Objekt. Die Auflistung aller Paare, die ein bestimmte Objekt enthalten, geschieht in zwei Schritten: Zuerst wird über eine Hashfunktion aus der Objekt-Adresse der Index des Objekts in der Tabelle berechnet. Zu diesem Index gehört eine Liste mit Referenzen. Dann wird jedes dieser referenzierten Paare aufgelistet. Wird die Länge der Tabelle als Potenz von Zwei gewählt, dann ist eine schnelle Hash-Funktion durch Shifts möglich.

Bei einer sequentiellen Auflistung der Paare, sind nur diejenigen interessant, die bewegte Objekte enthalten. Um nur diese Objektpaare zu selektieren werden Marken eingeführt. Die Marken werden nun nicht an die Paare selbst, sondern an die Hashtabelle der Objekte angefügt. Bei relativ vielen bewegten Objekten werden zuerst die betroffenen Objekte in der Tabelle markiert. Zur Ausgabe der Objektpaare wird dann die Paarliste sequentiell durchgegangen. Alle Paare, die markierte Objekte enthalten werden ausgeben.In der Abbildung A.4 sind alle notwendigen Felder und Verzeigerungen angedeutet. Neben erwähnten Variablen sind noch

weitere eingefügt, die die Verwaltung erleichtern. (Zum Beispiel die Tabellen-Indizes der Partner bei den Kollisionsdaten.) Im Folgenden sind noch die formalen Deklarationen der Datenstruktur aufgeführt:

```
#define KODA_ANZ 128         /* max. Anz. von Objekten in KoDaten */
#define KODA_ANZ_EXP 7       /* 2er-Logarithmus von KODA_ANZ  */

typedef struct cod_body *ObjektAdr;
struct KoDatum {
     ObjektAdr adr1, adr2;   /* Adressen der Objekte */
     struct MinDist  md;     /* Abstandsinfo fuer dieses Paar */
     struct kPoly_Init kp_init; /* Init. fuer kPoly-Abstaende */
     bool iflag;             /* = TRUE, falls kp_init gesetzt */
/* privat: */
     int partner1, partner2; /* Index in KoDaten der Obj. des Paars */
     struct KoDatum *next;   /* Zeiger auf naechstes Paar */
};
struct KoPartner {
     struct KoDatum *datum;  /* Zeiger auf Abst.info fuer Paar */
     struct KoPartner *next; /* Zeiger auf naechstes Obj. in Liste */
};
struct KoObjekt {
     ObjektAdr adresse;      /* Zeiger auf reales Objekt */
     struct KoPartner *partner;  /* Zeiger auf Obj,
                                    die mit Obj Paare bilden */
     bool marke;             /* = TRUE, falls das Obj markiert ist */
     struct KoPartner *akt;  /* aktueller Partner eines Objektes */
};
struct KoDaten_struct {
     struct MinDist global_md;   /* kuerzester Abst.vektor von allen Paaren
*/
     double bewegung;        /* Summe der zurueckgelegten Distanzen
                                 bewegter Obj */
     double max_dist;        /* groesste zuruecklegbare Dist. bewegter
                             Objekte bei einem Bewegungs-Schritt*/
/* privat: */
     struct KoObjekt obj[KODA_ANZ];   /* Hashtabellel mit Objekten */
     struct KoDatum *paar;   /* Zeiger auf Paarliste */
     struct KoDatum *akt;    /* akt. Paar mit einem markierten Obj */
};
typedef struct KoDaten_struct *KoDaten;
```

Die gesamte Datenverwaltung wird von vorgegebenen Funktionen abgedeckt. Damit gewinnt die Benutzung der Struktur an Sicherheit. Die Kollisionsdaten werden über Einfüge-Operationen in die Paarmenge eingegeben. Dadurch baut sich sukzessive die Datenstruktur auf. Über Listen-Operationen können alle Partner eines indizierten Objekts sequentiell ausgegeben werden. Die Liste verfügt über einen internen Zeiger der dazu verschoben wird. An der aktuellen Position des Zeigers kann der Partner ausgelesen und verändert werden. Löschen von Elementen ist bei der Kollisionserkennung nicht nötig. Zur sequentiellen Ausgabe markierte Kollisionsdaten, können zunächst einzelne Objekte oder alle zusammen markiert werden. Die Aufzählung der Paare erfolgt wiederum über Listen-Operationen.

```
int  koda_create();        /*erzeugt leere Paarmenge*/
int  koda_insert();        /*fügt Paar ein*/
bool koda_exist();         /*existiert dieses Paar schon ?*/

int  koda_reset();         /*init. Paarliste eines Obj.*/
bool koda_last();          /*ex. weiteres Paar zu diesem Obj.?*/
int  koda_get();           /*gebe aktuelles Paar eines Obj. aus*/
int  koda_next();          /*gehe zu nächstem Paar eines Obj.*/

int  koda_put();           /*ändere Wert des akt Paares eines Obj.*/

int  koda_mark_all();      /*setzt Markierung von allen Obj.*/
int  koda_mark();          /*markiert ein Obj.*/
int  koda_reset_marked();  /*setzt auf 1. markiertes Paar zurück*/
bool koda_last_marked();   /*ex. weiter markierten Paare?*/
int  koda_get_marked();    /*gebe aktuelles markiertes Paar aus*/
int  koda_next_marked();   /*gehe zu nächstem Paar mit Markierung*/
int  koda_print();         /*Ausgabe der Struktur*/
```

A.5 Programmiertechnik

Es sollen nun noch einige Aspekte der angewendeten Programmiertechnik beschrieben werden. Es ist eine Reihe von Erfahrungen, die bei dieser Arbeit gesammelt wurden und nicht unerwähnt bleiben sollen:

Die verwendete Programmiersprache C läßt relativ viele Freiheiten zu bei der Programmierung zu. Dies entsteht durch die Kombination von Elementen aus hohen und niederen Programmiersprachen. Um die Nachteile der niederen Sprachen zu eliminieren wurden zwei Konzepte angewendet. Zum Einen ist in einer Pascal-ähnlichen Form programmiert worden. Zum Anderen wurden möglichst viele Tools zur Unterstützung herangezogen. Im einzelnen sind dies "make", "indent" und "lint". Durch das Tool "make" werden diejenigen Teile des Quellcodes neu übersetzt, die eine Veränderung erfahren haben. Damit ist garantiert, daß nach Aufruf von "make" mit der aktuellen Version gearbeitet wird. Das Tool "indent" sorgt für eine einheitliche Formatierung des Quellcodes. Dies erleichtert erheblich die Fehlersuche. "lint" ist der sogenannte Programm-Verifizierer. Er ist eigentlich in enger Koppelung mit dem C-Kompiler gedacht. Einige im C-Kompiler fehlende Eigenschaften werden von "lint" abgedeckt. Zusammen mit dem C-Kompiler sind "indent" und "lint" im "make"-File eingebaut worden. Dies verlängert zwar die Übersetzung des Programmpakets bei der Entwicklung, aber durch den verringerten Zeitaufwand bei der Fehlersuche ist das voll gerechtfertigt.

