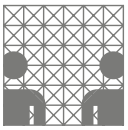
Diplomarbeit
im Studiengang Informatik

Physikbasierter Simulator für Greif- und Manipulationsverfahren mit Mehrfinger-Roboterhänden

am Arbeitsbereich für
Technische Aspekte Multimodaler Systeme,
Fachbereich Informatik,
Universität Hamburg

vorgelegt von
Hanno Scharfe

Dezember 2010



betreut von
Dr. Norman Hendrich
Prof. Bernd Neumann, Ph.D.

Abstract

This diploma thesis describes the development of a new simulator for grasping and manipulations in the field of service robotics. There are many technological improvements in the hardware. New complex robot hands have been developed, which feature similar possibilities as the human hand does. Existing simulators are not able to model the behaviour of these hands adequately. The simulator I developed permits this modelling by using rigid body dynamics.

Zusammenfassung

In dieser Diplomarbeit ist die Entwicklung eines neuen Simulators für Greif- und Manipulationsverfahren in der Servicerobotik dokumentiert. Es gibt dort viele Fortschritte im Bereich der Hardware. Insbesondere werden komplexere Roboterhände entwickelt, die ähnliche Fähigkeiten wie die menschliche Hand besitzen sollen. Existierende Simulatoren sind aber nicht in der Lage, eine angemessene Modellierung für das dynamische Verhalten dieser Roboterhände zu bieten. Der in dieser Diplomarbeit entwickelte Simulator ermöglicht diese Modellierung durch Verwendung der „Mechanik starrer Körper“.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Robotik beschäftigt sich mit von Menschen geschaffenen künstlichen Geräten, die selbständig komplexe Aktionen durchführen können. Erste mechanische Roboter wurden dabei schon in der Antike entworfen und teilweise auch hergestellt, allerdings waren deren Möglichkeiten damals noch sehr beschränkt. Einen wirklichen Durchbruch in der Roboterentwicklung gab es durch die Erfindung der programmierbaren Rechenanlagen. Seitdem ist man in der Lage, praktisch beliebige Algorithmen zu schreiben, die dann die Roboter steuern können. Heutzutage sind Roboter insbesondere in der Industrie unverzichtbar geworden. Hier werden sie für Aufgaben eingesetzt, die immer wiederholt durchgeführt werden sollen und bei denen eine hohe Genauigkeit und teilweise auch eine große Kraft benötigt wird. Diese Roboter arbeiten aber nur in sehr einfach strukturierten Umgebungen und können oft nicht ohne eine vom Menschen durchgeführte Umprogrammierung an eine neue Umgebung angepasst werden.

Ganz anders sind hingegen die Serviceroboter. Hierbei handelt es sich um Roboter, die sich auch unter Menschen aufhalten sollen. Sie müssen sich dafür in einer unbekannten, an den Menschen angepassten Umgebung zurechtfinden können, es darf keine Gefahr von ihnen ausgehen und sie müssen die Möglichkeit besitzen, mit Menschen zu interagieren. Hierzu ist allerdings noch eine große Menge an Grundlagenforschung erforderlich, da der Mensch den Robotern bei vielen alltäglichen Aufgaben wie dem Erkennen von Gegenständen, dem Bewegen auf zwei Beinen und dem Einsatz von Händen um Gegenstände zu ergreifen und als Werkzeuge zu verwenden noch deutlich überlegen ist. In einfachen Situationen sind diese Aufgaben bereits für Roboter durchführbar, aber bis Roboter die erforderliche Flexibilität erreichen, um sich unter Menschen aufhalten zu können, ist es noch ein weiter Weg. Die Forschung konzentriert sich deshalb auch bisher immer auf kleine, überschaubare Teilbereiche, die dann später einmal zu einem wirklich intelligenten System zusammengesetzt werden könnten.

Das europäische Forschungsprojekt HANDLE [HAN08], in dessen Kontext diese Diplomarbeit erstellt wurde, hat das Ziel, einen Roboter zu entwickeln, der ähnlich wie ein Mensch Gegenstände greifen und verwenden kann. Dazu werden menschliche Griffe aufgezeichnet, segmentiert und kategorisiert, um genauer zu verstehen, wie Menschen bestimmte Gegenstände greifen. Danach soll dieses Verhalten auf eine menschenähnliche Roboterhand, die Shadow Hand, übertragen werden. Durch Lernverfahren können die dafür notwendigen Handbewegungen dann verfeinert werden, um auch bei unbekannten Gegenständen ein sicheres Greifen zu ermöglichen.

1.1 Motivation

Für einen Menschen ist das Greifen von Gegenständen und die Verwendung dieser Gegenstände beispielsweise als Werkzeuge oftmals eine triviale Aufgabe, über die man meistens überhaupt nicht weiter nachdenkt. Auch vorher unbekannte Gegenstände kann man oft problemlos verwenden. Für einen Roboter ist jedoch schon das Greifen eine sehr schwierige Aufgabe, die bisher nur in wenigen Fällen überhaupt möglich ist. Oftmals fallen die Gegenstände dem Roboter aus der Hand oder die erreichten Griffe sind nicht besonders stabil. Auch kann es leicht vorkommen, dass gegriffene Gegenstände beschädigt werden. Das Verwenden von Gegenständen ist für Roboter sogar noch schwieriger und bisher auch nur sehr wenig erforscht. Für den Menschen alltägliche Aufgaben wie mit einem Schlüssel eine Tür zu öffnen, mit einem Stift zu schreiben oder ein Telefon zu bedienen sind für heutige Roboter eine noch nicht lösbare Aufgabe.

Die Probleme der Roboter haben dabei mehrere Ursachen. Eine Schwierigkeit ist die Hardware. Roboterhände sind bei weitem nicht so gut an die Umgebung angepasst wie die menschliche Hand. Die menschliche Hand besitzt sehr viele Freiheitsgrade, die größtenteils unabhängig und sehr genau kontrolliert werden können. Dies ist bei den meisten Roboterhänden noch nicht der Fall [BLMV08]. Entweder sie besitzen nur wenige Freiheitsgrade oder die Ansteuerung ist sehr träge und ungenau. Ein weiteres Problem ist die Sensorik. Die Haut der menschlichen Handinnenfläche besitzt etwa 60 Fühlkörperchen pro cm^2, an den Fingerspitzen sind es sogar um die 240 pro cm^2 [JV79]. Mit diesen Fühlkörperchen können unter anderem Berührungen, Druck, Vibrationen und Temperaturen festgestellt werden. Die besten Sensoren für Roboterhände haben hingegen derzeit maximal etwa 16 fühlende Flächen pro Fingerspitze, die auch nur auf Druck reagieren. Meistens existiert sogar nur ein einziger Sensor pro Fingerspitze.

Auch weitere Sensoren sind ein Problem. Menschen können schon über das Ansehen eines Gegenstandes oft eine genaue Vorstellung davon erhalten, wie sie diesen greifen können. Für einen Computer ist dagegen das Rekonstruieren von dreidimensionalen Szenen aus zweidimensionalen Bildern und das Extrahieren der vorhandenen Objekte mit ihren Formen, sowieso andererseits auch die Kombination mehrerer Sensoren zu einer multimodalen Weltsicht bisher nur für wenige einfache Fälle gelöst.

Ein weiterer Vorteil des Menschen besteht in der langen Lernphase. Neugeborene Babys sind noch nicht in der Lage, Objekte gut zu greifen und stabil in den Händen zu halten. Es dauert mehrere Jahre, in denen die Genauigkeit beim Umgang mit den Händen trainiert werden muss. Derartig lange Lernzyklen sind bei Robotern allein schon wegen der Abnutzung der Hardware nicht durchführbar, zudem existieren keine künstlichen Lernprogramme, die dies durchführen könnten.

Die Simulation von Robotern kann bei allen diesen Punkten behilflich sein. Zum Einen ist es hier möglich, derzeit noch nicht realisierbare Hardware virtuell zu erzeugen und zu verwenden. Auch die Simulation von hochauflösenden, genauen Sensoren ist leicht machbar. Zum Anderen ist durch die Simulation bereits ein internes Welt-

modell vorhanden, welches ja für die Simulation verwendet wird. Dieses Modell kann als Eingabe für Lernverfahren dienen und so genaue Informationen über Position, Form und Bewegungen von Gegenständen liefern, ohne dass dies aus Kamerabildern rekonstruiert werden müsste. Auch die Lernzeiten lassen sich deutlich erhöhen, dadurch dass man die Simulation auf mehreren Computern gleichzeitig durchführen kann. Insbesondere weil durchschnittliche Computer um Größenordnungen günstiger sind als die aufwändige Roboterhardware, bietet dies Vorteile.

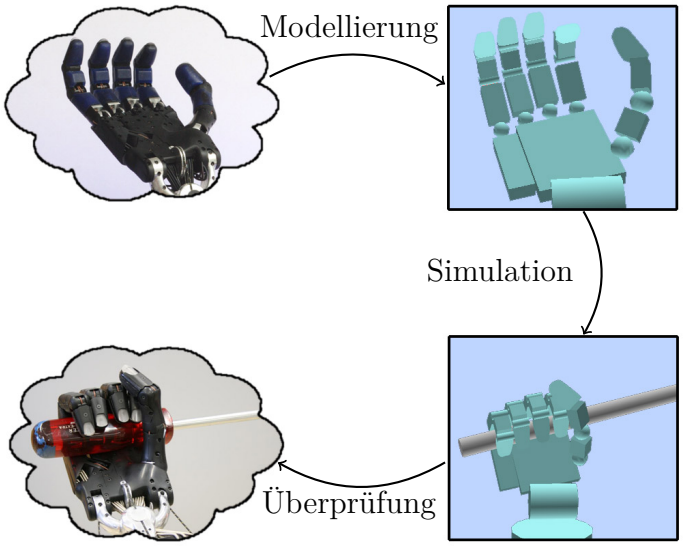


Abbildung 1.1: Modellierung und Simulation

Für die Simulation existieren hierbei viele verschiedene Abstraktionen der Realität, die auf unterschiedlichen Ebenen für unterschiedliche Zwecke arbeiten. Oftmals wird vollständig von dem physikalischen Verhalten der Umgebung abstrahiert, wodurch man Planungsalgorithmen testen kann. Oder es wird nur ein kleiner Teil der Physik modelliert, wie beispielsweise die Erkennung von Kollisionen und das Berechnen von möglichen ausgeübten Kräften in einer statischen Umgebung, wobei man aber auf ein dynamisches Verhalten der Welt verzichtet. Es gibt auch sehr genaue Simulationen, wie sie im Flugzeugbau und im Fahrzeugbau zur Berechnung der Stabilität und der Vorhersage des Verhaltens bei Unfällen verwendet werden. Jede dieser Abstraktionen ist nur für einige Einsatzzwecke verwendbar, da eine genauere Simulation der Realität auch zu einem größeren Aufwand bei der Berechnung führt. Eine zu genaue Darstellung läuft daher zu langsam ab oder benötigt aufwändige Hardware.

Vorhandene Simulatoren für Roboterhände führen üblicherweise Bewegungen nur über kinematische Ketten aus. Es wird dabei vernachlässigt, dass die Bewegung eines Gelenks auch zu Kräften auf die anderen Gelenke führt. Bei einfach aufgebauten Roboterhänden mit steifen Gelenken ist diese Modellierung angemessen, da eine gewisse Kraft auf ein Gelenk keine Auswirkungen zur Folge hat. Bei Mehrfinger-

Roboterhänden, die zusätzlich Gelenke mit einer nur geringen Steifigkeit besitzen, haben die Kräfte aber durchaus einen Einfluss auf das Verhalten. Es kann sich dadurch beispielsweise ein Gelenk von dem eingestellten Zielwinkel weg bewegen, da die ausgeübte Kraft nicht mehr ausreicht, um die gewünschte Position zu halten. Es kann auch vorkommen, dass ein Finger gegen einen anderen drückt und diesen damit bewegt.

Die in dieser Diplomarbeit verwendete Abstraktion der Physik kann es ermöglichen, derartige Kopplungen realitätsnah abzubilden. Dadurch sollen dann Experimente zum Greifen und Manipulieren von Gegenständen mit den komplexen Roboterhänden simulierbar werden.

1.2 Fragestellung

Die Fragestellung, die in dieser Diplomarbeit beantwortet wird, ist, inwieweit die in vielen Computerspielen verwendeten Echtzeit-Physiksimulatoren für die Simulation von Manipulationen mit Robotern verwendbar sind. Insbesondere sind hierbei Roboter mit Mehrfinger-Händen von Interesse, da deren Verhalten bisher nicht realitätsnah simuliert werden kann. Dazu muss zuerst ein Simulator entwickelt werden, der diese Physiksimulation verwendet und darüber hinaus wichtige Abstraktionen wie Sensoren, Motoren und Motoransteuerungen, die für die Robotik benötigt werden, bietet. Da in der Robotik, insbesondere bei der Verwendung von komplexen Roboterhänden, deutlich höhere Anforderungen an die Genauigkeit gestellt werden als bei Computerspielen, ist dann zu prüfen, wie genau und wie stabil die Ergebnisse dieser Simulation sind.

1.3 Ziele dieser Arbeit

Das erste Ziel dieser Arbeit ist die Entwicklung eines Robotersimulators. Da es mehrere Physiksimulations-Bibliotheken gibt, die frei verfügbar sind, ging es zuerst darum, eine dieser Bibliotheken auszuwählen. Die Bibliotheken mussten dafür untersucht und bewertet werden. Danach sollte ein Modell des im Arbeitsbereich TAMS (Technische Aspekte Multimodaler Systeme) der Universität Hamburg vorhandenen Roboters erstellt werden, welches bei dieser Simulation verwendet werden kann. Für diesen simulierten Roboter und den real existierenden Roboter sollte jeweils die gleiche Schnittstelle zur Ansteuerung geschrieben werden. Dadurch ist es leicht möglich, vergleichende Experimente durchzuführen. Zuletzt sollte dann die Qualität der Simulation unter verschiedenen Gesichtspunkten untersucht werden.

Bei dem Aufbau des Simulators war es wichtig, dass möglichst einfach weitere Gegenstände in die Simulation eingefügt werden können und auch die genaue Konfiguration des Roboters flexibel ist. Dies ist notwendig, da der Versuchsaufbau des Roboters derzeit noch nicht endgültig ist. Unter anderem ist es geplant, den Roboterarm, der aktuell auf dem Boden steht, möglicherweise an der Wand oder an der

Decke zu befestigen. Außerdem sollen an der Roboterhand unterschiedliche taktile Sensoren getestet werden. Während zum Beginn der Diplomarbeit noch keine taktilen Sensoren vorhanden waren, existieren derzeit relativ ungenaue Sensoren, die aber auch wieder ersetzt werden sollen. Auch in der Simulation sollen diese Änderungen ohne zu viel Aufwand möglich sein.

Nach der Erstellung des Simulators sollten dann verschiedene Experimente in der Simulationsumgebung durchgeführt werden. Dazu gehören unter anderem Versuche, bei denen Gegenstände bewegt werden. Hierbei ist eine formale Überprüfung schwer, weshalb nur überprüft wurde, ob bestimmte Aktionen überhaupt möglich waren. Weiterhin sollte untersucht werden, welchen Einfluss die verschiedenen Parameter der Physikbibliothek auf die Genauigkeit und die Geschwindigkeit der Simulation haben.

1.4 Verwandte Arbeiten

Es gibt in der Robotik eine Reihe von Simulatoren, die auf unterschiedliche Bereiche spezialisiert sind. Meistens wird bei diesen aber keine Dynamik simuliert, sondern die Roboter können nur mittels kinematischer Ketten in eine bestimmte Position gebracht werden, die dann untersucht wird.

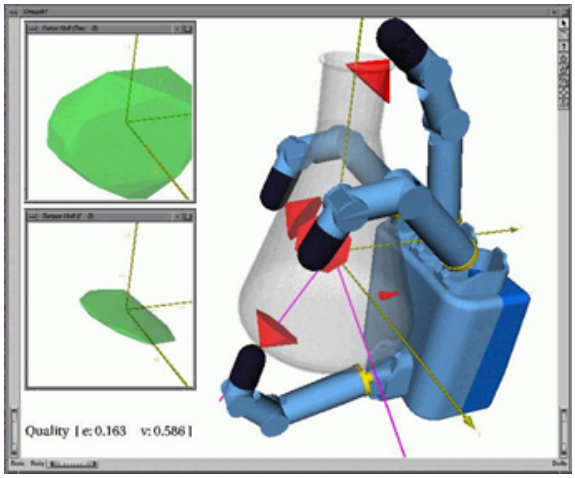


Abbildung 1.2: GraspIt!

Zu dieser Kategorie gehört GraspIt! [MM04]. GraspIt! kann, wenn die Position eines Gegenstandes und die Position einer Roboterhand bekannt sind, bestimmen bei welchen Gelenkstellungen die Finger der Roboterhand den Gegenstand berühren. Danach kann berechnet werden, ob der gewählte Griff stabil ist. Dies wird mit der force-closure-Methode bestimmt, bei der für jeden Kontaktpunkt bestimmt wird, welche Kräfte auf diesen Punkt wirken und ob die Kräfte die mögliche Normalkraft und die Reibungskräfte an diesem Punkt nicht überschreiten. Insbesondere wird

bei GraspIt! viel Wert auf eine genaue Beschreibung der Materialien gelegt, wobei auch „weiche“ Gegenstände mit gewissen Einschränkungen simuliert werden können. Komplexe Roboterhände können mit GraspIt! nur eingeschränkt simuliert werden, da keine dynamische Simulation der Hände möglich ist. In dem Screenshot zu GraspIt! ist ein gefundener Griff abgebildet. Die Kegel an den Kontaktstellen sind um so breiter, je größer die Reibung an dem Kontakt ist. Sie geben damit an, in welchem Bereich die Kräfte an dem Kontaktpunkt liegen müssen, um einen stabilen Griff zu erhalten.

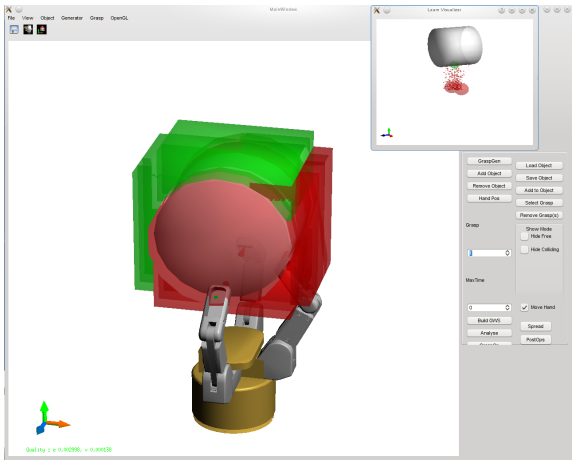


Abbildung 1.3: zgrasp

Der Simulator zgrasp [BL08] ist auf das Lernen von Griffen mit der Barrett-Hand optimiert. Es verwendet ähnlich wie GraspIt! ein kinematisches Modell der Robterhand. In diesem Simulator ist ein Algorithmus zum Bestärkenden Lernen implementiert, mit dem sehr schnell stabile Griffe gefunden werden können. Als Bewertungsfunktion hierfür kann nicht nur die Stabilität eines Griffes, sondern zusätzlich die Weiterverwendbarkeit nach dem Griff verwendet werden. In dem Screenshot zu zgrasp wird ein Griff der Barrett-Hand an einem Zylinder dargestellt. Die Farben geben hier an, welcher Bereich des Gegenstandes „frei“ für Interaktionen ist. Im grünen Bereich ist der Gegenstand nicht durch den gewählten Griff blockiert.

Ein weiterer Simulator, der sich allerdings derzeit noch in einem frühen Entwicklungsstadium befindet, ist OpenRAVE [Dia10]. Hier liegt der Fokus eher im Bereich der Planung. Es können durch die Verwendung von inverser Kinematik Bewegungspfade bestimmt und automatisch auf Kollisionen überprüft werden, wodurch sich ein darauf aufbauender Planungsalgorithmus um diese Probleme nicht kümmern muss. In OpenRAVE ist auch die Möglichkeit einer dynamischen Simulation vorhanden. Der Screenshot zu OpenRave zeigt einen Roboter, der eine geplante Manipulation durchführt. Die Simulation wird hierbei aber nur kinematisch durchgeführt. Es werden also keine Kräfte berücksichtigt. Die gegriffene Tasse ist in dem Fall in die

kinematische Kette des Roboterarms eingebaut und wird dadurch mit dem Arm mitbewegt.

Auf OpenRAVE baut das OpenGRASP Toolkit [LUD+10] auf. Dieses wird im Kontext des GRASP-Projektes [GRA08] entwickelt und bietet Funktionen um stabile Griffe zu erkennen und diese in den Planungsalgorithmen von OpenRAVE zu verwenden. Für OpenGRASP ist eine dynamische Physiksimulation geplant, aber bisher nicht realisiert.

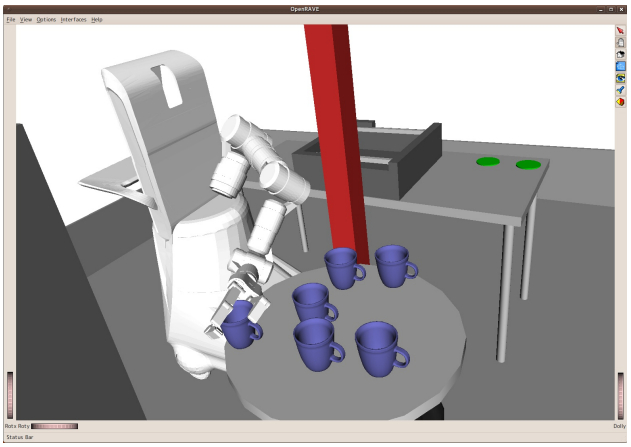


Abbildung 1.4: OpenRave

Im Rahmen des Player Project wird der Robotersimulator Gazebo [KH05] entwickelt. Dieser Simulator verwendet die Physikbibliothek ODE und ist auf mobile Roboter ausgerichtet. Hierfür sind verschiedene virtuelle Sensoren wie Laserscanner und Kameras verfügbar. In dem Screenshot zu Gazebo ist ein mobiler Roboter mit zwei Sensoren abgebildet. In dem Ausgabefenster des Laserscanners sieht man den gemessenen Abstand in der Messebene. In der Ausgabe der Kamera wird das aus der Position gesehene Bild dargestellt.

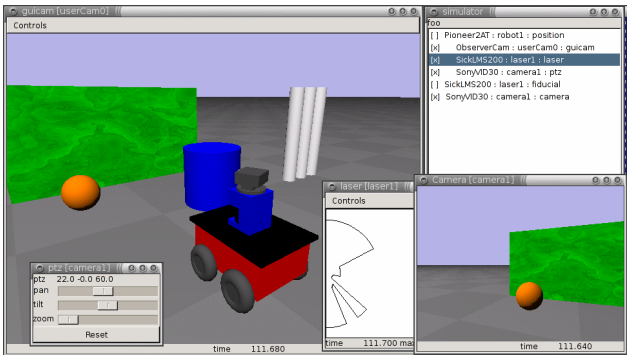


Abbildung 1.5: Gazebo

1.5 Aufbau dieser Arbeit

In dieser Arbeit wird die Entwicklung eines Robotersimulators dargestellt, der auf das Greifen und Manipulieren mit Mehrfinger-Roboterhänden spezialisiert ist.

In Kapitel 2 wird auf dieses Thema hingearbeitet. Zuerst wird dabei beschrieben, was das Greifen von Gegenständen von dem Manipulieren unterscheidet. Es wird dabei auch dargelegt, wieso das Manipulieren in existierenden Simulatoren nicht angemessen modelliert werden kann. Anschließend werden wichtige theoretische Grundlagen für den Aufbau des Simulators erklärt.

Kapitel 3 beschreibt den Versuchsaufbau, der simuliert werden soll. Der zu entwickelnde Simulator soll dabei allerdings nicht auf diesen Versuchsaufbau beschränkt sein, sondern auch andere Hardware unterstützen.

In Kapitel 4 wird aufbauend auf der zu verwendenden Hardware und dem theoretischen Hintergrund ein Entwurf für den Simulator beschrieben. Es werden dafür zuerst die wichtigsten Anforderungen dargestellt und dann der Entwurf entwickelt. Einige Details dieses Entwurfes werden dabei genauer beschrieben.

In Kapitel 5 wird dargestellt, welche existierende Software für die Realisierung des Simulators verwendet werden soll. Die getroffene Auswahl wird dort auch begründet. Insbesondere die Auswahl der zu verwendenden Physikbibliothek ist dabei für den Aufbau des Simulators entscheidend.

In Kapitel 6 wird dann die Realisierung des Entwurfes beschrieben. Hierbei wird nur auf die wichtigen Details eingegangen. Es werden auch einige Probleme bei der Umsetzung und deren Lösung dargestellt.

In Kapitel 7 wird dann das entwickelte System getestet. Es werden mehrere Testreihen durchgeführt, um wichtige Eigenschaften des entwickelten Simulators zu überprüfen, und es werden einige Manipulationsexperimente dargestellt.

Abschließend wird das Ergebnis dieser Arbeit kurz dargestellt und es werden Möglichkeiten beschrieben, auf dieser Arbeit aufzubauen.

2 Theoretische Grundlagen

In diesem Kapitel werden die Grundlagen erläutert, die für das Verständnis dieser Arbeit erforderlich sind. Zuerst wird ein Überblick über die technische Entwicklung von Roboterhänden gegeben und es wird der aktuelle Forschungsstand im Bereich des automatisierten Greifens und Manipulierens von Gegenständen beschrieben. Danach wird definiert, was unter einem Modell und einer Simulation zu verstehen ist und es wird dargestellt, wie reale Gegenstände mit ihren Interaktionen modelliert werden können. Hierbei wird auch auf einige wichtige physikalische Eigenschaften dieser Gegenstände eingegangen. Es wird dargestellt, wie die „Mechanik starrer Körper“, die für den Simulator verwendet werden soll, aufgebaut ist und wie eine Simulation damit durchgeführt werden kann. Anschließend wird auf einige wichtige Sätze der analytischen Geometrie eingegangen, die an verschiedenen Stellen dieser Arbeit verwendet werden und es wird ein kurzer Überblick über die Regelungstechnik gegeben, da zur Ansteuerung der Motoren der Roboterhand Regler benötigt werden.

2.1 Objektmanipulationen mit Robotern

2.1.1 Roboterhände

Der Aufbau der Hände eines Roboters bestimmt zu einem großen Anteil, mit welchen Gegenständen Interaktionen möglich sind. Bei Industrierobotern, die mit Greifern ausgestattet sind, haben diese Greifer eine Form, die genau an die zu greifenden Gegenstände angepasst ist. Diese Gegenstände lassen sich dann sehr schnell und stabil greifen. Sobald aber ein Gegenstand eine geringfügig abweichende Form besitzt, muss der Greifer ausgetauscht werden. Derartige Greifer besitzen meistens nur wenige Freiheitsgrade und auch nur wenige Sensoren, weil die Umgebung und auch die Positionen der Gegenstände vor dem Greifen meist ausreichend genau bekannt sind.

Serviceroboter hingegen sollen möglichst beliebige Gegenstände in einer „natürlichen“ Umgebung greifen können. Hierzu ist ein deutlich komplexerer Aufbau der Hand notwendig, da diese Gegenstände in sehr vielen Formen vorliegen können. Auch an die Sensorik werden höhere Anforderungen gestellt. Bei der Konstruktion der Hände für Serviceroboter orientiert man sich dabei meistens an der menschlichen Hand. Da der Aufbau der menschlichen Hand aber zu komplex für eine Realisierung als Roboterhand ist, muss man in einigen Bereichen Vereinfachungen vornehmen [SK08].

