In: 13. Fachgespräch Autonome Mobile Systeme (AMS'97), Stuttgart, 6.+7. Oktober 1997, Springer-Verlag, Reihe "Informatik Aktuell"

Schnelle Kollisionserkennung durch parallele Abstandsberechnung*

Dominik HENRICH, Stefan GONTERMANN und Heinz WÖRN
Institut für Prozeßrechentechnik und Robotik
Universität Karlsruhe, D-76128 Karlsruhe
E-mail: [dHenrich, Woern]@ira.uka.de

1. Einleitung

In vielen Anwendungsbereichen der Robotik, wie z.B. der Montage und Demontage, dem Recycling, der Chirurgie oder im Servicebereich, ist die Planung kollisionsfreier Bewegungen eine grundlegende Aufgabe. Allerdings können insbesondere bei Industrierobotern auf Grund des hohen Rechenaufwands die bisherigen Ansätze zur kollisionsfreien Bahnplanung nicht in der Praxis eingesetzt werden. Es zeigt sich, daß ein Großteil des Gesamtaufwands für die Kollisionserkennung benötigt wird. So wird zum Beispiel in [Metivier90] von einem Anteil bis zu 80% ausgegangen. Ein erfolgversprechender Ansatz für eine Beschleunigung der Bahnplanung ist daher zuerst in diesem Bereich zu suchen. In diesem Artikel geben wir eine schnelle Kollisionserkennung basierend auf einer parallelisierten Abstandsberechnung an.

Die einfachste Form der Kollisionserkennung ist der *Kollisionstest*, der nur ermittelt, ob die Objekte einer Umwelt, z.B. ein Roboter und verschiedene Hindernisse, miteinander kollidieren oder nicht. Ein Beispiel für ein solches Verfahren findet man in [Dobkin83]. Die etwas aufwendigere Alternative zum Kollisionstest ist die Berechnung von Abstandsvektoren. Der *Abstandsvektor* ist der Differenzvektor zweier Abstandspunkte, das heißt zweier Punkte mit minimalem Abstand auf den Objekten. Zwar kann es mehrere Paare von Abstandspunkten geben, aber der Abstandsvektor ist immer eindeutig. Eine Kollisionserkennung auf der Basis von Abstandberechnungen liefert also zusätzlich zu der Information, ob eine Kollision vorliegt, auch noch den Abstand und die Richtung zum nächsten Objekt (z.B. einem Hindernis).

Für die Anwendungen, in welche die Kollisionserkennung integriert wird, ergeben sich damit wesentlich mehr Möglichkeiten. Zum Beispiel kann eine Bewegungsplanung aus dem Abstand zu den Hindernissen eine sinnvolle Bewegungsgeschwindigkeit des Roboters ableiten, der sich in der Nähe von Objekten vorsichtiger bewegen muß als in einem Gebiet ohne Objekte. Ein weiterer Vorteil der Abstandsinformation ist die Möglichkeit Roboterbewegungen hierarchisch zu Planen [Sandmann97]. Allerdings gibt es zwischen den Verfahren zur Abstandsberechnung Unterschiede, was die Qualität der Berechnung und den dafür notwendigen Zeitaufwand betrifft. Dies hängt in erster Linie mit der zugrunde liegenden geometrischen Objektrepräsentation zusammen.

* Diese Arbeit wurde an dem Institut für Prozeßrechentechnik und Robotik (Leitung: o. Prof. Dr. U. Rembold, o. Prof. Dr. H. Wörn und Prof. Dr. R. Dillmann) durchgeführt. Weitere Informationen gibt es auf den Web-Seiten der PaRo-Gruppe (Parallele Robotik) unter http://wwwipr.ira.uka.de/~paro/.

Eine Form der Objektrepräsentation sind einfache Körper (*Primitive*), die sich durch eine konstante Anzahl skalarer Parameter beschreiben lassen, wie zum Beispiel Kugeln, Quader oder Zylinder. Zwischen Primitive lassen sich u.U. sehr schnelle Abstandsberechnungen durchführen. In der Literatur sind Ansätze zur Kollisionserkennung basierend auf Kugeln [Basta88, Lee87], Quadern [Meyer86] und Liniensegmenten [Lumelsky85] bekannt. Allerdings können durch Primitive die realen Objekte meist nur grob dargestellt werden. Desshalb ist auch nur eine untere Schranke für den realen Abstand ermittelbar. Der Approximationsfehler der Primitive überträgt sich somit auf die Lösungsqualität der Abstandsberechnung.

Wesentlich genauere Ergebnisse sind durch die Umweltmodellierung basierend auf konvexen Polyeder möglich, da sie sich durch die frei wählbare Anzahl von Flächen gut an die realen Objekte anpassen lassen.[1] Zudem ist bei Polyeder die direkte Verwendung von Geometriedaten aus CAD- oder Robotersimulations-Systemen möglich. Zwar lassen sich die Polyeder wiederum durch mehrere Primitive, wie z.B. Kugeln [Quinlan94, delPobil96] annähern, aber diese iterativen Approximationen sind zeitaufwendig [Sato96]. Effizienter sind die direkte Abstandsberechnungen zwischen konvexen Polyedern [Bobrow89, Gilbert88, Ong97].

Der Artikel ist wie folgt gegliedert: Nach der Einleitung wird nun in Abschnitt 2 ein schnelles sequentielles Verfahren zur Abstandsberechnung aufgezeigt. In Abschnitt 3 wird dann die Parallelisierung des Verfahrens vorgestellt. Abschließend wird in Abschnitt 4 die Leistungsfähigkeit der parallelen Kollisionserkennung mittels verschiedener Experimente analysiert.