Bei der Dokumentation wurde auf Konsistenz geachtet. In den Modulen wird immer nur an einer Stelle ein bestimmtes Objekt beschrieben. So befindet sich die Beschreibung der einzelnen Funktionen in deren Funktions-Köpfen. Sie ist an keiner anderen Stelle außer als Stichwort zu finden.

In jedem Modul sind standardmäßig zwei Sorten von Präprozessor-Anweisungen eingeführt. Sie werden angesteuert durch die Variablen `DEBUG` und `TRACE`. Sie können entweder definiert

sein oder durch Kommentarklammern ausgeschaltet werden. Dann sind sie dem Präprozessor nicht mehr bekannt. Ist die `DEBUG`-Option angeschaltet, dann werden Kontrollausgaben an den Benutzer geliefert. Diese Ausgaben sind mehr oder weniger ausführlich und in jedem Fall nur im Zusammenhang mit dem Quelltext zu verstehen. Bei der `TRACE`-Option wird jeder Eintritt in eine Funktion gemeldet. Dies geschieht durch eine Informationszeile (`TRACE: funktions_name()`). Bei beiden Optionen werden die Ausgaben auf die Standard-Ausgabe geschrieben.

Die Namensgebung der Funktionen ist selbsterklärend. Das wird dadurch erreicht, daß jede Funktion als Präfix den Namen des Moduls trägt, dem sie angehört. Damit ist die prinzipielle Aufgabe der Funktion geklärt. Der Rest des Namens besteht aus der speziellen Aufgabe, die die Funktion innerhalb des Moduls hat. Z. B. gehört `dist_kugel_kpoly()` dem Modul `distances.c` an und berechnet den Abstand zwischen einer Kugel und einem konvexen Polyeder.

Jede Funktion liefert als Funktionswert einen Ergebniscode zurück. Die Ausnahme bilden Funktionen, die immer den selben Wert als Ergebnis haben würden. Der Ergebniscode gibt mindestens an, ob die Funktion ihre Aufgabe korrekt erfüllt hat oder nicht. Ein differenzierterer Code zeigt zusätzlich noch die Ursache eines Fehlers oder die Art des Ergebnisses an. Damit kann die aufrufende Umgebung eine Fehlerbehandlung oder statistische Analysen durchführen. Im einfachsten Fall wird der Ergebniscode durch alle Inkarnationen bis zu dem Benutzer "hochgeschleift". Der Benutzer kann dann zumindest genau den Ort des Fehlers lokalisieren.

Eine Datei `Bugs.txt` beinhaltet eine Liste mit Versäumnissen bei der Implementierung. Davon ausgegangen, daß es in jeder Software Fehler oder Ungereimtheiten existieren, dann stellt eine solche Liste ein wichtiges Werkzeug dar. Sie ist für einen Benutzer der Software eine hilfreiche Anleitung, an der Arbeit des Vorgängers weiter zu arbeiten.

A.6 Anwendungsbeispiel

Das um die Kollisionserkennung erweiterte Simulationssystem kann für weitere Experimente eingesetzt werden. Zum Verständnis der Benutzung dieses Experimentier-Systems wird hier ein Beispiel zur Anwendung beschrieben. Dieses Beispiel kann auch als Vorlage zum Einsatz der implementierten Kollisionserkennung in einem anderen System dienen. Die Schritte, die das übernommene Simulationssystem betreffen, werden nur skizzenhaft beschrieben. Die dazugehörigen Referenzen sind angegeben.

Das Beispiel bespricht die Verwendung eines neuen Robotertyps und neue Hindernisse in dem Simulationssystem. Mit diesen neuen Objekten soll eine Bahn geplant und bei der Ausführung die on-line Kollisionserkennung durchgeführt werden. Dazu wird getestet welcher der Beschleunigungs-Ansätze am wirkungsvollsten ist.

1. Das neue Objekt (z. B. Armsegment oder Hindernis) wird in dem CAD-System Romulus entworfen oder liegt schon als Polyeder im Femgen-Format (`Name.fem`) vor. Von den Polyedern wird mit dem Befehl (auf der VAX)

   ```
   prerob.exe Name MaxPoints
   ```

   eine Approximation der konvexen Hülle berechnet (siehe [Beingesser90]). `MaxPoints` gibt dabei die maximale Anzahl von Polyedereckpunkten an, die verwendet werden soll. Es wird eine Datei `Name_hull.fem` erzeugt.

3. Für die Primitiven-Approximation werden mit dem Aufruf

   ```
   do_approx Name
   ```

   (`Name` ohne `.fem`) die Femgen-Dateien `Name.fem` und `Name_hull.fem` off-line eingelesen und eine möglichst günstige Kugel, Zylinder oder Quader berechnet. Zur Veranschaulichung des Ergebnisses wird ein ddd-Fenster mit drei Ansichten der Approximationen erzeugt. Die berechneten Primitive liegen dann als Dateien (`Name_kugel.ddd, Name_zylinder.ddd, Name_quader.ddd`) vor. Zur Information des Benutzers werden das Volumen und die Volumendifferenz der Primitiven angegeben.

4. Nun muß die Beschreibungs-Datei für den Roboter (`*.rob`) und für die Hindernisse (`*.env`) zusammengestellt werden. Auch ist eine Beschreibung der Kinematik des Roboters in einer roboterabhängien Datei nötig (siehe [Beingesser90]).

5. Bei Ansätzen mit hierarchischer Darstellung muß das Gütekriterium in dem Modul `kollerk.c` mit der Präprozessor-Variablen `KRITERIUM` eingestellt werden.

6. Falls dynamische Hierarchien eingesetzt werden sollen: Für den Roboter muß getestet werden welche optimale Baumstrukturen er annehmen kann. Dies geschieht mit der Funktion `kollerk_hier_test` aus dem Modul `kollerk.c`. Sie muß dazu in den Programmablauf bei `do_show_path` eingebunden werden. Die Häufigkeiten der optimalen Baumstrukturen werden am Bildschirm bzw. im Log-File des Simulationssystems ausgegeben.

7. Falls sich eine andere Häufigkeitsverteilung der optimalen Baumstrukturen ergibt, dann muß eine neue Auswahl-Funktion implementiert werden. Für den Puma260 war dieser Schritt unnötig, da die Auswahl-Funktion konstant gewählt werden konnte. Die optimale Baumstruktur wurde einmalig bei der statischen Kollisionserkennung (`kollerk_init_3`) von der gestreckten Armhaltung berechnet. Der Aufruf einer neuen Auswahl-Funktion muß vor der Aktualisierung der Hierarchie (`kollerk_update_hier`) in der dynamischen Kollisionserkennung (`kollerk_algo_7`) eingebaut werden.