Das wichtigste Merkmal bei der Struktur der menschlichen Hand sind die ein-

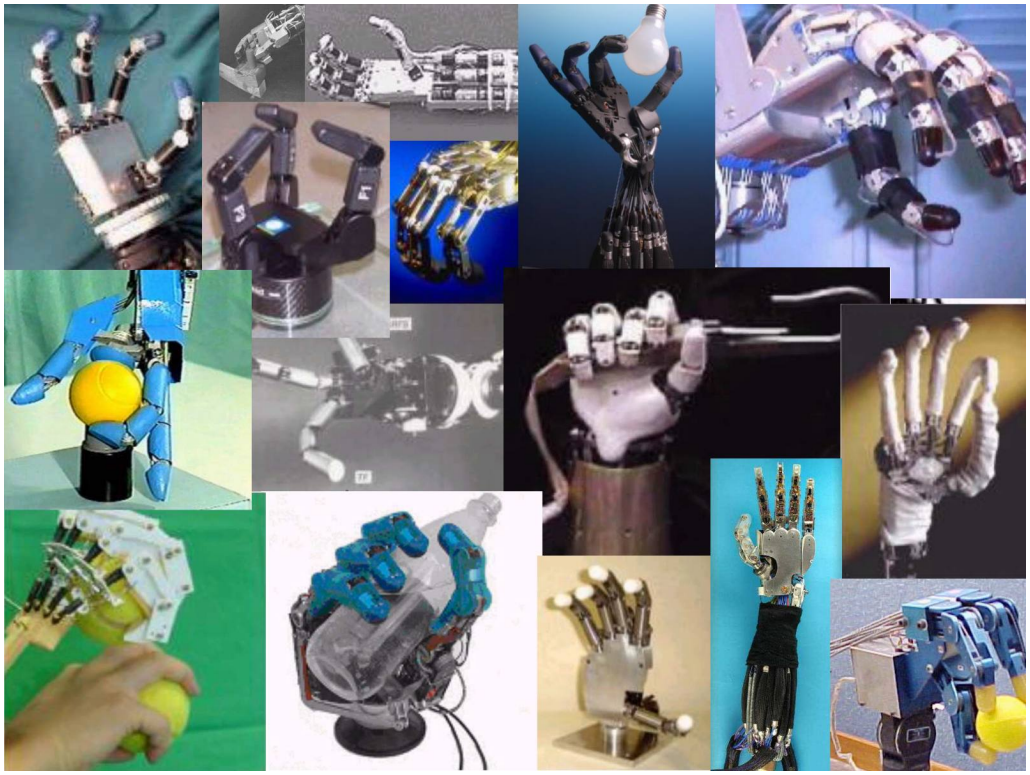


Abbildung 2.1: Verschiedene Hände für Serviceroboter (aus [BLMV08])

zeln beweglichen Finger und die Lage des Daumens. Dadurch, dass der Daumen im Vergleich zu den anderen Fingern entfernter positioniert ist und auch in Opposition zu den anderen Fingern gestellt werden kann, werden viele Griffe erst möglich. Bei den restlichen vier Fingern, die relativ gleichmäßig aufgebaut sind, kann man dagegen Vereinfachungen vornehmen, ohne die Fähigkeiten der Hand deutlich zu reduzieren. Deshalb besitzen viele Roboterhände einen Finger, der wie der Daumen beim Menschen unabhängig von den restlichen Fingern ist, und dann mindestens zwei weitere Finger, die teilweise aber nur gemeinsam bewegt werden können. Auch die Freiheitsgrade der einzelnen Finger werden oft reduziert. Beim Menschen besitzt jeder Finger drei Gelenke, mit denen er eingeknickt werden kann, und einen Freiheitsgrad, mit dem er ein wenig zur Seite gekippt werden kann. Bei Roboterhänden wird dieses oft auf einen oder zwei Freiheitsgrade reduziert. Es gibt aber auch komplexere Roboterhände, die fast alle Freiheitsgrade der menschlichen Hand besitzen. Bei diesen versucht man teilweise auch, die Form der menschlichen Hand möglichst genau nachzubilden.

Die menschliche Hand besitzt außerdem sehr viele taktile Sensoren. Dadurch ist es möglich, die Stabilität eines Griffes genau festzustellen und zu verändern. So ist man oftmals in der Lage, Gegenstände, die einem fast aus der Hand rutschen, doch noch zu halten. Man kann diese genauen Kontrollmöglichkeiten auch dafür nutzen,

einen Gegenstand absichtlich leicht in der Hand verrutschen zu lassen, um diesen in eine neue Position zu bringen. Bei Robotern sind diese sensorischen Fähigkeiten oftmals nur sehr eingeschränkt vorhanden, da die Herstellung von sehr kleinen und genauen Sensoren sehr aufwändig ist. Viele Roboterhände sind daher ganz ohne taktile Sensoren ausgestattet, oder es ist nur ein einzelner Sensor an jeder Fingerspitze vorhanden. Damit kann dann zwar festgestellt werden, ob ein Gegenstand überhaupt berührt wird und ob man den Druck auf einen Gegenstand noch erhöhen kann. Für viele Aktionen reichen diese sensorischen Fähigkeiten aber noch lange nicht aus. Es gibt auch in diesem Bereich einige neue Entwicklungen. Durch neuartige Materialien wird versucht, die Sensoren kleiner zu bauen und trotzdem die Haltbarkeit und die Messgenauigkeit zu erhalten. In [TW05] ist ein ausführlicher Überblick über verschiedene Sensorsysteme, die bei Robotern eingesetzt werden oder sich derzeit in der Entwicklung befinden.

Ein weiterer wichtiger Punkt ist die Möglichkeit, auf externe Kräfte durch Nachgeben zu reagieren. Dies wird auch als *Compliance* bezeichnet (Compliance ist übersetzbar als Dehnbarkeit oder als reziproke Steifigkeit). Roboter mit geringer Compliance, die also eine hohe Steifigkeit besitzen, ermöglichen zwar eine genauere Kontrolle der Position, aber bei möglichen Kollisionen kann es leicht zu Beschädigungen am Roboter oder der Umwelt kommen. Compliance kann man durch weniger steife Gelenke oder durch flexible Bauteile zu erzielen. Beim Menschen ist diese Flexibilität durch die Muskeln gegeben, die normalerweise nur leicht angespannt sind und daher schnell zu einem Nachgeben an den Gelenken führen. Durch eine stärkere Anspannung ist es aber trotzdem möglich, genaue und stabile Positionen zu erreichen. Außerdem sind einige Sensoren durch Reflexe direkt an die Steuerung der Muskeln gebunden. Dies ermöglicht sehr schnelle Reaktionen. Bei Robotern kann man ähnliches durch den Einsatz von pneumatischen Muskeln erreichen, da diese durch ihren Aufbau prinzipiell weniger steif sind und bei starker Belastung nachgeben können. Es ist aber auch eine genaue Kontrolle dieser Muskeln wichtig, um die Anspannung schnell zurücknehmen zu können, wenn dies notwendig wird. In [SSK08] wird beschrieben, wie durch die Verwendung von pneumatischen Muskeln die Gefahr für die Umgebung, die von einem Roboter ausgeht, reduziert werden kann. Es ist auch möglich, mit klassischen Antrieben durch Servomotoren eine akzeptable Compliance zu erreichen. Dazu müssen über Sensoren die auftretenden Kräfte genau erfasst werden, um dann sehr schnell in eine Reaktion umgesetzt werden zu können. Dieses Verfahren wird in [WP80] dargestellt.

Es gibt eine große Anzahl unterschiedlicher Roboterhände, die diese einzelnen Punkte unterschiedlich gut erfüllen. In [BLMV08] werden verschiedene aktuelle Roboterhände verglichen und nach verschiedenen Kategorien bewertet. In [Bic02] werden wichtige Ziele bei der Entwicklung von Roboterhänden dargestellt und es werden auch alternative Ansätze für das Design von Roboterhänden besprochen.

2.1.2 Greifen von Gegenständen

Das Greifen von Gegenständen ist Teil vieler aktueller Forschungsarbeiten. Oftmals werden Computermodelle der Gegenstände erzeugt, woraufhin dann das Modell mittels unterschiedlicher Verfahren auf mögliche Griffe untersucht wird. Von welchen Positionen aus Griffe ausprobiert werden kann dabei von der Form des Gegenstandes abhängen. Positionen, an denen ein Griff sowieso nicht stabil sein kann, werden dabei weggelassen. Auch braucht man an Stellen, die von der Roboterhand nicht erreichbar sind, keine Griffe auszuprobieren.

Da es den Servicerobotern möglich sein soll, praktisch beliebige Gegenstände zu greifen und zu verwenden, müssen diese Gegenstände auf mögliche Griffe untersucht werden. Bei unbekannten Gegenständen ist es sinnvoll, Griffe bekannter Gegenstände mit ähnlicher Form auszuprobieren. Diese können dann oftmals an den neuen Gegenstand angepasst werden. Oft reicht es auch, wenn ein Teil eines Gegenstandes erkannt wird, dies kann beispielsweise ein Henkel sein. Zusätzlich muss es aber auch möglich sein, vollständig unbekannte Gegenstände zu greifen. Hierfür kann man verschiedene mögliche Griffe ausprobieren und auf ihre Stabilität überprüfen. Sinnvoll ist dabei eine systematische Herangehensweise, um durch bestimmte Eigenschaften des Gegenstandes wie Symmetrien, unterschiedliche Materialeigenschaften und Formen die Anzahl der zu probierenden Griffe möglichst gering zu halten. In [MKCA03] wird beschrieben, wie man anhand generalisierter und vereinfachter Modelle Griffe an unbekannten Gegenständen ausprobieren kann.

Wichtig für ein späteres Verwenden der Gegenstände ist auch, dass man eine Einteilung nach möglichen Einsatzzwecken durchführt. Wenn man beispielsweise eine Tasse greifen will, um sie danach zu füllen, sollte man sie mit der Öffnung nach oben halten und diese Öffnung auch nicht mit der Hand verdecken. Möchte man diese Tasse jedoch in einen Schrank stellen, sollte man die Hand möglichst nicht an der Unterseite der Tasse haben, da diese ansonsten nicht sicher abgestellt werden kann. In [ASMC06] wird eine Untersuchung durchgeführt, welche Griffe Menschen an einem Gegenstand durchführen, wenn sie eine bestimmte Aktion danach durchführen sollen. Es konnte dabei eine Korrelation zwischen der geplanten Aktion und dem durchgeführten Griff festgestellt werden.

Um bei einer positionierten Hand zu überprüfen, ob von hier aus ein stabiler Griff möglich ist, müssen zuerst die verschiedenen Fingergelenke in die richtige Position gebracht werden. Bei Roboterhänden mit wenigen Freiheitsgraden ist dies einfach, da alle Gelenke soweit geschlossen werden, bis jeweils eine Kollision mit dem Gegenstand auftritt. Bei Roboterhänden mit sehr vielen Freiheitsgraden ist dies jedoch nicht so einfach möglich. Beispielsweise kann bei einer menschenähnlichen Hand die Position des Daumens sehr unterschiedlich sein. Einfach die Finger zu schließen, wird daher nicht funktionieren. Es kann auch einen Unterschied machen, ob man zuerst die äußeren oder die inneren Fingergelenke schließt. Ein Durchprobieren aller Möglichkeiten ist bei Händen mit vielen Freiheitsgraden also oftmals zu zeitaufwändig.

Bei einer Lösung dieses Problems wird die Tatsache ausgenutzt, dass bestimmte Gelenke meistens in einem gleichen Verhältnis zueinander bewegt werden. Viele theoretisch mögliche Fingerstellungen kommen also praktisch niemals vor. Mit Hilfe einer Vielzahl von gespeicherten Griffen wird aus dem gesamten Gelenkwinkelraum ein Unterraum berechnet, der in diesen Griffen häufiger vorkommt. Dieses Verfahren wird als *„Eigengrasp“* bezeichnet und wird in [CA09] untersucht. In [SFS98] wird gezeigt, dass bei der menschlichen Hand mit nur zwei Freiheitsgraden im Eigengrasp-Raum 80 Prozent der Varianz der Handstellungen beim Greifen abgedeckt werden.

Nach dem Schließen der Finger kann dann die Stabilität des Griffes überprüft werden. Ein Verfahren, um einen Griff auf Stabilität zu überprüfen wird in [MNP90] dargestellt. Hierbei wird angenommen, dass die Fingerkontakte einen unbegrenzten Druck aushalten, aber keine Reibung existiert. Es wird dann geprüft, ob sich das gegriffene Objekt unter dieser Bedingung noch bewegen kann. Wenn eine Bewegung nicht möglich ist, wird der Griff als „form-closure“ bezeichnet, da allein durch die Form des Gegenstandes und der Kontakte eine Bewegung ausgeschlossen werden kann. In dem Artikel wird außerdem eine Methode dargestellt, wie sich derartige Griffe konstruieren lassen. Dabei wird allerdings keine Rücksicht auf die Handgeometrie genommen, sondern es wird angenommen, dass die Finger beliebig positionierbar sind. Damit „form-closure“ möglich ist, werden mindestens drei Finger benötigt.

Bei einem weiteren Verfahren, welches in [Ngu86] beschrieben wird, werden zusätzliche Reibungskräfte berücksichtigt. Es wird an jedem Kontaktpunkt ein „Stabilitätskegel“ berechnet, in dem die Kraft, mit der der Gegenstand und der Finger zusammengedrückt werden, liegen muss. Dann wird bestimmt, ob man die Kräfte so bestimmen kann, dass an jedem Kontaktpunkt die Kraft in dem Stabilitätskegel liegt und die Gesamtkräfte auf den Gegenstand ausgeglichen sind. In diesem Fall kann der Griff als „force-closure“ beschrieben werden, da durch die Kompensation der Kräfte eine Bewegung verhindert wird.

Die erreichten Griffe lassen sich einteilen in *„Power Grasps“* und *„Precision Grasps“*. Bei einem Power Grasp wird der gegriffene Gegenstand nicht nur von den Fingern gehalten, sondern auch auf die Handfläche gedrückt. Durch die große Kontaktfläche, die die Hand mit dem Gegenstand hat, sind so sehr stabile Griffe möglich. Bei einem Precision Grasp hingegen wird der Gegenstand nur von den Fingern gehalten, oftmals sind sogar nur die äußersten Fingerglieder daran beteiligt. Dadurch kann man sehr genaue Griffe durchführen und hat oft viele Möglichkeiten, den Gegenstand zu verwenden, da nur ein kleiner Teil des Gegenstandes durch die Hand bedeckt ist. In [Cut02] wird diese Einteilung beschrieben, wobei die Griffe noch weiter unterschieden werden.

2.1.3 Manipulationen mit Gegenständen

Als *„Manipulationen“* werden diverse Aktionen zusammengefasst, die mit einem Gegenstand möglich sind. Dazu gehört das Bewegen des Gegenstandes in der Welt,

das Bewegen des Gegenstandes in der Hand, was auch als Umgreifen bezeichnet wird, das Verändern des Gegenstandes selbst, indem man diesen beispielsweise öffnet, oder das Kombinieren eines Gegenstandes mit einem weiteren Gegenstand. Manipulationen sind Teil vieler aktueller Forschungsarbeiten im Bereich der Robotik [Mas01, MLSS94]. Die Möglichkeiten aktueller Roboter in diesem Bereich sind aber noch sehr eingeschränkt. Dies liegt zum einen an der noch zu ungenauen Hardware, insbesondere im Bereich der Sensorik, und zum anderen auch an der komplexeren theoretischen Beschreibung. Es gibt bisher nur wenige Möglichkeiten, die Qualität einer Manipulation in Bezug auf Stabilität, Genauigkeit und Wiederholbarkeit zu definieren.

In [OSC02] wird ein Überblick über die Anforderungen an einen Roboter für die Manipulation von Gegenständen gegeben, außerdem werden dort Vorschläge gemacht, wie man einen Griff darauf hin überprüfen kann, ob bestimmte Manipulationen damit möglich sind. Es wird dort vorgeschlagen, einen Griff nach einem Maß für *„Manipulierbarkeit“* zu bewerten, welches die möglichen, aufgabenbezogenen Aktionen mit dem Gegenstand bewertet.

Während bei der Simulation von Griffen eine kinematische Simulation ausreichend ist, bei der also die Zeit nicht berücksichtigt wird, ist es bei der Simulation von Manipulationen notwendig, auch die Zeit zu berücksichtigen. Es ist beispielsweise oftmals notwendig, dass zu bestimmten Zeitpunkten ein Gegenstand nicht stabil gegriffen ist, wenn man diesen Gegenstand in der Hand bewegen will. Auch kommt es bei Bewegungen sehr stark auf die zeitliche Abfolge und die Geschwindigkeit an, um die gewünschten Ergebnisse zu erhalten. Die Manipulation mit nicht stabilen Griffen wird in [LM99] behandelt. Dort wird auch dargestellt, wie man die Bewegungen eines Gegenstandes in einer Hand beschreiben kann. Mögliche Bewegungen in einer Hand sind beispielsweise das Rollen, Rutschen oder Kippen.

2.2 Modelle und Simulationen

Wenn ein Experiment an einem System in der realen Welt nicht möglich ist, beispielsweise weil die Durchführung zu aufwändig, zu teuer, zu gefährlich oder auch einfach nicht beobachtbar wäre, kann man stattdessen das Experiment an einem Modell dieses Systems durchführen [Cel91]. Über das Verhalten dieses Modells lassen sich dann Rückschlüsse auf das Verhalten des realen Systems ziehen. Bei dem Erstellen des Modells muss eine geeignete Abbildung der Realität gefunden werden, die einerseits die Komplexität genügend reduziert, aber andererseits alle für das Experiment wesentlichen Eigenschaften erhält. Ein Modell ist dabei immer zweckbezogen und kann nicht alle Eigenschaften des realen Systems nachbilden. Aber auch bei den zu untersuchenden Eigenschaften ist oftmals eine gewisse Ungenauigkeit unvermeidlich, um die Komplexität des Modells überschaubar zu halten.

Wichtige Schritte um ein Modell zu bilden sind:

- Abgrenzung von der Umgebung: Nur für das Experiment bedeutende System-

bestandteile werden modelliert.

- Dekomposition/Aggregation: Die verbleibenden Bestandteile werden so zerlegt oder zusammengesetzt, dass möglichst geringe Wechselwirkungen zwischen einzelnen Teilen bestehen.

- Detailreduktion: Details, die keine oder nur geringe Auswirkungen auf das zu untersuchende Verhalten haben, können weggelassen oder vereinfacht werden.

- Klassifikation: Ähnliche Bestandteile werden in Klassen eingeordnet, so dass möglichst viele Eigenschaften in die Klasse ausgelagert werden können und nur die Unterschiede der Teile beschrieben werden müssen.

Modelle werden unterteilt in Blackbox-Modelle und Whitebox-Modelle. Bei Whitebox-Modellen ist die interne Struktur des realen Systems vollständig bekannt, wird aber zur einfacheren Untersuchung zu einem Modell vereinfacht. Bei einem Blackbox-Modell hingegen ist nur das Verhalten des System bekannt, aber nicht seine Struktur. Daher kann auch nur das Verhalten modelliert werden, was die Aussagekraft des Modells reduziert. In der Praxis werden häufig Mischformen eingesetzt, bei denen nur ein Teil der Strukturen bekannt ist und man oft gezielt darauf verzichtet, alle möglichen Eigenschaften genau festzustellen, weil der Aufwand hierzu zu groß wäre.

Bei der Modellierung eines Roboterarms beispielsweise sind oft die genaue Masse, die Form der Bauteile und die Position der Gelenke und Achsen bekannt. Andere Eigenschaften wie Verformungen der Bauteile unter Last sind hingegen oftmals nicht bekannt und können daher auch nicht modelliert werden. Allerdings sind die Verformungen oft auch so gering, dass die Modellierung nicht notwendig und auch gar nicht erwünscht ist, damit das Modell nicht zu umfangreich wird.

2.2.1 Simulationen

Eine Simulation dient zur Untersuchung von Systemen, die zu komplex für eine exakte formelmäßige Behandlung sind. Als ein wichtiger Bereich für Simulationen existieren die dynamischen Systeme, bei denen man eine Veränderung von miteinander wechselwirkenden Objekten über die Zeit betrachtet.

Zur Simulation wird von einem System ein Modell des Startzustandes erstellt, welches dann unter festgelegten Regeln über die Zeit verändert wird. Hierbei kann man unterscheiden zwischen zeitkontinuierlichen und zeitdiskreten Simulationen. Zeitdiskret bedeutet, dass man entweder feste Zeitschritte hat, bei denen jeweils der Zustand des Systems bestimmt wird, oder dass diese Zeitschritte unterschiedlich lang sind – abhängig von, im vorherigen Zustand bestimmten, erwarteten Ereignissen. Wenn Simulationsschritte nur durch Ereignisse ausgelöst werden, nennt man die Simulation auch ereignisgesteuert. Bei zeitkontinuierlichen Simulationen hingegen berücksichtigt man auch die Veränderungen zwischen den einzelnen Zeitschritten

und die Auswirkungen dieser Veränderungen auf das System. Dies ist aber in vielen Fällen zu komplex, um in Echtzeit berechnet zu werden. Aus diesem Grund wird in dieser Arbeit nur auf zeitdiskrete Simulationen eingegangen.

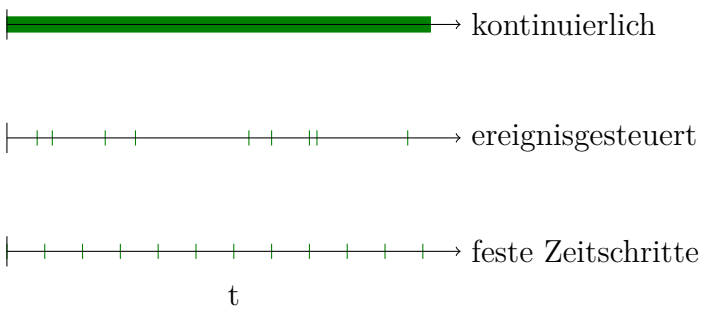


Abbildung 2.2: Zeitkontinuierliche und zeitdiskrete Simulationen

Simulationen können grundsätzlich auch ohne Computer durchgeführt werden. Insbesondere im militärischen Bereich wurden aufgrund der hohen Kosten und Risiken eines realen Konfliktes schon früh Simulationen in der Form von Planspielen oder Brettspielen eingesetzt [Pia00]. Damit konnten mögliche Taktiken ausprobiert werden und es war möglich, deren Auswirkungen abzuschätzen. Durch die Verbreitung von Computern ab der Mitte des 20. Jahrhunderts wuchs die Bedeutung von Simulationen erheblich, da sie nun mit vergleichsweise geringem Aufwand durchführbar waren. Heutzutage werden Simulationen in sehr vielen Bereichen eingesetzt. Sie dienen zur Vorhersage des Wetters, um die Stabilität architektonischer Entwürfe zu überprüfen, um Verformungen von Fahrzeugen bei Unfällen zu bestimmen und zu vielem mehr. Besonders wichtig sind Simulationen auch in den Naturwissenschaften, beispielsweise zur Überprüfung des Verhaltens komplexer chemischer Moleküle oder um die Auswirkungen quantenphysikalischer Wechelwirkungen vorherzusagen.

Allerdings sind den Simulationen auch Grenzen gesetzt. Grundsätzlich ist eine Simulation nur so gut wie das Modell, das als Grundlage dient. Es ist leicht möglich, versehentlich bei der Modellierung Eigenschaften wegzulassen, die doch deutliche Auswirkungen haben könnten. Einige Systeme zeigen außerdem ein instabiles oder sogar chaotisches Verhalten, bei denen eine kleine Veränderung der Startzustände nach einer kurzen Zeit zu sehr unterschiedlichen Zuständen führt. Hierdurch wird die Vorhersagekraft einer Simulation stark reduziert. Oftmals werden deshalb Simulationen mit kleinen Veränderungen der Startzustände wiederholt ausgeführt, wodurch sich ein chaotisches Verhalten mit großer Wahrscheinlichkeit erkennen lässt. Außerdem kann man dadurch bei einem nicht chaotischen Verhalten feststellen, mit welcher Wahrscheinlichkeit ein bestimmtes Ereignis oder ein bestimmter Endzustand zu erwarten ist. Diese statistische Untersuchung des Verhaltens wird wird auch als *„Monte-Carlo-Simulation“* bezeichnet.

2.3 Physikalische Modelle

Alle physikalischen Theorien sind Modelle, die einen Teil der Realität mit guter Genauigkeit annähern. Für die Implementierung eines Simulators ist es daher sinnvoll, von diesen Modellen auszugehen und sie soweit zu vereinfachen, dass eine Simulation mit akzeptablem Aufwand möglich ist. Als Ansatz bietet sich bei der Robotik die *„klassische Mechanik“* [Str87] an, da relativistische und quantenmechanische Eigenschaften hier vernachlässigbar sind.

2.3.1 Klassische Mechanik

Das Grundkonzept der klassischen Mechanik beruht auf den *„Massepunkten“*. Ein Massepunkt ist ein Objekt, das eine positive, endliche Masse m aber keine Ausdehnung und damit auch keine Oberfläche besitzt. Zusätzlich kann man für jeden Massepunkt zu jedem Zeitpunkt eine Position und eine Geschwindigkeit feststellen. Die Position und die Geschwindigkeit sind dabei vektorielle Größen, sie beschreiben also nicht nur einen Wert, sondern auch eine Richtung. Die einzelnen Massepunkte gehorchen dabei den Newtonschen Gesetzen:

1. Ohne Krafteinwirkung verändern sie ihre Geschwindigkeit nicht.
2. Bei einer Krafteinwirkung ist die Geschwindigkeitsänderung $\vec{a}$ proportional zur Grösse der Kraft $\vec{F}$: $\vec{F} = m * \vec{a}$.
3. Wenn ein Massepunkt auf einen anderen Massepunkt eine bestimmte Kraft ausübt, so erfährt dieser eine Gegenkraft von gleicher Größe und umgekehrter Richtung.

Über den Ursprung der Kräfte wird in der Klassischen Mechanik nichts gesagt. Wenn man die Massepunkte als ein Modell für einzelne Atome ansieht, ist es sinnvoll, als Kräfte die Gravitationskraft und die elektromagnetische Kraft zu verwenden.

Weitere wichtige Sätze der klassischen Mechanik sind die Energieerhaltung und die Impulserhaltung. Die Energieerhaltung besagt, dass die Gesamtenergie eines abgeschlossenen Systems immer konstant bleibt. Diese Energie kann aber beliebig auf die verschiedenen Massepunkte verteilt sein und kann in unterschiedlichen Formen vorliegen. Am wichtigsten hierbei ist die *„kinetische Energie'"*, die ein Teilchen aufgrund seiner Geschwindigkeit besitzt. Die kinetische Energie kann man mit der Formel $E = \frac{1}{2}mv^2$ berechnen, wobei v die Geschwindigkeit ist. Außerdem ist die *„potentielle Energie“* von Bedeutung, die auch als Lageenergie bezeichnet wird. Diese Energie besitzt ein Masseteilchen durch seine Lage in einem Kraftfeld wie beispielsweise der Gravitationskraft oder der elektromagnetischen Kraft.

Die Impulserhaltung besagt, dass der Gesamtimpuls eines abgeschlossenen Systems konstant bleibt, welcher für ein Teilchen durch $\vec{p} = m\vec{v}$ bestimmt ist. Der Impuls ist im Gegensatz zur Energie eine vektorielle (gerichtete) Größe. Unter Berücksichtigung der Energieerhaltung und der Impulserhaltung ist ein abgeschlossenes

System der Klassischen Mechanik vollständig bestimmt. Dies bedeutet, dass diese Sätze zur genauen Vorhersage jeglichen Verhaltens in diesem System ausreichen.

Abgeschlossen ist ein System, wenn keinerlei Effekte von außerhalb des betrachteten Systems Auswirkungen auf die Energien oder Impulse haben. In vielen Fällen sind Systeme aber bei Simulationen nicht abgeschlossen. Es wird beispielsweise Energie durch das Betreiben eines Motors hinzugefügt.

Die Darstellung der einzelnen Massepunkte als Atome ist für einen Simulator oftmals nur begrenzt geeignet, da selbst kleine Objekte schon aus einer sehr großen Anzahl von Atomen bestehen und eine Berechnung daher äußerst aufwändig wäre. Nur bei sehr kleinen Objekten wie beispielsweise Molekülketten ist diese Simulation angemessen.

Bei größeren Gegenständen bietet es sich an, die Massepunkte, die aufgrund der vorhandenen Kräfte aneinander gebunden sind und sich daher nur wenig relativ zueinander bewegen, als einen zusammenhängenden Körper zu beschreiben. Man erhält dann ausgedehnte Objekte mit veränderlichen Formen. Die entstehenden Strukturen sind in ihren Eigenschaften dabei sehr komplex. Es gibt regelmäßige Kristallstrukturen, die vergleichsweise einfach zu berechnen sind, aber auch unregelmäßige (amorphe) Strukturen, die oftmals nur statistisch anzunähern sind. Die Eigenschaften dieser Objekte werden in der *„Physik der kondensierten Materie“* erforscht und es ist auch möglich, ihr Verhalten mit guter Genauigkeit zu simulieren. Bei dieser Simulation existieren weitere Energieformen, die aber auf die grundlegenden Energien der klassischen Mechanik zurückführbar sind. Dies ist unter anderem die *„thermische Energie“*, die in einem Körper gespeichert werden kann und die *„Spannenergie“* die beim Verformen eines Gegenstandes von diesem aufgenommen wird.