2. Sequentielle Abstandsberechnung

In diesem Abschnitt wird das schnellste bekannte Verfahren zur Abstandsberechnung vorgestellt und erweitert. Es bildet die Grundlage für den nachfolgenden parallelen Algorithmus. Die Auswahl eines vielleicht besser parallelisierbaren aber langsameren Algorithmus ist nicht sinnvoll, weil dann die Parallelisierung erst den Rückstand zu dem schnellsten sequentiellen Algorithmus aufholen muß, bevor sie eine Beschleunigung erzielen kann.

Zusammen mit den Überlegungen aus der Einleitung ist der Algorithmus zur Abstandsberechnung zwischen konvexen Polyedern [Gilbert88] und der Approximation durch statische Hierarchien [Faverjon89] bzw. dynamische Hierarchien [Henrich92] am erfolgversprechensten. Es lassen sich damit sehr kurze Berechnungszeiten erzielen. Zum Verständnis wird die Grundidee nun kurz vorgestellt.

Um die Anzahl der zeitaufwendigen Abstandsberechnungen zwischen Polyedern zu reduzieren, kann man off-line eine Approximations-Hierarchie aufbauen. Dazu werden die Objekte einer Szene paarweise zusammengefaßt und durch neue Objekte konservativ approximiert. Die neu entstandenen Objekte werden ebenfalls paarweise zusammengefaßt. Dieses Vorgehen wird solange fortgesetzt, bis im letzten Schritt ein

[1] Die Einschränkung auf konvexe Polyeder ist hier unbedeutend, da sich jeder (nichtkonvexe) Polyeder durch eine Vereinigung von konvexen Polyeder darstellen läßt.

Objekt entsteht, das alle einzelnen Objekte enthält. Die Baumstruktur in Abb. 1 verdeutlicht den Aufbau der Hierarchie für den Roboterarm Puma260.

Unbewegte Objekte werden in einer *statischen* Hierarchie zusammengefaßt. Eine daraus resultierende optimale Hierarchie kann einmal zu Beginn aufgebaut und muß im Verlauf der Kollisionserkennung nicht mehr verändert werden. Damit ist eine vollständige Berechnung der statischen Hierarchie off-line möglich. Dagegen muß eine aus bewegten Objekten aufgebaute *dynamische* Hierarchie nach jeder Bewegung eines enthaltenen Objektes aktualisiert werden, da sich mit der Lage der zusammengefaßten Objekte auch ihre Approximation ändern kann. (Für weitere Details siehe [Henrich91]). Mit achsen-parallelen Bounding-boxes als Primitive ist die geometrische Aktualisierung hinreichend schnell durchführbar.

Durch die Einführung der (statischen bzw. dynamischen) Hierarchien wird die Abstandsberechnung zu einer Bestensuche in einem Baum. Dabei enthält jeder Knoten des Suchbaums ein Objekt aus der statischen Hindernishierarchie und ein Objekt aus der dynamischen Roboterhierarchie. Einen Knoten zu expandieren bedeutet seine beiden Objekte jeweils in ihre Teilobjekte zu zerlegen und damit neue Objektpaare zu bilden. Für jedes dieser Paare ist eine Abstandsberechnung durchzuführen. Die Abb. 2 zeigt das Vorgehen für eine Szene mit einem Roboter aus zwei Bauteilen und einer Umwelt mit vier Hindernissen.

Die Bestensuche beginnt mit der Expansion des obersten Knotens des Suchbaums (*Wurzel*). Nachdem die Abstandsberechnung für alle durch die Expansion entstandenen Nachfolger durchgeführt wurde, werden sie aufsteigend sortiert in eine Prioritätenliste (*Open-Liste*) eingefügt. Bis die Suche terminiert, wird anschließend immer der "beste" Knoten, d.h. der mit dem kleinsten berechneten Abstand, aus der Open-Liste entnommen, expandiert und die neuen Nachfolgerknoten entsprechend der Abstände ihrer Objekte in die Open-Liste einsortiert.

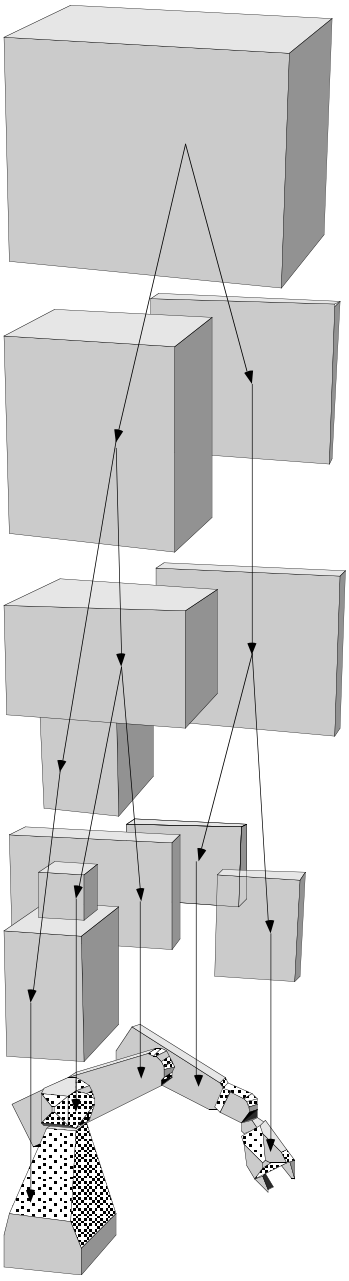

Abb. 1: Hierarchische Robotermodellierung durch Bounding-boxes basierend auf konvexen Polyedern