8. In dem Modul `kollerk.c` wird dann der gewünschte Beschleunigungs-Ansatz durch die Präprozessor-Variable `ALGO` ausgewählt. Der Abbruch bei einem Abstand Null wird mit der Variable `ABBRUCH` unabhängig davon eingeschaltet. Mit der Variable

`STATISTIK` kann die Anzahl der benutzen Primitiven nach der Bewegung ausgegeben werden.

9. Das Modul `kollerk.c` wird mit dem Befehl `make` in dem seinem Katalog übersetzt.

10. In dem Katalog mit dem Quellcode des Simulationssystems `COD` wird mit dem Befehl `make` das Gesamtsystem zusammengebunden (siehe [Beingesser90]).

11. Nach dem Aufruf des Simulationssystems können Anfangs- und Endpositionen der Arme eingestellt werden (siehe [Hanke90]). Auch eine Darstellung der Szene in den verschiedenen Primitiven ist möglich. Die Bahn wird mit ausgeschaltetem Kollisionstest erzeugt, d. h. Durchdringungen können auftreten. Mit dem Menü kann die Schrittweite der Arme festgelegt werden. Sie entspricht der Schrittweite der Kollisionserkennung. Vor der Ausführung einer Bahn wird gefragt, ob eine Zeitmessung erwünscht ist. Während der Darstellung der Bewegungsbahn auf dem Bildschirm, wird nach jedem Schritt der Abstandsvektor mit dem eingestellten Ansatz berechnet. Der Vektor wird numerisch und graphisch ausgegeben.

12. Möchte man einen anderen Beschleunigungs-Ansatz anwenden, dann wird mit dem Schritt 8 fortgefahren.

Literaturverzeichnis

Zitierte Literatur

[Adolphs] Adolphs P., Horsch T., Olomski J., Höhn G.,
CARO - ein Konzept für die kollisionsvermeidende on-line Bahnplanung für einen 6-achsigen Roboter,
TH Darmstadt, Fachgebiet Regelsystemtheorie und Robotik, interner Bericht.

[Beingesser90] Beingesser U.
Kollisionserkennung und Trajektorienplanung im dreidimensionalen Raum mit Hilfe von konvexen Hüllen,
Diplomarbeit, Universität Karlsruhe, 1990.

[Bobrow89] Bobrow J. E.,
A Direct Minimization Approach for Obtaining the Distance Between Convex Polyhedra,
Int Jour. of Rob. Res., Vol 8, No 3, S 65, 1989.

[Duda72] Duda R., Hart P.,
Pattern Classification and Scene Analysis,
Wiley, New York, 1972.

[Dobkin85] Dobkin D. P. , Kirkpatrick D. G.,
A linear Algorithm for Determining the Separation of Convex Polyhedra,
Journal of Algorithms, Vol 6, S 381, 1985.

[Faverjon89] Faverjon B.,
Hierarchical object models for efficient anti-collision algorithms,
IEEE, 1989.

[Foley83] Foley J. D., vanDam A.,
Fundamentals of Interactive Computer Graphics,
Addison Wesley, 1983.

[Freund88] Freund E., Hoyer H.
Real-time pathfinding in multirobot systems including obstacle avoidance,
Int. Jour. of Rob. Res., Vol 7, No 1, S 42, Feb. 1988.

[Gilbert88] Gilbert E. G., Johnson D. W., Keerthi S.,
A fast procedure for computing the distance between complex objects in three-dimensional space,
IEEE Journal of Rob. and Autom., Vol 4, No 2, April 1988.

[Hanke91] Hanke M.,
Planung kollisionsfreier Trajektorien für einen Zweiarmmanipulator,
Diplomarbeit, Universität Karlsruhe, 1991.

[Khatib86] Khatib O.,
Real-time obstacle avoidance for manipulators and mobile robots,
The Int. Jour. of Rob. Res., Vol 5, No 1, Spring 1986.

[Lumelsky85] Lumelsky V. J.,
On fast computation of distance between line segments,
Inform. Proc. Let., Vol 21, No 2, Aug. 1985, S 55.

[Meyer86] Meyer W.,
Distances between boxes: Application to collision detection and clipping,
Proc. IEEE Int. Conf. Rob. Autom., S 587, 1986.

[Orlowski85] Orlowski M.,
The Computation of the distance Between Polyhedra in 3-Space,
SIAM Conf. on Geometric Modelling and Robotics, Albany, New York, 1985.

[Preparata85] Preparata F. P., Shamos M. I.,
Computational Geometry,
Springer, 1985.

Weitere Literatur

Hier sind noch weitere Literaturreferenzen angegeben, die bei der Recherche angefallen sind. Die mit einem Punkt (•) bezeichneten Referenzen wurden gesichtet. Sie tragen aber nicht direkt zu der Arbeit bei.

Abramowski S.
Collision Avoidance for Nonrigid Object
Lecture Notes in Computer Science, Springer 1988

Abramowski S.
Exakte Algorithmen zur Bewegung gelenkgekoppelter Objekte zwischen festen Hindernissen
Dissertation Universität Karlsruhe, 1989

Abramowski S., Müller H.
Collision Avoidance for Nonrigid Objects
ZOR-Zeitschrift für OR, Physica-Verlag Heidelberg, Vol 32, S 165-186, 1988

Ahuja N., et al.
Inference detection and collision Avoidance among 3-dimensional Objects
Proc. 1st Annual National Conf. AI, S 44, August 1980

Anderson K. R.
A reevaluation of an Efficient Algorithm for Determining the Convex Hull of a finit Planar Set
IPL, Vol 7, S 53, 1987

Andersson l. E., Stewart N. F.
Maximal Distance for Robotic Simulation - The Convex Case
Jour. of Optimization and Applications, Vol 57, No 2, S 215, 1988

Barraco A., Xing D. M.
Determination of Trajectories with Obstacle Avoidance
NATO ASI Series, Vol F29,. 1987, S 361

Barr R. O.
An Efficient Computational Procedure for Generalized Quadratic Programming Problems
SIAM Jour. Contr., Vol 7, S 415, 1969

Barr R. O., Gilbert E. G.
Some Efficient algorithms for a Class of Abstract Optimization Problems
IEEE Trans. Autom. Contr. , Vol 14, S 640, 1969

Benson R. V.
Euclidean Geometry on and Convexity
McGraw Hill, 1966

Borenstein J., Koren Y.
Real-Time Obstacle Avoidance for Fast Mobile Robots
IEEE Trans on SMC, Vol 19, No 5, S 1179, 1989

