

VISUM: Ein Virtual-Reality-System für das Testen und Trainieren von humanoiden Robotern

M. Baas, M. Fautz, D. Finkenzeller, S. Preuß, S. Thüring und A. Schmitt

Institut für Betriebs- und Dialogsysteme, Universität Karlsruhe,
Am Fasanengarten 5, 76131 Karlsruhe, Germany

{baas, fautz, dfinken, stpreuss, thuering, aschmitt}@ira.uka.de

Kurzfassung

Es werden die Probleme diskutiert, die bei der Entwicklung eines offenen Virtual-Reality-Systems zu lösen sind, in dem humanoide Roboter agieren sowie getestet und trainiert werden sollen. Neben den Standard-Funktionalitäten der VR-Systeme wie z.B. Navigation in 3D-Welten sind zusätzlich kinematische und dynamische Simulationen zu unterstützen, die gesamte Roboter-Sensorik sowie die im humanoiden Roboter integrierte Software bzw. Handlungsintelligenz. Diese komplexen Anforderungen können nur durch eine modulare und offene Softwarearchitektur erfüllt werden. Der zurzeit existierende Prototyp VISUM (Virtuelle Simulationsumgebung) wird kurz vorgestellt.

1 Einleitung

Bei technisch komplexen Szenarien ist es seit Langem üblich, das Verhalten von Systemen durch Simulationen zu überprüfen, weil dies in vielen Fällen kostengünstiger ist als bei Testverfahren mit realen Systemen. Die Simulation erlaubt auch qualitative Aussagen über das Verhalten oder die Funktionsweise von Systemen oder Teilsystemen sowie deren Optimierung, die noch gar nicht real vorliegen, sondern sich noch in Konstruktion befinden. Bei der im SFB 588 betriebenen Forschung wird u.a. angestrebt, humanoide Roboter zu simulieren. Dies ist nur möglich, wenn Kinematik, Dynamik, Sensorik sowie die „Intelligenz“ des humanoiden Roboters mit einer simulierten Umwelt zusammenspielen. Diese Anforderungen simultan zu erfüllen, erfordert die Realisierung einer virtuellen Realität (VR) mit erheblich erweitertem Modellierungsspektrum. Entwickelt werden soll ein offenes VR-System, das folgende Leistungsmerkmale aufweist:

- komplexe mechanische Modellierung: Starre Körper, Roboterarme, Greifbewegungen, gelenkgekoppelte Mehrkörpersysteme, Kollisions- und Stoß-Effekte, Servo-Antriebe, Pneumatik, Reibungseffekte usw.
- simulierte Sensorik und Messtechnik: Berührungs-, Ultraschall-, Infrarot-Sensoren, Kraftsen-

soren, simulierte Videokamera-Augen, Akustik-Simulation usw.

- Softwareintegration („Software in the loop“)
- VR-Immersion für Menschen: Interagieren mit der VR-Simulation sowohl aus externer Sicht als auch in der direkten Immersion, also Face-to-Face mit dem humanoiden Roboter.

In diesem Papier wird der Stand der Arbeiten an diesem Teilprojekt beschrieben und es werden einzelne Aspekte etwas genauer betrachtet.

2 Modellierung von VR-Szenen mit Mechanik

2.1 Geometrie, Bilderzeugung, Visuelle Simulationen

Die Modellierung der visuellen Darstellung beschränkt sich auf die Definition von geometrischen Primitiven, meist einem Dreiecksnetz, welches die sichtbare Hülle der zu simulierenden Objekte darstellt. Dieses wird mit Farbeigenschaften, Texturen, Spiegelung und Opazität verfeinert. Diese Art der Modellierung ist die Grundlage aller VR-Systeme.

2.2 Kinematik

Für die kinematische, noch nicht dynamische Bewegung der Objekte müssen Attribute vorgegeben werden, die deren Beweglichkeit im Sinne der kinematischen Randbedingungen einschränken. Des Weiteren muss meist eine hierarchische Ordnung der Objekte erstellt werden, die die gemeinsamen Bewegungen organisieren und die Freiheitsgrade der einzelnen Bewegungen bestimmen. Eine Tür oder Schublade im Küchenschrank wäre ein Unterobjekt des Schrankes mit daraus folgenden eingeschränkten Bewegungsmöglichkeiten (nur eine Drehung bzw. Translation möglich). Schließlich müssen kinematische Antriebe erlaubt werden, also Ortsveränderungen ohne den Einfluss von Kräften und trägen Massen. Kinematische Bewegungen sind z.B. typisch für Computeranimationen, sie können sowohl sehr natürlich als auch sehr unnatürlich aussehen.