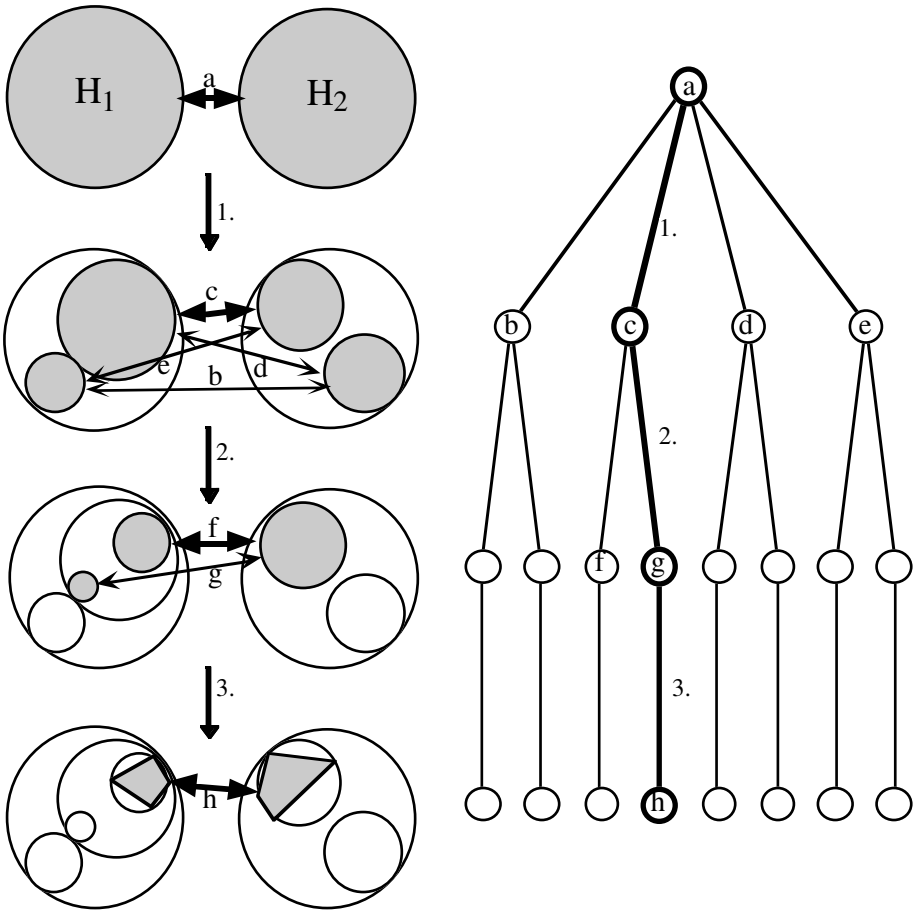

Abb. 2: Links: Suche nach dem kürzesten Abstand zwischen Objekten der Hindernishierarchie H_1 und der Roboterhierarchie H_2. Rechts: Der dazugehörende vollständige Suchbaum mit entsprechenden Objektpaaren a, …, h als Knoten. Fette Linien: Expansion der Knoten

Die Terminierung der Bestensuche hängt davon ab, ob man eine Abstandsberechnung, einen (booleschen) Kollisionstest oder eine abgeschwächte Abstandsberechnung durchführen will. Bei der *abgeschwächten* Abstandsberechnung wird nach dem genauesten Abstand gesucht, jedoch bei größtmöglicher Vermeidung der zeitaufwendigen Abstandsberechnungen zwischen Polyedern. Im einzelnen ergeben sich folgende Terminierungsbedingungen:

- **Terminierung der Abstandsberechnung:** Bei einer Kollision zweier Polyeder, ansonsten wenn der kleinste Abstand zweier Polyeder kleiner ist als der Abstand der Objekte des ersten Knotens in der Open-Liste (keine Kollision).
- **Terminierung des Kollisionstests:** Bei einer Kollision zweier Polyeder oder wenn der Abstand der Objekte des ersten Knotens in der Open-Liste größer Null ist.

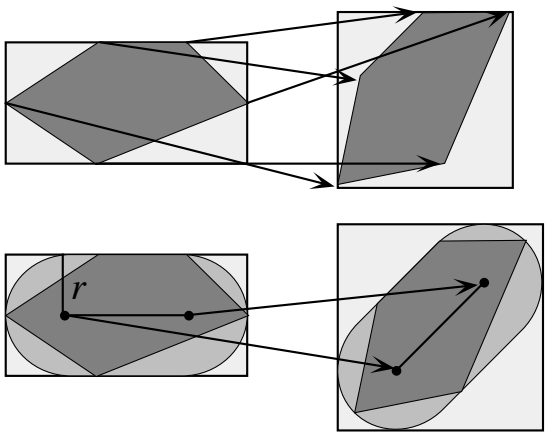

Abb. 3: Aktualisierung der Bounding-box eines Polyeders, dessen Position und Orientierung, z.B. nach einer Roboterbewegung, verändert hat. Oben: Reine Bounding-box-Approximation. Unten: Zusätzliche Verwendung von r-Zylinder

- **Terminierung der abgeschwächten Abstandsberechnung:** Bei einer Kollision zweier Polyeder ansonsten wenn der Abstand der Objekte des ersten Knotens in der Open-Liste größer Null ist und als nächstes der Abstand zweier Polyeder berechnet werden müßte (keine Kollision).