Boyse J. W.
Interference Detection among Solids ans surfaces
CACM, Vol 22, No 1, S 3, 1979

• Buckley C, E.
A Foundation for the "Flexible Trajectory" Approach to Numeric Path-Planning
NATO ASI Series, Vol 29, 1987, S 389

Buckley C. E., Leifer L. J.
A proximity metric for continous path planning
Prox. Int. Joint conf. AI, S 1096, 1985, Los Angeles

• Buckley S. J.
Fast motion planning for multiple moving robots
IEEE Int. Conf. on Rob. and Aut., 1989

• Cameron S. A., Culley R. K.
Determining the Minimum Translational Distance Between two Convex Polyhedra
Proc. IEEE Int. Conf. Rob. Autom., 1986

Campo M.
On-line Kollisionsvermeidung bei kinematisch redundanten Robotern
Techn. Rept. Jahresbericht, TH Aachen, 1987

Campo M., Ameling W.
Real-time collision avoidance for kinematically redundant manipulators
Preprints of the IFAC-Sympos., Karlsruhe, 1988, S 25

Canny J. F.
Collision Detection for Moving Polyhedra
Proc. European Conf. on AI, 1984

Chaud D. R., Kapur S. S.
An Algorithm for Convex Polytopes
JACM, Vol 17, S 78, 1970

Chazelle B.
Convex Partitions of Polyhedra
SIAM Journal Computation, Vol 13, No 3, August 1984

Chen A. C., Vidysagar M.
Optimal-Control of Robotic Manipulators in the Presence of Obstacles
Jour. of Rob. Systems, Vol 7, No 5, S 721, 1990

Colbaugh R., Glass K. L., Seraji H.
Obstacle Avoidance for Redundant Robots Using Configuration Control
Jour. of Rob. Systems, Vol 6, No 6, S 721, 1989

Comba P. G.
A procedure for detecting intersections of three-dimensional objects
JACM, Vol 15, No 3, S 354, 1986

Culley R. K., Kempf K. G.
A Collision Detection Algorithm Based on Velocity and Distance
IEEE Int. Conf. on Rob. Autom., San Franzisco 1986

• Dai F.
Collision-Free Motion of an Articulated Kinematic Chain in a Dynamic Environment
IEEE Comp. Graphics and Applications, Vol 9, No 1, S 70, 1989

• Delamadrid J. F., Gini M. L.
Path Tracking Through Uncharted Moving Obstacles
IEEE Trans. on SMC, Vol 20, No 6, S 1408, 1990

Diehl R.
Algorithmische Kollisionsprüfung für die Simulation von Bewegungsabläufen
Abschlußbericht, Inst. für Rechneranwendung in Planung und Konst., Karlsruhe, 1987

Dimarzio E. A.
Nonintersecting Random-Walk in the Presence of Nonspherical Obstacles
Physical Rewiew Letters, Vol 64, No 23, S 2791, 1990

• Dobkin D. P. , Kirkpatrick D. G.
Fast Detection of Polyhedral Intersection
Theoretical Comp.. Science, Vol 27, 1983, S 241

• Duffy N. D., Allan D., Herd J. T.
Anticollision Techniques in Multirobot Environments
Electronics and Comm. Engin. Jour., Vol 1, No 5, S 196, Sept./ Okt. 1989

Elzinga D. J., Hearn D. W.
Geometrical solutions for some minimax location problems
Transportation Science, Vol 6, S 379, 1972

Erdmann M., Lozano-Perez T.
On Multiple Moving Objects
IEEE Int. Conf. on Rob. Autom., San Franzisco 1986

• Faverjon B.
Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator
IEEE, 1984

• Faverjon B.
Object level Programming of Industrial Robots
IEEE, 1986

• Faverjon B., Tournassoud P.
A CAD System for Multi-Robot Coordination
Proc. of the NATO Int. advanced Research Workshop on Languages for sensor-based Control in Robotics, 1986, Pisa

• Faverjon B., Tournassoud P.
A local based approach for path planning of manipulators with a high number of degrees of freedom
IEEE, 1987

• Faverjon B., Tournassoud P.
Motion Planning for Manipulators in Complex Environments
IEEE, 1988

Fletcher r. W., Goldenber A. A.
Collision Avoidance for Robot Manipulators - Application to Catia/IBM 7565 Interface
Jour. of Rob. Systems, Vol 5, No 2, S 125, 1988

Fortune S., et al.
Coordinated Motion of two Robot Arms
Proc. IEEE Conf. Rob. Autom., San Franzisco, CA, 1986, S 216

Freund E.
Hierarchical Control of Guided Collision Avoidance for Robots in assembly Automation
Proc. 4th Int. Conf. on Assembly Automation, Japan, Sept. 1983, S 91

Freund E.
Collision Avoidance in Multi-Robot Systems
Proc. 2nd Int. Symposium on Robotic Research, Kyoto, August 1984

• Freund E., Borgolte U.
Ein Algorithmus zur Kollisionserkennung und -vermeidung bei Robotern mit Zylindrischem Arbeitsraum
Robotersyseme, No 6, S 1, 1990

Freund E., Hoyer H.
Collision Avoidance for Industrial Robots with Arbitrary Motion
Int. Jour. of Rob. Systems, Vol 1, No 4, 1984, S 314

• Freund E., Hoyer H.
Ein Verfahren zur automatischen Kollisionsvermeidung für Roboter
Robotersysteme, No 1, 1985, S 67

Freund E., Hoyer H.
On the Online Solution of the Findpath Problem in Multirobot Systems
Rob. Research: 3rd Int. Symp., Ed: Faugeras, et al., Cambridge, 1986, S 253

• Freund E., Hoyer H.
Pathfinding in multi-robot systems: solution and applications
IEEE, 1986

• Freund E., Hoyer H.
Automatische Bahnbestimmung in Echtzeit für Robotersysteme
Robotersysteme, No 3, 1987, S 89

Freund E., Hoyer H.
Roboterforschung: Entwicklung zu neuem Anwendungsbereich
Proc. 4th Europ. Kongreßmesse für techn. Autom. (Komm-Tech), Essen, 1987

Fu L. C., Liu D. Y.
An Efficient Algorithm for Finding a Collision-Free Path Among Polyhedral Obstacles
Jour. of Rob. Systems, Vol 7, No 1, S 129, 1990

• Gilbert E. G., Foo C. P.
Computing the distance between spooth objects in three dimensional space
IEEE, 1989

• Gilbert E. G., Johnson D. W.
Distance Functions and their Application to Robot Path-Planning in the Presents of Obstacles
JRA, Vol 1, S 21, Feb. 1985

• Gilbert E. G., Johnson D. W., Keerthi S. S.,
A Fast Procedure for Computing the Distance Between Complex Objects in Three-dimensional Space
JRA, Vol 4, No 2, April 1988.