Ausgedehnte Gegenstände besitzen zusätzlich einen Drehimpuls, der die Rotation des Gegenstandes beschreibt. Der Gesamtdrehimpuls ist genauso wie die Energie und der Impuls eine Erhaltungsgröße, darf sich also bei einem abgeschlossenen System nicht ändern.

Simulationen mit Modellen der Physik kondensierter Materie werden oft durchgeführt, um Materialeigenschaften zu überprüfen oder um Wechselwirkungen zwischen unterschiedlichen Materialien festzustellen. Aufgrund der Komplexität ist eine Berechnung in Echtzeit auf normalen Computern allerdings nicht möglich.

2.4 Mechanik starrer Körper

Als weitere Vereinfachung kann man annehmen, dass die einzelnen Körper vollständig starr und damit unverformbar sind. In diesem Fall kann man den inneren Aufbau der einzelnen Körper vernachlässigen und es verbleiben nur sehr wenige Eigenschaften, die berücksichtigt werden müssen. Diese Eigenschaften kann man einteilen in statische Eigenschaften, die während der Simulation unverändert bleiben und dynamische Eigenschaften, die in jedem Zeitschritt neu berechnet werden. Zu den statischen Eigenschaften gehören die Masse und der Trägheitstensor des

Körpers, die Position des Schwerpunktes, die Form der äußeren Hülle und die Oberflächenbeschaffenheit mit Werten wie dem Reibungskoeffizienten und dem Restitutionskoeffizienten, die bei einer Kollision von Bedeutung sind. Zu den dynamischen Eigenschaften gehören die Position, die Rotation, die Geschwindigkeit und die Rotationsgeschwindigkeit des Körpers [Ebe01].

2.4.1 Schwerpunkt, Trägheitstensor und Masse

Für alle Berechnungen außer bei Kollisionen von Gegenständen ist die genaue Form eines Gegenstandes nicht wichtig. Man kann bei allen auf den Gegenstand einwirkenden Kräften die resultierenden Geschwindigkeitsänderungen bestimmen, indem man nur Schwerpunkt, Trägheitstensor und Masse kennt. Wenn eine Kraft an einem beliebigen Punkt eines Körpers wirkt, wird ein Teil dieser Kraft in eine lineare Beschleunigung des Körpers umgewandelt und ein anderer Teil wirkt als Drehmoment. Durch das Drehmoment wird die Rotation des Körpers verändert. Mit dem Trägheitstensor, einer 3x3 Matrix, lässt sich der Widerstand des Körpers gegen eine Veränderung der Rotation und damit die neue Rotationsgeschwindigkeit bestimmen. Sollte die Kraft genau in Richtung des Schwerpunktes wirken, wird nur eine lineare Beschleunigung übertragen.

2.4.2 Reibungskoeffizient

Der Reibungskoeffizient gibt an, wie groß die maximale Reibungskraft zwischen zwei Körpern ist, die mit einer gegebenen Normalkraft aufeinander gedrückt werden. In der einfachsten Form, die hier verwendet wird, lautet die Formel für die resultierende maximale Reibungskraft $F_R = \mu_1 \cdot \mu_2 \cdot F_N$, wobei μ die Reibungskoeffizienten der Körper sind und F_N die Normalkraft. Die Reibungskraft wirkt dabei immer senkrecht zur Normalkraft und entgegengesetzt der relativen Bewegung der beiden Körper. Dadurch wird die relative Bewegung der Körper verringert. Diese Darstellung ist aber eine recht ungenaue Vereinfachung der Realität. μ hängt eigentlich auch von der Kombination beider Materialien ab. Zusätzlich muss man die Haftreibung bei nicht vorhandener Bewegung von der Gleitreibung bei vorhandener Bewegung unterscheiden.

2.4.3 Restitutionskoeffizient

Der Restitutionskoeffizient (auch Stoßzahl genannt) gibt an, wie viel Energie bei einer Kollision von zwei Körpern erhalten bleibt und zu einer Reflexion führt. In der Realität wird bei einem Zusammenstoß immer ein Teil der Energie in Wärme oder eine dauerhafte Verformung umgewandelt und geht dadurch für die Bewegung verloren. In der Simulation wird einfach ein Teil der Energie abgezogen. Ein Restitutionskoeffizient von 1 bedeutet dabei, dass keine Energie verloren geht, und 0 bedeutet, dass sämtliche Energie verloren geht, es also nicht zu einer Reflexion kommt.

2.5 Simulation der Mechanik starrer Körper

Da die Mechanik starrer Körper mit relativ geringem Aufwand simulierbar ist, wird sie anderem bei Computerspielen und bei der Erzeugung von computeranimierten Filmen verwendet. Es ist dazu notwendig, die in der simulierten Welt vorhandenen Gegenstände, Verbindungen zwischen diesen Gegenständen und zusätzliche externe Kräfte, die beispielsweise durch Motoren oder die Gravitation hervorgerufen werden, angemessen zu modellieren [WB97].

2.5.1 Modellierung der Gegenstände

Die Form der Gegenstände, die in der Simulation vorkommen sollen, kann auf zwei unterschiedliche Arten modelliert werden. Entweder sind die Gegenstände zusammengesetzt aus geometrischen Primitiven oder sie werden durch ein Polygonnetz beschrieben. Geometrische Primitive sind einfache Formen, die mit wenigen Parametern dargestellt werden können. Hierzu zählen unter anderem Kugeln, Quader und Zylinder. Durch das Zusammensetzen dieser Primitive können dann auch komplexe Formen geschaffen werden. Ein Polygonnetz hingegen ist eine Liste von Polygonen, welche die Oberfläche beschreiben. Ein Polygon wiederum besteht aus einer Liste von Punkten im dreidimensionalen Raum, die alle auf einer beliebigen Ebene liegen müssen. Durch das Verbinden dieser Punkte erhält man den Umriss der Polygonfläche. Praktisch verwendet werden meistens nur Dreiecke und keine komplexeren Polygone, da hierbei die Berechnung, ob ein Punkt in der durch das Polygon beschriebenen Fläche liegt, sehr einfach ist. Für die Berechnung von Kollisionsantworten ist es notwendig, Polygonen auch eine Ausrichtung zu geben. Es muss definiert sein, welches die Außenseite ist. Die Außenseite legt man hierbei oft als die Seite, auf der die Punkte im Uhrzeigersinn verbunden werden, fest. Bei einer Definition eines Gegenstandes durch ein Polygonnetz muss man dieses beachten, ansonsten erfolgt bei einer Kollision keine Abstoßung, sondern eine Anziehung der Objekte.

(a) Kugel (b) Quader (c) Zylinder

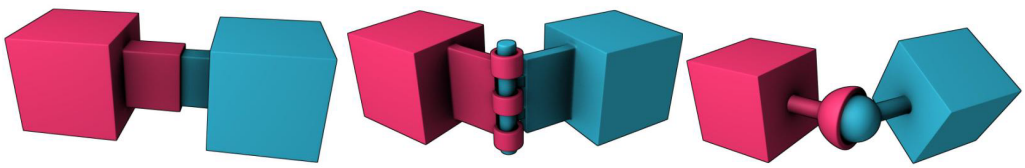
Abbildung 2.3: Verschiedene geometrische Primitive

Gegenstände können auch als statisch definiert sein. Dies bedeutet, dass sie sich niemals bewegen, also bei einer Kollision zwar der andere Gegenstand von dem statischen Gegenstand abprallt, diesen aber dadurch nicht in Bewegung versetzt. Statische Gegenstände sind üblicherweise der Boden und die Wände, die in der Realität eine sehr hohe Masse besitzen würden.

2.5.2 Modellierung von Verbindungen

Verbindungen zwischen zwei Gegenständen werden durch Constraints („Zwangsbedingungen“) modelliert. Constraints werden durch jeweils eine Transformation relativ zu den Gegenständen positioniert. Sie schränken dann die relative Lage der Gegenstände zueinander in einem oder mehreren Freiheitsgraden ein. Überlicherweise bestehen zwischen zwei Gegenständen 6 Freiheitsgrade, davon drei Freiheitsgrade der Translation und drei Freiheitsgrade der Rotation. Die interne Darstellung eines Constraints ist eine beliebige Anzahl von linearen Gleichungen oder Ungleichungen. Für jeden vollständig eingeschränkten Freiheitsgrad wird eine Gleichung verwendet und für Freiheitsgrade, die auf einen bestimmten Bereich eingeschränkt werden, werden zwei Ungleichungen verwendet.

Es gibt eine Reihe von häufig verwendeten Constraints. Wenn man alle Freiheitsgrade beschränkt, erhält man eine starre Verbindung, bei der also keine relative Bewegung mehr möglich ist. Wenn nur ein Freiheitsgrad der Translation nicht beschränkt wird, erhält man einen Schieber. Wenn ein Freiheitsgrad der Rotation nicht beschränkt wird, erhält man ein Scharnier, welches bei der Modellierung der Roboter die wichtigste Verbindung darstellt. Sollten nur die Freiheitsgrade der Translation beschränkt werden, erhält man ein Kugelgelenk. In Abbildung 2.4 sind einige Constraints visualisiert. In der Simulation ist es nicht notwendig, dass die Gegenstände sich berühren wie in der Abbildung, sondern die Transformation eines Constraints zeigt üblicherweise auf eine Stelle außerhalb des Gegenstandes. Als Form des Gegenstandes würden in den Beispielen also nur die Würfel existieren. Durch eine Berührung der Gegenstände würde die Simulation komplexer und damit langsamer werden.



(a) Schieber (b) Scharnier (c) Kugelgelenk

Abbildung 2.4: Unterschiedliche Constraints (aus [Cou10])

2.5.3 Modellierung von Aktuatoren

Aktuatoren wie beispielsweise Motoren werden in der Simulation nur dadurch modelliert, dass zu jedem Zeitschritt eine Kraft auf die jeweils betroffenen Gegenstände ausgeübt wird. Ansonsten sind sie in der Simulation selbst nicht vorhanden. Durch einen Motor, der an einem Scharnier angebracht ist, wird beispielsweise eine Rotationskraft auf beide damit verbundenen Gegenstände ausgeübt.

2.5.4 Genereller Ablauf der Simulation

Zu jedem Zeitschritt werden bei der Simulation verschiedene Aktionen ausgeführt.

1. Beschleunigung der Gegenstände durch vorhandene externe Kräfte.
2. Vorhersage der Bewegung der Gegenstände aufgrund ihrer aktuellen Geschwindigkeit.
3. Bestimmung von nach der Bewegung auftretenden Kollisionen und Einfügen von Kollisionsconstraints.
4. Lösen der Constraints.
5. Neue Gegenstandspositionen und Geschwindigkeiten festlegen.

Es ist wichtig, dass die Gegenstände in Schritt 2 noch nicht bewegt werden. Dort wird zunächst nur die Bewegung bestimmt, die die Gegenstände ohne Beschränkungen durch Kollisionen oder Verbindungen durchführen würden. In Schritt 4 wird dann anhand der vorhergesagten und der alten Position bestimmt, welche Bewegung wirklich möglich ist. Bei einer Kollision muss der Gegenstand beispielsweise reflektiert werden oder wenn zwei Gegenstände verbunden sind, können diese sich nur gemeinsam bewegen.

2.5.5 Behandlung von Kollisionen

Für die Detektion von Kollisionen werden unterschiedliche Algorithmen verwendet. Details dazu lassen sich unter anderem in [Mir98] nachlesen. Für jede der erkannten Kollisionen wird ein Constraint eingefügt, welcher aus einer Ungleichung besteht. Diese Ungleichung ist erfüllt, wenn die Kollision aufgehoben wurde. Dieses Constraint wird danach genauso behandelt wie andere Constraints, die die Bewegung von Gegenständen einschränken. Es wird aber wieder entfernt, sobald die Kollision nicht mehr existiert.

Dadurch, dass alle simulierten Gegenstände absolut starr sind, existieren oftmals nur einzelne Punkte als Kontakte. In der Realität würden hier ausgedehnte Kontaktflächen entstehen, da die meisten Gegenstände zumindest ein wenig verformbar sind. Bei einem Punktkontakt ist es nicht möglich, ein Reibungsmoment, also eine

Reibung, die die Rotation um einen Punkt bremst, zu übertragen. Dies ist eine deutliche Einschränkung bei der Realitätsnähe der Simulation. Eine Lösung für dieses Problem sind „weiche Oberflächen“, die in [CLA07] vorgestellt werden. Hierbei wird jedoch nicht die Geometrie der Gegenstände verändert, sondern es werden Kontaktpunkte an Stellen mit einem nur geringen Abstand generiert. Die gesamte Kraft wird dann auf alle diese Punkte verteilt, wobei der Abstand der Gegenstände an den Punkten berücksichtigt wird.

2.5.6 Lösen der Constraints

Im Constraintsolver findet die eigentliche Arbeit der Simulation statt. Hier müssen alle Verletzungen von Constraints durch unmögliche Bewegungen aufgehoben werden. Bei dem Lösen muss auch ein korrektes physikalisches Verhalten garantiert werden, also die Energieerhaltung, Impulserhaltung und Drehimpulserhaltung müssen erfüllt sein.

Es handelt sich bei den simulierten Systemen aber im allgemeinen nicht um abgeschlossene Systeme:

- Durch Motoren wird zusätzliche Energie in das System eingebracht, der Impuls bleibt aber erhalten. In der Realität handelt es sich hier um die Umwandlung (potentieller) elektrischer Energie in kinetische Energie.
- Durch die Gravitation wird der Gesamtimpuls und die Gesamtenergie des Systems verändert. Dies liegt daran, dass die Beschleunigung hier nicht wie in der Realität durch die Anziehung zweier Gegenstände zustande kommt, also der Umwandlung von potentieller Energie in kinetische Energie, sondern die Beschleunigung aus dem „nichts“ entsteht.
- Durch Reibung und Kollisionen geht ein Teil der Energie verloren. Die Energie würde normalerweise in thermische Energie umgewandelt, die aber nicht modelliert wird.
- Bei der Kollision mit statischen Gegenständen wird der Gesamtimpuls verändert. In der Realität gibt es keine statischen Gegenstände und auch bei deutlichen Größenunterschieden wie beispielsweise der Kollision eines Balles mit einem Planeten wird bei einer Kollision immer der Impuls der beiden Gegenstände verändert.

Bei allen diesen Fällen ist die Änderung der Energie bzw. des Impulses allerdings genau berechenbar und kann beim Lösen berücksichtigt werden.

Für eine Echtzeitsimulation ist das exakte Lösen der Constraints zu aufwändig. Es werden daher Vereinfachungen vorgenommen und die Lösung wird iterativ berechnet. Hierbei wird dann nach einer festgelegten Anzahl von Iterationsschritten abgebrochen. Es kann dann dazu kommen, dass nach dem Iterieren noch immer einige Constraints verletzt sind. Oftmals sind diese Ungenauigkeiten relativ gering,

aber in einigen Fällen kann es zu einem deutlich fehlerhaften Verhalten der Simulation führen. Das verwendete Lösungsverfahren ist für diese Diplomarbeit aber nicht relevant, da es auch nicht modifiziert werden soll. Eine ausführliche Beschreibung befindet sich in [MY88] und [Cat05].

2.6 Analytische Geometrie

Die analytische Geometrie verwendet, als Teilgebiet der Geometrie, algebraische Hilfsmittel (hauptsächlich aus der linearen Algebra) zur Lösung geometrischer Probleme. Da die physikalische Simulation zu großen Teilen auf den Sätzen der Geometrie basiert (neben einigen grundlegenden Sätzen der klassischen Physik), ist die analytische Geometrie hier von großer Bedeutung. Die für diese Arbeit wichtigsten Sätze werden hier kurz dargestellt. Die mathematischen Begründungen dazu lassen sich in jedem Buch zur linearen Algebra nachlesen, für diese Arbeit wurde hauptsächlich [Koe97] verwendet.

2.6.1 Vektoren

Ein Vektor dient zur Angabe der Position eins Punktes oder zur Definition einer Parallelverschiebung. Vektoren bestehen im dreidimensionalen Raum aus drei Zahlen, die jeweils die Strecke entlang einer Achse des Koordinatensystems beschreiben.

$$\vec{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{2.1}$$

Zwei Vektoren können komponentenweise addiert werden, was eine Hintereinanderausführung von zwei Parallelverschiebungen bedeutet und sie können mit einer Zahl multipliziert werden, was eine Mehrfachausführung der selben Verschiebung bedeutet. Die Länge eines Vektors lässt sich mit folgender Formel bestimmen:

$$l(\vec{v}) = |\vec{v}| = \sqrt{v_x^2 + v_y^2 + v_z^2} \tag{2.2}$$

2.6.2 Normierte Vektoren

Ein normierter Vektor ist ein Vektor mit der Länge 1. Dieser lässt sich als Angabe einer Richtung interpretieren. Vektoren lassen sich normieren, indem man sie mit $1/|\vec{v}|$ multipliziert (ausser beim Nullvektor, der keine definierte Richtung besitzt). Zwischen zwei normierten Vektoren lässt sich der Winkel bestimmen, indem man das Skalarprodukt bildet:

$$\cos(\alpha) = \vec{v} \cdot \vec{w} = \sqrt{v_x \cdot w_x + v_y \cdot w_y + v_z \cdot w_z} \tag{2.3}$$

2.6.3 Transformationen

Alle Objekte in der Simulation sollen die Koordinaten der jeweils untergeordneten Objekte in einem lokalen Koordinatensystem darstellen können. Untergeordnete Objekte sind für Gegenstände beispielsweise die einzelnen geometrischen Grundformen, aus denen der Gegenstand zusammengesetzt ist, aber auch die Verbindungsstellen zu anderen Gegenständen. Der Vorteil bei der Darstellung als lokales Koordinatensystem ist, dass diese Koordinaten während der Simulation niemals verändert werden müssen, wenn sich der Gegenstand als Ganzes bewegt. Unter anderem für die Berechnung von Kollisionen und auch zur grafischen Darstellung der Gegenstände ist es notwendig, die Koordinaten in ein globales Koordinatensystem umrechnen zu können. Um Vektoren von einem Koordinatensystem in ein anderes umzurechnen, werden *„homogene Transformationen“* [Ebe01] verwendet.

Eine Transformation ist eine Matrix mit 4x4 Elementen. Um einen Vektor $\vec{v}$ zu transformieren, wird dieser mit der Transformationsmatrix $\mathbf{A}$ multipliziert. Damit dies möglich ist, muss der Vektor um einen vierten Wert erweitert werden, welcher auf 1 gesetzt wird. Dies ist notwendig, da eine Verschiebung des Nullpunktes bei einer Transformation keine lineare Abbildung ist. Bei einer Multiplikation mit einer 3x3-Matrix wären also nur Rotationen möglich. Durch die Erweiterung auf eine vierte Koordinate, die nicht 0 ist, wird dieses Problem umgangen. Nach der Transformation erhält man wiederum einen vierdimensionalen Vektor, bei dem der letzte Wert auch immer 1 ist. Dieser Vektor wird dann wieder auf einen dreidimensionalen Vektor verkleinert, indem der letzte Wert weggelassen wird. Die Transformationsmatrix selbst besteht aus der 3x3 Rotationsmatrix, einem 1x3 Translationsvektor und einer konstanten vierten Zeile.

$$\mathbf{A} \cdot \vec{v} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} v_x \\ v_y \\ v_z \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11}v_x + r_{12}v_y + r_{13}v_z + t_x \\ r_{21}v_x + r_{22}v_y + r_{23}v_z + t_y \\ r_{31}v_x + r_{32}v_y + r_{33}v_z + t_z \\ 1 \end{pmatrix} \tag{2.4}$$

Für die Transformation sind folgende Matrizen von grosser Bedeutung:

Identität

Eine Transformation mit dieser Matrix lässt einen Vektor unverändert.

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.5}$$

Translation

Eine Translation verschiebt den Ursprung des Koordinatensystems. Zu allen Vektoren wird also bei der Transformation der selbe Wert hinzuaddiert.

$$\mathbf{T}(\vec{t}) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

Rotation

Eine Rotation lässt den Ursprung des Koordinatensystems an der selben Position, ändert aber die Richtungen der Koordinatenachsen. Hierbei gibt es drei Spezialfälle für die Rotation um jeweils eine der Koordinatenachsen:

$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.7}$$

$$\mathbf{R}_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.8}$$

$$\mathbf{R}_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.9}$$

Diese Matrizen werden als elementare Transformationen bezeichnet.

Matrixmultiplikation

Eine beliebige Positionierung des Koordinatensystems lässt sich durch die Hintereinanderausführung elementarer Transformationen erreichen. In diesem Fall ist es auch möglich, die einzelnen Transformationsmatrizen vorher zu multiplizieren und damit eine einzige Matrix für die gesamte Transformation zu erhalten. Dadurch lässt sich bei einer großen Anzahl von zu transformierenden Vektoren die Geschwindigkeit deutlich steigern.

$$\mathbf{B} \cdot (\mathbf{A} \cdot \vec{v}) = (\mathbf{B} \cdot \mathbf{A}) \cdot \vec{v} \tag{2.10}$$

Zu jeder so erzeugten Transformationsmatrix $\mathbf{A}$ lässt sich eine inverse Transformation $\mathbf{A}^{-1}$ finden, die einen Vektor vom verschobenem Koordinatensystem wieder

in das ursprüngliche System transformiert. Bei elementaren Translationen und Rotationen verändert man dazu einfach die Vorzeichen des Rotationswinkels bzw. des Translationsvektors. Bei zusammengesetzten Transformationen lässt sich die inverse Transformation dadurch bestimmen, dass man die elementaren Transformationen invertiert und die Matrizen dann in entgegengesetzter Reihenfolge multipliziert.

$$(\mathbf{C} \cdot \mathbf{B} \cdot \mathbf{A})^{-1} = \mathbf{A}^{-1} \cdot \mathbf{B}^{-1} \cdot \mathbf{C}^{-1} \tag{2.11}$$

Es ist aber auch möglich, eine Transformationsmatrix ohne Kenntnis der elementaren Transformationen zu invertieren, dazu wird oftmals der Gauß-Jordan-Algorithmus verwendet.

2.7 Kinematische Ketten

Eine kinematische Kette ist ein System aus starren Körpern, die durch Gelenke verbunden sind, also beispielsweise ein Roboterarm. Üblicherweise ist von einem dieser Körper die Position bekannt und man möchte etwas über die Position eines anderen Gliedes der Kette erfahren.

Bei der Vorwärtstransformation verwendet man das Wissen über die Gelenkwinkel, um die Position eines Körpers zu erfahren. Dies wird bei dem Simulator unter anderem dafür benötigt, um die Startpositionen der einzelnen Gegenstände zu bestimmen. Dabei müssen jeweils die Transformationen von einem Gelenk zum nächsten bekannt sein, was aber üblicherweise vorausgesetzt werden kann. Zur Berechnung werden dann für die Gelenke jeweils die passenden Transformationen gebildet. Bei einem Scharnier wird beispielsweise eine Rotationsmatrix verwendet. Danach werden die Transformationen alle in der Reihenfolge der kinematischen Kette multipliziert, um die Transformation vom Start bis zum Ende zu erhalten.

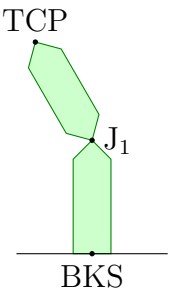


Abbildung 2.5: Eine einfache kinematische Kette

In Abbildung 2.5 ist ein einfaches Beispiel für die Vorwärtstransformation dargestellt. Hier sind zwei Körper durch ein Gelenk verbunden, welches um die lokale z-Achse rotieren kann. Der untere Körper ist dabei unbeweglich und der obere Kör-

per kann über das Gelenk bewegt werden. Um die Position des TCP[1] zu bestimmen, müssen zuerst die Transformationen vom Basiskoordinatensystem (BKS) zum Gelenk J_1 und vom Gelenk J_1 zum TCP bekannt sein. Die Transformation des Gelenkes ist $R_z(\alpha)$, wobei α der Winkel des Gelenkes ist. Die zusammengesetzte Transformation ist daher

$$T_{TCP}^{BKS} = T_{J_1}^{BKS} \cdot R_z(\alpha) \cdot T_{TCP}^{J_1} \tag{2.12}$$

Dieses Beispiel kann problemlos auf beliebig viele verkettete Körper erweitert werden.

2.8 Regelungstechnik

Die Regelungstechnik beschäftigt sich, als Teilgebiet der Automatisierungstechnik, mit selbständig ablaufenden Vorgängen, die im Wesentlichen aus Messen, Steuern und Regeln bestehen [Unb92, HD04]. Hierbei sind insbesondere die Regelkreise von Bedeutung. Regelkreise sind rückgekoppelte Systeme, die eine messbare Größe, Regelgröße genannt, auf einem Sollwert halten sollen. Dazu wird der Sollwert mit dem gemessenen Wert, der auch Istwert genannt wird, verglichen und dadurch die aktuelle Regelabweichung bestimmt. Dann wird durch den Regler anhand der Abweichung eine Stellgröße berechnet. In der Regelstrecke wird die Stellgröße verwendet, um beispielsweise Motoren anzutreiben, um die Abweichung zu verringern. Die Regler müssen dabei immer auf das Verhalten der Regelstrecke angepasst sein, um das gewünschte Verhalten zu erreichen.

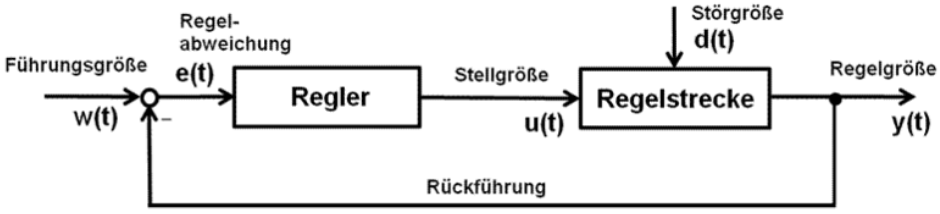


Abbildung 2.6: Ein Regelkreis (aus [Reg09])

In der Robotik werden Regler unter anderem dafür eingesetzt, um bei einem Gelenk einen bestimmten Winkel einzustellen. Die Regelgröße ist hierbei der Winkel und die Stellgröße kann beispielsweise die Eingangsspannung für einen Motor an dem Gelenk sein. Eine Regelung ist relativ kompliziert, da Sensoren und Aktuatoren ein Zeitverhalten haben, also verzögert reagieren. Durch das Anlegen einer

[1] Tool Center Point (Punkt im Zentrum des Werkzeuges) ist in der Robotik eine gebräuchliche Bezeichnung für die Position eines Endeffektors. Welcher Punkt dieses Endeffektors genau damit gemeint wird, ist unterschiedlich.

Spannung an einen Motor wird sich dieser erst langsam bewegen und dann bei gleicher Spannung weiter beschleunigen. Auch nach dem Abschalten der Spannung wird er nicht sofort stoppen, sondern nur langsam abbremsen. Sollte man einen Motor also erst abschalten, wenn der Zielwinkel erreicht wird, wird dieser Winkel kurz darauf überschritten werden, was wieder eine Motorbewegung in die entgegengesetzte Richtung erforderlich macht. Hierbei würde also eine periodische Schwingung entstehen, die sich im schlimmsten Fall aufschaukeln würde. Ein weiterer Fehler kann durch externe Störungen auftreten. So kann es sein, dass durch externe Kräfte wie Reibung oder die Gravitation der erwünschte Winkel niemals erreicht wird, da die Motorspannung bei einem kleinen Fehler nur sehr gering ist und dadurch der Motor nicht genügend Kraft entwickelt um die Abweichung zu beheben.

In der Regelungstechnik werden daher Regler entwickelt, die dieses fehlerhafte Verhalten kompensieren können. Grundsätzlich lassen sich Regler danach einteilen, ob diese ein lineares und zeitinvariantes (LZI) Verhalten aufweisen oder nicht. Zeitinvarianz bedeutet, dass ein zeitliches Verschieben des Eingangssignals auch beim Ausgangssignal nur zu derselben Zeitverschiebung führt und die Ausgangsfunktion ansonsten gleich bleibt. Linearität bedeutet hierbei, dass, wenn der Verlauf des Ausgangssignals für mehrere Eingangssignale bekannt ist, eine beliebige Linearkombination dieser Eingangssignale auch zu einer Linearkombination der bekannten Ausgangssignale führen muss. Dies lässt sich auch durch Formeln darstellen, wobei $s_i(t)$ die Eingangssignale, $g_i(t)$ die Ausgangssignale und a_i die Linearfaktoren sind. $\mathcal{T}$ ist die Übertragungsfunktion und t_0 eine beliebige Zeitverschiebung. Folgendes muss also erfüllt sein:

$$\sum_i a_i \mathcal{T}\{s_i(t)\} = \mathcal{T}\left\{\sum_i a_i s_i(t)\right\} \tag{2.13}$$

$$\mathcal{T}\{s_i(t-t_0)\} = g_i(t-t_0) \tag{2.14}$$

Dies ist bei realen Reglern nie exakt der Fall, da der Regelbereich aus physikalischen Gründen begrenzt sein muss. Es ist aber möglich, in einem gewissen Bereich ein annähernd lineares Verhalten zu erreichen. Bei idealisierten Reglern, die also nur durch mathematische Funktionen beschrieben werden, ist ein lineares Verhalten aber möglich und dieses dient oft als Annäherung für das reale Verhalten. Auch die realen Regler werden dann meistens als linear und zeitinvariant beschrieben, wobei zusätzlich der zulässige Bereich für den Betrieb des Reglers und die Abweichungen vom linearen Verhalten in diesem Bereich angegeben werden. Zur mathematischen Beschreibung eines Regler wird üblicherweise das Übergangsverhalten angegeben. Das Übergangsverhalten ist eine Funktion, oftmals in Form einer Integral- oder Differentialgleichung, die den Wert der Stellgröße abhängig von der Regelabweichung angibt.