Nach der Bewegung eines Objektes in der dynamischen Hierarchie kann sich die Lage der Polyeder verändert haben. Um die dynamische Hierarchie aktualisieren zu können, müssen zuvor alle bewegten Objekte in ihre neue Position transformiert werden. Die Transformation von Polyeder kann je nach Anzahl der Polyedereckpunkte sehr rechenintensiv sein, da für jeden Eckpunkt eine Matrix-Multiplikation durchzuführen ist (siehe Abb. 3 oben).

Um den Aufwand für die Transformationen zu verringern, werden hier in den dynamischen Hierarchien alle Roboterobjekte durch *r*-Zylinder off-line approximiert. Ein *r-Zylinder* ist dabei die sphärische Erweiterung eines Geradenstücks im Raum um den Radius *r*. Somit kann für *r*-Zylinder die Abstandsberechnung zwischen Geradenstücke aus [Lumelsky85] verwendet werden. Bei einer Neuberechnung der Bounding-boxes werden zunächst nur die *r*-Zylinder in die neue Lage transformiert, so daß sich die Zahl der Punkte, für die eine Matrix-Multiplikation durchgeführt werden muß, auf zwei pro Objekt reduziert. Die *r*-Zylinder werden dann in einem zweiten Schritt durch Boundingboxes approximiert (siehe Abb. 3 unten). Eine Transformation der Polyeder ist damit nur notwendig, wenn zwischen diesen eine Abstandsberechnung durchgeführt werden muß, d.h. wenn die entsprechenden r-Zylinder kollidieren.

Durch dieses Vorgehen reduziert sich die Zahl der Matrix-Multiplikationen und der Aufwand für die Neuberechnung der Bounding-boxes gegenüber der reinen Boundingbox-Hierarchie erheblich. Allerdings wird aufgrund des zusätzlichen Primitivs der Approximationsfehler der Bounding-boxes in der Summe größer. Wegen des größeren Approximationsfehler muß dann häufiger auf die wesentlich aufwendigere Abstandsberechnung zwischen konvexen Polyedern zurückgegriffen werden. Jedoch sind die Einsparungen bei der Transformation größer als der Mehraufwand bei der Abstands-

berechnung. Insgesamt wird eine Beschleunigung der Kollisionserkennung um ca. 30% erreicht (siehe [Katz96]). Außerdem zeigen die Ausführungen in nächsten Abschnitt, daß dieser Ansatz eine Verschiebung von Arbeit aus einem Bereich, der nicht parallelisiert werden kann, in einen Bereich, der parallelisiert werden kann, bewirkt. Dadurch ist bei einer parallelen Berechnung mit einer weiteren Verbesserung zu rechnen.

3. Parallele Abstandsberechnung

Eine Analyse der sequentiellen Kollisionserkennung aus dem vorigen Abschnitt zeigt, daß eine Parallelisierung nur im Bereich der Bestensuche sinnvoll möglich ist [Gontermann97]. Der Aufbau der dynamischen Hierarchie läßt sich auf Grund der überwiegend iterativen Berechnungen kaum parallelisieren und alle anderen aufwendigen Berechnungen, wie z.B. der Aufbau der statischen Hierarchie, werden off-line durchgeführt. Natürlich ließen sich auch elementare Operationen (z.B. Matrix-Multiplikationen) parallelisieren, allerdings ist dieser Ansatz wenig erfolgversprechend, da eine Implementierung auf industriell interessanten nachrichtengekoppelten Parallelrechnern mit einem zu hohem Kommunikationsaufwand verbunden wäre.

Für die Parallelisierung der sequentiellen Abstandsberechnung wird die Bestensuche um drei Mechanismen erweitert. Dazu gehören die lokale Open-Listen, eine geeignete Kommunikationsstrategie und ein Verfahren zur Erkennung der Terminierung. Außerdem erhält ein ausgezeichneter Prozessor (*Master*) den Wurzelknoten und prüft die Terminierung der restlichen Prozessoren (*Slaves*).

Zur Zwischenspeicherung der schon bekannten Knoten verwaltet jeder Prozessor seine eigene lokale Open-Liste[2]. Jeder Prozessor expandiert dann asynchron den besten Knoten seiner Open-Liste und sortiert die Nachfolgerknoten in diese wieder ein. Damit stellt sich die Frage nach einem Verfahren, mit dem die Knoten am besten auf die einzelnen Open-Listen verteilt bzw. untereinander ausgetauscht werden können.

Für den Knotenaustausch wird hier die *Zufallskommunikation* aus [Kumar94] verwendet. Hierbei sendet jeder Prozessor in regelmäßigen Abständen einige seiner besten Knoten an einen zufällig ausgewählten anderen Prozessor. Allerdings wird das Verfahren in zwei Punkten modifiziert. Zum einen verteilen die Prozessoren ihre besten Knoten nicht auf andere Prozessoren, sondern jeder Prozessor fordert bei Bedarf Knoten von einem zufällig ausgewählten anderen Prozessor an (*Random polling*). Dies hat den Vorteil, daß ein Prozessor nicht von einem zufällig ankommenden Knoten "überrascht" werden kann, was für die Feststellung der Terminierung von entscheidender Bedeutung ist. Zum anderen lassen sich die Lastverteilung und damit die Laufzeit der parallelen Kollisionserkennung maßgeblich verbessern, wenn man den Master-Prozessor höher gewichtet als die Slave-Prozessoren und er dadurch häufiger für eine Knotenanforderung ausgewählt wird. Das liegt daran, daß der Master-Prozessor den Wurzelknoten expandiert und daher als erster einer Knotenanforderung nachkommen kann.