• Gilbert E. G., Hong S. M.
A New Algorithm for Detectiong the Collision of Moving Objects
IEEE Int. Conf. Rob. Autom., S 8, 1989

Gouzenes L.
Strategies for Solving Collision-Free Trajectory Problems for Mobile an Manipulating Robots
Int. Journal of Robotic Research, Vol 3, No 4, 1984

Grechanovsky E., Pinsker I.
An Algorithm for Moving a Computer Controlled Manipulator while Avoiding Obstacles
Proc. 8th IJCAI, Karlsruhe, 1983

Grunbaum B.
Convex Polytopes
1967

Habib M. K., Yuta S.
Efficient Online Path Planning Algorithm and Navigation for a Mobile Robot
Int. Jour. of Electronics, Vol 69, No 2, S 187, 1990

Hasegawa T.
Collision Avoidance Using Characterized Description of Free Space
Proc. Int. Conf. on Advanced Robotics, Tokyo, September 1985

• Hasegawa T., Terasaki
Collision Avoidance: Devide and Conquer Approach by Space Characterization and Intermediate Goals
IEEE Trans on SMC, Vol 18, No 3, Mai/Juni 1988

Hayward V.
Fast Collision Detection Scheme by Recursive Decomposition of a Manipulator Workspace
IEEE Int. Conf. Rob. Autom., San Franzisco, 1986

Hörmann A., Hörmann K.
Planung kollisionsfreier Greifoperationen: Analyse der Objektgeometrie
Robotersysteme, Vol 6, S 39-50, 1990

• Hörmann A., Meier W., Schloen J.,
A Control Architecture for an Advanced Fault-Tolerant Robot System
Proc. Int. Conf. of Intelligent Autonomous Systems IAS 2, Vol 2, S 576, Amsterdam 1989.

• Hörmann K.,
Ein Verfahren zur Planung kollisionsfreier Bahnen für Industrieroboter
Dissertation, Universität Karlsruhe, 1987

• Hörmann K., Werling V.
Planung kollisionsfreier Greifoperationen: Kollisionsfreie Bahnplanung für Greifer und Manipulator
Robotersysteme, Vol 6, S 119-125, 1990

• Hörmann K., Werling V.
Ein Verfahren zur Planung von Feinbewegungen
Robotersysteme, Vol 5, S 17-28, 1989

Hopcroft J., Joseph D., Whitesides S.
On the Movement of Robot Arms in 2-Dimensional bounded Regions
SIAM Journal on Computing, Vol 14, S 315, 1985.

• Horsch T., Adolphs P.
Collision Detection Between Robot and Environment - a Hierarchical Approach
Int Conf. on Autom., Rob. and Comp. Vision, Singapur, 1990

Hoyer H.
Verfahren zur automatischen Kollisionsvermeidung von Robotern im koordinierten Betrieb
Dissertation, Universität Hagen, 1984

Hoyer H.
Online Collision Avoidance for Industrial Robots
Proc. 1st IFAC Symp. Robot Control, Barcelona, 1985, S 477

Hsu P., Hauser J., Sastry S.
Dynamic Control of Redundant Manipulators
Journal of Robotic Systems, Vol 6, No 2, S 133, 1989

Johnson d. W., Gilbert W. G
Minimum time Robot Path Planning in the Precence of Obstacles
Proc IEEE Conf. on Decision and Control, S 1748, 1985

Johnson G. H., Gram C.
A Simple Algorithm for Building the 3 dimensional Convex Hull
BIT, Vol 23, S 146, 1983

Kant K., Zucker S.
Toward an Efficient Trajectory Planning: The Path-Velocity Decomposition
Int. Jour. Rob. Res., Vol 5, S 72, No 3, 1986

Kant K., Zucker S.
Planning collision-free trajectory in time-varying environment
IEEE Int. Conf. on Rob. and Aut., 1988, S 1644

Kedem K., Sharir M.
An Efficient Motion Planning Algorithm for a Convex Polygonal Object in 2-Dimensional Polygonal Space
Technical Report No 253, CS Departement Courant Institute, Oktober 1986.

Khatib O., LeMaitre J. F.
Dynamic controll of manipulators in operational space
3rd CISM-IFToMM, S267, Udine, Sept 1978

Kircanski M. Vukobratovic M.
Contribution to Control of Redundant Robotic Manipulators in an Environment with Obstacles
Int. Jour. of Rob. Res., Vol 5, No 4, S 112, 1987

• Klingert A. J.,
Kollisionsvermeidung für Gelenkgekoppelte kreisförmige Objekte in der Ebene
Diplomarbeit Universität Karlsruhe, 1988.

Krogh B. H.
Feedback Obstacle Avoidance Control
Proc. 21th Allerton Conf. University of III, Oktober 1983

• Krüger G.
Informatik I und II, Skriptum zur Vorlesung
WS 1985/86 und SS 1986

Kuntze H. B., Schill W.
Methods for Collision Avoidance in Computer Controlled Industrial Robots
Proc. 12th ISIR, Paris, 1982

Le N. M.
Algorithmen zur exakten Planung kollisionsfreier Bewegung für Objekte
Dissertation, Universität Karlsruhe, 1990

Lee B. H.
Constraints Identification in Time-Varying Obstacle Avoidance for Mechanical Manipulators
IEEE Trans. on SMC, Vol 19, No 1, S 140, 1989

• Lee B. H., Chien Y. P.
Time-Varying Obstacles Avoidance for Robot Manipulators: Approaches and Difficluties
Proc. IEEE Int. Conf. Rob. Autom., 1987, S 1610

• Lee B. H., Lee C. S. G.
Collision-Free Motion Planning of Two Robots
IEEE Trans. on SMC, Vol 17, No 1, Januar/Februar 1987

• Leinen D.
Planung und graphische Visualisierung von kollisionsfreien Greifbewegungen für Industrieroboter
Diplomarbeit Universität Karlsruhe, 1989.

Liu G., Palmer R. J.
Efficient Field Courses Around an Obstacle
Jour. of Agricultural Engineering Research, Vol 44, No 2, S 87, 1989

Lozano-Perez T.
Spatial Planning: A Configuration Space Approach
IEEE Trans. on Comp., Vol c-32, No 2, Februar 1983

Lozano-Perez T.
A Simple Motion Planning Algorithm for General Robot Manipulators
Proc. AAAI, S 626, 1986

Lozano-Perez T., Wesley M. A.
An Algorithm for Planning Collision Free Paths among Polyhedral Obstacles
CACM, Vol 22, No 10, S 560, 1979.