Lineare zeitinvariante Regler lassen sich auch durch die Angabe ihrer Sprungantwort vollständig beschreiben. Die Sprungantwort ist die Ausgabe des Reglers bei Eingabe der Sprungfunktion $\sigma(t)$, mit $\sigma(t) = 0$ für $t < 0$ und $\sigma(t) = 1$ für $t >= 0$. Die

Beschreibung durch die Sprungantwort ist möglich, da sich jede Funktion mit endlich vielen Diskontinuitäten und ohne Polstellen durch eine Linearkombination von zeitlich verschobenen Sprungfunktionen beliebig genau annähern lässt. Aufgrund der Linearität des Reglers ist dann auch die Antwort auf diese Funktion beliebig genau anzunähern. Die Sprungantwort wird auch als Übergangsfunktion bezeichnet.

Es gibt auch die Möglichkeit, innerhalb des Regelbereiches ein nichtlineares Verhalten zu verwenden. Diese nichtlinearen Regler führen in bestimmten Anwendungsfällen zu einem deutlich besseren (stabileren) Verhalten der Regelgröße, allerdings ist der Entwurf und die Beschreibung dieser Regler auch aufwändiger.

Ein Spezialfall sind Open-Loop-Regelstrecken. Bei diesen existiert keine Rückkopplung, so dass eine Korrektur von fehlerhaften Werten nicht möglich ist. Daher sind Open-Loop-Regler nur in Fällen verwendbar, wo sich der Wert der Regelgröße ohne Messung berechnen lässt. Unter anderem bei Schrittmotoren ist es möglich, derartige Regler zu verwenden.

2.8.1 P-Regler

P-Regler (Proportional-Regler) sind die einfachsten möglichen linearen Regler. Hierbei ist der Wert der Stellgröße $e(t)$ immer proportional zur Regelabweichung $u(t)$.

$$u(t) = K_p * e(t) \tag{2.15}$$

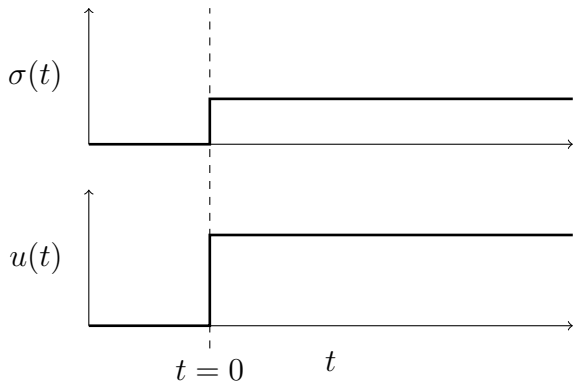


Abbildung 2.7: Die Sprungfunktion und Sprungantwort eines P-Reglers

Regler dieses Typs können zwar einen Fehler reduzieren, aber oftmals nicht ganz beseitigen. Wenn die Regelabweichung klein genug ist, wird irgendwann die Stellgröße nicht mehr ausreichen, um vorhandene Störungen auszugleichen. Der Einsatz von P-Reglern ist daher nur sinnvoll, wenn keine unvorhersehbaren Störungen existieren, oder eine gewisse Regelabweichung akzeptabel ist.

2.8.2 PID-Regler

Ein PID-Regler (Proportional-Integral-Differential-Regler) besitzt außer dem proportionalen Glied, das auch ein P-Regler besitzt, zusätzlich ein integrierendes Glied und ein differenzierendes Glied. Für alle diese Glieder kann jeweils ein konstanter Gewichtungsfaktor beliebig gesetzt werden. Die Differentialgleichung des PID-Reglers ist

$$u(t) = K_P \left[e(t) + \frac{1}{T_N} \int_0^t e(\tau) d\tau + \ T_V \frac{d}{dt} e(t) \right] \tag{2.16}$$

Hierbei gibt K_P den proportionalen Faktor an, K_P/T_N ist der Integrationsfaktor und $K_P * T_V$ der Differentiationsfaktor. T_N wird auch als Nachstellzeit bezeichnet und T_V als Vorhaltzeit.

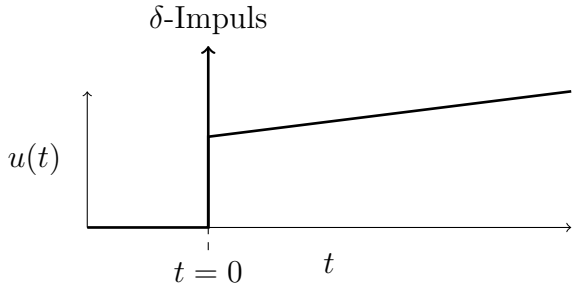


Abbildung 2.8: Sprungantwort eines PID-Reglers

In Abbildung 2.8 ist die Antwort eines PID-Reglers auf eine Sprungfunktion dargestellt. Das differenzierende Glied bewirkt bei einer Änderung der Regelabweichung einen kurzen, schnellen Impuls. In der Theorie wäre dieser Impuls bei einer nicht kontinuierlichen Änderung des Regelfehlers für einen Moment unendlich groß, was mathematisch einer Stoßfunktion (auch δ-Impuls genannt) entspricht. Dies ist praktisch nicht realisierbar, daher erfolgt stattdessen in der Praxis eine zeitlich ausgedehnte Wirkung aber mit endlicher Stärke. Danach erreicht die Funktion einen gewissen Wert, der durch den P-Anteil begründet ist und steigt dann langsam unbegrenzt weiter an. Dies wird durch den I-Anteil hervorgerufen.

Das integrierende Glied, welches bei einem konstanten Regelfehler einen immer größeren Wert „aufsammelt“, kann damit langfristig beliebige endliche Abweichungen kompensieren. Bei einer zu großen Gewichtung des I-Gliedes kann es aber zum Überschwingen kommen. Integrationsglieder können außerdem einen Wind-up-Effekt verursachen. Dieser Effekt tritt auf, wenn der Sollwert nicht oder nur sehr langsam erreichbar ist. In diesem Fall ergibt die Integration über eine längere Zeit einen sehr großen Betrag. Wenn dann der Sollwert wieder auf einen erreichbaren Wert gestellt wird, dauert es sehr lange, bis der aufintegrierte Wert wieder abgebaut ist, was zu einer Verzögerung bei der Reaktion führt. Verhindern kann man dies,

indem man den Integrator auf einen Maximalwert begrenzt, der nicht überschritten werden kann, oder den Wert mit der Zeit langsam von selbst absinken lässt. In diesen Fällen kann man aber wie bei einem P-Regler das Problem haben, dass bestimmte gewünschte Sollwerte niemals erreicht werden.

PID-Regler werden in der Praxis sehr häufig eingesetzt und sind auch in der Shadow Hand zur Steuerung der Finger vorhanden. Daher sollen diese auch im Simulator verfügbar sein. Es ist auch möglich, einen PID-Regler als P-Regler zu verwenden, indem man $1/T_N$ und T_V auf 0 setzt.

In einer Simulation mit Zeitschritten der Länge T lässt sich das Übergangsverhalten eines PID-Reglers folgendermaßen darstellen:

$$u_{i+1} = K_P \left[e_{i+1} + \frac{T}{T_N} \sum_{j=0}^{i+1} e_j + \frac{T_V}{T}(e_{i+1} - e_i) \right] \tag{2.17}$$

2.8.3 Regelkreise

Das Verhalten eines Regelkreises hängt nicht nur von der Struktur des Reglers, sondern auch von der Struktur der Regelstrecke ab. Die Regelstrecke besteht dabei meistens aus einem Aktuator und einem dazu gehörigen Sensor. In der Regelstrecke gibt es oftmals eine Zeitverzögerung bis eine Reaktion auf die Stellgröße erfolgt. Diese Verzögerung wird Totzeit genannt. Nach der Totzeit folgt dann eine Reaktion, die im einfachsten Fall eine Veränderung der Regelgröße proportional zur Stellgröße ist. Zusätzlich wirken Störgrößen $d(t)$ auf die Regelstrecke ein. Störgrößen haben verschiedenste Ursachen. Bei den Gelenkreglern eines Roboters sind dies unter anderem die Gravitation, die Massenträgheit, Reibungskräfte oder auftretende Kollisionen. Diese Störgrößen hängen manchmal auch vom Wert der Regelgröße ab, wodurch eine weitere (unerwünschte) Rückkopplung erzeugt wird. Diese zusätzlichen Rückkopplungen können einen Regelkreis oft instabil machen. In diesem Fall kann es zu unkontrollierten Schwingungen kommen.

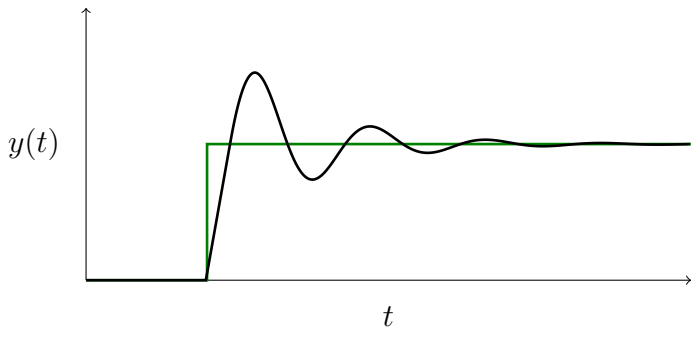


Abbildung 2.9: Sprungantwort eines Regelkreises mit P-Regler und ohne Störgröße

In Abbildung 2.9 ist das Ausgangsverhalten eines einfachen Regelkreises dargestellt. Hier ist nur ein P-Regler vorhanden und es existiert keine Störgröße. Nach dem

Ändern des Sollwertes ist zuerst ein deutliches Überschwingen erkennbar, wonach sich das Ausgangssignal langsam stabilisiert.

Auf den Regelkreis, dessen Verhalten in Abbildung 2.10 dargestellt ist, wirkt hingegen eine konstante Störgröße. Deshalb wird dort ein PI-Regler verwendet. An dem Graph der Stellgröße ist deutlich zu erkennen, dass diese, auch wenn kein Regelfehler mehr existiert, nicht auf 0 zurückgeht. Dies wird durch den I-Anteil des Reglers verursacht und ermöglicht das Kompensieren der Störgröße.

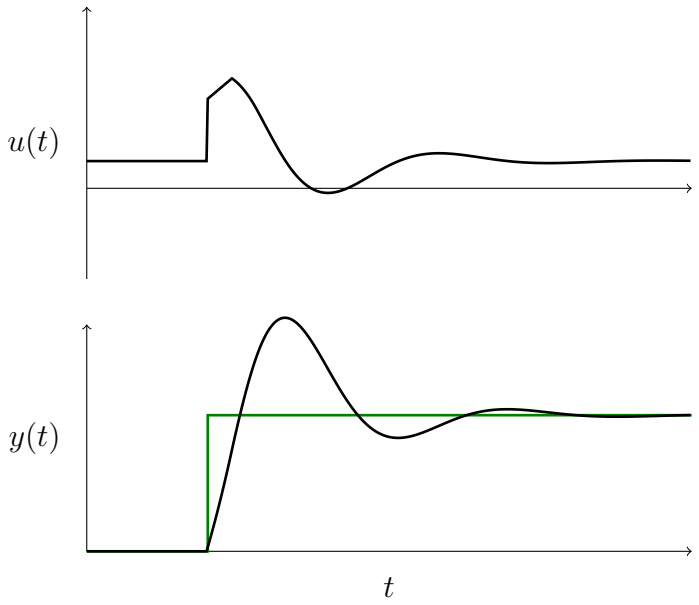


Abbildung 2.10: Sprungantwort eines Regelkreises mit Störgröße und Totzeit

3 Verwendete Hardware

Obwohl der in dieser Arbeit beschriebene Simulator prinzipiell nicht auf eine spezielle Hardware beschränkt sein soll, so wird er doch entwickelt, um insbesondere die Shadow Dextrous Hand und den dazugehörigen Versuchsaufbau des Projekts HANDLE zu unterstüzen. Die vorhandene Hardware soll im Simulator modelliert werden und es sollen Experimente und Messungen damit durchgeführt werden. Der Versuchsaufbau besteht aus einem PA10-6C Roboterarm, der derzeit mit seiner Basis auf dem Boden steht, einer Shadow Dextrous Hand C5M, die an dem Arm angebracht ist, einem CyberGlove Datenhandschuh als Eingabegerät, einem speziellen Tisch und einigen Gegenständen, die von der Hand gegriffen werden können. Außerdem existieren am Versuchsaufbau noch Kameras, die verwendet werden, um die Positionen der Gegenstände und der Hand zu erkennen. Die Kameras werden in dieser Diplomarbeit allerdings nicht verwendet.

3.1 Shadow Dextrous Hand

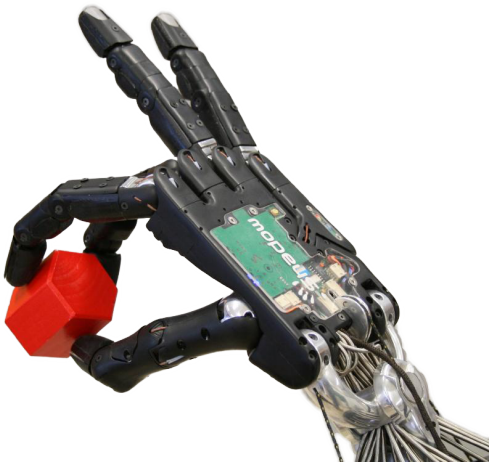


Abbildung 3.1: Die Shadow Dextrous Hand mit einem Holzklotz

Die Shadow Dextrous Hand C5M (kurz Shadow Hand) [Sha08] der Shadow Robot Company ist eine antropomorphe Roboterhand. Das Ziel bei der Entwicklung war, die menschliche Hand in Größe, Form und Funktionalität möglichst genau nachzuahmen. Die Hand wiegt etwa 4.2 kg und ist vom Unterarm bis zu den Fingerspitzen

ca 45 cm lang, wobei der Schwerpunkt etwa 12 cm von der Montageplatte des Unterarms entfernt ist. Der Unterarm hat an der breitesten Stelle einen Durchmesser von 13.5 cm. Die kinematische Struktur der Hand ist in Abbildung 3.2 dargestellt. Die dort vorhandenen Größen und Winkel sollen auch für die Erstellung des Modells verwendet werden. Die Hand besitzt fünf Finger und insgesamt 24 Gelenke. Zwei Gelenke befinden sich am Handgelenk, fünf Gelenke am Daumen und jeweils vier an den anderen vier Fingern. Unterhalb des kleinen Fingers befindet sich noch ein Gelenk in der Handfläche. Alle Finger bis auf den Daumen sind identisch aufgebaut und bei diesen sind die äußersten zwei Gelenke verbunden und lassen sich nur gemeinsam ansteuern. Beim Daumen sind alle Gelenke einzeln steuerbar.

Jedes Gelenk hat einen eindeutigen Namen. Diese Name beginnt mit zwei Zeichen als Abkürzung des Fingers (TH,FF,MF,RF,LF) bzw. des Handgelenkes (WR). Das dritte Zeichen ist immer ein „J“ für Joint (Gelenk). Das vierte Zeichen ist eine Ziffer von 1 bis 5, die das jeweilige Gelenk an dem Finger bzw. Handgelenk identifiziert. Die Nummerierung beginnt bei dem Gelenk, welches am entferntesten vom Arm ist.

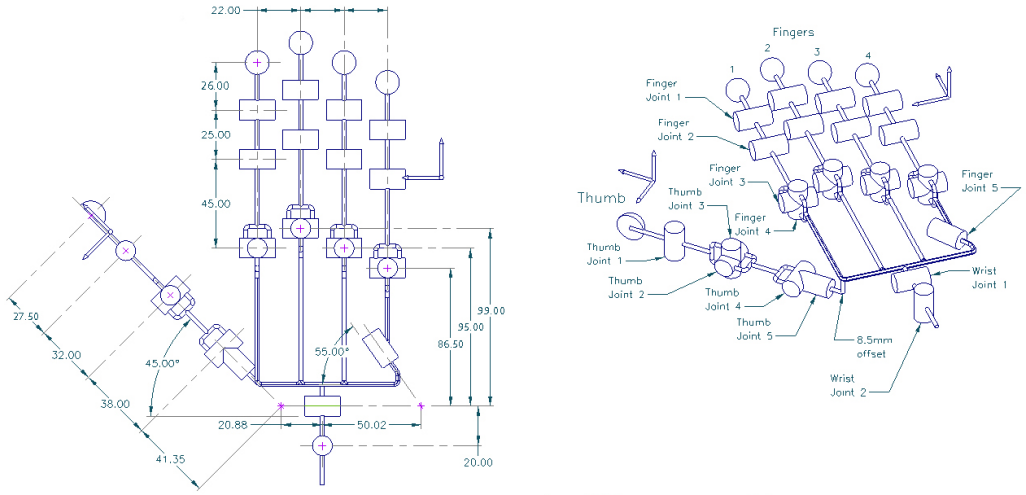


Abbildung 3.2: Kinematische Struktur der Shadow Dextrous Hand (aus [Sha08])

Jedes Gelenk ist über Metallsehnen mit jeweils zwei pneumatischen Muskeln verbunden, die sich im Unterarm befinden. Von diesen Muskeln ist jeweils einer dafür zuständig, das Gelenk zu öffnen (Extensor), während der andere das Gelenk schließt (Flexor). Die Muskeln werden an eine externe Druckluftversorgung mit einem Druck von etwa 3 bar angeschlossen. Dieser Aufbau hat den Vorteil, dass in der Hand selbst, wo nur wenig Platz ist, nur ein kleiner Teil der Technik eingebaut werden muss. Durch die Verwendung von pneumatischen Muskeln sind die Finger weniger steif und geben bei Belastungen automatisch nach, wodurch sowohl die Hand als auch die gegriffenen Gegenstände vor Schäden durch zu hohe Kräfte bewahrt werden. Der Druck in jedem Muskel wird über zwei PID-Regler gesteuert, einer für den Lufteingang und einer den Luftausgang.

Die Hand besitzt eine große Anzahl an Sensoren. Für jeden Muskel existiert ein Drucksensor, für jedes Gelenk existiert ein Sensor für den aktuellen Gelenkwinkel und derzeit befindet sich an jeder der fünf Fingerspitzen ein taktiler Sensor. Die taktilen Sensoren arbeiten aber nur eingeschränkt, da für einige dieser Sensoren eine sehr hohe Kraft notwendig ist, damit sie ansprechen. Es ist geplant, hier später genauere Sensoren einzusetzen. Die taktilen Sensoren, die später dort eingesetzt werden sollen, werden jeweils 34 sensitive Flächen besitzen, mit denen jeweils Gewichte im Bereich zwischen 10 und 1000 Gramm messbar sein sollen [Sha08].

Abbildung 3.3: Die Shadow Dextrous Hand mit dem Unterarm mit pneumatischen Muskeln und der Steuerungselektronik mit den Ventilen

Angesteuert wird die Roboterhand von einem PC über den Can-Bus. Von der Shadow Robot Company wird eine Programmierschnittstelle (API) bereitgestellt, die auf einem PC mit Linux verwendet werden kann. Bei der Ansteuerung lassen sich alle Regler einzeln einschalten und abschalten. Es können für jeden Regler die PID-Faktoren festgelegt werden. Auch der Deadband-Bereich, der bestimmt um welchen Winkel der Zielwinkel eines Gelenks vom aktuellen Winkel abweichen darf, bis die Regler aktiviert werden, ist änderbar. Für jedes Gelenk ist es möglich, den Zielwinkel festzulegen und den aktuell gemessenen Winkel abzufragen. Außerdem kann man die Messwerte der taktilen Sensoren an den Fingerspitzen und Werte der Drucksensoren in den Muskeln abfragen.

3.2 PA 10 Roboterarm

Der PA 10-6C von Mitsubishi Heavy Industries [PA00] ist ein Roboterarm, der häufig in der Forschung eingesetzt wird, da er in etwa die Form eines menschlichen Armes nachbildet. Er besitzt sechs Gelenke: drei Drehgelenke und drei Schwenkgelenke. Alle Gelenke verwenden einen Antrieb mit Harmonic-Drive-Getriebe, welches eine hohe Steifigkeit der Gelenke garantiert. Die Gesamtlänge des Armes beträgt 1317 mm. Der Arm wiegt 38 kg. Die maximale Traglast beträgt nach Herstellerangaben 10 kg.

Abbildung 3.4: Der PA 10-6C Roboterarm

An einen Steuerungsrechner auf dem die Software RCCL (Robot Control C Library) [HP86] läuft, wird er über ARCnet angebunden. Die Steuerungssoftware sendet alle 10 ms Daten an den Arm, um die Gelenke und die Bremsen anzusteuern, und erhält die aktuell gemessenen Gelenkwinkel zurück. Es ist auch möglich, einen RCCL-Serverprozess zu starten, über die man dann von anderen Programmen aus den Arm kontrollieren kann. RCCL enthält Funktionen, über die man die Zielpositionen des Arms im Gelenkraum und im kartesischen Raum vorgeben kann. Da intern nur eine Steuerung im Gelenkraum möglich ist, müssen kartesische Koordinaten von RCCL mit inverser Kinematik umgerechnet werden. Sollte die gewünschte Zielposition nicht erreichbar sein oder eine Singularität auftreten, wird die Ausführung abgebrochen.

Nr	Gelenk	Mechanisches Limit	Software Limit	Geschwindigkeit
1	S1	$\pm 180°$	$\pm 177°$	± 1 rad/s
2	S2	$-67° \ldots 127°$	$-64° \ldots 124°$	± 1 rad/s
3	E1	$-113° \ldots 164°$	$-107° \ldots 158°$	± 2 rad/s
4	E2	$\pm 270°$	$\pm 255°$	$\pm 2\pi$ rad/s
5	H1	$\pm 180°$	$\pm 165°$	$\pm 2\pi$ rad/s
6	H2	$\pm 270°$	$\pm 255°$	$\pm 2\pi$ rad/s

Tabelle 3.1: Die Limits der Gelenke des PA10-6C (aus [PA00])

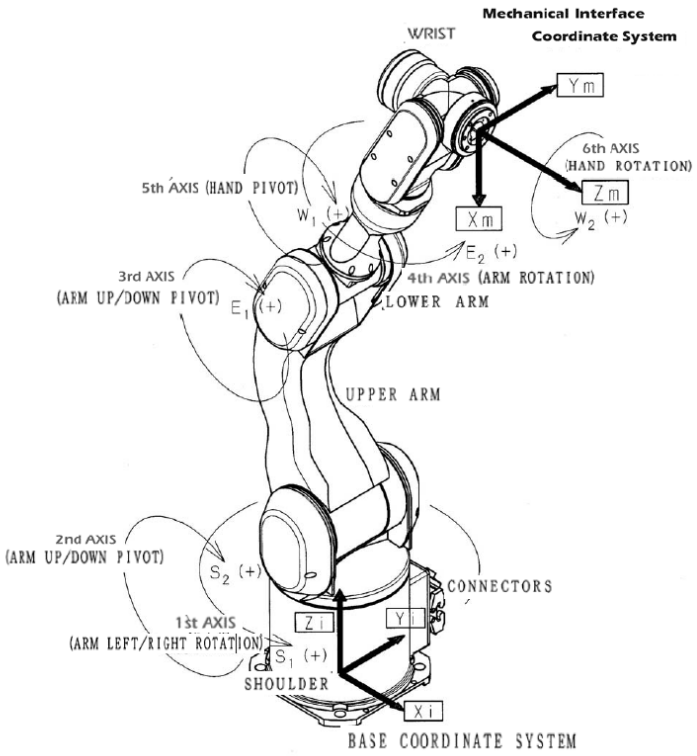


Abbildung 3.5: Die Gelenke und Koordinatensysteme des PA 10-6C (aus [PA00])

In Abbildung 3.5 sind die Gelenke des Roboterarmes abgebildet. In einer Konfiguration, in dem der Greifer direkt nach der Montageplatte folgt, sind die Gelenke 1 und 2 hierbei das „Schultergelenk“, die Gelenke 3 und 4 sind das „Ellbogengelenk“ und die Gelenke 5 und 6 das „Handgelenk“. Da die Shadow Hand selbst auch noch einen Unterarm und ein Handgelenk besitzt, könnte man in dieser Konfiguration allerdings auch die Gelenke 5 und 6 als Ellbogen bezeichnen. In Tabelle 3.1 sind die mechanischen Grenzen der Gelenke und die softwareseitig eingestellten Grenzen aufgelistet. Zusätzlich wird auch die maximale Geschwindigkeit und die maximale Beschleunigung durch die Software begrenzt, wobei diese Grenzen allerdings veränderbar sind. Der Arm wird derzeit nur mit sehr geringen Geschwindigkeiten angesteuert, um bei Fehlern noch reagieren zu können. In Abbildung 3.6 sind die Abmessungen des Armes dargestellt, die zur Erstellung des Simulationsmodells dienen.

Bei Experimenten mit dem Roboterarm stellte sich heraus, dass das Gelenk 5 zu schwach ist, obwohl die angebaute Hand nur ca. 5 kg wiegt. Durch die Länge der Hand ist aber vermutlich das Trägheitsmoment zu hoch. Als Folge davon treten bei höheren Geschwindigkeiten starke Oszillationen auf. Bei niedrigen Geschwindigkeiten ist die Steuerung aber noch möglich.

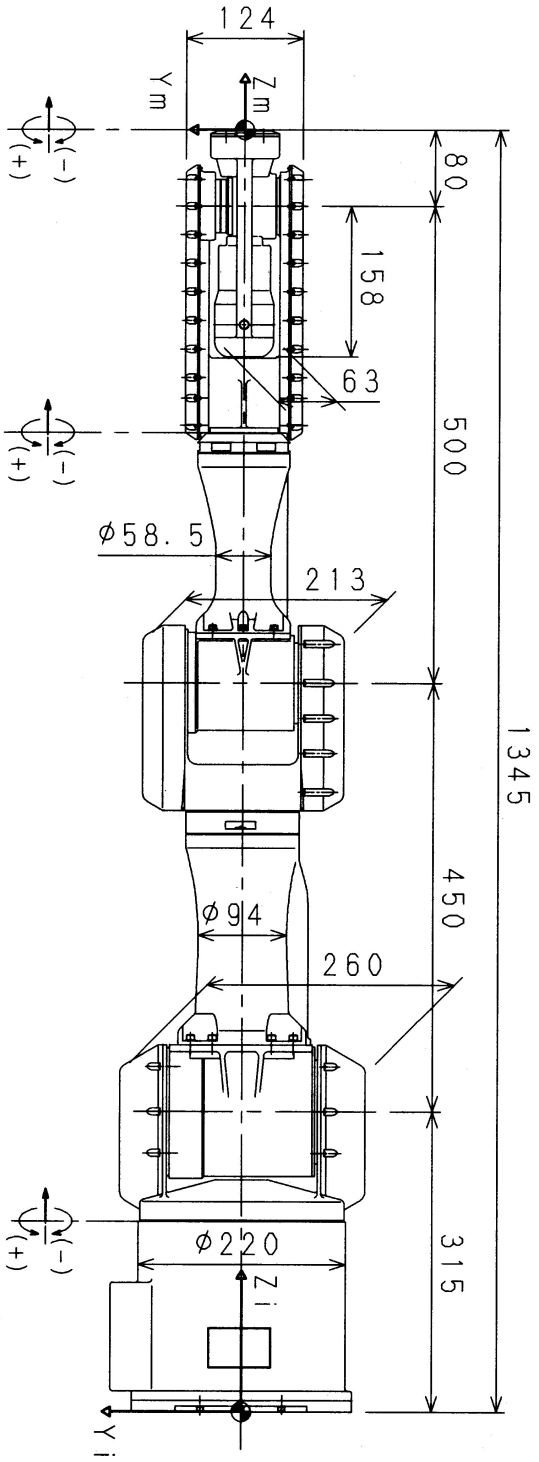


Abbildung 3.6: Die Abmessungen des PA 10-6C (aus [PA00])

3.3 CyberGlove Datenhandschuh

Der CyberGlove der Immersion Corporation [Vir98] ist ein Handschuh, der mit speziellen Sensoren ausgestattet ist, um die Stellung der Fingergelenke auszulesen. Er wurde als spezielles Eingabegerät für Anwendungen in der Virtuellen Realität entwickelt. Der Handschuh besitzt insgesamt 22 Dehnungsmesstreifen als Sensoren, davon 3 an jedem Finger, jeweils einen Sensor zwischen zwei Fingern, einen Sensor für die Krümmung der Handfläche und 2 Sensoren am Handgelenk. Die Sensoren, die in den Handschuh eingenäht sind, liefern keine Winkel, sondern nur unkalibrierte Widerstandswerte zurück. Deshalb ist bei der Verwendung immer zuerst eine Kalibrierung notwendig. Angeschlossen wird der CyberGlove über eine serielle Schnittstelle. Er kann durch ein einfaches Protokoll direkt angesprochen werden.