---

2 Eine zentral verwaltete, globale Open-Liste führt zwangsläufig zu einem Engpaß der Berechnungen, welcher eine obere Schranke für die maximale Anzahl an Prozessoren darstellt, die noch zu einer Beschleunigung führen [Huang90].

Für die parallele Abstandsberechnung durch Bestensuche muß die sequentielle Terminierungsbedingung erweitert werden, da hier jeder Prozessor seine eigene lokale Open-Liste besitzt. Damit ergeben sich im einzelnen folgende drei Abbruchbedingungen:

(1) Bei **Abstandsberechnung**: Wenn der kleinste Abstand zweier konvexer Polyeder kleiner ist als der Abstand der Objekte des besten Knotens in der Open-Liste.
Bei **Kollisionstests**: Wenn der Abstand der Objekte des ersten Knotens in der Open-Liste größer Null ist (keine Kollision).
Bei **abgeschwächter Abstandsberechnung**: Wenn der Abstand der Objekte des ersten Knotens in der Open-Liste größer Null ist und als nächstes der Abstand zweier konvexer Polyeder berechnet werden müßte.

(2) Es können keine weiteren Knoten von anderen Prozessoren eintreffen.

(3) Bei einer Kollision oder wenn Bedingung (1) und (2) für alle Prozessoren gelten.

Um festzustellen, ob alle Prozessoren die Bedingungen (1) und (2) erfüllen, werden zwei Ansätze aus [Kumar94] kombiniert. Dazu werden alle Prozessoren zu einem virtuellen Ring verbunden. Falls für den Master-Prozessor die beiden ersten Abbruchbedingungen erfüllt sind, dann sendet er seinem Nachbarn im Ring einen weißen Token. Falls für die nachfolgenden Prozessoren die beiden ersten Abbruchbedingungen erfüllt sind, dann wird der Token so weitergeleitet, wie er empfangen wurde; ansonsten wird auf jeden Fall ein schwarzer Token weitergeleitet. Wenn der Token wieder beim Master-Prozessor ankommt, erkennt dieser an der Farbe des Tokens, ob die Bestensuche beendet ist. Dabei zeigt ein weißer Token die Terminierung, und ein schwarzer Token die Fortsetzung der Suche an. Für alle Prozessoren gilt dabei, daß nach dem Verschicken eines weißen Tokens keine Knotenanfragen mehr gestellt werden dürfen, damit keine Arbeit auf den Prozessor zukommt.

4. Experimentelle Ergebnisse

Die parallele Kollisionserkennung wurde auf der IPR-ParaStation, einem nachrichtengekoppelten System aus neun 133MHz-Pentium-PCs, implementiert [Wurll97]. Die ParaStation-Karte ermöglicht eine parallele Kommunikation der PCs in einem 2-dimensionalen zyklischen Gitter ((2D-Torus) [Warschko96]. Für die folgenden Experimente wurde die sog. Port-Schnittstelle verwendet, welche für Nachrichten der Größe 1 bis 256 byte eine Latenzzeit zwischen 15 und 56 µs sowie einen maximalen Durchsatz von ca. 12 MB/s erreicht.

Um die verschiedenen Variationen der Kollisionserkennung zu testen wurden einige mit dem Robotersimulationssystem ROBCAD erzeugte Umweltmodelle verwendet [Katz96]. Dazu gehört zunächst das Modell BOTTLENECK mit sieben Objekten, die im Durchschnitt aus 7,4 Eckpunkten bestehen. Neben diesem relativ kleinen Umweltmodell wurde eine industrielle Umgebung verwendet, die aus 40, 60, 80, 100 und 157 Objekten mit durchschnittlich 9 Eckpunkten besteht. Außerdem wurde durch das wiederholte Einlesen der industriellen Umgebung mit 100 Objekten (WBK100), eine Umwelt mit 1.000 Objekten (WBK1000) erzeugt.

In allen Testumgebungen wurde von dem Roboter Puma260, der durch zehn konvexe Polyeder mit durchschnittlich 30 Eckpunkten modelliert ist, eine vorher festgelegte

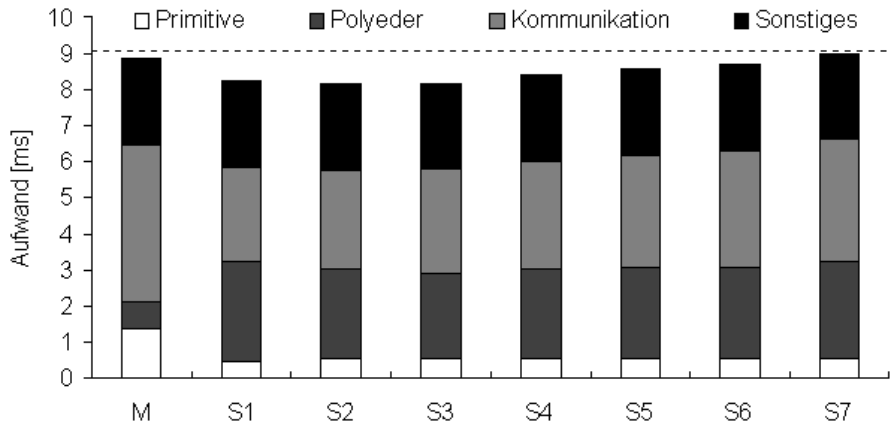

Abb. 4: Die Lastverteilung der acht Prozessoren (M, S1, ..., S7) bei einer parallelen Kollisionserkennung in der WBK1000-Umgebung. Außerdem der Zeitaufwand eines jeden Prozessors für die verschiedenen anfallenden Aufgaben (gestrichelte Linie: Gesamtberechnungszeit)