2.3 Dynamik

Für eine realitätsnahe dynamisch-mechanische Simulation von Bewegungen in einer VR-Welt müssen bei den einzelnen beteiligten Körpern zunächst deren Massen, die wirkenden Kräfte, kinematischen Verkettungen, die Schwerkraft usw. einkalkuliert werden. Anstatt kinematischer, also trägheitslos ausgeführter Bewegungen gibt es jetzt nur noch Bewegungen, wenn Antriebskräfte oder von außen einwirkende Kräfte vorhanden sind.

Korrekte dynamische Mechaniksimulation setzt auch den korrekt simulierten Impulsaustausch bei Kollisionen voraus. Möchte man komplexere Kollisionen simulieren, müssen die Objekte noch mit Reibungskoeffizienten ihrer Oberfläche versehen werden, um Reibung und unelastische Impulsübertragungen zu simulieren.

Spätestens bei komplexeren dynamisch simulierten VR-Welten wird es zu einem Problem, die gesamte Simulationskette bestehend aus Mechanik, Kinematik, Kollisionserkennung und Bilderzeugung noch in Realzeit durchführen zu können.

2.4 Weitere Objektattribute in einer physikalisch verfeinerten VR-Simulation

In vielen Fällen reicht die Simulation der Mechanik auf der Basis gekoppelter Mehrkörpersysteme mit korrekter Kollisionsauflösung aus, insbesondere für die Simulation humanoider Roboter. Allerdings müssen auch alle vom humanoiden Roboter benutzten Sensor-Signale zur Verfügung gestellt werden. Die meisten Sensor-Signale wie z.B. Laser-Abstandsscanner, Ultraschall-Abstandsmessung und Bilderzeugung für die Augen-Kameras des Roboters können aus der aktuellen Konfiguration der 3D-Welt ohne großen Rechenaufwand abgeleitet werden.

Erhebliche Erweiterungen der mechanischen Welt wären erforderlich, wenn auch folgende Attribute physikalisch hinreichend plausibel mitgeführt werden sollen:

1. realistische Lichtausbreitung (Global-Illumination-Verfahren),
2. Akustik und Schallausbreitung,
3. Temperatur (für Infrarot-Kameras bzw. Sensoren),
4. elastische Verformungen,
5. Materialzerstörungen bei zu großen Kräften,
6. hochfrequente mechanische Schwingungen,
7. elektromagnetische Wellenausbreitung,
8. Flüssigkeiten in Gefäßen, also z.B. Kaffeekannen und Kaffeetassen.

Bei dem zunächst angestrebten Küchenszenario sind die Attribute gemäß 1. und 8. wichtig und sollten längerfristig in die VR-Welt integriert werden.

3 Wirklichkeitsnähe mechanischer Simulationen

Ein fundamentales Problem wirklichkeitsnaher physikalisch modellierter VR-Welten stellt die für eine numerische Simulation benötigte hohe Rechenzeit dar. Solange

man nur simulieren will, ohne den Anspruch an Realzeit-Ausführung, existiert das Problem nicht. Soll jedoch ein Mensch in die VR-Welt immersiert werden (vgl. Abschnitt 5), der mit dem humanoiden Roboter interagiert, so ist zu fordern, dass die gesamte Simulation in Realzeit oder besser sogar schneller ausgeführt werden kann. Grundsätzlich bieten sich hier zwei Strategien an, um Realzeit-Ausführung zu erreichen:

1. Mehrprozessor-Systeme mit ausreichender Rechenleistung,
2. Wirklichkeitsnähe durch schnelle approximierende Verfahren der physikalischen Simulationen.

Im Folgenden werden insbesondere Möglichkeiten der Approximation erörtert. Generell können Simulationen in zwei Klassen eingeteilt werden, je nachdem, welchen Zweck sie erfüllen:

1. Simulationen, um Vorhersagen zu treffen bzw. Entscheidungen über reale Dinge zu fällen.
2. Simulationen, um dem Menschen (oder einer Maschine) die reale Welt vorzutäuschen.

Bei der ersten Klasse steht die Genauigkeit im Vordergrund, da sich die Simulation immer auf eine reale Gegebenheit oder ein reales (bzw. ein noch zu erstellendes) Produkt bezieht. Beispiele hierfür sind die Wettervorhersage, eine Hochwasservorhersage oder ein simulierter Crashtest.

Die zweite Klasse von Simulationen hat zum Ziel, dem Menschen (bzw. der Maschine, z.B. dem humanoiden Roboter) die Realität genau genug vorzutäuschen. Im Gegensatz zur ersten Klasse steht hierbei nun nicht mehr die Simulation im Vordergrund, sondern der *immersierte* Mensch (oder eben die Maschine). Dadurch ist die Genauigkeit der Simulation nur noch insoweit relevant, wie sie vom Menschen beurteilt werden kann. Beispielsweise kann die exakte Bewegungsbahn eines Körpers vom Menschen nicht vorhergesagt werden, weshalb in einer Simulation eine von vielen „plausiblen“ Bahnen ausgewählt werden kann. Aus diesem Grund steht bei dieser Klasse von Simulationen nicht die Genauigkeit sondern vielmehr die Geschwindigkeit der Simulation im Vordergrund, da eine Immersion des Menschen nur dann möglich ist, wenn die Bildwiederholrate der virtuellen Bilder hoch genug ist.