Lunde E., Balchen J. G., Egeland O.
Dynamic Control of Kinematically Redundant Robotic Manipulators
Modelling Identification and Control, Vol 8, No 3, S 159, 1987

Lyusternik L.
Convex Figures and Polyhedra
Dover, 1963

Lumelsky V. J.
Detection of Mutual Interference in Multirobot Systems
IEEE Trans. SMC, 1987

Mayorga R. V., Wong A. K. C, Ma K. S.
An Efficient Local Approach for the Path Generation of Robot Manipulators
Jour. of Robotic Systems, Vol 7, No 1, S 23, 1990

Nelson R. C., Aloimonos J. Y.
Obstacle Avoidance Using Flow Field Divergence
IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol 11, No 10, S 1102, 1989

• O´Donnell P. A., Lozano-Perez T.
Deadlock-free and collision-free coordination of two robot manipulators
IEEE, 1989

Preparata F. P., Hong S. J.
Convex Hulls of finite Sets of Points in Two and Three Dimensions
CACM, Vol 20, S 87-93, Februar 1977

Reif J. H,
Complexity of the Mover´s Problem and Generalizations
IEEE Found of Computer Science 20th Conf., S421, New York 1979

Reif J. H., Sharir
Motion Planning in the Presents of Moving Obstacles
Proc. 20th IEEE Symp. Found. Comp. Sci., S 144, 1985

Rogers D. F.
Procedural Elements for Computer Graphics
McGraw Hill International Edition, 1988

Schwartz J. T.
Finding the minimum distance between two convex polygons
Inform. Proc. Letters, Vol 13, No 4-5, S 168, 1981

Schwartz J. T., Sharir M.
On the Pianos Mover´s Problem III
Int. Journal of Robotics Research, Vol 2, No 3, Herbst 1983

• Sciavicco L., Sicilliano B.
A Solution Algorithm to the Inverse Kinematic Problem for Redundant Manipulators
IEEE Jour. of Rob. and Autom., Vol 4, No 4, 1988

• Shih C. L., Lee T. T., Gruver W. A.
A Unified Approach for Robot Motion Planning with Moving Polyhedra l Obstacles
IEEE Trans. on SMC, Vor 20, No 4, S 903, 1990

Shiller Z., Dubowsky S.
Robot Path Planning with Obstacles, Actuator, Gripper, and Payload Constraints
Int. Jour. of Rob. Res., Vol 8, No 6, S 3, 1989

Smith R. C.
Fast Robot Collision Detection Using Graphics Hardware
SRI Project 7234, Technical Note, Juni 1985

Soetadji T.
A Cube-based Approach to finde-Path for an Autonomous Mobil Robot
Info III, 1986

Sperl R.
Implementierung eines Verfahrens zur Planung kollisionsfreier Bahnen für 6-achsige Industrieroboter
Diplomarbeit, Universität Karlsruhe, 1987

Sutner K., Maass W.
Motion Planning Among Time-Dependent Obstacles
Acta Informatica, Vol 26, No 1/2, S 93, 1988

Takeuchi T, Nagai Y., Enomoto N.
Fuzzy Control of an Mobile Robot for Obstacle Avoidance
Information Sciences, Vol 45, No 2, S 231, 1988

Tornero J., Hamlin J., Kelley R. B.
Spherical object representation and fast distance computation for robotic applications
Proc. IEEE Int. Conf. on Rob. and Aut., 1991

Tournassoud P.
A Strategy of Obstacle Avoidance and Application to Multirobot Systems
Proc. IEEE Conf. Rob. Autom., San Franzisco, CA, 1986, S 1224

Tychonievich L., et al.
A Maneuvering-Board Approach to Path Planning with Moving Obstacles
Proc. Int. Joint Conf. AI, 1989, S 1017

Uchiyama M.
Control of Robot Arms
JSME Int. Jour. Series III-Vibration Control Engineering for Industry, Vol 32, No 1, S 1, 1989

Udupa S.
Collision Detection and Avoidance in Computer Controlled Manipulation
Proc. 5th Int. Joint Conf. AI, S 737, 1977

Volpe R., Khosla P.
Manipulator Control with Superquadric Artificial Potential Functions
IEEE Trans. on SMC, Vol 20, No 6, Nov./Dez. 1990

Wang C. C., Li C.
A General Robot Path Verification Simulator with Motion Time Profile
Proc. 9th ICPR, S 186, 1989

Wolfe P.
Finding the Nearest Point in a Polytope
Math. Programming, Vol 11, S128, 1987

Wolfson E., Schwartz E. L.
Computing Minimal Distances on Polyhedral Surfaces
IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol 11, No 9, S 1001, 1989

Wu Y. F., Schlag M. D. F., Widmayer P., Wong C.K.
Rectilinear Shortest Paths and Minimum Spanning-Trees in the Presence of Rectilinear Obstacles
IEEE Trans. on Comp., Vol 36, No 3, S 321, 1987

Yao A. C.
On constructing minimum spanning trees in k-dim. space and related problems
SIAM Journal of Comp., Vol 11, No 4, S 721, 1982

Zaharakis s. C., Guey A.
Time Optimal Robot Navigation via the Slack Set Method
IEEE Trans. on SMC, Vol 20, No 6, S 1396, 1990

Zuo L. C., Huang Y. Y., Hall E. L.
Region Filling Operations with Random Obstacle Avoidance for Mobile Robots
Jour. of Rob. Systems, Vol 5, No 2, S 87, 1988

Glossar

Die Beschreibungen der Begriffe sollen keine allgemeine Definition darstellen, sondern deren Verwendung in dieser Arbeit verdeutlichen. Einige Begriffe werden im Text formal oder ausführlicher beschrieben.

Abstandspunkte *(points of minimal distance)* Die Abstandspunkte sind Punkte mit minimalem Abstand auf zwei Objekten. Die Punkte sind im Allgemeinen nicht eindeutig, d. h. für zwei gegebene Objekte kann es mehrere Paare von Abstandspunkten geben. Die Abstandspunkte sind das Ergebnis der Abstandsberechnung. Aus ihnen wird der Abstandsvektor berechnet. (Kapitel 1.3)

Abstandsvektor *(collision vector)* Der Abstandsvektor ist der Differenzvektor zweier Abstandspunkte. Für konvexe Objekte ist er im Gegensatz zu den Abstandspunkten eindeutig, da er nur die Richtung und den kleinsten Abstand, nicht aber die Lage im Raum angibt. Bei der Kollisionserkennung bietet die Berechnung des Abstandsvektors gegenüber einem reinen Test auf Kollision den Vorteil, daß die darauf folgende Kollisionsvermeidung wirkungsvoller gestaltet werden kann. Dies wird durch die zusätzliche Abstands- und Richtungsinformation erreicht. Im Gegensatz zum Kollisionsvektor gibt der Abstandsvektor keine Durchdringungen an. (Kapitel 1.3)