Bahn abgefahren. Diese Bahn setzt sich aus 119 Roboterkonfigurationen zusammen, die so gewählt sind, daß der Roboter in ca. 50% der Fälle mit Objekten in der BOTTLENECK-Umwelt kollidiert. Um Ungenauigkeiten in der Zeitmessung auszugleichen, wurde die Kollisionserkennung für jede Konfiguration 100 mal aufgerufen, so daß insgesamt 11.900 Kollisionserkennungen pro Testlauf zustande kommen.

Das Maß für die durch eine parallele Berechnung erzielte Beschleunigung ist der *Speedup*. Er ist definiert als der Quotient zwischen der Berechnungszeit des schnellsten bekannten sequentiellen Algorithmus und der Berechnungszeit des parallelen Algorithmus für p Prozessoren. Die Obergrenze für den erreichbaren Speedup ist daher die Anzahl p der Prozessoren (*linearer Speedup*). Diese Obergrenze läßt sich allerdings real nicht erreichen. Einige Gründe dafür sind: die Kommunikation zwischen den Prozessoren, der Verwaltungsaufwand, eine ungünstige Lastverteilung, sequentielle Anteile im parallelen Algorithmus und die durch Parallelisierung entstehende Mehrarbeit.

Um nun eine genauere Abschätzung für den erreichbaren Speedup zu erhalten, kann der Anteil der einzelnen Arbeitsschritte am Gesamtaufwand bestimmt werden. Für die BOTTLENECK-Umwelt hat eine Zeitmessung des sequentiellen Algorithmus die in Tabelle 1 aufgeführten Ergebnisse ergeben. Unter der idealisierenden Annahme, daß der Zeitaufwand für die Bestensuche durch die Parallelisierung nicht mehr ins Gewicht fällt, erhält man damit einen maximal erreichbaren Speedup von 49,39 s / (0,51 s + 9,39 s + 0,26 s) = 4,86. Bei dem hier eingesetzten Parallelrechner mit neun Prozessoren, kann sich der Zeitaufwand für die Bestensuche (5,52 s + 33,18 s + 0,53 s = 39,23 s) bei einer idealen Parallelisierung höchstens um den Faktor 9 verringern, d.h. auf ca. 4,36 s. Der maximal erreichbare Speedup beträgt damit nur noch 49,39 s / (0,51 s + 9,39 s + 0,26 s + 4,36 s) = 3,40. Analog beträgt für die WBK1000 Umwelt mit 1.000 Hindernissen der maximale Speedup 26,95 und auf einen Parallelrechner mit neun Prozessoren 6,94.

Aufgabe	Anzahl Aufrufe	Zeit pro Aufruf [µs]	Gesamt zeit [s]	Anteil am Gesamt-aufwand
Verwaltung	11.900	42,864	0,51	1,03 %
Bewegung				
Roboterposition aktualisieren	11.900	789,085	9,39	19,01 %
dyn. Hierarchie berechnen	11.900	21,432	0,26	0,52 %
Bestensuche				
Expandieren von Knoten	1.013.000	5,445	5,52	11,17 %
Abstand zw. Polyeder	155.000	214,054	33,18	67,18 %
Abstand zw. Primitiven	852.500	0,626	0,53	1,08 %
Gesamt	11.900	4.150,000	49,39	100,00 %

Tabelle 1: Anteile der wichtigsten Aufgaben der sequentiellen Kollisionserkennung am Gesamtaufwand für die BOTTLENECK-Umwelt.

Eine wichtige Voraussetzung für die effiziente Parallelisierung ist die ausgewogene Lastverteilung. In Abb. 4 ist der Aufwand der einzelnen Prozessoren für die Abstandsberechnung zwischen Primitiven und konvexen Polyedern, für die Kommunikation und für die sonstigen Aufgaben zusammengerechnet. Zu den "sonstigen Aufgaben" gehört die Initialisierung, d.h. die Aktualisierung der Roboterposition und der Aufbau der dynamischen Hierarchie, und die globale Programmschleife mit der Feststellung der Terminierung ("Verwaltung"). Es ist zu erkennen, daß die Prozessoren sehr gut ausgelastet sind, da der Abstand bis zur gestrichelten Linie, die den Gesamtaufwand für eine Kollisionserkennung markiert, bei allen Prozessoren relativ gering ist. Wartezeiten der einzelnen Prozessoren z.B. infolge einer leeren Open-Liste können damit nahezu ausgeschlossen werden.

Die Abb. 5 zeigt die Berechnungszeiten und die Beschleunigung für drei unterschiedlich große Umweltmodelle (BOTTLENECK, WBK100, WBK1000). Die Berechnungen wurden sequentiell und parallel mit 2, 4 und 8 Prozessoren durchgeführt. Die Ergebnisse zeigen, daß die relative Beschleunigung durch Parallelisierung mit der Anzahl von Hindernissen zunimmt. Dies liegt einerseits an der Reduktion des sequentiellen Anteils im parallelen Gesamtalgorithmus von ca. 20% auf unter 2%. Andererseits verbessert sich das Verhältnis Kommunikation (Knotenaustausch) zu Kalkulation (Abstandsberechnungen). Der erreichte Speedup für eine Umgebung mit 10 bzw. 1000 Hindernissen beträgt 1,5 bzw. 3. Dagegen ist der bei dieser Parallelisierung maximal mögliche Speedup 3,4 bzw. 6,94. Somit wird etwa die Hälfte der bei dieser Parallelisierung maximal erreichbaren Beschleunigung erzielt.