3.1 Simulationsmethoden für Mehrkörpersysteme

Für die dynamische Simulation von Mehrkörpersystemen sind in der Vergangenheit viele Softwaresysteme entwickelt worden. Ein weit verbreitetes System ist z.B. *Adams*. Fast alle bekannt gewordenen Ansätze basieren darauf, dass ein System von Differenzialgleichungen aufgestellt wird, welches nach zum Teil komplexen symbolischen Umformungen mit Verfahren wie Runge-Kutta über die Zeitachse integriert werden kann. An den klassischen Verfahren ist nachteilig, dass sie bei wesentlichen Änderungen der Randbedingungen wie z.B. Kollisionen und Umorganisationen des Mehrkörpersystems auch

das Differenzialgleichungssystem verändern müssen, was sehr zeitaufwändig sein kann.

Wir haben uns daher das Ziel gesetzt, die dynamische Simulation für Mehrkörpersysteme auf ganz direktem Weg und ohne Rückgriff auf Systeme von Differenzialgleichungen durchzuführen. Wir verfolgen dabei die Strategie, einen Körper durch Massenpunkte zu approximieren und den einzelnen Massenpunkt als ein mit seiner mechanischen Umgebung interagierendes Subsystem aufzufassen. Das hat insbesondere den Vorteil, dass Sonderfälle wie bei Kollisionen, Reibung etc. problemlos in die Simulation einbezogen werden können [12]. In einer ersten Studie wurde das Verfahren eingesetzt, um einen Roboterarm bis herunter auf die Ebene der Servos zu simulieren [11]. Die Realzeitfähigkeit und die Stabilität des Verfahrens sind derzeit jedoch noch unbefriedigend. Daher ist die Frage offen, welche dynamischen Simulationsverfahren in das VR-System integriert werden.

3.2 Bewertung von Simulationsmethoden

Wie können nun Simulationsmodelle miteinander verglichen werden? Oder: wie kann gezielt eine geeignete Methode zur Bewertung von Simulationsverfahren entwickelt werden?

Ein numerisches Modell ist immer nur eine Vereinfachung der Realität. Deshalb muss zunächst der Zweck der Simulation festgelegt werden, um überhaupt eine Bewertungsgrundlage zu haben. Wofür soll die Simulation eingesetzt werden? Müssen Vorhersagen gemacht werden können? Wenn ja, welche? Welche physikalischen Modelle müssen in der Simulation enthalten sein und welche Effekte müssen berücksichtigt werden?

Nachdem Zweck und Ziel der Simulation feststehen, kann eine konkrete Methode nach den folgenden Gesichtspunkten bewertet werden:

- **Plausibilität:** Liefert die Simulation Ergebnisse, die unter passenden Umständen auch in der Realität so beobachtet werden könnten?
- **Effizienz:** Welche Komplexität besitzt der Algorithmus (O-Kalkül)? Zeitvergleich mit anderen Algorithmen bei gleicher Simulationsaufgabe.
- **Numerische Stabilität:** Basiert der Algorithmus auf numerisch instabilen Verfahren (steife Gleichungssysteme)? Kommt es in der Praxis zu Divergenzen?
- **Vorhersagefähigkeit:** Ist es möglich, anhand eines Simulationsergebnisses Vorhersagen über ein reales System zu treffen? Dabei muss vorher genau festgelegt werden, was überhaupt vorhergesagt werden soll und was nicht.

Ein weiteres nahe liegendes Bewertungskriterium wäre das der *physikalischen Korrektheit*. Ist eine Simulationsmethode physikalisch korrekt oder nicht? Was bedeutet das überhaupt und wie kann man es überprüfen?

Am einfachsten wäre es, wenn man ein Experiment in der Realität und in der Simulation durchführen und die Ergebnisse vergleichen könnte. Hier wird man auf das Problem stoßen, die Anfangsbedingungen nicht exakt identisch setzen zu können. Bei chaotischen Systemen ist

es nicht einmal in der Realität möglich, ein Experiment mehrmals mit gleichem Ausgang auszuführen. Ein exakter Vergleich mit einer Simulation ist in diesen Fällen von vornherein nicht möglich.