Der Handschuh wird bisher als ein Eingabegerät für die Steuerung der Shadow Hand verwendet. Er soll deshalb auch für die Ansteuerung des Simulators verwendet werden. Außerdem dient er zur Aufzeichnung von Greif- und Manipulationsaktionen, die vom Menschen durchgeführt werden. Die aufgezeichneten Daten können dann später für Lernverfahren verwendet werden, bei denen der Roboter durch die Imitation menschlicher Bewegungen lernen kann.

Abbildung 3.7: Der CyberGlove bei einem Experiment. Am Daumen sind die eingenähten Sensoren gut erkennbar

3.4 Arbeitstisch und Gegenstände

Der Arbeitstisch wurde so umgebaut, dass sich im Arbeitsbereich der Hand anstelle einer festen Holzoberfläche eine Polystyrolschaumstoffplatte befindet. Dies hat den Vorteil, dass bei einer möglichen Kollision die Roboterhand nicht beschädigt wird. Als Gegenstände zum Greifen existieren unter anderem verschiedene farbige Holzklötze mit einfachen Formen, die auch als Kinderspielzeug Verwendung finden. Mit diesen Holzklötzen sollte der Simulator getestet werden.

Im Rahmen des Projektes HANDLE sind Experimente mit verschiedenen weiteren Gegenständen geplant, die größtenteils auch im Simulator dargestellt werden können [HAN09]. Zu diesen Experimenten gehört das Greifen eines Plastikbechers, der dabei nicht verformt werden soll, das Bedienen einer Fernbedienung, das Greifen von Stoffen, das Verwenden von Schlüsseln und das Öffnen einer Flasche. Nicht geeignet ist der Simulator dabei für verformbare Gegenstände, da diese nicht in der gewählten Physiksimulation dargestellt werden können. Bei einigen Experimenten wird eine sehr hohe Genauigkeit bei der Kollisionserkennung und den resultierenden Kräften benötigt. Dies ist unter anderem bei dem Lösen einer Schraube mit einem Schraubendreher der Fall. Es ist bisher nicht klar, ob die Genauigkeit der Simulation dafür ausreichen kann. Möglicherweise ist diese Genauigkeit erreichbar, wenn man die Länge der Zeitschritte sehr klein wählt. In diesem Fall würde eine Simulation aber auch sehr langsam ablaufen.

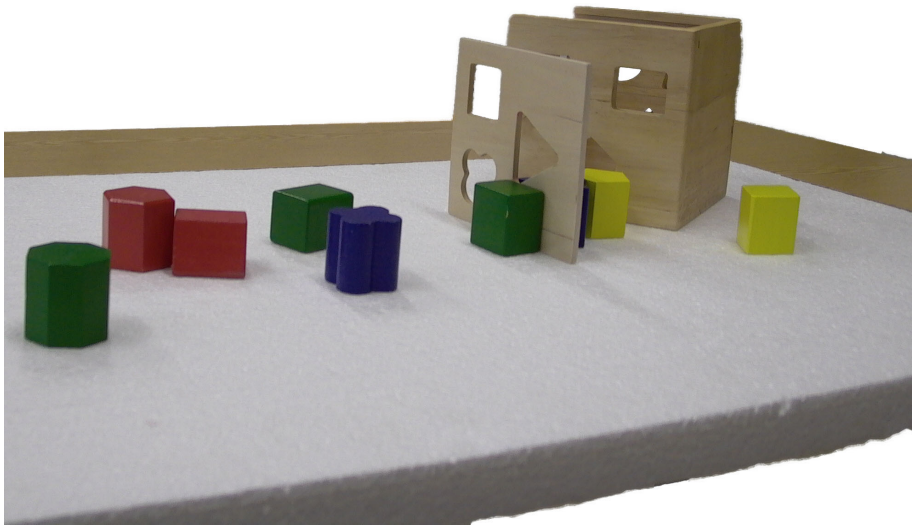


Abbildung 3.8: Der Arbeitstisch mit Holzklötzen aus der Sicht des Roboters

4 Entwurf des Simulators

In diesem Kapitel werden die Überlegungen für den Entwurf des Simulators dargestellt. Es mussten mehrere Anforderungen an den Entwurf berücksichtigt werden. Einige interessante Aspekte davon werden dabei genauer beschrieben. Im nächsten Kapitel wird dann eine Auswahl unter den existierenden Softwarebibliotheken und Technologien, die für den Aufbau des Simulators verwendet werden sollen, getroffen.

4.1 Anforderungen

Bei dem Entwurf des Simulators gab es verschiedene Anforderungen. Einige ergaben sich direkt aus den Zielen der Diplomarbeit. Bei anderen stand die Bedienbarkeit des Simulators im Vordergrund oder das Ermöglichen von späteren Erweiterungen.

4.1.1 Dynamisch bewegte Roboterhand

Bei einfach aufgebauten Roboterhänden mit wenigen Freiheitsgraden reicht es normalerweise aus, diese als kinematische Kette zu simulieren. Dabei wird angenommen, dass die Roboterhand eine so große Steifigkeit besitzt, dass die gewünschten Gelenkwinkel jederzeit eingehalten werden können. Bei Mehrfinger-Roboterhänden mit vielen Gelenken an jedem Finger, die zusätzlich nur eine geringe Steifheit haben, weicht eine rein kinematische Simulation aber stark von dem realen Verhalten ab. Dies liegt daran, dass die Gelenke ihre eingestellten Winkel oftmals nicht erreichen. Es kann auch vorkommen, dass durch eine Kraft auf ein Gelenk ein anderes Gelenk am selben Finger mitbewegt wird. Bei der Shadow Hand ist dieses Verhalten sehr deutlich festzustellen, da die pneumatischen Muskeln zu einer geringen Steifigkeit führen.

Dies kann nur simuliert werden, indem alle Gelenke der Hand von einer dynamischen Physiksimulation gesteuert werden. Dann ist es möglich, genau wie in der Realität die Kräfte an den Gelenken zu begrenzen. Durch große externe Kräfte, wie sie beispielsweise durch eine Kollision mit einem Gegenstand entstehen, werden dann die Gelenke auch vom eingestellten Zielwinkel wegbewegt. Dies ist ein deutlicher Unterschied zum Simulator GraspIt!, bei dem die gegriffenen Gegenstände dynamisch bewegt werden können, aber die Roboterhände nur kinematisch gesteuert werden.

4.1.2 Modellierung wichtiger Elemente

Alle wichtigen Bestandteile, die in dem realen Versuchsaufbau vorkommen, sollen im Simulator angemessen modelliert werden können. Dies baut natürlich auf den

Möglichkeiten der Physikbibliothek auf. Grundobjekte sind also die unverformbaren Körper, die außer ihrer Form jeweils einen Schwerpunkt, eine Masse, einen Trägheitstensor und einen Reibungskoeffizienten besitzen.

Auf einer übergeordneten Abstraktionsebene kann man Gegenstände sehen, die aus einem oder mehreren dieser unverformbaren Körper zusammengesetzt sind. Diese sollen zusätzlich einen Namen zur eindeutigen Identifikation in der Simulation besitzen. Gegenstände sind beispielsweise einzelne Fingerglieder, Armglieder oder auch frei bewegliche Objekte, die von dem Roboter gegriffen werden sollen.

Zusätzlich werden simulierte Gelenke benötigt, mit denen einzelne Gegenstände in einer bestimmten Position zueinander gehalten werden können. Für Gelenke soll jeweils angegeben werden können, welche Freiheitsgrade diese besitzen und wo die Limits der einzelnen Freiheitsgrade liegen. Die Gelenke erzeugen dann die Constraints, die für die Physiksimulation benötigt werden.

Auf eine Simulation der pneumatischen Muskeln an den Gelenken wurde verzichtet, weil keine Möglichkeit bestand, das Verhalten der Muskeln einzeln zu untersuchen. Stattdessen werden diese von simulierten Motoren bewegt. Auch die Motoren des Armes sollen simuliert werden. Bei den Motoren muss es möglich sein, eine maximal verfügbare Kraft und eine erwünschte maximale Geschwindigkeit anzugeben. Über diese Werte kann dann auch die Steifigkeit des Gelenks gesteuert werden.

Die Motoren sollen über Regler angesteuert werden. Die Regler geben den Motoren dann anhand des Regelfehlers die gewünschte Kraft vor. Es sollen zuerst nur PID-Regler simuliert werden, da diese sehr häufig verwendet werden. Es soll aber auch möglich sein, andere Regler einzubauen, um beispielsweise das Verhalten der Muskeln über die Regler zu simulieren.

Außerdem werden noch Sensoren benötigt. An der Shadow Hand sind Winkellagesensoren, taktile Sensoren und Drucksensoren in den Muskeln verbaut. Die Drucksensoren müssen nicht simuliert werden, weil auch die pneumatischen Muskeln in der Simulation nicht existieren. Die Winkellagesensoren sind sehr wichtig, da diese als Eingabewerte für die Regler dienen. Die taktilen Sensoren sollen simuliert werden, aber werden für die Simulation selbst nicht benötigt. Sinnvolle Einsatzmöglichkeiten für die taktilen Sensoren sind aber bei automatischen Steuerungen der Hand gegeben. Diese Steuerungen können dann über die vorliegenden Kräfte entscheiden, welche Aktionen durchzuführen sind.

4.1.3 Anpassbare simulierte Welt

Da es ein Ziel der Simulation war, dass verschiedene Versuchsaufbauten dargestellt und ausprobiert werden können, muss es leicht möglich sein, in dem Simulator neue Objekte hinzuzufügen oder die Positionen und Verbindungen der einzelnen Objekte zu verändern. Als Lösung dieses Problems sollen die vorhandenen Gegenstände nicht fest einprogrammiert sein. Stattdessen sollen Konfigurationsdateien verwendet werden, in denen alle Objekte beschrieben werden. Diese Datei kann dann einfach bearbeitet werden. Zusätzlich ist es von Vorteil, wenn in dieser Datei auch auf wei-

tere Dateien verwiesen werden kann. Dadurch ist es möglich, die Beschreibung der Welt aufzuteilen. Bestimmte Gegenstände, die immer wieder vorkommen, wie beispielsweise der Roboterarm, müssen dann nur an einer Stelle beschrieben werden und können in unterschiedlichen Versuchsbeschreibungen verwendet werden.

4.1.4 Unterstüzung des Princeton Shape Benchmark

Der Princeton Shape Benchmark [SMKF05] ist eine kategorisierte Sammlung von 3D-Modellen diverser Gegenstände, die für unterschiedliche Simulationen geeignet sind. In mehreren Forschungsarbeiten, in denen maschinelles Greifen erforscht wird, findet der Princeton Shape Benchmark Verwendung. Durch die Verwendung dieser Gegenstände ist es möglich, unterschiedliche Simulatoren, Roboterkonfigurationen und Lernalgorithmen besser zu vergleichen. Eine Unterstüzung dieses Benchmarks im Simulator ist daher sehr hilfreich.

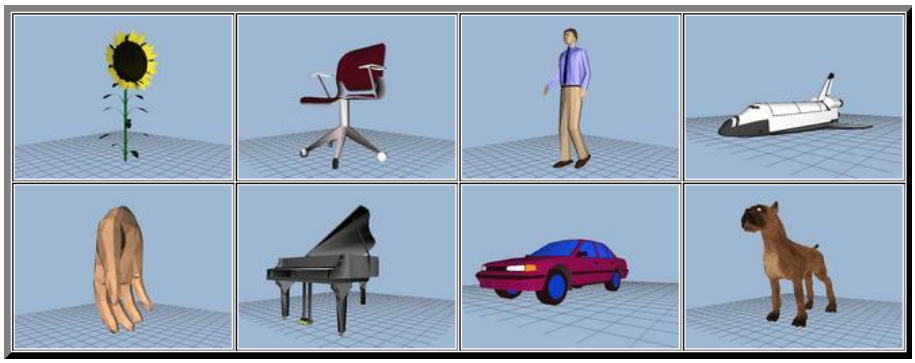


Abbildung 4.1: Gegenstände aus dem Princeton Shape Benchmark (aus [SMKF05])

4.1.5 Aufzeichnung des Simulationsverlaufes

Um die Experimente auswerten zu können, sollen möglichst viele Daten, die bei der Simulation erzeugt werden, aufgezeichnet werden können. Interessant sind hierbei beispielsweise Positionen und Geschwindigkeiten der Gegenstände, Ausgaben der Sensoren wie Gelenkwinkel und Kollisionskräfte und auch die genaue Position der einzelnen Kollisionen. Diese Daten müssen nach jedem Zeitschritt aufgenommen und dann in einer externen Datei für die spätere Auswertung abgespeichert werden können.

4.1.6 Gleiche Schnittstellen wie beim Roboter

Damit bei Steuerungsprogrammen ein leichtes Umschalten zwischen dem realen Roboter und der Simulation möglich ist, sollen jeweils die gleichen Schnittstellen angeboten werden. Dadurch müssen dann Programmteile wie die Ansteuerung des Robo-

ters durch den CyberGlove oder mögliche Lernprogramme nur einmal geschrieben werden und können ohne Änderung auf beiden System verwendet werden. Bei dem realen Roboter lagen diese Schnittstellen größtenteils fest, da diese durch die Software der Shadow Hand vorgegeben wurden. Der Simulator sollte also diese Schnittstelle auch implementieren.

4.1.7 Unterstützung unterschiedlicher Eingabemethoden

Die Steuerung soll durch unterschiedliche Eingabemethoden möglich sein. Eine dieser Eingabemethoden ist eine grafische Darstellung der einzelnen Regler mit der Möglichkeit, diesen ihren Sollwert vorzugeben. Eine andere Möglichkeit soll die Steuerung über den CyberGlove sein. Bei diesem erhält man unkalibrierte Werte der Dehnungsmessstreifen als Ausgabe, die dann kalibriert werden müssen und wiederum als Eingabe für die Regler dienen. Als weitere Möglichkeit soll das Laden von vorher abgespeicherten Simulationszuständen möglich sein. Hierbei werden die Gegenstände nicht durch die Physiksimulation an ihre neuen Positionen bewegt, sondern die geladenen Werte werden direkt als Positionen gesetzt.

4.1.8 Ausgabe als 3D-Grafik

Um den Simulationsverlauf beobachten zu können, muss der aktuelle Zustand der Welt als 3D-Grafik ausgegeben werden. Wichtig ist hierbei auf jeden Fall eine Darstellung der Gegenstände an der entsprechenden Position. Außerdem muss man die Position der Kamera in der Welt bewegen können. Wenn beispielsweise der Arm zu einem zu greifenden Gegenstand bewegt wird, ist ein Überblick über die Szene vorteilhaft. Wenn man aber einen Gegenstand dann greifen will, ist eine detailliertere Darstellung dieses Gegenstandes notwendig und die Umgebung ist weniger interessant. Um mögliche Verdeckungen zu vermeiden, muss es eine Möglichkeit geben, durch Gegenstände hindurchzusehen. Dies kann dadurch erreicht werden, dass nur die Kanten der Gegenstände dargestellt werden aber die Flächen durchsichtig sind.

Weiterhin sollen auch noch aktuelle Kollisionspunkte angezeigt werden können. Dadurch wird das Verhalten der Simulation verstehbar. Ebenso können mögliche Fehler in der Simulation so leichter entdeckt werden.

Für automatische Lernverfahren ist eine Ausgabe als 3D-Grafik nicht immer notwendig, so dass es von Vorteil ist, wenn hierbei die Ausgabe abschaltbar ist. Dadurch lässt sich dann die Simulation ein wenig beschleunigen.

4.2 Überblick über den Entwurf

In Abbildung 4.2 ist der auf den Anforderungen aufbauende Entwurf dargestellt. Der Kern der Anwendung ist der Simulator. Dieser bietet mehrere Schnittstellen an. Zum einen ist dies die Konfigurationsschnittstelle, über die beim Start des Simulators die Daten über in der Simulation vorhandene Objekte geladen werden können.

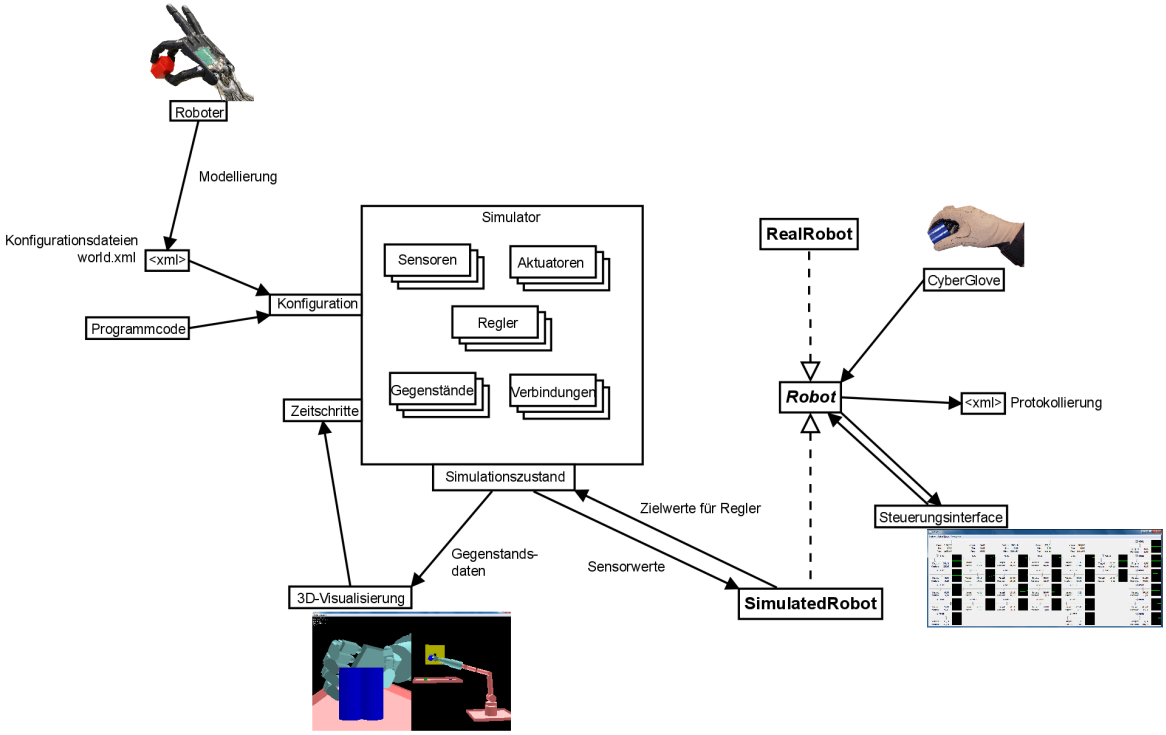


Abbildung 4.2: Überblick über den Simulatorentwurf

Diese Daten können dabei aus externen Dateien stammen oder von Programmcode erzeugt werden. Zusätzlich ist es möglich, einige Simulationsparameter zu verändern. Änderbare Parameter sind unter anderem die Länge der internen Zeitschritte und die Anzahl an Iterationen für den Constraintsolver.

Eine weitere Schnittstelle dient zum Auslösen von Zeitschritten. Dadurch, dass der Verlauf der Zeit nicht direkt im Simulator verwaltet wird, kann eine Simulation beliebig schnell oder langsam ausgeführt werden. Dies ist nur durch die Rechengeschwindigkeit des Computers begrenzt.

Die zur Laufzeit wichtigste Schnittstelle des Simulators ermöglicht es, auf Gegenstände zuzugreifen und deren aktuelle Werte abzufragen oder zu verändern. Als wichtigste abfragbare Parameter existieren hierbei die Sensorwerte und die Positionen der Gelenke und der Gegenstände. Diese Schnittstelle wird von allen Visualisierungen und Steuerungsmodulen verwendet. Auch zum Aufzeichnen der Simulation in einer Datei wird diese Schnittstelle eingesetzt. Steuerungsmodule können die Sollwerte der Regler festlegen, oder Motoren direkt kontrollieren.

Die Steuerung läuft dabei über die Robot-Schnittstelle, die einen gemeinsamen Zugriff auf den Simulator und den realen Roboter anbietet. Über diese Schnittstelle sind nicht alle Daten der Simulation greifbar. Es existieren hier nur Regler, deren Sollwerte festgelegt werden können und Sensoren, deren Meßwerte abgefragt werden

können. Dies ist dasselbe, was auch die Schnittstelle zur Shadow Hand anbietet. Über diese Schnittstelle läuft die Ansteuerung durch den CyberGlove und das Kontrollfeld, in dem die einzelnen Gelenke konstrolliert werden können. Diese Schnittstelle wird später noch genauer beschrieben.

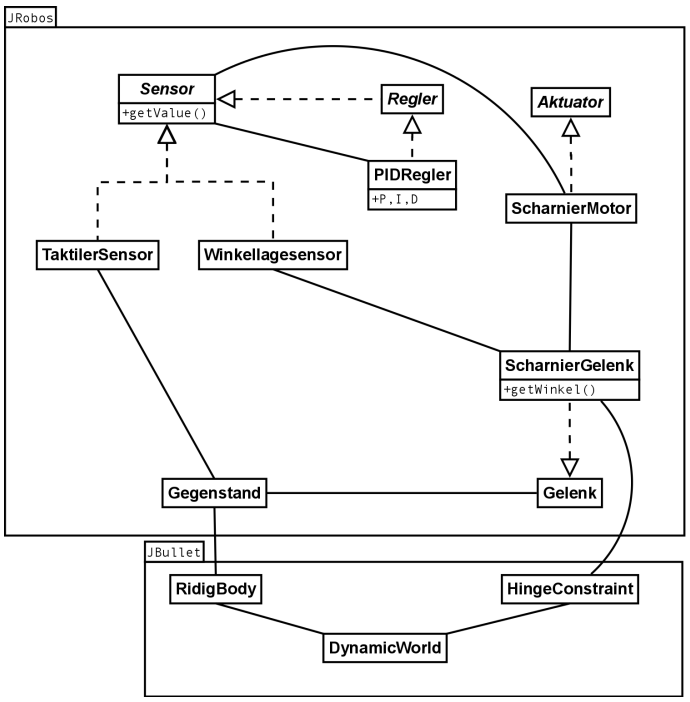


Abbildung 4.3: Klassendiagramm des Simulationskerns

Abbildung 4.3 zeigt ein Klassendiagramm mit den wichtigsten Klassen des Simulationskerns. Die Klassen im Package JRobos werden in dieser Diplomarbeit geschrieben. Die Klassen im Package JBullet stammen aus der Physikbibliothek. Es werden Klassen für die modellierten Gegenstände, Gelenke, Sensoren, Aktuatoren und Regler benötigt.

Gelenke sind jeweils mit einem oder zwei Gegenständen verbunden und sorgen für einen Zusammenhalt dieser Gegenstände. Für die Physiksimulation werden sie als Constraints dargestellt. Sensoren bieten als gemeinsame Schnittstelle die Möglichkeit, den aktuellen Meßwert abzufragen. Die Quelle der Sensordaten ist je nach Typ des Sensors unterschiedlich. Für einen Winkellagesensor ist die Quelle ein Gelenk, von dem der Winkel bestimmt werden kann. Für einen taktilen Sensor hingegen werden die Kollisionsinformationen eines Gegenstandes ausgewertet. Regler werden intern auch als Sensoren angesehen, da diese genauso wie Sensoren einen abfragbaren Wert zur Verfügung stellen. Zusätzlich können sie aber auch einen anderen Sensor als Quelle für den Istwert der Regelung verwenden. Der Sollwert kann von außerhalb des Simulators festgelegt werden. Als Aktuatoren existieren bisher nur Motoren. Diese akzeptieren ebenso wie die Regler auch einen Wert, der entweder

von einem Sensor oder von ausserhalb der Simulation stammen kann. Zusätzlich muss hier noch angegeben werden, auf welchem Gelenk der Motor arbeiten soll.

4.3 Durchführung der Simulation

Zum Durchführen einer Simulation werden an den Simulatorkern externe Zeitsignale gesendet, wodurch die Ausführungsgeschwindigkeit angepasst werden kann. Über die Steuerungsoberfläche können diese Zeitschritte einzeln ausgelöst werden und es ist möglich, ein automatisches Auslösen zu aktivieren. In dem Fall wird versucht, die Simulation in Echtzeit ablaufen zu lassen.

Im Simulator selbst wird dieser Zeitschritt in einzelne Schritte mit gleicher Länge unterteilt. Dies ist notwendig, damit die Genauigkeit der Simulation selbst konstant bleibt. Durch eine Unterteilung in kürzere Schritte ist eine Verbesserung der Genauigkeit zu erreichen, was allerdings auf Kosten der Ausführungsgeschwindigkeit geht. Im Kapitel „Experimente“ wird dies genauer untersucht.

In jedem einzelnen Schritt werden nun mehrere Aktionen durchgeführt:

1. Aktivierung der Physiksimulation und Bewegen der Gegenstände

2. Aktualisierung der Sensorwerte aufgrund des neuen Simulationszustandes

3. Aufrufen einer externen Funktion. Diese kann Logik enthalten, um beispielsweise die Sollwerte der Regler zu verändern und um den aktuellen Zustand aufzuzeichnen

4. Setzen der Motorkräfte für den nächsten Zeitschritt unter Verwendung der Sensorwerte und der Positionen der Gegenstände

4.4 Funktionsweise der Sensoren

4.4.1 Winkellagegeber

Die Winkellagegeber geben den aktuellen Winkel eines Scharnier-Gelenkes zwischen zwei Gegenständen zurück. Dieser Winkel lässt sich berechnen, indem man bei beiden Gegenständen jeweils einen normalisierten Richtungsvektor $\vec{v}$ im Koordinatensystem $\mathbf{T}$ des Gegenstandes erzeugt, der orthogonal auf der Drehachse steht und der bei beiden Objekten bei einem Winkel $\alpha = 0$ in dieselbe Richtung zeigt. Bei einer Rotation zeigen diese Vektoren dann in eine unterschiedliche Richtung. Beide Vektoren müssen nun in das globale Koordinatensystem transformiert werden, woraufhin der Winkel durch Bilden des Skalarproduktes zwischen diesen Vektoren bestimmt werden kann.

$$\vec{v}_1' = \mathbf{T}_1^{-1} * \vec{v}_1 \tag{4.1}$$
$$\vec{v}_2' = \mathbf{T}_2^{-1} * \vec{v}_2 \tag{4.2}$$
$$\alpha = cos^{-1}(\vec{v}_1' * \vec{v}_2') \tag{4.3}$$

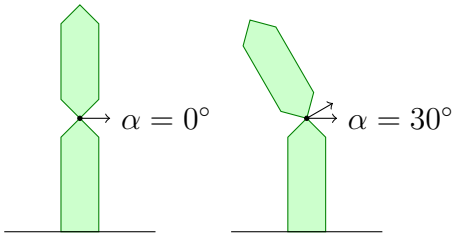


Abbildung 4.4: Bestimmung des Winkels eines Scharnier-Gelenkes

4.4.2 Taktile Sensoren

Taktile Sensoren sollen die äußeren Kräfte, die durch Kollisionen auf einen Teil eines Gegenstandes wirken, messen. In der Simulation werden dazu alle Kollisionspunkte des Gegenstandes mit der jeweiligen Richtung und Kraft abgefragt. Dieses wird von der Physikbibliothek bereitgestellt. Danach wird für alle Kollisionspunkte bestimmt, ob dieser Punkt in der sensitiven Fläche des Sensors liegt. Die Fläche kann entweder der ganze Gegenstand sein, oder ein Teil der Oberfläche in einem kugelförmiger Ausschnitt. Sollte diese Bedingung erfüllt sein, wird das Skalarprodukt zwischen der Richtung der Kraft und der Ausrichtung des Sensors gebildet. Hieraus lässt sich der Winkel zwischen beiden berechnen, der in diesem Fall ein Faktor zur Bestimmung der Kraft in Richtung des Sensors darstellt. Sollte dieser Faktor größer als 0 sein wird er mit der Stärke der Kollision multipliziert und zu dem Sensorwert hinzuaddiert.