Abstandsvektor, kürzeste *(shortest collision vector)* Für die Kollisionserkennung ist vor allem der kürzeste Abstandsvektor von Bedeutung. Aber es wird nicht immer nur der kürzeste betrachtet. Um in einer Szene den kürzesten Abstandsvektor zu finden, können alle möglichen berechnet und der kürzeste davon ausgegeben werden. In dieser Arbeit kann man drei kürzeste Abstandsvektoren unterscheiden: zwischen zwei Objekten, zwischen zwei Kollisionsklassen und innerhalb der gesamten Szene. (Der Abstandsvektor zwischen zwei Objekten ist per Definition immer der kürzeste.) Welcher der drei gemeint ist, geht aus dem jeweiligen Kontext hervor. (Kapitel 2.1)

Approximation *(approximation)* Eine Approximation ist eine Annäherung eines Objekts durch ein anderes, meist einfacheres Objekt. Die Abstandsberechnung kann mit den Approximationen meistens schneller durchgeführt werden als mit den Objekten selbst. Der resultierende Abstand ist dann aber nur eine Abschätzung des realen Abstands. (Kapitel 2.3 und 3.1)

Approximation, konservative *(conservative approximation)* Die konservative Approximation eines Objektes enthält das Volumen des angenäherten Objektes vollständig. Bei einer Abstandsberechnung mit konservativen Approximationen ist eine *untere* Abschätzung des realen Abstands möglich. D. h. der Abstand zwischen den konservativen Approximationen ist immer *kleiner* als der zwischen den ursprünglichen Objekten. (Kapitel 3.1)

Approximation, Primitiven- *(primitive approximation)* Die Primitiven-Approximation ist die Approximation eines Objekts durch ein oder mehrere Primitive. Als Primitive kann z. B. gewählt werden: Quader, Zylinder, Kugel, Bounding-Box. Damit wird das Objekt in unterschiedlichen Genauigkeiten dargestellt. Für die Kollisionserkennung beschleunigen die verschiedenen Genauigkeitsstufen die Abstandsberechnung erheblich. (Kapitel 3)

Approximations-Fehler *(approximation error)* Der Approximations-Fehler ist der Darstellungsfehler, der durch die Verwendung einer Annäherung an ein Objekt entsteht.

Hier wird als Approximations-Fehler die Volumendifferenz zwischen dem Original-Objekt und seiner Approximation benutzt. (Kapitel 3.2)

Bounding-Box *(bounding box)* Die Bounding-Box ist ein achsenparalleler Quader. Als Approximation eines Objektes gibt die Bounding-Box die extreme Ausdehnung des Objekts in jede Koordinatenrichtung an. Bei der Kollisionserkennung zeigt sich die Bounding-Box als wirkungsvollste Primitiven-Approximation zur Abstandsberechnung zwischen den realen Objekte. (Kapitel 3.1)

Fortschreibung, Abstands- *(distance update)* Bei der Abstands-Fortschreibung wird statt den Abstandsvektor zwischen zwei Objekten neu zu berechnen ein gegebener Vektor nach einer Bewegung eines Objekts aktualisiert. Diese Aktualisierung kann durch die größte zuückgelegte Distanz des bewegten Objektes geschehen. Überschreitet der fortgeschriebene Abstand den Wert Null, dann muß der Abstandsvektor neu berechnet werden. (Kapitel 2.3, [Faverjon89])

Fortschreibung, globale Abstands- *(global distance update)* Bei dieser Abstands-Fortschreibung wird der kürzeste Abstandsvektor der *gesamten* Szene fortgeschrieben. Es handelt sich nur um *einen* einzelnen Vektor. (Kapitel 2.3)

Fortschreibung, lokale Abstands- *(local distance update)* Bei dieser Abstands-Fortschreibung wird nicht nur der kürzeste Abstandsvektor der gesamten Szene, sondern es werden *alle* Abstandsvektoren zwischen *jedem* einzelnen Objektpaaren aktualisiert. Es handelt sich um mehrere Vektoren. (Kapitel 2.3)

Hash-Funktion *(hash-function)* "Um Datensätze eines Datenbestandes, die durch Schlüssel identifiziert sind, mit kleinstmöglichem Aufwand wiederzufinden, kann man die Datensätze so speichern, daß sich aus dem jeweiligen Schlüsselwort die Adresse des zugehörigen Satzes errechnet. Eine Hash-Funktion ist formal eine Abbildung $H: S \rightarrow A$ einer Menge von Schlüsselwörtern S in eine Menge A von Adressen, die einen zusammenhängenden Adreßraum bilden."* Bei den dynamischen Hierarchien ist die Hash-Funktion eine der denkbaren Implementierungen zur Auswahl der optimalen Baumstruktur für eine bestimmte Konfiguration. (Kapitel 4.5)

Hierarchie *(hierarchy)* "Eine Hierarchie ist ein sich nach Art eines Stammbaumes verzweigendes System von Mengen/Teilmengen. Formal heißt ein System $\boldsymbol{H} = (A, B, ...)$ von Teilmengen $A, B, ...$, einer Objektmenge $\boldsymbol{O}$ eine Hierarchie, wenn gilt: (1) $\boldsymbol{O}$ gehört zu $\boldsymbol{H}$; (2) Gehören A, B zu $\boldsymbol{H}$, so sind A und B entweder disjunkt oder es ist eines im anderen enthalten; (3) Alle einelementigen Teilmengen gehören zu $\boldsymbol{H}$."* (Kapitel 2.3 und 4.1)

Hierarchie, dynamische *(dynamic hierarchy)* Bei den dynamischen Hierarchien verändern sich die Inhalte an den Knoten oder die Baumstruktur während des Betrachtungszeitraums. Hier werden die dynamischen Hierarchien als Kompositions-Hierarchien von dynamischen Kollisionsklassen wie z. B. Roboterarmen eingesetzt. Nach jedem Bewegungsschritt des Arms muß die optimale Baumstruktur mit einer Auswahl-Funktion gefunden und die Kompositionen an den Hierarchie-Knoten aktualisiert werden. (Kapitel 4.6)

Hierarchie, Kompositions- *(composition hierarchy)* Die Kompositions-Hierarchie ist eine hierarchische Darstellung einer Menge von Objekten. Jeder Knoten ist die Komposition aller seiner Unter-Hierarchien. An den Blättern befinden sich die ursprünglichen Objekte.

---

* aus "Lexikon der Informatik und Datenverarbeitung", H.-J. Schneider (ed), Oldenburg Verlag 1991, 3. Auflage.