Wenn die physikalische Korrektheit ohne Vergleich mit der Realität gezeigt werden soll, so müssen zunächst die physikalischen Modelle und Randbedingungen festgelegt werden, bzgl. derer die Korrektheit nachgewiesen werden soll. Beispielsweise könnte gefordert werden, dass ein Modell zur Simulation starrer Körper korrekt bzgl. der Energieerhaltung sein soll, d.h. die Summe der kinetischen und potentiellen Energie bleibt konstant. Die wenigsten Modelle werden dieser Überprüfung standhalten können, sobald äußere Einflüsse und Kollisionen im Spiel sind. Man stelle sich einen Würfel vor, den man zunächst ruhig in der Hand hält und dann auf einen Tisch fallen lässt. Bei einer „realistischen“ Simulation wird der Würfel evtl. ein paar Mal umher springen, über den Tisch rollen und letztendlich wieder zur Ruhe kommen. Der Zustand des Tisches würde sich dabei vermutlich nicht ändern. Am Anfang und am Ende des Versuchs ist der Würfel in Ruhe, besitzt damit also keine kinetische Energie. Da man ihn zu Beginn über den Tisch gehalten hatte, war die potentielle Energie am Anfang höher als am Ende des Versuchs, d.h. es ging Energie verloren. Die Simulation scheint also nicht korrekt bzgl. der Energieerhaltung zu sein. Dennoch kann diese Simulation als realistische und sinnvolle Simulation bezeichnet werden, da sie plausibel wirkt und einen Effekt beobachten lässt, der in der Realität auch beobachtbar ist, nämlich, dass der Würfel nicht unendlich lange weiterrollt. In der Energiebilanz wurden aber nicht alle Energieformen berücksichtigt. Durch Reibung entsteht Wärme. Durch Kollisionen entstehen Deformationen. Diese Verluste wurden in der Energiebilanz ignoriert. Um nun aber z.B. die thermische Energie berücksichtigen zu können, müssen im Szenenmodell Temperaturdaten enthalten sein. Bei der Simulation starrer Körper wird dies sicherlich in den wenigsten Fällen sinnvoll sein. Daraus folgt, dass man die Energieerhaltung in diesem Fall nicht nachweisen kann. Erst wenn man auf die Reibung verzichten würde, könnte im Würfelbeispiel die Energieerhaltung überprüft werden. Der Würfel müsste sich dann mit entsprechender Geschwindigkeit unendlich lange weiter bewegen. Allerdings wird niemand behaupten wollen, dass diese Simulation dann realistischer sei als diejenige mit Reibung. Die Energieerhaltung kann also nur in idealisierten Fällen als Kriterium verwendet werden, z.B. dann, wenn keine äußeren Einflüsse wirken. Bei komplexeren und damit praxisbezogenen Situationen ist dieses Kriterium nicht mehr anwendbar.

Das obige Beispiel hat gezeigt, dass es im Allgemeinen sehr schwierig ist, geeignete Kriterien zu finden, um eine Simulation auf physikalische Korrektheit hin zu überprüfen. Es ist daher ratsam, eine Simulation daraufhin zu beurteilen, ob sie ihren zuvor festgelegten Zweck in der Praxis erfüllt oder nicht.

Da die exakte „physikalische Korrektheit“ nicht festgestellt werden kann, sollte dieser Begriff durch den der „Plausibilität“ oder der „Vorhersagefähigkeit“ ersetzt werden:

Eine Simulation kann als plausibel angesehen werden, wenn das Ergebnis unter passenden Umständen auch in der Realität beobachtet werden könnte.

Die Überprüfung dieses Kriteriums verlässt sich allerdings auf die Erfahrung des prüfenden Menschen und kann damit nicht als objektives Vergleichskriterium herangezogen werden.

Die Vorhersagefähigkeit kann dadurch überprüft werden, dass man das entsprechende Experiment auch in der Realität durchführt und mit dem Simulationsergebnis vergleicht. Hier muss allerdings darauf geachtet werden, dass man nicht eine Größe vorhersagen möchte, die sich überhaupt nicht vorhersagen lässt (z.B. die Konfiguration eines chaotischen Systems).

4 Simulation mechatronischer Systeme

Bisher befassten wir uns nur mit passiver Mechanik, d.h., in der Simulation legten wir Wert darauf, dass Objekte plausibel miteinander interagieren. Das bedeutet, dass die Objekte kein Eigenleben besitzen. Dass dies für eine Simulation mechatronischer Systeme nicht ausreicht, liegt auf der Hand, da diese aktive Systeme sind. Deshalb ist es notwendig, eine Technik zu entwickeln, mit der die Objekte ein Eigenleben erlangen.