Die maximale Beschleunigung wird nicht erreicht, da bei der parallelen Kollisionserkennung mehr Abstandsberechnungen durchgeführt werden, als bei dem sequentiellen Algorithmus. Durch das gewichtete Random-polling werden die Knoten des Master-Prozessors schnell verteilt und (evtl. bessere!) Knoten der Slave-Prozessoren nur sehr langsam. Das Expandieren von schlechteren Knoten führt zu einer deutlichen Zunahme der Abstandsberechnungen zwischen konvexen Polyedern. Beispielsweise werden bei der parallelen Kollisionserkennung mit acht Prozessen mehr als doppelt so viele Polyederabstände berechnet, als bei der sequentiellen Berechnung (siehe Abb. 6).

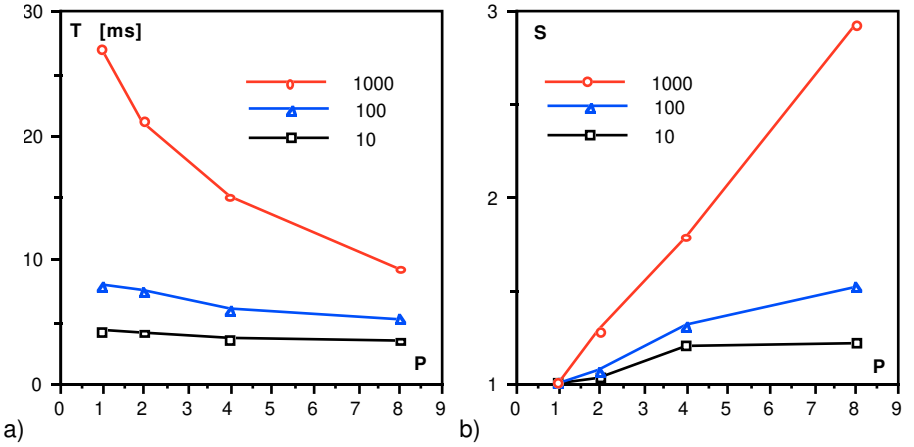

Abb. 5: Experimentelle Ergebnisse der parallelen Kollisionserkennung mit P Prozessoren in Umgebungen mit ca. 10, 100 und 1000 Hindernissen (BOTTLENECK, WBK100, WBK1000): a) Laufzeiten T und b) Beschleunigungen S (Speedups) bei P Prozessoren

5. Zusammenfassung und Ausblick

Durch die hier betrachtete Abstandberechnung auf Polyederbasis steht einer übergeordneten Anwendung mehr Information zur Verfügung als mit dem Kollisionstest. Insbesondere ermöglicht die geometrische Modellierung mittels konvexer Polyeder die Daten aus CAD- oder Robotersimulations-Systemen direkt zu verwenden und somit den Abstand sehr genau zu berechnen. Zur Beschleunigung werden statische und dynamische Bounding-box-Hierarchien eingesetzt. Außerdem reduziert die zusätzliche Approximation durch r-Zylinder die aufwendige Transformation der bewegten Polyeder.

Die Parallelisierung der Bestensuche nach dem kürzesten Abstand wird durch eine lokale Open-Liste je Prozessor ermöglicht. Der dabei notwendige Austausch von Suchbaumknoten mittels einem gewichteten Random-polling verteilt die Last nahezu perfekt. Dabei kann die Terminierung der Bestensuche durch einen 2-farbigen Token im virtuellen Ring zuverlässig festgestellt werden. Die Experimente zeigen eine Beschleunigung der Abstandsberechnung insbesondere bei wachsender Umweltgröße. Insgesamt ist die Kollisionserkennung für Industrieroboter auch bei vielen Hindernissen in weniger als 10 ms möglich.

Die vorgestellte Kollisionserkennung kann in verschiedene Richtungen erweitert werden. Einerseits gilt es die bei der parallelen Abstandsberechnung entstehende Mehrarbeit zu reduzieren. Wird andererseits die Kollisionserkennung im Zusammenhang mit der übergeordneten Anwendung betrachtet, dann lassen sich z.B. durch Abstandsfortschreibung [Sandmann97] noch weitere Beschleunigungen erreichen. Schließlich ist eine weitere Beschleunigung möglich, wenn die sich bewegenden Objekte und die Art ihrer Bewegung gezielt ausgenutzt wird, wie es schon für den