4.5 Funktionsweise der Aktuatoren

Als Aktuatoren existieren bei dem realen Roboter die Motoren des Roboterarmes und die pneumatischen Muskeln der Roboterhand. Diese Aktuatoren verhalten sich deutlich unterschiedlich. Bei den Motoren ist eine Ansteuerung sehr einfach, da diese fast ohne Zeitverzögerung eine große Kraft ausüben können. Die pneumatischen Muskeln hingegen besitzen ein sehr komplexes Verhalten. Bei einem Befüllen eines Muskels mit Luft steigt dabei die Kraft langsam an, bis der maximale Druck erreicht ist. Wenn sich der Muskel zusammenzieht, sinkt allerdings der Druck wieder wodurch die ausgeübte Kraft sinkt. Dadurch, dass jedes Fingergelenk von zwei Muskeln gesteuert wird, wird das Verhalten nochmals komplizierter. Hierbei gibt es dann verschiedene Möglichkeiten für den Druck in beiden Muskeln, die dieselben aufsummierten Kräfte auf die Finger ausüben. Für ein genaues Modell müsste also auch

der Druck in beiden Muskeln simuliert werden. Aufgrund der Komplexität soll im Simulator aber nur ein vereinfachtes Modell verwendet werden, welches sich ähnlich wie ein Elektromotor verhält.

4.6 Schnittstelle zur Shadow Hand

Die gemeinsame Schnittstelle, über die Kontrollprogramme den Simulator und die Shadow Hand ansprechen können, ist grösstenteils durch die vorgegebene Schnittstelle der Shadow Hand, die in der Programmiersprache C geschrieben ist, festgelegt. Damit der Zugriff auch von Java aus möglich ist, musste diese Schnittstelle erstmal mit Hilfe von JNI [Lia99] zugänglich gemacht werden. Dabei wurden die Funktionen in Klassen gekapselt, so dass der Zugriff objektorientiert erfolgen kann.

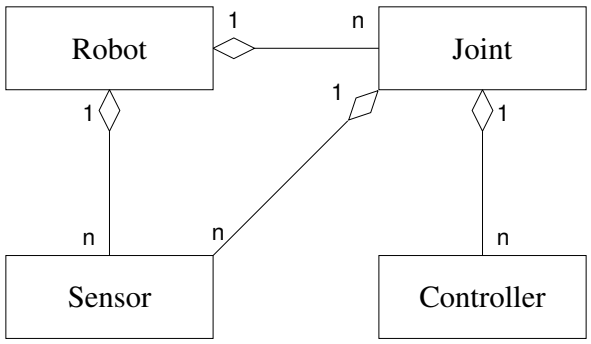


Abbildung 4.5: Klassendiagramm der Schnittstelle zur Shadow Hand

In Abbildung 4.5 ist das Klassendiagramm dieser Schnittstelle dargestellt. Auf der obersten Ebene befindet sich die Roboter-Klasse (Robot). Diese bietet Zugriff auf die Liste der vorhandenen Gelenke (Joint). Jedes Gelenk repräsentiert dabei ein existierendes Gelenk der Shadow Hand. Ein Gelenk bietet wiederum Zugriff auf die vorhandenen Sensoren und die Regler (Controller) für die pneumatischen Muskeln. Sensoren gibt es unter anderem für den aktuellen Winkel und den Druck in den Muskeln. Aber auch der eingestellte Zielwinkel ist als Sensor modelliert. Diesem Sensor kann man von dem Programm aus einen Wert zuweisen, während die anderen Sensoren nur gelesen werden können. An den Reglern können verschiedene Einstellungen vorgenommen werden. Diese können ganz abgeschaltet werden und man kann die PID-Faktoren verändern. Außerdem lässt sich einstellen, wie genau der gemessene Winkel am Zielwinkel sein muss, um von den Reglern als akzeptabel eingestuft zu werden (Deadband). Dies ist notwendig, da die Steuerung und auch die Sensorwerte nur eine begrenzte Genauigkeit bieten können. Sollte man also den als akzeptabel bewerteten Bereich zu klein wählen, würde das Gelenk andauernd

nachgeregelt werden müssen. Dies hätte einen erhöhten Verschleiss der Hardware zur Folge, ohne die Genauigkeit zu verbessern.

Es gibt auch Sensoren, die direkt vom Roboter aus zugreifbar sind. Dies sind die taktilen Sensoren, die ja an keinem Gelenk angebracht sind und dadurch nicht anders abgefragt werden könnten. Die taktilen Sensoren sind bisher unkalibriert, wohingegen die Drucksensoren und die Winkellagesensoren der Gelenke kalibriert sind.

Für diese Schnittstelle in Java existieren außer dem Simulator zwei Implementierungen. Die eine ist die Verbindung zur Shadow Hand, die JNI verwendet und nur auf einem Rechner, auf dem auch die entsprechenden Treiber installiert sind, lauffähig ist. Hier werden alle Funktionen der Schnittstelle unterstüzt. Zusätzlich existiert noch eine Implementierung, die nur dazu gedacht ist, die Schnittstelle selbst zu testen. Diese liefert für alle Sensoren sinnvolle Werte zurück und lässt auch eine Änderung des Zielwinkels zu. Allerdings werden die Werte durch ein sehr stark vereinfachtes Modell berechnet, welches nicht sehr realitätsnah ist.

Bei dem Simulator sollen alle Funktionen der Schnittstelle unterstüzt werden, außer der Einstellung des Deadbandes. Außerdem werden keine Sensoren vorhanden sein, die den Zustand der pneumatischen Muskeln abfragen, da diese in der Simulation nicht modelliert werden sollen.

5 Verwendete Software

In diesem Kapitel wird die vorhandene Software beschrieben, die für die Entwicklung des Simulators verwendet werden soll. Es wird außerdem begründet, wieso diese Auswahl getroffen wurde. Die Begründungen basieren aus den Anforderungen an den Simulator und dem Entwurf aus Kapitel 4.

5.1 Programmiersprache

Als Programmiersprache wurde Java ausgewählt. Ein Grund war, dass hierfür eine Vielzahl von Programmbibliotheken zur Verfügung stehen. Außerdem sind in Java geschriebene Programme plattformunabhängig, sie funktionieren also ohne Änderungen auf beliebigen Betriebssystemen. Von der Performance ist Java etwa mit C++ vergleichbar, bietet aber Vorteile, da es einfacher und damit schneller zu programmieren und zu debuggen ist.

5.2 Java Native Interface

Das Java Native Interface (JNI) [Lia99] ist ein Framework, über das von Java aus der Zugriff auf plattformspezifische Funktionen ermöglicht wird. Es ist direkt in die Programmiersprache Java integriert und bietet eine standardisierte Anwendungsprogrammierschnittstelle (API) an. Um JNI zu verwenden, ist es notwendig, eine Adapterbibliothek in einer Programmiersprache wie C oder C++ zu schreiben. Diese Bibliothek kann dann von Java aus geladen werden und die vorhandenen Funktionen werden automatisch an Javafunktionen gebunden. Benötigt wird dieses insbesondere, um hardwarenahe oder betriebsystemnahe Funktionen zu verwenden, beispielsweise für den Zugriff auf bestimmte Systemgeräte und Treiber.

Für den Simulator wird JNI verwendet, um die Shadow Hand und den Cyberglove anzusteuern.

5.3 Auswahl der Physikbibliothek

Da für den Simulator eine vorhandene Physikbibliothek verwendet werden sollte, mussten die existierenden Physikbibliotheken auf ihre Verwendbarkeit überprüft werden, um dann die beste auszuwählen. Grundsätzliche Voraussetzungen waren, dass der Einsatz unter Linux und Windows möglich ist und dass die Verwendung der Bibliothek kostenlos ist. Daher standen folgende Physikbibliotheken zur Auswahl:

- ODE (Open Dynamics Engine) [ODE01]/ ode4j [Zä09]
- Bullet [BUL06]/ JBullet [JBU08]
- Newton Game Dynamics [JS]

Die Open Dynamics Engine wird bereits seit 2001 entwickelt und findet deshalb in vielen Forschungsprojekten, Simulatoren und Computerspielen Verwendung. Seit einigen Jahren geht die Entwicklung allerdings nur noch sehr langsam voran. Ode4j, eine Portierung nach Java, existiert seit 2009 und bietet die meisten Funktionen von ODE.

Die Entwicklung der Bullet Physics Library wurde 2006 gestartet, die Portierung nach Java, JBullet, wird seit 2008 erstellt. Bullet wird in sehr vielen neueren Computerspielen und 3D-Modellierungswerkzeugen verwendet. Ein Entwicklungsziel bei Bullet ist, viele neuere Konzepte zur Dynamiksimulation einzusetzen. Es wird auch versucht, die besten Funktionen aus anderen Physikbibliotheken bei Bullet einzubauen.

Newton Game Dynamics verwendet laut Dokumentation im Gegensatz zu den meisten Physiksimulatoren einen deterministischen Constraintsolver, der eine bessere Stabilität bieten soll. Details über diesen Constraintsolver waren aber nicht herauszufinden, da der Sourcecode nicht verfügbar ist.

Bewertet wurden die Physikbibliotheken nach folgenden Eigenschaften:

- Erweiterbarkeit
- Verfügbarkeit von Dokumentationen
- Vorhandene Funktionalität
- Geschwindigkeit der Simulation
- Genauigkeit der Simulation

Die Erweiterbarkeit hängt hauptsächlich von der Lizenz ab, unter der die Bibliothek zur Verfügung steht. Newton Game Dynamics ist zwar kostenlos, allerdings ist der Quellcode nicht frei verfügbar (Freeware), während Bullet und ODE unter einer Open Source-Lizenz stehen und daher auch der Quellcode angesehen und modifiziert werden kann. Die Verfügbarkeit des Quellcodes ist auch wichtig, um Ursachen für ein fehlerhaftes Verhalten und mögliche undokumentierte Auswirkungen von Funktionsaufrufen herauszufinden.

Die Dokumentation war bei den meisten Bibliotheken erschreckend unvollständig. Bei Newton Game Dynamics war nur zu wenigen Funktionen überhaupt eine Dokumentation vorhanden und diese war oft auch fehlerhaft oder überholt, was um so schwerer wiegt, da der Quellcode nicht verfügbar ist. Bei Bullet ist größtenteils nur die Verwendung der Funktionen dokumentiert aber nicht das interne Verhalten. Die ausführlichste Dokumentation existiert für ODE. Bei Bullet und ODE ist es

aber oftmals nötig, den Quelltext anzusehen, um das Verhalten einer Funktion zu verstehen.

Die Funktionalität der einzelnen Bibliotheken unterscheidet sich deutlich. Grundlegende Funktionen wie konvexe Körper und die wichtigsten Arten von Constraints sind überall vorhanden. Konkave Körper werden aber nur von Bullet vollständig unterstützt, in ODE existiert hierfür allerdings bereits eine experimentelle Implementierung. ODE hat die größte Anzahl an vordefinierten Typen von Constraints. Dies bietet Vorteile, da man damit auch komplexe Verbindungen zwischen zwei Körpern ohne weitere Hilfsmittel modellieren kann. Sollte ein benötigter Constraint nicht vorhanden sein, muss man diesen entweder selbst implementieren oder aus mehreren einfacheren Hilfsconstraints zusammensetzen, was allerdings zu Genauigkeitsproblemen führen kann. Ein Konzept welches nur in Bullet existiert, betrifft die Motoren. Diese können direkt an den mit Constraints verbundenen Körpern Kräfte erzeugen und damit reale Motoren simulieren. Bei ODE und Newton Game Dynamics ist es erforderlich, diese Motorkräfte in einem zusätzlichen Schritt selbst zu berechnen.

Die Genauigkeit und die Geschwindigkeit der einzelnen Physiksimulatoren wurde in [BB07] mit mehreren Testreihen für unterschiedliche Bereiche untersucht. Dabei stellte sich heraus, dass jede Bibliothek in einzelnen Bereichen größere Genauigkeitsprobleme hat. Insbesondere Newton Game Dynamics zeigte sehr große Abweichungen von den physikalisch korrekten Werten. ODE und Bullet waren oft relativ ähnlich in der Genauigkeit, aber bei ODE funktioniert die Kollisionserkennung nicht zuverlässig. Dadurch können Objekte durcheinander durchfallen. Von der Geschwindigkeit war Bullet am besten, gefolgt von Newton Game Dynamics. ODE benötigte die längste Zeit für die Berechnungen. Insgesamt hat Bullet bei der Untersuchung am besten abgeschnitten.

Am wichtigsten für die Simulation ist eine sichere und stabile Erkennung von Kollisionen. Insbesondere mit konkaven Formen, die bei vielen zu simulierenden Gegenständen vorkommen, haben alle Physiksimulationen außer Bullet große Schwierigkeiten. Die Wahl fiel aufgrund dieser Ergebnisse auf die Physikbibliothek Bullet. Anstatt einen JNI-Adapter für Bullet zu schreiben, wurde allerdings die Java-Portierung JBullet verwendet, welche sich nur in wenigen Details von Bullet unterscheidet. Der größte Unterschied ist dabei das Fehlen von einigen seltener verwendeten Funktionen.

5.4 3D-Darstellung

Für die grafische Ausgabe gibt es eine Vielzahl von brauchbaren Bibliotheken. Besonders in Betracht gezogen wurden Java3D [JAV, SRD00] und LWJGL (Lightweight Java Game Library) [LWJ].

Java3D bietet Vorteile, da es auf einem sehr hohen Abstraktionsniveau arbeitet und daher relativ leicht und einfach zu verwenden ist. Hier werden anzuzeigende Ge-

genstände in einen Szenengraph eingehängt. Der Szenengraph enthält innere Knoten mit Transformationen und Bedingungen, mit denen die Anzeige zu steuern ist. Als Blattknoten sind die eigentlichen Geometriedatenobjekte enthalten, die dann angezeigt werden sollen. Wie aus diesem Szenengraph dann eine gerenderte Szene erzeugt wird, ist in Java3D verborgen, und man muss sich darum nicht kümmern.

LWJGL hingegen bietet keine eigenen Abstraktionen an, sondern bietet nur in Java eine direkte Anbindung an OpenGL. OpenGL [Gro97] ist eine standardisierte Programmierschnittstelle zur Erzeugung von 3D-Grafiken, welche auf einer sehr hardwarenahen Ebene arbeitet. Hier gibt es keinen Szenengraph, sondern alle Operationen wie das Löschen der Szene, das Zeichnen von Gegenständen, das Setzen von Transformationen müssen direkt als Befehl eingegeben werden. OpenGL funktioniert dabei intern als Zustandsmaschine. Für jeden Befehl ist also definiert, bei welchen Zuständen dieser verwendet werden darf und welche Auswirkungen dies auf den Nachfolgezustand hat.

Gemeinsam ist bei Java3D und LWJGL, dass beides nicht plattformunabhängig ist, aber auf den meisten aktuellen Computern ohne Änderungen verwendet werden kann. Außerdem ist mit beiden Systemen ein sehr schnelles Rendern der Grafik möglich, die Geschwindigkeitsunterschiede sind hierbei eher unbedeutend.

Bei dem Simulator fiel die Wahl aus zwei Gründen auf LWJGL. Zum einen hat ein Szenengraph in diesem Kontext keinen praktischen Nutzen, da durch die Physiksimulation bedingt sich immer alle Objekte sich auf derselben Ebene befinden. Eine Hierarchie von Objekten ist dabei nicht vorgesehen. Außerdem liegen bei vielen Gegenständen die Polygondaten direkt vor, da diese ja auch für die Physikberechnungen benötigt werden, wodurch eine Darstellung mit LWJGL auch sehr einfach ist. Der zweite Grund ist, dass die Beispiele bei JBullet auch LWJGL verwenden. Dadurch waren nur geringe Anpassungen notwendig, um die Objekte des Simulators zu zeichnen.

5.5 XML

XML (Extensible Markup Language, „erweiterbare Auszeichnungssprache“) [BPSM+08] ist eine Auszeichnungssprache, in der besonders hierarchische Strukturen gut darstellbar sind. Sie ist mit dem Ziel entworfen worden, von Menschen und auch von Maschinen einfach und effizient lesbar und schreibbar zu sein. Eine Auszeichnungssprache ist eine formale Sprache, in der Daten beschrieben werden können. Bei einigen Auszeichnungssprachen ist auch eine Beschreibung enthalten, wie diese Daten dargestellt oder verarbeitet werden sollen. Dies ist bei XML jedoch nicht der Fall.

Ein XML-Dokument enthält eine Baumstruktur, die aus inneren Knoten (Tags), Eigenschaften dieser Knoten (Attribute) und Blattknoten (Text) besteht. In dem Listing 5.1 ist ein XML-Dokument dargestellt. Jeder innere Knoten wird dabei mit einer spitzen Klammer begonnen, worauf der Name des Knotens folgt. Dahinter

können dann Attribute mit einem beliebigen Wert folgen. Nach der schließenden spitzen Klammer folgen die untergeordneten Knoten, von denen in diesem Fall vier Stück vorhanden sind. Der erste enthaltene Knoten ist ein Blattknoten, mit dem Inhalt `inhalt1`. Danach folgt ein weiterer innerer Knoten, der in diesem Fall aber keine weiteren untergeordneten Knoten hat. Dies erkennt man an dem Schrägstrich vor der schließenden Klammer. Am Ende eines Knotens, der nicht direkt geschlossen wird, muss noch einmal in spitzen Klammern dessen Name folgen, wobei vor dem Namen der Schrägstrich stehen muss.

Listing 5.1: Ein XML-Dokument

```
<tag1 attribut1="wert1" attribut2="wert2">
  inhalt1
  <tag2 attribut3="wert3" />
  <tag3>inhalt2</tag3>
  inhalt3
</tag1>
```

XML wird sehr oft eingesetzt, um Daten zwischen verschiedenen Programmen auszutauschen oder um Menschen die Möglichkeit zu bieten, gespeicherte Daten anzusehen und zu verändern. Bei dem Simulator soll XML für alle externen Dateien verwendet werden, also zur Beschreibung der simulierten Welten, zum Aufzeichnen von Simulationsverläufen und zum Speichern von einzelnen Zuständen.

6 Implementierungsdetails

In diesem Kapitel werden einige interessante Details der in der vorliegenden Diplomarbeit realisierten Implementierung genauer beschrieben. Insbesondere bei Abweichungen zu dem vorher vorgestellten Entwurf wird darauf eingegangen und es wird begründet, wieso diese Abweichungen sinnvoll waren. Die einzelnen Abschnitte dieses Kapitels sind untereinander nicht zusammenhängend, sondern folgen jeweils als detaillierte Beschreibung eines Abschnittes aus dem Entwurf.

6.1 Definition der Roboter und der Umgebung

Der in der Simulation vorhandene Roboter und die weiteren Gegenstände werden in einer Welt-Definitionsdatei beschrieben. Diese Dateien werden von Hand in einem XML-Format erstellt und sollen dann alle für den Simulator benötigten Informationen enthalten. Es war also notwendig, ein möglichst kurzes und auch leicht zu schreibendes Format hierfür zu entwickeln. Da in XML hierarchische Strukturen sehr gut ausdrückbar sind, wurden diese Hierarchien auch für die Beschreibungsdateien verwendet.

6.1.1 Weltbeschreibung

Auf der obersten Ebene der Welt-Definitionsdatei (`world.xml`) gibt es die Möglichkeit, einige Einstellungen für die Konfiguration der Simulation vorzunehmen. Unter anderem können hierbei die initiale Position der Kamera und einige für die Physikbibliothek wichtige Parameter festgelegt werden. Außer der Konfiguration werden dann noch die zu erzeugenden Gegenstände mit ihren Eigenschaften angegeben.

6.1.2 Prototypen und Instanzen

Um mehrfache sehr ähnliche Beschreibungen von Gegenständen zu vermeiden, wurde die Definition von Gegenständen in einen Prototyp und eine Instanz aufgeteilt. Eine Instanz, die jeweils auf einen Prototyp verweist, enthält dabei außer einem Namen nur die Information, an welcher Position sich der Gegenstand befinden soll und mit welchen Gegenständen er verbunden werden soll. Die restlichen Daten enthält der Prototyp, wobei mehrere Instanzen denselben Prototypen verwenden können. Der Prototyp enthält Informationen über die Form und Masse des Gegenstandes, sowie über die vorhandenen Verbindungsstellen, Sensoren, Aktuatoren und Regler. Sobald eine Instanz eines Gegenstandes erzeugt werden soll, wird dann der Prototyp

verwendet, um die weiteren untergeordneten Objekte zu erzeugen. Positioniert werden alle Objekte über dasselbe Verfahren. Es kann jeweils eine Position angegeben werden und eine Rotation um eine beliebige Achse. Hierzu muss der Normalenvektor der Rotationsachse angegeben werden und ein Winkel in Grad. Positionierungen von untergeordneten Objekten sind dabei immer relativ zu den übergeordneten Objekten. Ein Beispiel für eine Definition von Prototypen und Instanzen ist in Listing 6.1 zu sehen. Längeneinheiten werden dabei in Zentimeter angegeben und Gewichte in Gramm. Für Winkel wird das Gradmaß verwendet.

Listing 6.1: Definition von Prototypen und Instanzen

```
<prototype name="prot1">
  <geometry type="...">...</geometry>
  <connector type="hinge" name="c0">...</connector>
</prototype>
<instance name="obj1" prototype="prot1">
  <position>110 72 0</position>
</instance>
<instance name="obj2" prototype="prot1">
  <position>120 72 10</position>
  <rotationaxis>1 0 0 45</rotationaxis>
</instance>
```

Anstatt die Definition eines Prototypen direkt in der Weltdatei anzugeben, kann man auch auf eine weitere Datei verweisen. Dadurch ist es möglich, Bibliotheken von Gegenständen anzulegen, die dann von mehreren Welt-Beschreibungen verwendet werden.

Prototypen können zusätzlich auch aus mehreren Gegenständen zusammengesetzt sein. Dies erreicht man dadurch, dass innerhalb eines Prototypen wiederum Instanz-Deklarationen vorkommen. Diese werden dann bei der Instanziierung des Prototypen zusätzlich erzeugt. Dadurch können mit einer einzigen Instanzdeklaration mehrere Gegenstände erzeugt und auch verbunden werden. Bei der Roboterdefinition wird dieses unter anderem bei der Definition der Finger verwendet. Dadurch, dass ja vier Finger der Roboterhand identisch sind, muss die Beschreibung des zusammengesetzten Fingers nur einmal existieren, und es werden dann alle vier Finger daraus erzeugt.

6.1.3 Geometriedaten

Jeder Prototyp kann mehrere Geometriedatenblöcke enthalten. Diese werden dann als ein zusammenhängender, starrer Gegenstand erzeugt. Positionen von Geometriedaten sind jeweils relativ zum Koordinatensystem angegeben. Geometrieobjekte sind dabei entweder geometrische Primitive (Quader, Kugel oder Zylinder), oder sie haben eine aus einem Polygonnetz bestehende Form. Diese Polygondaten sind dann in einzelnen externen Dateien gespeichert und können in verschiedenen For-

maten vorliegen. Unterstützt werden sollen auf jeden Fall die Formate „Open Inventor“ [IV] und „Wavefront Object“ [OBJ] , in denen die Beschreibungen der Shadow Hand vorliegen. Zusätzlich wird das „Object File Format“ [OFF] unterstüzt, welches vom Princeton Shape Benchmark verwendet wird. In Listing 6.2 wird ein Prototyp definiert, dessen Geometrie aus einem Zylinder und einem Polygonnetz zusammengesetzt ist. Bei Polygondaten ist es möglich, eine Skalierung anzugeben, die beim Laden des Geometrie verwendet wird. Dies ist notwendig, da unterschiedliche Skalierungen gebäuchlich sind.

Listing 6.2: Definition von Geometriedaten

```
<prototype name="prot1">
  <geometry type="cylinder">
    <radius>1</radius>
    <length>30</length>
    <mass>50</mass>
    <position>0 0 0</position>
  </geometry>
  <geometry type="mesh">
    <file>f1.obj</file>
    <scale>0.1 0.1 0.1</scale>
    <mass>50</mass>
    <position>0.0 -1.35 0.0</position>
    <rotationaxis>0 0 1 90</rotationaxis>
  </geometry>
</prototype>
```

6.1.4 Verbindungen

Eine Verbindung zwischen zwei Gegenständen wird in ihrer Beschreibung aufgeteilt. Bei jedem Prototypen können hierbei Verbindungspunkte definiert werden. Diese haben auch eine relative Positon im Koordinatensystem des Prototypen und einen Typ (Unbeweglich, Scharnier, . . .). Bei der Erzeugung einer Instanz ist es dann möglich, diese Verbindungspunkte jeweils mit einem Verbindungspunkt eines anderen Gegenstandes zusammenzusetzen. Eine der Verbindungen kann als „Basisverbindung“ ausgezeichnet werden. Die besondere Bedeutung dieser Basisverbindung liegt darin, dass ein Gegenstand, der mit einer Basisverbindung an einen anderen Gegenstand angekoppelt wird, automatisch relativ zu diesem positioniert wird. Dadurch ist es nicht notwendig, beispielsweise bei der Verbindung einer Roboterhand mit einem Roboterarm die genaue Position und Größe des Armes zu kennen. Die Hand wird in diesem Fall automatisch am Ende des Armes platziert, wenn bei dem Arm eine Verbindungsstelle an der richtigen Position existiert. Bei einer Definition wie in Listing 6.3 würde dann der Gegenstand `obj2` automatisch relativ zu `obj1` positioniert werden und mit einem Scharnier zu diesem verbunden werden.

Listing 6.3: Eine Scharnier-Verbindung zwischen Gegenständen

```
<prototype name="prot1">
  <geometry type="...">...</geometry>
  <connector type="hinge" name="c0">
    <position>0 42 0</position>
    <limit>-45 30</limit>
  </connector>
</prototype>
<prototype name="prot2">
  <geometry type="...">...</geometry>
  <baseconnector type="hinge">
    <position>0.0 -1 0.0</position>
  </baseconnector>
</prototype>

<instance name="obj1" prototype="prot1">
  <position>110 72 0</position>
</instance>
<instance name="obj2" prototype="prot2">
  <connectTo>obj1</connectTo>
</instance>
```

6.1.5 Sensoren, Aktuatoren und Regler

Eine Beschreibung der an einem Gegenstand vorhandenen Sensoren, Aktuatoren und Reglern ist ebenfalls in der Prototypdefinition vorhanden. Hier können die genauen Eigenschaften dieser Objekte festgelegt werden. Bei einer Erzeugung eines Gegenstandes aus diesem Prototyp werden angehängte Objekte automatisch mit erzeugt.

6.2 Berechnung des Schwerpunktes eines Gegenstandes

Für eine realitätsnahe Simulation starrer Körper ist es wichtig, dass der Schwerpunkt jedes Gegenstandes bekannt ist. Der Schwerpunkt eines beliebigen Gegenstandes berechnet sich allgemein nach der Formel

$$\vec{x}_s = \frac{1}{G} \int_V \vec{x} \cdot g(\vec{x}) \cdot \rho(\vec{x}) \mathrm{d}V \text{ mit } G = \int_V g(\vec{x}) \cdot \rho(\vec{x}) \mathrm{d}V. \tag{6.1}$$

Hierbei ist $g(\vec{x})$ die Schwerkraft, und $\rho(\vec{x})$ die Dichte des Gegenstandes an dem Punkt $\vec{x}$. Das Integral wird dabei über das gesamte Volumen des Gegenstandes gebildet. Bei der Simulation kleiner Gegenstände kann man davon ausgehen, dass die Schwerkraft an allen Punkten gleich ist. Wenn außerdem nichts über die Dichte des

Gegenstandes bekannt ist, kann man hier nur annehmen, dass sie überall innerhalb der Hülle konstant ist.