4.1 Software-Servos

Um einen Roboterarm in der Simulation bewegen zu können, benötigen wir Gelenke, die Armglieder miteinander verknüpfen [11]. Zusätzlich werden Motoren benötigt, die Kräfte auf die Gelenke ausüben und sie bewegen. Die hierfür bei realen Robotern eingesetzten Servos werden über einen Regler angesteuert, der es ermöglicht, Bewegungen kontrolliert auszuführen und Positionen anzufahren bzw. zu halten. Ein von außen eingestellter Sollwert – in unserem Fall der Gelenkwinkel – wird mit dem gemessenen Istwert verglichen und im Rahmen der Leistung des Servos korrigiert. Dies ist ein ständig aktiver Regelungszyklus, denn auch bei Ruhe muss eine Kraft aufgewandt werden, um z.B. den Roboterarm in der Ruheposition zu halten.

In der Simulationen müssen deshalb die beiden Komponenten Servo und Regler mit folgenden Eigenschaften simuliert werden:

1. Servo:
 - a) Ausüben eines veränderlichen Drehmoments,
 - b) Sensoren zur Winkelmessung.
2. Regler:
 - a) Einstellbarkeit des Winkel-Sollwertes,
 - b) Begrenzung des maximalen Drehmoments,
 - c) Drehmomentregelung.

Die Eigenschaften 1.a), 1.b) und 2.a) sind in unserer direkten mechanischen Simulation mit Massepunkten leicht zu realisieren. Da wir uns in einer virtuellen Umgebung bewegen, besitzen wir die 3D-Modelle aller Objekte und können somit die Winkel zwischen den Objekten berechnen. Das Drehmoment unter Punkt 1.a), das der Servo aufbringen muss, kann als Kraftvektor

bestimmter Länge angegeben werden. Seine Richtung und Stärke (Länge) wird von der Regelung 2.c) bestimmt. Sein maximaler Betrag wird durch den Wert in Punkt 2.c) begrenzt.

Bei realen Systemen ist die Regelung im Allgemeinen in Software auf Microcontrollern realisiert. Für die Simulation bedeutet das, dass die Regelungs-Software mit geringem Aufwand in einen Software-Regler umgesetzt werden kann. Die Software-Lösung des Servos und des Reglers muss dem Verhalten realer Servos und Regler (wie sie z.B. im zu simulierenden Roboterarm vorkommen) angepasst werden.

Bei unseren Experimenten hat sich ein Problem mit der Abtastrate ergeben: Die Häufigkeit, mit der der Regler den Istwert (1.b)) des Servos überprüft, ist wesentlich höher (die Abtastrate liegt im kHz-Bereich) als die Schrittweite der Mechanik-Simulation, die bei 25 oder 50 Iterationsschritten pro Sekunde liegt.

Sind die oben genannten Anforderungen zur Zufriedenheit gelöst, dann ist die Grundlage geschaffen, einen virtuellen Roboterarm dynamisch zu simulieren und damit auch alle wirksamen Kräfte offen zu legen.

4.2 Zweck der Simulation mechatronischer Systeme

Es stellt sich die Frage, weshalb man mechatronische Systeme simulieren möchte. Für obiges Beispiel „Software-Servo“ gilt, dass man z.B. einen Roboterarm erst in der Simulation entwickelt und testet, bevor man ihn wirklich baut. In der Simulation lassen sich schnell verschiedene Konstruktionen realisieren und ausprobieren. Die Simulation eröffnet die Möglichkeit, Parameter wie Gewicht und Größe der Armglieder, einfach zu variieren und verschieden starke Servos einzusetzen. Anschließend kann man den Arm testen und die maximale Nutzlast ermitteln. Somit lässt sich Zeit und Geld sparen. Auch in Hinsicht Sicherheit für Mensch und Maschine ist die Simulation sinnvoll, denn man kann die Software zur Steuerung des Armes am virtuellen Modell entwickeln und ausprobieren. Bei gravierenden Fehlfunktionen in der Software kommt deshalb nichts und niemand zu Schaden.

4.3 Software in the loop

In der Simulation sollen natürlich nicht nur Servos, sondern auch andere Komponenten des Roboters, z.B. seine Sensoren, getestet werden. Auch diese müssen in Software nachgebildet werden. Der Roboter erhält dann virtuelle Kameras, Abstandssensoren usw. Die zugrunde liegende Software, die später im realen Roboter zum Einsatz kommt, kann nicht unterscheiden, ob sie ihre Daten aus der Simulation oder vom realen Roboter erhält. Ebenso verhält es sich mit den Ausgabedaten. Man kann hierbei von „Software in the loop“ (SWIL) sprechen.