Werden die Kompositionen an den Knoten durch Approximationen dargestellt, dann bewirkt die Kompositions-Hierarchie aus der Sicht der Kollisionserkennung eine starke Reduzierung der Objektanzahl. Dabei wird die zusammengefaßte Objektmenge in unterschiedlichen Genauigkeitsstufen dargestellt. Bei der Kollisionserkennung mit mehreren Armen bieten sich die Kompositions-Hierarchien als Darstellung der Kollisionsklassen an. (Kapitel 4.1)

Hierarchie, optimale Kompositions- *(optimal composition hierarchy)* Eine Kompositions-Hierarchie ist dann optimal, wenn sie auf jeder Ebene den Wert eines vorgegebenen Gütekriteriums minimiert. Durch das Gütekriterium wird die Baumstruktur der optimalen Hierarchie gesteuert. Es bestimmt welche Objekte der Hierarchie auf welcher Ebene zusammengefaßt werden. Durch ein günstig gewähltes Gütekriterium kann die Kollisionserkennung mit optimalen Kompositions-Hierarchien weiter beschleunigt werden. Aus der Sicht der Kollisionserkennung wird dann nicht nur die Anzahl der Objekte reduziert, sondern auch die uninteressanten Objekte aus der Betrachtung früh ausgeschieden. (Kapitel 4.1)

Kollisionserkennung *(collision detection)* Die Kollisionserkennung erfolgt hier nur im kartesischen Raum. Sie kann entweder nur ein einfacher Test auf Kollision sein oder eine Abstandsberechnung beinhalten. Der Test berechnet, ob eine Berührung oder Durchdringung zweier Objekte vorliegt oder nicht. Die Abstandsberechnung erlaubt eine mögliche Kollision schon im Voraus zu detektieren. (Kapitel 2.1)

Kollisionsklasse *(collision class)* Die Kollisionsklasse ist eine Zusammenfassung von Objekten, zwischen denen keine Kollisionserkennung berechnet wird. Die Klassen werden so gewählt, daß sie nur Objekte enthalten zwischen denen entweder keine Kollisionen auftreten oder die Kollisionen durch andere (einfachere) Mechanismen vermieden werden. Durch die Einführung von Kollisionsklassen wird das bisherige Weltmodell der Kollisionserkennung mit einem Roboterarm für mehrere Arme modifiziert. Bei mehreren Armen bildet ein Arm für einen anderen Arm auch eine Hindernis, sodaß die Zweiteilung des Weltmodells nicht aufrecht erhalten werden kann. Es wird stattdessen zwischen zwei zu Kollisionsklassen zusammengefaßten Armen der Abstandsvektor berechnet. (Kapitel 2.4)

Kollision *(collision)* Eine Kollision tritt dann auf, wenn sich zwei oder mehr Objekte berühren oder durchdringen. (Kapitel 1.2)

Kollisionsvektor *(collision vector)* Der Kollisionsvektor gibt zwischen zwei Objekten die kürzeste Translation an, den eines der Objekte zurücklegen muß, um das andere zu berühren. Mit einem Kollisionsvektor kann daher auch der Grad einer Durchdringung angegeben werden. Er bezeichnet dann einen "negativen" Abstand. Für die Kollisionserkennung wird hier nur der Abstandsvektor verwendet. (Kapitel 1.3)

Kollisionsvermeidung *(collision avoidance)* Der Begriff der Kollisionsvermeidung soll nicht allgemein definiert werden. Er wird nur im Zusammenhang der hier verwendeten Art von Kollisionserkennung benutzt. Dabei soll die Kollisionsvermeidung die Berechnung einer Ausweichbewegung von einer vorgegeben Bewegungsbahn bezeichnen, nachdem durch die Kollisionserkennung eine mögliche Kollision auf der Bahn erkannt wurde. (Kapitel 1.1)

Komposition *(composition)* Die Komposition ist eine Zusammenfassung mehrerer Objekte zu logisch einem Objekt. Wie diese Zusammenfassung dargestellt wird, ist offen gelassen. Zum Beispiel kann eine Komposition durch einen Polyeder oder ein Primitiv repräsentiert werden. Durch Kompositionen wird aus der Sicht der Kollisionserkennung die Anzahl der dargestellten Objekte stark reduziert. Für die Kollisionserkennung bedeutet das eine Beschleunigung der Abstandsberechnungen. (Kapitel 4.1)

off-line *(indirekte Arbeitsweise)*, "Arbeitsart mit Rechensystemen, wobei die Daten nicht direkt in ein Rechensystem ... eingegeben werden."* Hier soll unter off-line Berechnung der Prozeß verstanden werden, der *vor* dem Prozeß der Kollisionserkennung für eine Bewegung abläuft. (Kapitel 2.2)

on-line *(direkte Arbeitsweise)* "Arbeitsart mit Rechensystemen, wobei die Daten direkt über eine Datenleitung eingegeben werden"*. Hier soll unter on-line Berechnung der Prozeß verstanden werden, der (quasi-) parallel mit dem Prozeß der Kollisionserkennung für eine Bewegung abläuft. (Kapitel 2.2)

Polyeder *(polytope/polyhedron)* Der Polyeder ist ein Volumen, das durch ebene Flächen begrenzt ist. Rechner-intern können die Polyeder durch verzeigerte Listen von Eckpunkten, Kanten und Flächen beschrieben werden. (Kapitel 3.1)

Polyeder, konvexer *(convex polytope/polyhedron)* Bei dem konvexen Polyeder liegt jede Verbindung von zwei innen liegenden Punkten auch im Innern des Polyeders. Rechner-intern können die konvexen Polyeder durch verzeigerte Listen von Eckpunkten, Kanten und Flächen oder durch eine Menge von Hyperebenen beschrieben werden. Jedes durch einen Polyeder dargestellte Objekt kann auch durch eine Menge von konvexen Polyedern dargestellt werden. Die Konvexität der Polyeder erleichtert dabei die Abstandsberechnung. Hier dienen die konvexen Polyeder als Grundmodellierung, also als genaueste Darstellung der realen Objekte. (Kapitel 3.1)

Primitiv *(primitive)* Ein Primitiv ist ein einfach beschreibbares Volumen. Beispiele sind Kugel, Zylinder, Quader oder Bounding-Box. Hier dienen Primitive zur vereinfachten Darstellung realer Objekte. Sie beschleunigen als Approximation erheblich die Abstandsberechnung zwischen den Objekten. (Kapitel 3.1)

Schrittweite Die Schrittweite gibt an, nach welcher zurückgelegten Distanz des Roboterarms spätestens die Kollisionserkennung durchgeführt werden soll. Die Distanz ist hier als Norm in dem von den Gelenkwinkeln aufgespannten Konfigurationsraum angegeben. Sie ist ein Maß für das Ausmaß einer Bewegung des Roboterarms. (Kapitel 2.1 und 5.4)

---

* aus "Lexikon der Informatik und Datenverarbeitung", H.-J. Schneider (ed), Oldenburg Verlag 1991, 3. Auflage.