Bei den geometrischen Primitiven ist es wegen deren punktsymmetrischen Aufbaues einfach, diesen Schwerpunkt zu bestimmen. Der Schwerpunkt ist hier immer identisch mit dem Symmetriezentrum. Bei Polygonformen ist dies jedoch nicht der Fall. Es ist hierbei möglich, dem Simulator den Schwerpunkt eines Gegenstandes anzugeben, wenn dieser bekannt ist. Sollte der Schwerpunkt nicht angegeben sein, wird ein vereinfachtes Modell verwendet, um diesen zu berechnen. Diese Vereinfachung ist nötig, da eine Integration über das Volumen eines beliebig geformten dreidimensionalen Gegenstandes zu aufwändig wäre. Es wird dabei angenommen, dass sich die gesamte Masse des Objektes auf der Oberfläche verteilt. Man kann dann zuerst den Schwerpunkt jedes Polygons der Oberfläche bestimmen. Die Masse eines Polygons ist dann proportional zu seiner Fläche. Da die vorhandenen Polygone in der Simulation immer Dreiecke sind, deren Kantenlängen bekannt sind, lässt sich zur Berechnung der Satz von Heron anwenden. Nach diesem ist die Fläche $F = \sqrt{s(s-a)(s-b)(s-c)}$ wobei a, b, c die Kantenlängen sind und $s = \frac{a+b+c}{2}$ dem halben Umfang entspricht. Der Schwerpunkt eines Dreieckes ist $S = \frac{1}{3}(A + B + C)$, wobei A, B, C die Koordinaten des Eckpunkte sind. Anschließend wird der Schwerpunkt des Gegenstandes ermittelt. Dazu wird folgende Formel verwendet:

$$\vec{x}_s = \frac{1}{m} \sum_i \vec{x}_i m_i \tag{6.2}$$

Hier sind $\vec{x}_i$ die Schwerpunkte der Dreiecke und m_i die jeweilige Masse der Dreiecke. Das Ergebnis ist dann der Vektor, der den Schwerpunkt der Polygonform enthält.

6.3 Implementierung der taktilen Sensoren

Zur Bestimmung des Sensorwertes der taktilen Sensoren sollten alle Kollisionen berücksichtigt werde, die auf einem Gegenstand festgestellt werden. Bei Experimenten mit der Simulation ergab sich allerdings das Problem, dass diese Sensorwerte sehr stark schwanken, da die simulierten Objekte sich niemals vollständig in Ruhe befinden. Diese Unruhe der Gegenstände ist bei einer Simulation mit Zeitschritten unvermeidlich. Die Ursache hierfür ist, dass wenn eine Kollision zwischen zwei Gegenständen erkannt wurde, sich diese Gegenstände schon ein kleines Stück ineinander bewegt haben. Deshalb wird dann eine Kraft erzeugt, die diesen Fehler wieder beheben soll. Hierdurch bewegen sich die Gegenstände dann aber wieder leicht auseinander,weshalb dann kurzzeitig keine Kollision mehr besteht. In einfachen Situationen mit kurzen kinematischen Ketten würden diese Schwingungen aber nach kurzer Zeit zur Ruhe kommen. Damit die Sensoren trotzdem brauchbare Messwerte liefern, muss der eingehende Wert daher geglättet werden. Dafür kann man jedem Sensor einen Faktor (kleiner 1) angeben, mit dem jeweils der vorherige Messwert

multipliziert wird, wonach dann der neue Messwert hinzuaddiert wird. In Abbildung 6.1 sind diese Sensorwerte ungeglättet und geglättet mit einem Faktor von 0.9 dargestellt.

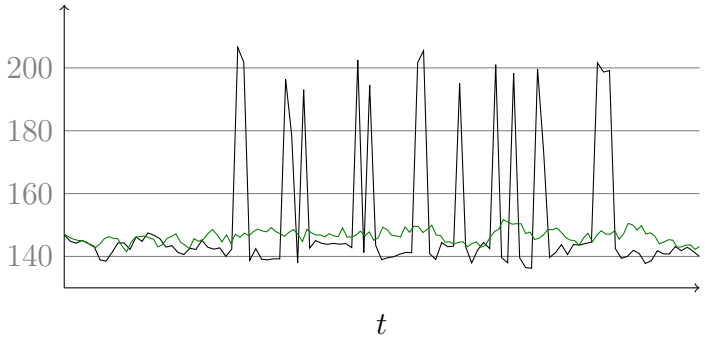


Abbildung 6.1: Verlauf der Sensorwerte über die Zeit

Zur Anpassung an reale Sensoren gibt es noch die Möglichkeit, den erhaltenen Messwert mit einem beliebigen konstanten Faktor zu multiplizieren und einen Offset hinzuzuaddieren. Danach kann der Wert noch auf einen bestimmten Bereich eingegrenzt werden. Dies ist sinnvoll, um Sensoren zu simulieren, die erst bei einer gewissen Kraft ansprechen. Alle simulierten taktilen Sensoren haben innerhalb ihres Bereiches immer eine lineare Kennlinie, eine genauere Anpassung an reale Sensoren ist daher derzeit nicht möglich.

Auf vergleichende Experimente mit den derzeit an der Hand angebrachten taktilen Sensoren wurde verzichtet, da diese so spät ansprechen, dass sie praktisch unbrauchbar sind.

6.4 Anpassung der Aktuatoren an die Simulation

6.4.1 Dynamische Motoren

Als Aktuatoren existieren derzeit nur Motoren, die Rotationskräfte auf mit einem Gelenk verbundene Gegenstände ausüben. Die Möglichkeit, diese Rotationskräfte auf einen Gegenstand auszuüben, ist direkt in der Physiksimulation vorhanden. Bullet erwartet hier, dass man die erwünschte Zielgeschwindigkeit in rad/s und den maximal möglichen Impuls, um diese Geschwindigkeit zu erreichen, angibt. Der maximale Impuls bleibt hierbei über den Verlauf der Simulation konstant, und die Zielgeschwindigkeit soll zur Steuerung dienen. Hierfür wurden PID-Regler implementiert, denen man den gewünschten Winkel als Sollwert vorgeben kann Es wird dann die Differenz zum aktuell gemessenen Wert als Regelabweichung verwendet und berechnet, welche Zielgeschwindigkeit an die Physiksimulation weitergegeben wird.

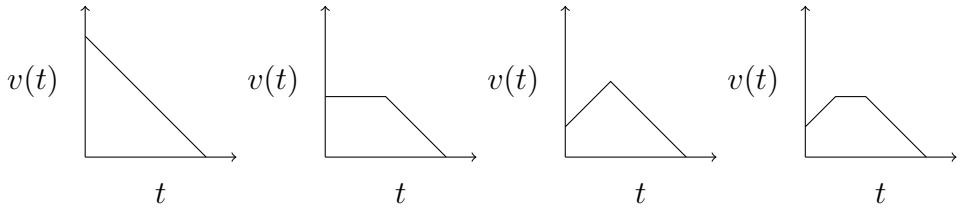
Bei der Simulation traten allerdings zwei Probleme auf. Zum einen ist es nicht vorgesehen, dass in Gelenken eine Reibung berechnet wird. Dieses Problem lässt sich

dadurch verringern, dass bei den PID-Reglern der Faktor für das D-Glied erhöht wird. Dadurch wird bei jeder Bewegung des Gelenkes eine Gegenkraft erzeugt, die diese Bewegung bremst. Allerdings wird dann auch durch die Änderung des Sollwertes eine Reaktion ausgelöst. Als Alternative wurde deshalb zusätzlich zu dem Regler noch eine Funktion eingebaut, die nur auf Änderungen des gemessenen Winkels reagiert und dann eine Gegenkraft erzeugen soll. Es wird also die Winkeländerung mit einem einstellbaren negativen Faktor multipliziert und zusätzlich auf die vom Regler bestimmte Zielgeschwindigkeit addiert. Dies ermöglicht dann eine brauchbare, allerdings nicht ganz realistische, Simulation der Reibung (siehe auch Abschnitt 7.1.2).

Das zweite Problem war, dass bei dem Testen des Robotermodells im Simulator, bei dem also die Hand und der Arm zusammen simuliert werden sollten, eine große Ungenauigkeit bei der Simulation der Finger auftrat. Die Ursache hierfür liegt am Prinzip, wie die Bewegungen bestimmt werden. Durch den Constraintsolver werden alle Constraints gleichermaßen behandelt und es wird versucht, diese zu erfüllen. Tritt nun beispielsweise bei einem Armgelenk ein kleiner Fehler auf und dieser wird korrigiert, entsteht dadurch ein großer Fehler an den Fingern, relativ zur Größe der Finger. Dieser Fehler tritt also immer auf, wenn die Größenunterschiede zwischen den verbundenen Objekten zu groß werden – auch in der Dokumentation zu Bullet wird erwähnt, dass man möglichst einen Größenfaktor von 10 nicht überschreiten sollte. Bei dem Roboter ist aber der Unterschied zwischen der Länge eines Fingergliedes mit etwa 2 cm und der Länge eines Armstückes mit bis zu 50 cm mehr als doppelt so groß. Es war möglich, die Fehler durch eine Erhöhung der Iterationen des Solvers zu verringern, was allerdings zu einer Verlangsamung der Simulation führt. Die Lösung bestand dann darin, den Roboterarm nicht mehr durch die Physiksimulation zu bewegen, sondern als statisches Objekt zu modellieren und dann mit Vorwärtskinematik die Postion der Armglieder zu berechnen. Dies ist auch sinnvoll, da der Roboterarm eine sehr große Steifigkeit besitzt und daher gut vorherzusagen ist. Eine kinematische Simulation reicht hier also völlig aus.

6.4.2 Kinematische Motoren

Der Simulator unterstützt deshalb eine zweite Art von Aktuatoren, die für die Gelenke des Armes verwendet werden. Diesen Aktuatoren wird direkt der Zielwinkel vorgegeben und es wird dann unter der Verwendung von Beschleunigungskurven die Bewegung berechnet. Diese Beschleunigungskurven verhindern ein ruckartiges Bewegen, da die maximale Beschleunigung limitiert werden kann. Für die Armgelenke muss dem Simulator beim Laden der Welt jeweils die maximale Geschwindigkeit und die maximale Beschleunigung vorgegeben werden. Dies entspricht genau den Möglichkeiten der Gelenke des realen Roboterarmes. In jedem Zeitschritt wird dann die Änderung der Geschwindigkeit und die Änderung der Position berechnet, wobei mehrere Fälle in Betracht gezogen werden müssen. Diese Fälle, die in Abbildung 6.2 dargestellt sind, können dann wie folgt unterschieden werden:



(a) Fall 1 (b) Fall 2 (c) Fall 3 (d) Fall 4

Abbildung 6.2: Mögliche Fälle bei der Berechnung der Beschleunigung

1. Es muss sofort gebremst werden, um am Zielpunkt zu stoppen. Dieser Fall tritt auch ein, wenn es nicht mehr möglich ist, am Zielpunkt zu stoppen, weil während der Bewegung des Armes der Zielpunkt verändert wurde.

2. Es kann erst mit der aktuellen Geschwindigkeit weiterbewegt werden, bis dann gestoppt werden muss. Dieser Fall tritt ein, wenn bereits die maximale Geschwindigkeit erreicht wurde.

3. Es wird zuerst beschleunigt und danach direkt gebremst. Dieser Fall tritt ein, wenn die maximale Geschwindigkeit noch nicht erreicht wurde und auch am Punkt der höchsten Geschwindigkeit nicht überschritten wird.

4. Es wird bis zur maximalen Geschwindigkeit beschleunigt, dann diese Geschwindigkeit beibehalten und später gebremst.

Zur Bestimmung, ob der erste Fall vorliegt, muss nur die Strecke berechnet werden, die mindestens noch zurückgelegt werden muss. Wenn Δs die zurückzulegende Strecke ist, v der Betrag der aktuellen Geschwindigkeit, v_{max} die maximale Geschwindigkeit und a_{max} die maximale Beschleunigung, lässt sich dies folgendermaßen durchführen:

$$t_{min} = v/a_{max} \tag{6.3}$$

$$s_{min} = vt_{min} - \frac{1}{2}a_{max}t_{min}^2 \tag{6.4}$$

Wenn der erste Fall nicht vorliegt, muss der zweite Fall vorliegen, wenn die maximale Geschwindigkeit bereits erreicht ist.

Die Bestimmung der letzten beiden Fälle ist etwas aufwändiger. Hierzu müssen die Formeln für die Geschwindigkeit und die zurückgelegte Strecke zu einem Zeitpunkt t aufgestellt werden, unter der Annahme, dass am Punkt t_x die maximale Geschwindigkeit vorliegt. Wenn man nun die Geschwindigkeit am Zeitpunkt t auf 0 setzt und die zurückgelegte Strecke auf Δs, kann man durch Umformungen zu den Werten von t und t_x gelangen. Nun muss die Geschwindigkeit zum Zeitpunkt

t_x berechnet und mit der Maximalgeschwindigkeit verglichen werden. Sollte die Geschwindigkeit $v(t_x)$ größer als die Maximalgeschwindigkeit sein, handelt es sich um Fall 4. Ansonsten liegt Fall 3 vor.

$$s(t) = vt_x + \frac{1}{2}a_{max}t_x^2 + (v + at_x)(t - t_x) - \frac{1}{2}a(t - t_x)^2 = \Delta s \tag{6.5}$$

$$v(t) = v + at_x - a(t - t_x) = 0 \tag{6.6}$$

Durch Verwendung dieser Formeln ist es möglich, ein Gelenk an exakt der gewünschten Position zu stoppen und es ist auch möglich, jederzeit während einer Bewegung ein neues Ziel festzusetzen.

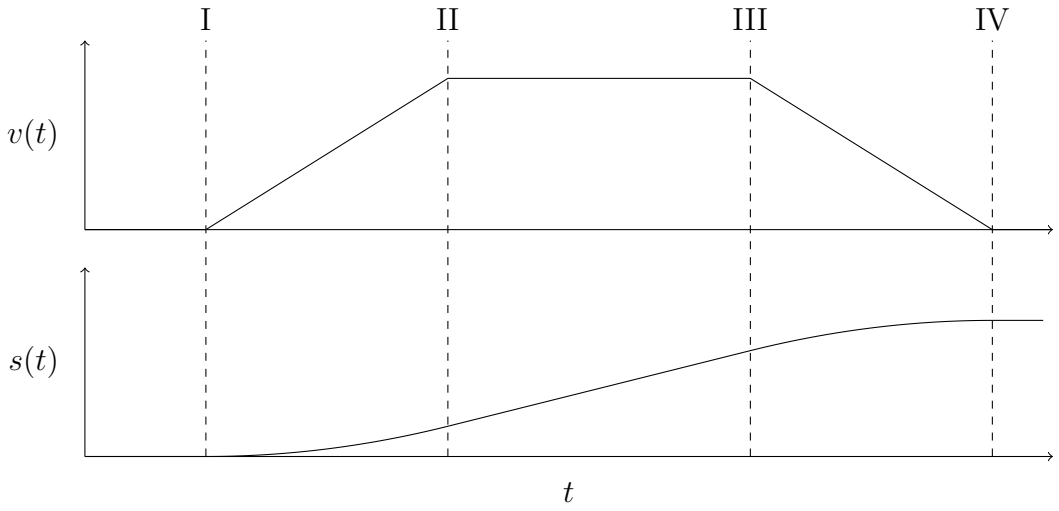


Abbildung 6.3: Bewegungskurve eines kinematisch gesteuerten Gelenks

In Abbildung 6.3 ist als Beispiel eine vollständige Bewegungskurve dargestellt. In diesem Fall beginnt die Geschwindigkeit bei 0. An dem Punkt I, an dem ein Bewegungsbefehl gegeben wird, wird bestimmt, um welchen der vier Fälle es sich hier handelt. Dies ist Fall 4, daher wird die Position von Punkt II bestimmt, an dem die maximale Geschwindigkeit a_{max} erreicht wird. Bis hier wird mit a_{max} beschleunigt. Sobald Punkt II erreicht ist, wird wiederum der Fall bestimmt, wobei das Ergebnis nun Fall 2 ist. Es wird also bestimmt, ab welcher Stelle wieder gebremst werden muss (Punkt III) und bis dort mit konstanter Geschwindigkeit weiterbewegt. Ab Punkt III tritt nun Fall 4 in Kraft, wobei bis Punkt IV abgebremst wird, um den Zielpunkt dann zu erreichen.

6.5 Erstellung des Robotermodells

Da für den Simulator ein möglichst realistisches Modell des Roboters erstellt werden musste, wurden hier zuerst die einzelnen starren Bestandteile modelliert und diese

dann mit Gelenken verbunden. Für die Positionen und Begrenzungen der Gelenke wurden die in den Handbüchern [PA00] und [Sha08] beschriebenen Werte verwendet. Damit die Simulation mit akzeptabler Geschwindigkeit läuft, wurden bei der Form der einzelnen Bestandteile nicht die genauen Formen des realen Roboters nachgebildet. Hier wurde stattdessen an den meisten Stellen auf einfache geometrische Primitive zurückgegriffen, die aber von den Abmessungen ähnlich wie in der Realität sind. Nur bei den äußersten Fingergliedern, bei denen eine genaue Form sehr wichtig ist, wurde das von der Shadow Robot Company zur Verfügung gestellte 3D-Modell als Grundlage verwendet. In Abbildung 6.4 ist das erstellte Handmodell dargestellt.

Die Masse der einzelnen Teile wurde geschätzt, da hierzu keine genauen Daten vorlagen. Auch die Kräfte, die auf die Fingergelenke durch die pneumatischen Muskeln ausgeübt werden, waren nicht bekannt. Deshalb wurden auch hier Werte verwendet, die in der Simulation ein ähnliches Verhalten wie in der Realität ergeben sollten. Das sehr komplexe Verhalten der Fingergelenke, das durch die zwei pneumatischen Muskeln in Verbindung mit vier Reglern hervorgerufen wird, konnte damit allerdings nicht nachgebildet werden. Um das reale Verhalten nachzubilden, wären nichtlineare Regler notwendig.

Die Gelenke des Roboterarmes werden alle über kinematische Motoren gesteuert. Der Roboterarm selbst kann also nicht durch die Physiksimulation bewegt werden. Hierdurch ist allerdings die erzielte Genauigkeit bei der Positionierung deutlich höher und entspricht auch eher der Positionierung des realen Armes. Kollisionen des Armes mit der Umgebung sind in der Realität sowieso zu vermeiden, da dies schnell zu einer Beschädigung führen würde. Eine korrekte Simulation einer solchen Kollision ist daher auch nicht notwendig.

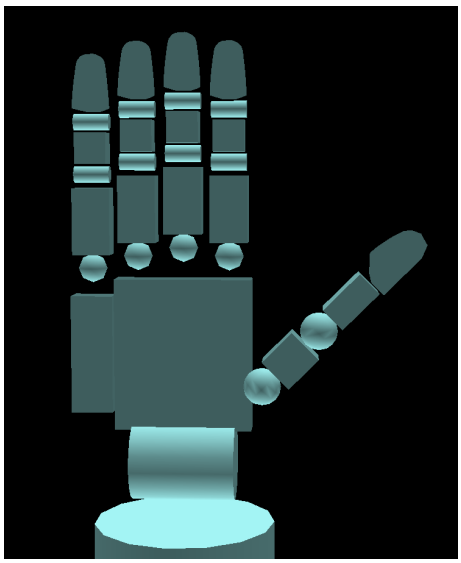


Abbildung 6.4: Das erstellte Modell der Hand

Für die, in dieser Diplomarbeit durchzuführenden, Experimente mussten zusätzlich einige Gegenstände modelliert werden. Hierbei wurden teilweise reale Gegenstände als Vorbild genommen. Einer der Gegenstände ist ein Würfel mit 4 cm Kantenlänge und 20 g Gewicht. Ein weiterer Gegenstand ist ein Block aus vier zusammengesetzten Zylindern mit jeweils einem Radius von 1,2 cm und einer Länge von 5 cm. Dieser wiegt 20 g. Diese beiden Gegenstände existieren auch mit einer ähnlichen Form als Holzklötze in der realen Versuchsumgebung und sie sind als Versuchsgegenstände im HANDLE-Projekt vorgesehen. Daneben gibt es noch einen größeren Block aus vier Zylindern mit jeweils 2,2 cm Durchmesser und 8 cm Länge, der insgesamt 100 g wiegt. Dieser wurde verwendet, um das Greifen von größeren Gegenständen zu testen. Als letztes existiert noch ein Zylinder von 30 cm Länge mit einem Radius von 1 cm und einem Gewicht von 50 g.

Die Definition weiterer Objekte ist unproblematisch. Durch den in 6.1.2 und 6.1.3 beschriebenen Prozess können beliebige Objekte des Princeton Shape Benchmark hinzugefügt werden.

6.6 Ansteuerung des CyberGlove

Die simulierte Roboterhand soll auch über den CyberGlove steuerbar sein. Daher war es notwendig, dessen Daten für die Simulation verfügbar zu machen. Der CyberGlove, der über ein serielles Kabel an den PC angeschlossen wird, akzeptiert dabei das in [Vir98] beschriebene Protokoll. Von diesem Protokoll wird nur der Befehl, der die aktuellen Sensorwerte abfragt, verwendet. Diese Werte sollen zur Ansteuerung der einzelnen Gelenke verwendet werden. Hierbei war zu beachten, dass die Anordnung der Sensoren nicht der Anordnung der Gelenke der Shadow Hand entspricht und dass die Sensorwerte unkalibriert sind.

Zur Ansteuerung des CyberGlove wird ein Programmteil in der Programmiersprache C benötigt, der dann über JNI mit dem restlichen Programm verbunden wird. Die Daten stehen dann unkalibriert in Java zur Verfügung. Zur Kalibrierung wird ein einfaches Verfahren verwendet. Es werden pro Sensor jeweils der maximale und der minimale erhaltene Wert gespeichert und diese Werte werden als Vollausschlag gewertet. Mit Hilfe dieser Werte werden alle weiteren Werte auf einen Bereich von 0 bis 1 normiert.

$$s_{norm} = (s - s_{min})/(s_{max} - s_{min}) \tag{6.7}$$

Es handelt sich hierbei um eine lineare Interpolation. Für jedes Gelenk wird dann vorgegeben, aus welchem dieser Sensoren es seine Werte erhalten soll. Dabei kann zusätzlich eine weitere Skalierung und ein Offset angegeben werden. Der hierbei erhaltene Wert wird dann auf die Begrenzungen des Gelenkes umgerechnet.

$$j = (s_{norm} \cdot scale + offset) \cdot (j_{max} - j_{min}) + j_{min} \tag{6.8}$$

Durch dieses Verfahren ist es möglich, mit nur geringem Kalibrierungsaufwand brauchbare Sensorwerte zu erhalten. Es ist nur notwendig, dass der Benutzer des

Handschuhs alle Gelenke einmal bis zu den Grenzen der eigenen Hand bewegt. Danach können die Daten an den Simulator oder an die Shadow Hand übertragen werden.

Die Ansteuerung der Shadow Hand hiermit war allerdings problematisch, da es zu einer deutlichen Verzögerung der Fingerbewegung kommt. Diese Verzögerung liegt vermutlich an der Hand selbst, da derartige Verzögerungen im Simulator nicht auftraten. Eine Ursache des Problems sind die pneumatischen Muskeln. Wenn hier bei mittleren Gelenkwinkeln beide Muskeln mit Luft gefüllt sind, führen kleine Änderungen am Luftdruck nur zu sehr geringen Auswirkungen und die Finger reagieren sehr träge. Im Kapitel Abschnitt 7.2 wird dieses Verhalten näher analysiert.

6.7 Benutzungsschnittstellen des Simulators

Um den Zustand des Roboters und der Welt in der Simulation für den Benutzer darzustellen, besitzt der Simulator ein Aufgabefenster, in dem die dreidimensionalen Gegenstände abgebildet werden. Dieses Ausgabefenster ist in Abbildung 6.5 dargestellt.

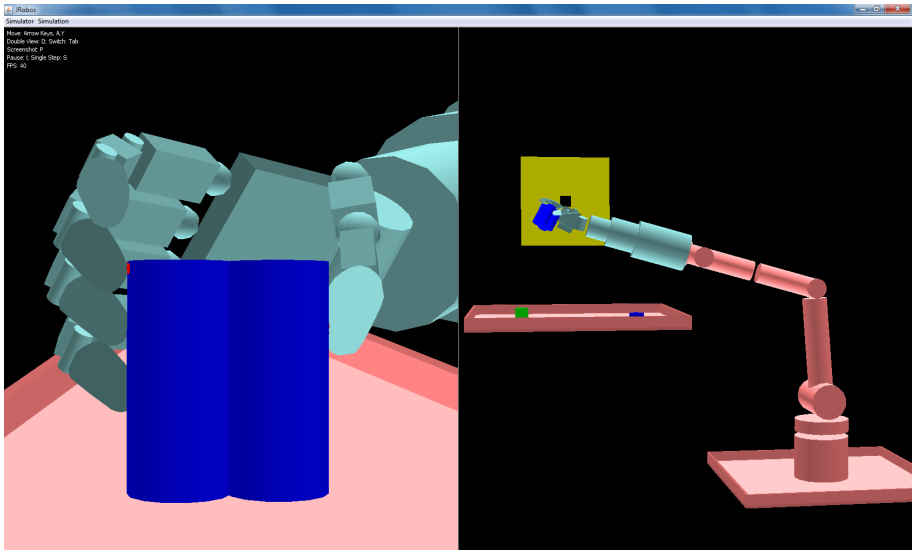


Abbildung 6.5: 3D-Ausgabefenster des Simulators. Split-Screen mit zwei unterschiedlichen Ansichten.

Zur Darstellung wird hier LWJGL verwendet, welches eine schnelle Ausgabe ermöglicht. LWJGL erwartet, dass man jeweils zuerst eine Matrix angibt, welche die Position des Gegenstandes bestimmt. Hierbei kann die Positionsmatrix verwendet werden, die von der Physiksimulation geliefert wird. Danach müssen die Polygondaten des Gegenstandes übertragen werden. Geometrische Primitive mit runden Formen, also Kugeln und Zylinder, kann man nicht direkt darstellen. Aus diesen muss

zuerst ein Polygonnetz erzeugt werden. Dies ist notwendig, da die Grafikhardware nur ebene Flächen von Polygonen schnell zeichnen kann. Bei einem Zylinder wird die Mantelfläche dazu in 16 ebene Flächen eingeteilt. Auch die Oberseite und die Unterseite werden jeweils in 16 Flächen eingeteilt. Hierdurch erscheint der Zylinder allerdings nicht mehr vollständig rund. Bei einer sehr stark vergrößerten Darstellung sind die einzelnen ebenen Flächen erkennbar. Bei einer Kugel wird ähnlich verfahren. Hier muss die Aufteilung aber in zwei Achsen erfolgen. Kugeln werden durch ein Netz aus 64 Dreiecken dargestellt. Diese Polygonnetze müssen für jeden Gegenstand nur einmal erzeugt werden. Danach werden sie gespeichert und können dann beliebig oft zur Darstellung verwendet werden, da sie jeweils in lokalen Koordinaten angegeben sind.

Nach der Ausgabe der Gegenstände werden noch alle Kontaktpunkte, die von der Physiksimulation berechnet wurden, als kleine rote Kugeln dargestellt. Dadurch kann man leicht erkennen, ob ein Finger wirklich einen Kontakt zu einem Gegenstand hat und man sieht auch die genaue Position dieses Kontaktes.

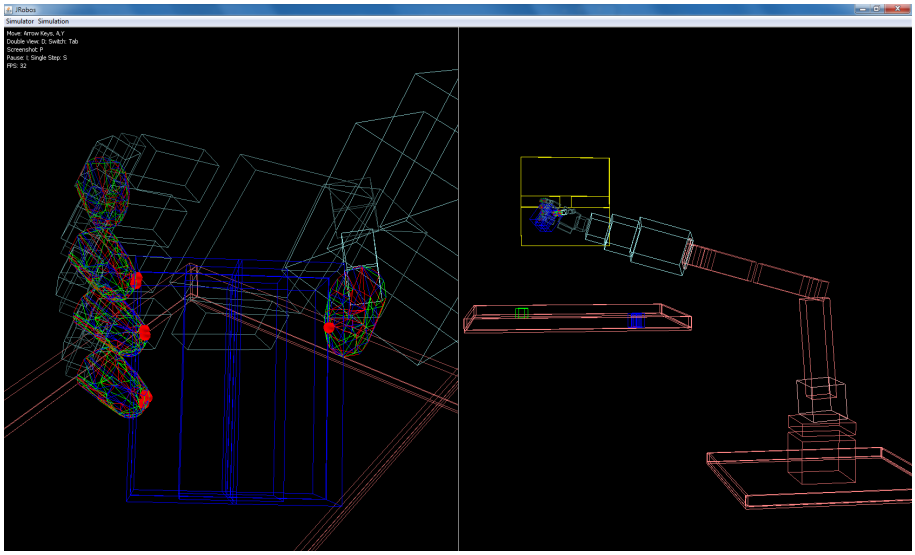


Abbildung 6.6: 3D-Ausgabefenster des Simulators. Anzeige als Wireframe.