Ein kurzer Abstecher in die Thematik der „Hardware in the loop“ (HWIL) soll dies verdeutlichen. Hier wird einem abgeschlossenen System (im Allgemeinen eine autonome Hardware-Komponente) die Realität vorgegaukelt, indem es mit simulierten Eingabedaten gespeist wird (vgl. [5] und [6]). Im Gegensatz hierzu erhält in unserem Fall eine (Steuerungs-) Software Sensordaten einer simulierten Hardware. Allgemein soll in der Simula-

tion die Roboterhardware so detailliert wie möglich nachgebildet werden. Der Roboter, bzw. die Robotersteuerung erfährt über seine virtuellen Sensoren und Kameras eine virtuelle Welt, von der er „glaubt“, dass er in ihr agiert. In unserem Sinne ist das eine erweiterte Form von SWIL, und zwar „Robot in the loop“.

5 Interaktivität durch Immersion

Um Interaktivität und Kommandierung zwischen dem virtuellen Roboter und dem realen Menschen zu ermöglichen, ist es nötig, den Menschen in die virtuelle Welt zu immersieren und dem Roboter das Vorhandensein des realen Menschen innerhalb der virtuellen Simulationsumgebung vorzutauschen. Dies ist nötig, damit der Roboter die Möglichkeit hat, den Menschen mit seinen virtuellen Sensoren (Kameras, Ultraschallsensoren etc.) wahrzunehmen, und auf ihn zu reagieren (z.B. bei einer Kollisionsgefahr oder bei einer gemeinsamen Handlung).

Hierfür bieten sich eine Reihe von VR-Technologien an. Die am meisten angewandte Hardwarekombination für die direkte Immersion des Menschen besteht aus einem Datenanzug (elektromagnetisch, mechanisch oder mit optischen Markern), einem Datenhandschuh und einem 3D-Sichtsystem (HMD). Der Datenanzug gewährleistet die Steuerung des Avatars, also eines zugrunde liegenden 3D-Modells bzw. Skeletts durch Berechnung von Position und Ausrichtung definierter Punkte und Gelenke in Echtzeit.

Diese VR-Systeme sind sehr teuer und in ihrem Einsatz zeitintensiv und unpraktikabel, da insbesondere der Datenanzug an jeden Benutzer angepasst werden muss.

5.1 3D-Modell

Wir verfolgen derzeit eine völlig andere Lösung. Wir wollen sowohl die visuelle Erscheinung als auch die Bewegung eines Menschen mit mehreren Kameras - ohne die Verwendung von Markern - in Echtzeit erfassen. Als Basistechnologie wird die Volumenschnitttechnik eingesetzt, die kürzlich in einer Dissertation an unserem Institut zur Praxisreife entwickelt wurde [2]. Das Verfahren basiert auf dem Konzept der visuellen Hülle von 3D-Objekten (vgl. [4]).

Eine solche Approximation und damit das geometrische Menschmodell kann dann wie folgt bestimmt werden: Die Rückprojektion der Silhouettenkontur in den R^3 aus Sicht der Kamera bildet eine so genannte Sichtpyramide. Schneidet man die aus unterschiedlichen Ansichten gewonnenen Sichtpyramiden, so erhält man eine Approximation der Gestalt des Menschen. Hierfür soll das in [2] entwickelte *VisOR* Verfahren („visuelle-Hülle-basierte Oberflächenrekonstruktion“) verwendet werden, das im Gegensatz zu voxelbasierten Volumenschnittverfahren (vgl. [1] und [9]), deren Volumendaten für eine Visualisierung zuerst in ein Oberflächenmodell überführt werden müssen, bereits bei der Schnittberechnung die gewünschte Repräsentationsform erzeugt. Darüber hinaus ist dieses Verfahren so effizient, dass es (auf heutigen Rechnern) eine Berechnung der Volumenschnitte aus bis zu acht Ansichten bei fein aufgelösten Silhouettenkonturen in Echtzeit verspricht [7].



Abbildung 1: Integration einer menschlichen Figur in eine virtuelle Küche. Die Figur wurde aus acht Videobildern rekonstruiert.

Insbesondere zur Simulation von visuellen Sensoren ist die Generierung von realistischen Kamerabildern aus beliebigen Ansichten des geometrischen Menschmodells erforderlich. Dies wird durch Projektion der Kamerabilder auf das geometrische Menschmodell und synthetische Generierung der gewünschten Ansichten erreicht [8].

Für das Erkennen von Gesten kann es zudem erforderlich sein, eine Extraktion der genauen Bewegung über eine Anpassung eines Gelenkmodells an das geometrische Modell vorzunehmen.