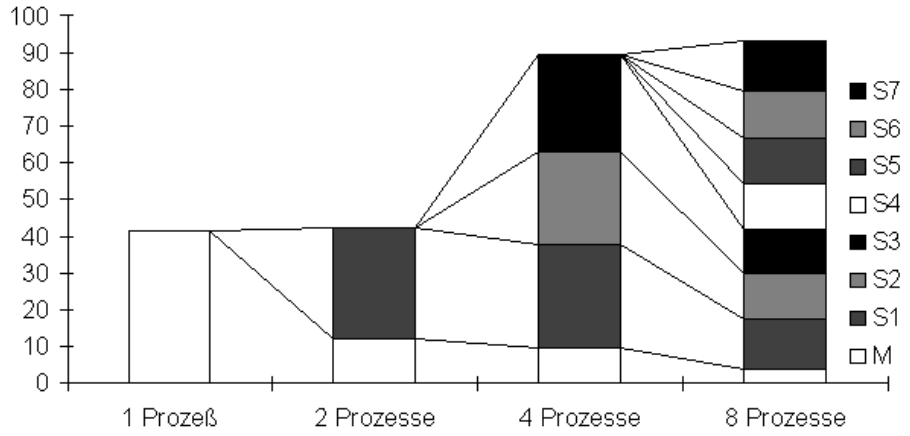

Abb. 6: Anzahl der Abstandsberechnungen zwischen konvexen Polyedern in der WBK1000-Umgebung für die Prozessoren M, S1, …, S7

Kollisionstest [Baginski97] und für die Berechnung von Polyederabständen [Ong97] vorgeschlagen wurde.

6. Literaturverzeichnis

[Baginski97] Baginski B., Efficient Dynamic Collision Detection using Expanded Geometry Models, Submitted to IROS 1997.

[Basta88] Basta R.A., Mehrotra R., Varansi M.R., Detecting and avoiding collisions between two robot arms in a common workspace, Robot Control Theory and Applications, 1988, p. 185-192.

[Bobrow89] Bobrow J.E., A Direct Minimization Approach for Obtaining the Distance Between Convex Polyhedra, Int. Journal of Rob. Res., 1989, Vol. 8, No. 3, p. 65-78.

[delPobil96] del Pobil A.P., Pérez M., Martínez B., A Practical Approch to Collision Detection Between General Objects, Proc. IEEE Int. Conf. on Robotics and Automation, Minneapolis, Minnesota, April 1996, p. 779 - 784.

[Dobkin83] Bobkin D.P., Kirkpatrick D.G., Fast Detection of Polyhedral Intersection, Theoretical Computer Science, Vol. 27, 1983, p. 241-253.

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

[Gilbert88] Gilbert E.G., Johnson D.W., Keerthi, S.S., A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space, IEEE Journal of Robotics and Automation, Vol. 4, No. 2, April 1988, p. 193-203.

[Gontermann97] Gontermann S., Parallele Kollisionserkennung, Diplomarbeit, Universität Karlsruhe, 1997.

[Henrich91] Henrich D., On-line Kollisionserkennung mit hierarchisch modellierten Hindernissen für ein Mehrarm-Robotersystem, Diplomarbeit, Universität Karlsruhe, 1991.

[Henrich92] Henrich D., Cheng X., Fast Distance Computation for On-Line Collision Detection with Multi-Arm Robots, IEEE Int. Conf. on Robotics and Automation, Nizza, France, May 1992, p. 2514-2519.

[Huang90] Huang S-R., Davis L. S., 1990, "Speedup analysis of centralized parallel heuristic search algorithms", Int. Conf. on Parallel Processing, vol 3, pp 18-21.

[Katz96] Katz G., Integration eines parallelen Bewegungsplaners in ROBCAD, Diplomarbeit, Universität Karlsruhe, Dezember 1996.

[Kumar94] Kumar V., Grama A., Gupta A., Karypis G., Introduction to Parallel Computing, The Benjamin/Cummings Publishing Company, Inc., 1994.

[Lee87] Lee B.H., Lee C.S.G., Collision-Free Motion Planning of Two Robots, IEEE Transaction on Systems, Vol. SMC-17, Number 1, 1987, p. 21-32.

[Lumelsky85] Lumelsky V. J., On fast computation of distance between line segments, Inform. Proc. Let., Vol 21, No 2, Aug. 1985, p. 55 - 61.

[Metivier90] Metivier C., Urbschat R., Run-time statistical analysis of a robot motion planning algorithm, Early draft, Stanford University, ERL 445, Stanford, C.A., 1990.

[Meyer86] Meyer W., Distance between boxes: Applications to collision detection and clipping, Proc. IEEE Int. Conf. on Robotics and Automation, 1986, p. 597-602.

[Ong97] Ong C.J., Gilbert E.G., The Gilbert-Johnson-Keerthi Distance Algorithm: A Fast version for Incremental Motions, Proc. IEEE Int. Conf. on Robotics and Automation, Albuquerque, New Mexiko, April 1997, p. 1183 - 1189.

[Quinlan94] Quinlan S., Efficient Distance Computation between Non-Convex Objects, IEEE Int. Conf. on Robotics and Automation, 1994, 4, p. 3324-3329.

[Sandmann97] Sandmann S., Parallele Bewegungsplanung für Industrieroboter in dynamischer Umgebung, Diplomarbeit, Universität Karlsruhe, März 1997.

[Sato96] Sato Y., Hirata M., Maruyama T., Arita Y., Efficient Collision Detection using Fast Distance-Calculation Algorithms for Convex and Non-Convex Objects, Proc. IEEE Int. Conf. on Robotics and Automation, April 1996, p. 771-778.

[Warschko95] Warschko T.M., Tichy W.F, Herter C.G., Efficient Parallel Computing on Workstation Clusters, Technical Report 21/95.

[Wurll97] Wurll C., Henrich D., Ein Workstation-Cluster für paralleles Rechnen in Robotik-Anwendungen, APS'97: 4. ITG/GI-Fachtagung Arbeitsplatz-Rechensysteme, Koblenz-Landau 1997.