5.2 Sichtsystem

Das Problem der 3D-Sicht wurde bisher meist mit HMDs (Head Mounted Displays) gelöst. Es schottet den Nutzer von der Realität ab und erleichtert ihm so das Eintauchen in die synthetische Welt. Eine weitere Alternative stellen Stereoprojektionssysteme dar. Diese können bis auf sechs Projektionswände erweiterbar sein (CAVE). Im Gegensatz zu HMDs bieten Stereoprojektionssysteme mehreren Benutzern die Möglichkeit, die Simulation wahrzunehmen. Mit dieser Technologie werden auch die Nebenwirkungen der HMDs, die unnatürliche Entkopplung von Akkomodation und Vergenz sowie der sensorische Konflikt zwischen Gleichgewichtssinn und visuellem System verhindert.

Bisher wurden die Daten der Stereoprojektionen von Graphiksubsystemen teurer Graphikrechnern geliefert. Als Alternative bietet sich die Verwendung mehrerer Standard-PCs an. Hierbei stellt aber die Synchronisation und die Verteilung der Daten ein Problem dar, das mit speziellen Protokollen für das verteilte Visualisieren [3] oder durch synchronisierbare Szenegraphen wie z.B. OpenSG [10] gelöst werden kann.

6 Architektur des offenen VR-Systems

In den vorherigen Abschnitten wurden die Einzelteile betrachtet, aus denen ein VR-System zusammengesetzt ist. Dies umfasste sowohl die geometrische als auch die dynamische Modellierung der VR-Welt und deren Bestandteile, sowie eine Anbindung des Menschen als

Benutzer, so dass er auf die VR-Welt Einfluss nehmen kann. Zuletzt muss die Welt in angemessener Weise dargestellt werden können. All diese Komponenten müssen in einem Softwaresystem zusammengefasst werden. Einige allgemeine Anforderungen an solch ein System sind:

- leichte Erweiterbarkeit,
- offene API,
- leichtes Debugging,
- leichte Bedienung,
- einfache Integration der Robotersteuersoftware.

Vor allem in einem prototypischen System stehen die Erweiterbarkeit und die Offenheit im Vordergrund. Bei der Entwicklung von Algorithmen sollten hier vom System keine unnötigen Barrieren aufgebaut werden, die die Produktivität negativ beeinflussen würden.

Aus diesem Grund bietet es sich an, zunächst keine systemnahe Sprache wie C/C++ zu verwenden, sondern auf eine höhere Sprache wie Python zurückzugreifen. Durch eine dynamische Sprache wie Python erhält man ein erhebliches Maß an Flexibilität, was der Erweiterbarkeit und Offenheit des Systems zugute kommt. Lediglich der Aspekt der Echtzeit tritt in Konflikt mit einer interpretierten Sprache, die je nach Anwendung bis zu einige Größenordnungen langsamer als C/C++ sein kann. Hier bietet Python jedoch mehrere Ansätze, um ein Programm zu beschleunigen (Pyrex, Psyco, Weave, SWIG, Sip, Boost).

Der bisher entwickelte VR-Prototyp wurde in Python geschrieben und besteht aus einem vorgegebenen Satz von abstrakten Basisklassen, von denen abgeleitet wird, um die eigentliche Funktionalität des Systems bereitzustellen. Die abgeleiteten Klassen werden als Plugins in das System integriert, so dass das eigentliche Kernprogramm nicht mehr angefasst werden muss.

Die Hauptklassen der VR-Umgebung sind:

- *Scene*: Sie dient als Container für die Objekte in der VR-Welt.
- *Node*: Alle Objekte, die in einer Szene gespeichert werden können, sind vom Typ Node.
- *Controller*: Ein Controller bewegt einen Knoten, hier ist also die Dynamik enthalten.
- *Renderer*: Er stellt die Szene auf dem Bildschirm dar (z.B. via OpenGL).

Zusätzlich gibt es noch Klassen, die der Interaktivität mit der Umgebung dienen, die hier aber nicht aufgelistet sind. Spezielle Szeneknoten sind der Agent (Objekte, die eine eigene Verhaltenslogik besitzen), Sensoren (liefern Eingänge für einen Agenten) und Aktoren (treiben einen Agenten an). Die eigentliche Simulation findet in einem Controller-Objekt statt. Jeder Controller kann eine beliebige Anzahl Knoten bewegen, so dass auch unterschiedliche Simulationen innerhalb einer VR-Welt angewandt werden können. Ein Knoten ohne Controller dient lediglich als statische Kulisse. Der Ablauf eines einzelnen Simulationszyklus ist in Diagramm 1 dargestellt. Die ersten drei Schritte betreffen je die Sensoren, die Agenten und die Aktoren, danach werden sämtliche

Controller ausgeführt und damit die Knoten bewegt, und schließlich wird das Bild von einem Renderer erzeugt.

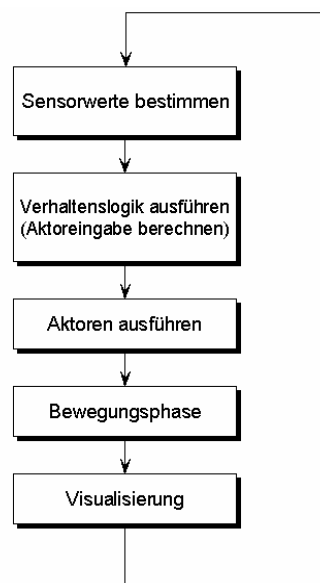


Diagramm 1: Simulationszyklus

Durch die dynamische Eigenschaft von Python können den Szeneknoten auch noch während der Laufzeit neue Attribute oder sogar Methoden hinzugefügt werden. Beispielsweise kann ein Controller-Plugin den zu kontrollierenden Knoten Attribute verpassen, die es zur Simulation benötigt (z.B. Reibungskoeffizienten, Numerikparameter usw.).

Ein Screenshot der VR-Umgebung mit einem Armtec-Roboterarm ist in Abbildung 2 zu sehen.

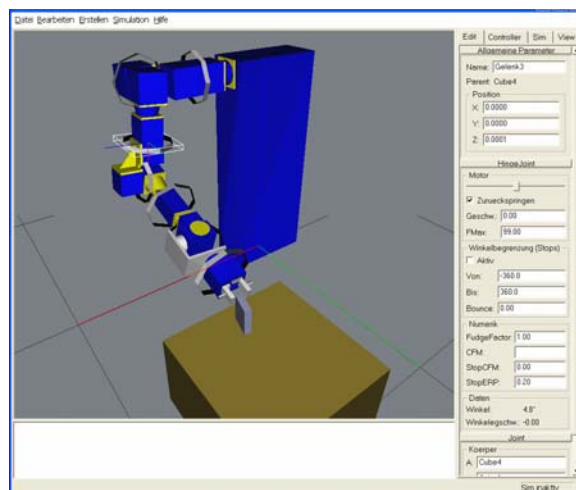


Abbildung 2: Simulation eines Armtec-Arms in der prototypischen VR-Umgebung

7 Danksagung

Wir bedanken uns bei den Mitarbeitern der verschiedenen Teilgruppen des SFB 588 „humanoide Roboter“ für die gute Zusammenarbeit und die intensive fachliche Kommunikation bei der Durchführung des Projektes.

Literatur

- [1] A. Bottino und A. Laurentini. Experimenting with nonintrusive motion capture in a virtual environment. In *The Visual Computer*, Bd. 17, S.14-29. Springer-Verlag 2001.
- [2] M. Fautz. *Objekt- und Texturrekonstruktion mit einer robotergeführten Kamera*. Dissertation, Universität Karlsruhe (TH), Fakultät für Informatik, Oktober 2002.
- [3] J. Frahm, J. Evers-Senne und R. Koch. Network Protocol for Interaction and Scalable Distributed Visualization. In *Proceedings of the First International Symposium on 3D Data Processing Visualization and Transmission (3DPVT02)*, 2002.
- [4] A. Laurentini. The visual hull concept for silhouette-based image understanding. In *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 16, pp. 150-162, 1994.
- [5] M. Gomez. *Hardware-in-the-Loop Simulation*. www.embedded.com/story/OEG20011129S0054, 2001 (gesehen am 18.04.2002).
- [6] J. Leddin. *Simulation Takes Off with Hardware*. www.embedded.com/story/OEG20020321S0028, 2002 (gesehen am 18.04.2002).
- [7] W. Matusik, C. Buehler und L. McMillan. Polyhedral Visual Hulls for Real-Time Rendering. In *Proceedings of the Twelfth Eurographics Workshop on Rendering*, S. 115–125, Juni 2001.
- [8] S. Moezzi, L. Tai und P. Gerard. Virtual View Generation for 3D Digital Video. In *IEEE MultiMedia*, 4, Nr. 1, S. 18-26, Januar-März 1997.
- [9] S. Penny, J. Smith und A. Bernhardt. Traces: Wireless full body tracking in the CAVE. In *Ninth International Conference on Artificial Reality and Telexistence (ICAT'99)*, Dezember 1999.
- [10] M. Roth. *Integration paralleler Rendering-Verfahren für lose gekoppelte Systeme in OpenSG*. Technischer Bericht, Fraunhofer-Institut für Graphische Datenverarbeitung -IGD-, Darmstadt, 2002.
- [11] F. Schelling. *Anwendung von Massenpunktverfahren für die Echtzeitsimulation eines humanoiden Roboterarms*. Diplomarbeit, Universität Karlsruhe, Mai 2002.
- [12] A. Schmitt und S. Thüring. *Ein vereinfachtes numerisches Verfahren für die mechanische Simulation in Virtual-Reality-Systemen*. Interner Bericht 2000-26, Universität Karlsruhe, Fakultät für Informatik, Dezember 2000.