In einer zweidimensionalen Ansicht einer dreidimensionalen Welt können oftmals wichtige Teile verdeckt sein. Da es bei der Steuerung des Roboters aber sehr wichtig ist, den Zustand der Welt möglichst genau feststellen zu können, wurde die Möglichkeit eingebaut, die Darstellung aufzuteilen. Damit kann dann die Welt aus zwei unterschiedlichen Positionen angezeigt werden. Außerdem wurde eine Option eingebaut, mit der die Oberflächen der Gegenstände ausblendbar sind. Es werden dann nur die Konturen der Polygone angezeigt. Dies wird auch als Wireframe bezeichnet. In Abbildung 6.6 ist der Simulator im Wireframe-Modus dargestellt. Gut sichtbar sind hier auch die roten Kontaktpunkte.

Damit man auch ohne Verwendung des Roboters in der Welt Gegenstände bewegen kann, gibt es noch die Möglichkeit, diese mit der Maus zu verschieben. Hierzu muss man einen Gegenstand anklicken und kann diesen dann verschieben. Da die Maus aber nur ein zweidimensionales Eingabegerät ist, ist diese Verschiebung nur in der Bildebene möglich. Es ist als nicht möglich, einen Gegenstand anzuklicken und näher heranzuziehen.

Um die Sollwerte der Winkel vorzugeben und um die aktuellen Winkel abzufragen, gibt es ein Steuerungsinterface. Dies ist in Abbildung 6.7 dargestellt. Mit diesem Interface lässt sich sowohl der Simulator als auch der reale Roboter steuern. Es werden hier alle Gelenke mit ihrem Namen und der Möglichkeit, die Zielwinkel zu verändern, angezeigt. Außerdem kann man einen Verlauf der Winkel über die Zeit anzeigen lassen, um das Verhalten des Roboters besser untersuchen zu können. Über diese Schnittstelle ist es auch möglich, die Daten aufzuzeichnen und die Steuerung über den Cyberglove zu aktivieren.

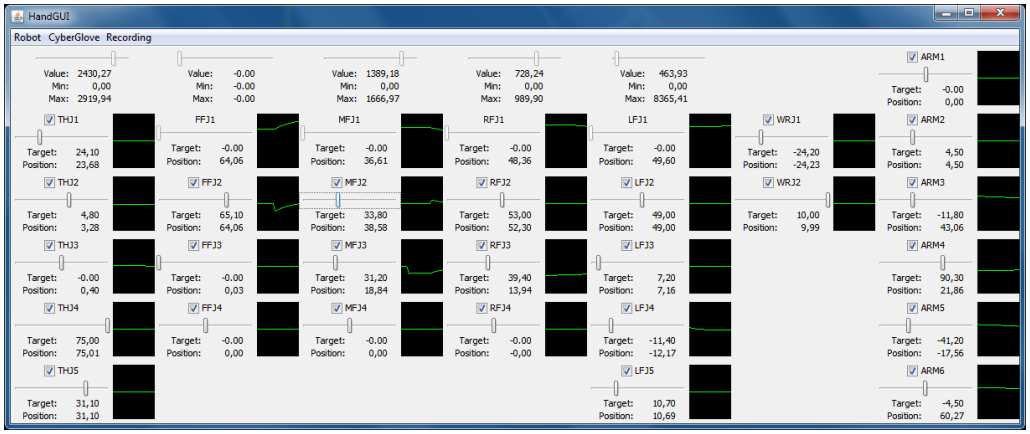


Abbildung 6.7: Steuerung des Simulators

7 Messungen und Experimente

Um die Genauigkeit der Simulation zu überprüfen, wurden unterschiedliche Messungen und Experimente durchgeführt. Eine der Messreihen sollte zeigen, welche Werte für die verschiedenen Genauigkeitsparameter der Simulation sinnvoll sind. Diese Parameter sind die Länge der Zeitschritte und die Anzahl an Iterationen des Constraintsolvers. Eine weitere Messung diente dazu, das Verhalten der dynamisch bewegten Gelenke festzustellen und mit den Gelenken der Shadow Hand zu vergleichen. Mit den darauffolgenden Experimenten, bei denen einige Gegenstände mit dem simulierten Roboter bewegt wurden, konnte gezeigt werden, dass der Simulator funktionsfähig ist und sich für Manipulationen mit Mehrfinger-Roboterhänden eignet. Bei der Durchführung der Experimente war es auch möglich, einige Ungenauigkeiten bei der Simulation auf ihre Auswirkungen zu überprüfen.

7.1 Auswirkungen unterschiedlicher Simulationsparameter

Es gibt bei der Physiksimulation Bullet zwei Parameter, die große Auswirkungen auf die Genauigkeit der Simulation haben. Dies sind die Länge der Zeitschritte und die Anzahl an Iterationen des Constraintsolvers.

Durch kürzere Zeitschritte werden die Bewegungen pro Zeitschritt kleiner, so dass mögliche Kollisionen schneller erkannt werden. Außerdem können die Regler, die mindestens einen Zeitschritt Verzögerung haben, schneller reagieren. Da aber in jedem Zeitschritt ein vollständiger Simulationszyklus berechnet werden muss, wobei alle Objekte auf Kollisionen überprüft werden, führen kürzere Zeitschritte auch zu einer deutlich geringeren Simulationsgeschwindigkeit.

Durch eine Erhöhung der Iterationen des Constraintsolvers wird ebenfalls die Genauigkeit erhöht. Hierdurch lassen sich die Constraintfehler besser lösen. Mögliche Constraintverletzungen werden kleiner oder vollständig behoben. Die Erhöhung der Iterationen hat keine so großen Auswirkungen auf die Geschwindigkeit der Simulation, da hierbei außer dem Lösen der Constraints keine Aktionen mehrfach durchgeführt werden müssen.

Es wurden mehrere Testreihen durchgeführt, bei denen jeweils einer dieser Parameter verändert wurde. Standardmäßig sind die Zeitschritte auf 1/60 s und die Anzahl der Iterationen auf 10 eingestellt. Diese Werte sind für die Simulation in Computerspielen ausreichend und ermöglichen dabei eine Berechnung in Echtzeit. Bei der Robotersimulation war jedoch schnell festzustellen, dass mit diesen Werten die Fehler viel zu groß sind. Deshalb wurden die Zeitschritte auf 1/500 s und die Iterationen des Constraintsolvers auf 80 festgelegt.

7.1.1 Benötigte Rechenzeit

Zuerst wurde die benötigte Rechenzeit bestimmt, um eine Sekunde realer Zeit zu simulieren. Die Rechenzeit ist abhängig von dem verwendeten Computersystem. Bei den Tests wurde ein Computer mit Intel Core2Duo 6600 mit 2,4 GHz Taktfrequenz, 4 GB Arbeitsspeicher, Windows 7 64bit und der 64bit Java Virtual Machine 1.6.0-22 von Sun verwendet.

Bei den in Abbildung 7.1 dargestellten Testreihen wurde die Anzahl Iterationen des Constraintsolvers konstant bei 80 gehalten und die Länge der Zeitschritte wurde verändert. Dargestellt ist hierbei jeweils, in welcher Relation der Zeitverlauf der Simulation zum realen Zeitverlauf steht. Ein Wert von 0,5 bedeutet also, dass für die Simulation von einer Sekunde realer Zeit jeweils zwei Sekunden Rechenzeit benötigt werden. Bei der ersten Testreihe wurde die Hand bewegt, aber ohne dass Kollisionen der Finger existieren. Bei der zweiten Testreihe berührten alle fünf Finger die Oberfläche eines Gegenstandes. In Abbildung 7.2 sind diese Kollisionen für alle Finger außer dem Daumen dargestellt. Jeder Finger berührt hier mit 4 Kontaktpunkten den Gegenstand. Der Gegenstand selbst ist in der Darstellung unsichtbar, um die Kollisionen erkennen zu können. In diesem Fall sind wegen der Kontakte aufwändigere Berechnungen notwendig, wodurch die Simulation langsamer abläuft. Im praktischen Einsatz, wenn also nur einige der Finger etwas berühren, wird die Simulationsgeschwindigkeit zwischen diesen beiden Fällen liegen.

Deutlich sichtbar ist hier, dass die Simulation auf dem verwendeten Computersystem nicht in Echtzeit ablaufen kann, wenn die Anzahl an Zeitschritten pro Sekunde über 100 liegt. Zu beachten ist hierbei aber, dass aktuelle Computer leistungsfähiger sind und daher bei diesen auch mehr Zeitschritte pro Sekunde möglich wären.

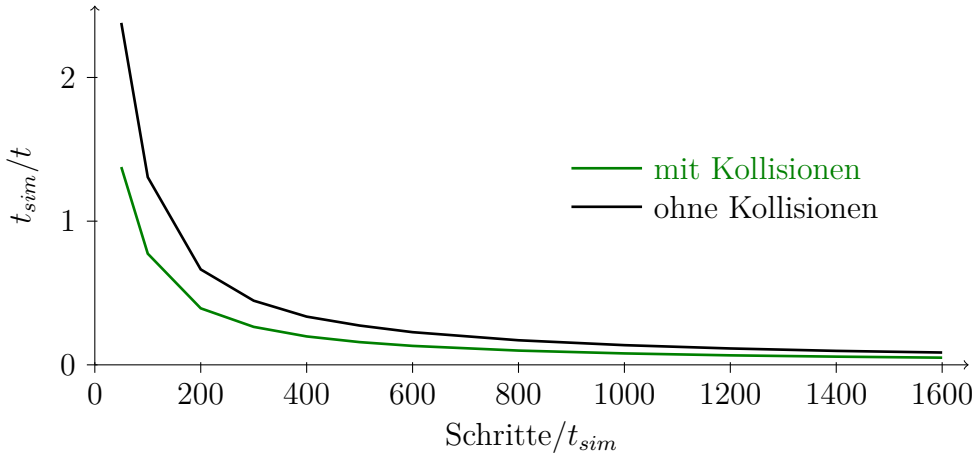


Abbildung 7.1: Geschwindigkeit der Simulation bei unterschiedlichen Zeitschritten

In Abbildung 7.3 sind zwei ähnliche Testreihen dargestellt. In diesem Fall wurde die Anzahl der Zeitschritte pro Sekunde konstant bei 500 gehalten und die Iterationen des Constraintsolvers wurden verändert. Hier ist zu erkennen, dass insbesondere

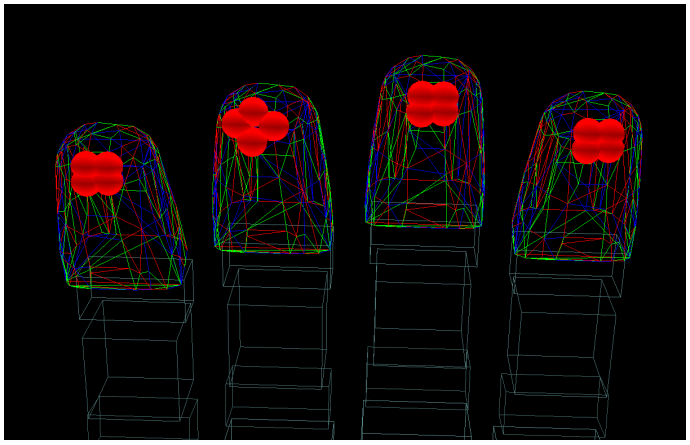


Abbildung 7.2: Darstellung der Kollisionen, die bei der Überprüfung der Rechenzeit vorlagen

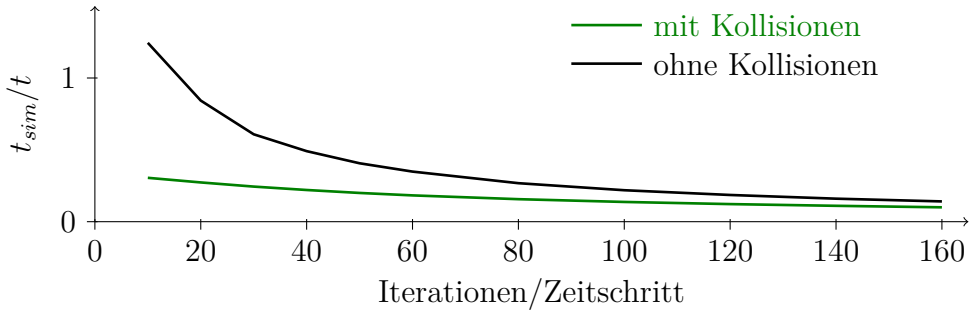


Abbildung 7.3: Geschwindigkeit der Simulation bei einer unterschiedlichen Anzahl Iterationen des Constraintsolvers

bei Kollisionen die Erhöhung der Iterationen nur geringe Auswirkungen hat. Ein großer Anteil der Zeit wird hierbei also nicht für die Behandlung der Constraints verwendet, sondern für die Kollisionserkennung. Da üblicherweise bei der Simulation auch Kollisionen vorliegen, ist es also sinnvoll, die Anzahl der Iterationen zu erhöhen. Dadurch kann die Genauigkeit stark verbessert werden, ohne die Berechnungen zu sehr zu verlangsamen.

7.1.2 Untersuchung der Fingerbewegungen beim Bewegen des Roboterarmes

Bei diesem Experiment wurden alle Gelenke der Hand und des Roboterarmes zuerst auf 0° gesetzt. Nach 500ms wurde das Gelenk 2 des Roboterarmes, mit dem dieser geschwenkt werden kann, Richtung -10° bewegt und nach weiteren 1500ms dann in Richtung 10°. Die Hand wird also zuerst rückwärts und danach vorwärts bewegt. Gemessen wurde dann, wie weit die Winkel der unteren Fingergelenke von ihrem eingestellten Wert abweichen. Gelenk 2 des Armes wurde deshalb verwendet,

weil dadurch der größte Hebelarm besteht und die Kräfte auf die Fingergelenke am größten sind. Bei den Fingergelenken wurde FFJ3 ausgewählt. Dies ist das untere Gelenk, mit dem der Zeigefinger eingeknickt werden kann (siehe Abschnitt 3.1). Die Abweichungen kommen dadurch zustande, dass in in den simulierten Gelenken keine Reibungskräfte existieren, die eine Bewegung verhindern würden. Die Reibungskraft, die mit einem Regler simuliert wird, reduziert die Bewegung der Gelenke zwangsläufig erst einen Zeitschritt später.

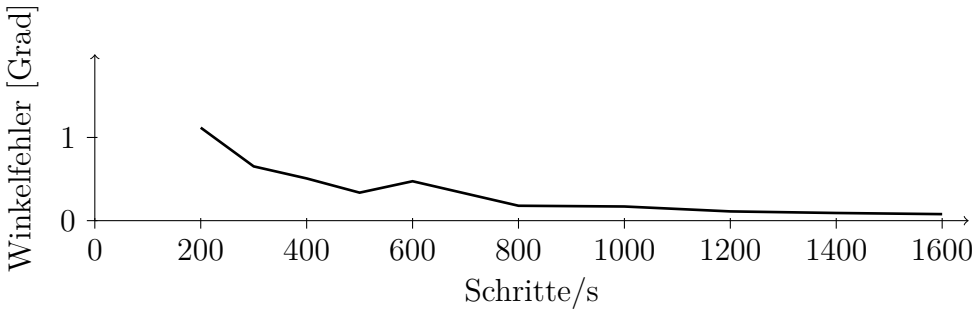


Abbildung 7.4: Winkelfehler eines Fingergelenks beim Bewegen des Armes mit unterschiedlichen Zeitschritten

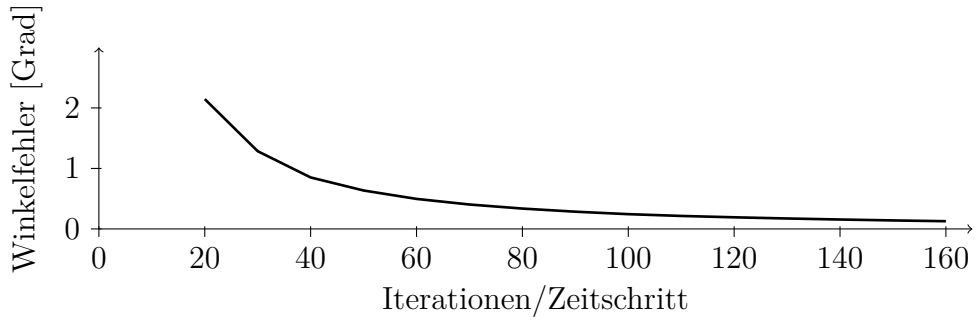


Abbildung 7.5: Winkelfehler eines Fingergelenks beim Bewegen des Armes mit einer unterschiedlichen Anzahl Iterationen des Constraintsolvers

In Abbildung 7.4 wird der maximale Fehler der Gelenkwinkel bei unterschiedlichen Zeitschritten und immer bei 80 Iterationen des Constraintsolvers dargestellt. Abbildung 7.5 zeigt diesen Fehler bei unterschiedlich vielen Iterationen und festen Zeitschritten von 1/500 s. Die ersten Messwerte wurden hierbei weggelassen, da ansonsten die restlichen Werte nicht mehr erkennbar wären. Bei 100 Zeitschritten pro Sekunde lag der Fehler bei 4.17 Grad. Deutlich sichtbar ist, dass der Fehler mit beiden Varianten gut reduzierbar ist. Der verbleibende Fehler von 0,2° ist akzeptabel, da die Winkelgeber der realen Hand eine geringe Auflösung und Wiederholbarkeit aufweisen.

Auf den ersten Blick überraschend ist dabei der Anstieg des Fehlers bei 600 Zeitschritten pro Sekunde. In 7.6 ist der zugehörige Verlauf des Fehlers über die Zeit bei Zeitschritten von 1/500 s und 1/600 s dargestellt. Die erste Stelle, an der ein größerer Fehler auftritt, ist beim Start der Rückwärtsbewegung der Finger. Beim anschließenden Bremsen und der darauf folgenden Vorwärtsbewegung tritt kein großer Fehler auf. Dies liegt daran, dass hier die Grenze des Gelenkes erreicht wurde und die Fehler also durch den Constraintsolver korrigiert wurde. Erst wenn die Vorwärtsbewegung abgebremst wird, kommt es wiederum zu einem größeren Fehler. Hierbei ist nun sichtbar, dass der Fehler bei Zeitschritten von 1/600 s nur kurzzeitig größer ist, als bei den kürzeren Zeitschritten.

Die genaue Ursache dieses Verhaltens konnte nicht festgestellt werden, vermutlich liegt es aber daran, dass die Simulation mit Zeitschritten arbeitet und nicht kontinuierlich ist. Dadurch kommt es sehr stark darauf an, wann genau eine Bewegung gestartet wird. Die kontinuierliche kinematische Berechnung der Armbewegung kann dazu führen, dass der Arm zu einem beliebigen Zeitpunkt zwischen zwei Zeitschritten gestartet wird. Wenn nun die Armbewegung genau nach einem Zeitschritt gestartet wird, so ist bis zum darauf folgenden Schritt schon eine weite Bewegung erfolgt, auf die noch keine Reaktion der Regler folgen konnte. In diesem Fall wird der auftretende Fehler also größer. Durch eine leichte Verschiebung der Startzeit für die Armbewegung änderte sich das Verhalten, so dass der deutliche Ausschlag nicht mehr auftrat. Dies bestätigt die oben gemachte Vermutung.

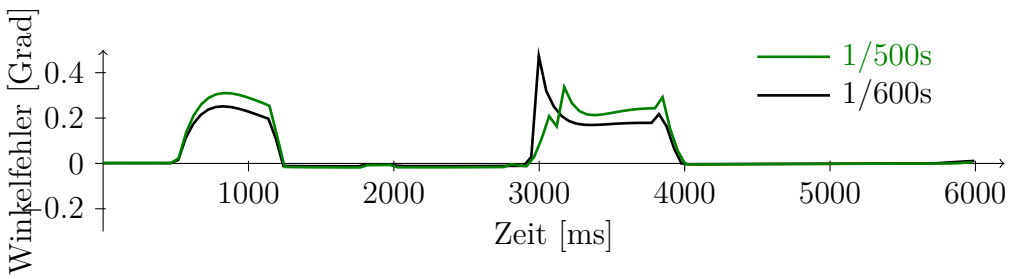


Abbildung 7.6: Winkelfehler eines Fingergelenks beim Bewegen des Armes bei 500 und 600 Zeitschritten pro Sekunde und 80 Iterationen pro Zeitschritt

7.1.3 Untersuchung der Stabilität eines Griffes

Sehr wichtig ist auch die Stabilität der möglichen Griffe mit dem Simulator. Es kann hier vorkommen, dass sich trotz ausreichender Reibungskräfte die Gegenstände langsam bewegen. Dies liegt insbesondere an der Simulation mit Zeitschritten. Die genaue Ursache dafür wurde in Kapitel 6 bei der Implementation der taktilen Sensoren erklärt. Auch bei diesem Experiment wurden wieder die Länge der Zeitschritte und die Anzahl an Iterationen des Constraintsolvers verändert. In einer Messreihe wurde ein Gegenstand so gehalten, dass dieser nach unten aus den Fingern rutschen

könnte. Der gewählte Griff ist in Abbildung 7.7 dargestellt. Es handelt sich hier um einen Precision Grasp mit vier Fingern. Es wurde dann festgestellt, wie weit sich dieser Gegenstand in 6 Sekunden nach unten bewegt. Bei einer zweiten Messreihe wurde zusätzlich der Arm innerhalb der 6 Sekunden bewegt, so dass durch die zusätzlichen Beschleunigungskräfte eine Bewegung wahrscheinlicher wurde.

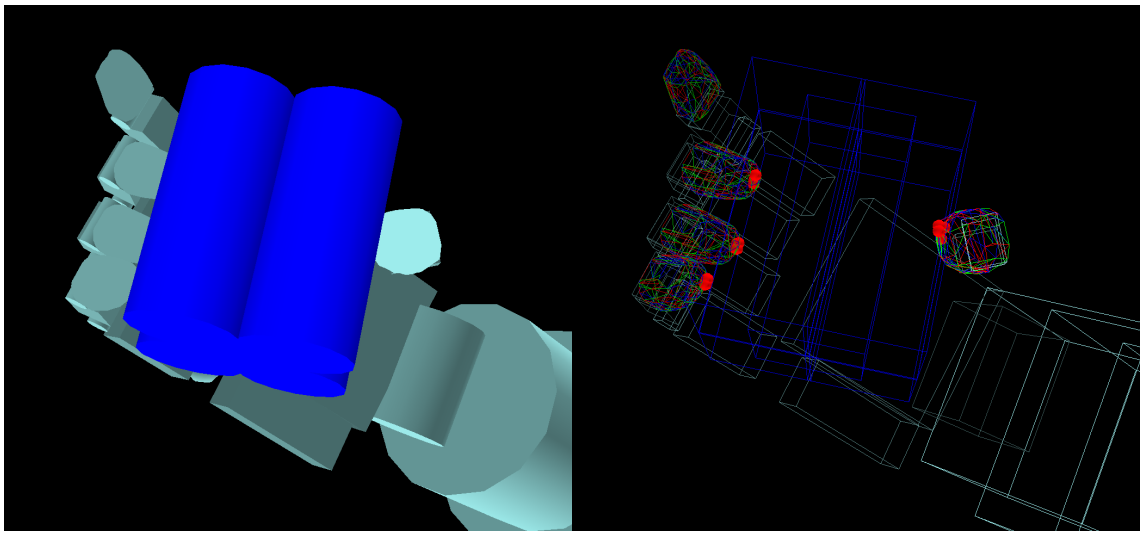


Abbildung 7.7: Gewählter Griff, bei dem die Bewegung des Gegenstandes getestet wurde. Normale Darstellung und als Wireframe

In Abbildung 7.8 sind die Testreihen bei einer unterschiedlichen Anzahl von Zeitschritten dargestellt. Bei unter 200 Zeitschritten pro Sekunde fiel der Gegenstand aus der Hand, weshalb dies hier nicht dargestellt ist. Deutlich sichtbar ist, dass der Fehler zuerst deutlich geringer wird, sich aber bei noch kürzeren Zeitschritten kaum etwas ändert. Bei Bewegungen des Armes bleibt der Fehler länger höher, da die Beschleunigungen hier größer sind. Sinnvoll erscheint anhand der Ergebnisse eine Wahl der Zeitschritte zwischen 500 und 1000 Schritten pro Sekunde, um einen brauchbaren Kompromiss zwischen Simulationsgeschwindigkeit und Stabilität zu erreichen. Über 1000 Schritte pro Sekunde sind bei den vorhandenen Beschleunigungen nicht notwendig.

In Abbildung 7.9 sind die Testreihen bei konstanten 500 Zeitschritten pro Sekunde und einer unterschiedlichen Anzahl Iterationen des Constraintsolvers dargestellt. Hier sieht man, dass eine Erhöhung der Iterationen für die Stabilität des Griffes keine Vorteile bringt. Bei einer Bewegung des Armes nimmt die Stabilität sogar ab, wenn man die Iterationen erhöht. Die Ursachen für die Verschlechterung konnten nicht geklärt werden.

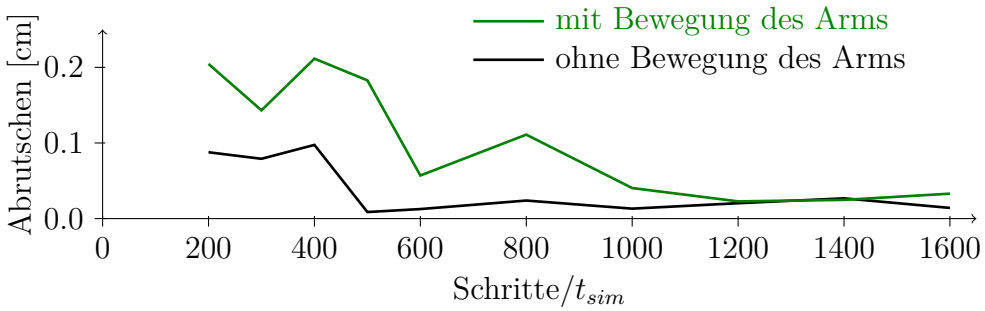


Abbildung 7.8: Herunterrutschen eines Gegenstandes bei unterschiedlichen Zeitschritten

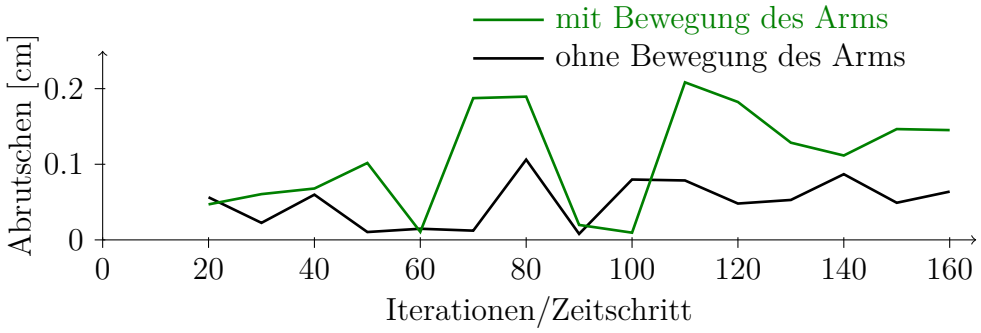


Abbildung 7.9: Herunterrutschen eines Gegenstandes bei einer unterschiedlichen Anzahl Iterationen des Constraintsolvers

7.2 Bewegung der Gelenke der Hand

Bei diesem Experiment sollte die Bewegung der Gelenke der Shadow Hand untersucht werden. Dies ist wichtig, weil das Verhalten der pneumatischen Muskeln sehr komplex ist. Die Messungen werden anschließend mit Messungen der simulierten Gelenke verglichen. Zur Aufzeichnung der Daten wurde ein Steuerungsprogramm geschrieben, welches ein Gelenk zuerst an die jeweilige Startposition fährt. Danach wird einige Sekunden gewartet, bis keine Bewegung mehr stattfindet. Dann wird der Zielwinkel eingestellt und die Daten werden für 4 Sekunden aufgezeichnet. Als Start und Ziel wurden nacheinander alle erreichbaren Gelenkwinkel in Schritten von 5 Grad eingestellt.

In den Abbildungen 7.10 und 7.11 sind zwei der aufgezeichneten Bewegungskurven dargestellt. Hier wurde jeweils das Gelenk 2 des Mittelfingers angesteuert. Da das Gelenk 1 mit diesem verbunden ist, wird es ebenso bewegt. In Abbildung 7.10 findet eine weite Bewegung des Fingers von 10 Grad bis 75 Grad statt, während im anderen Fall nur eine Bewegung um 5 Grad stattfindet.

Deutlich ist hier zu sehen, dass bei einem größeren Unterschied zwischen dem Startwinkel und dem Zielwinkel die Bewegung nach viel kürzerer Zeit beginnt. Dies

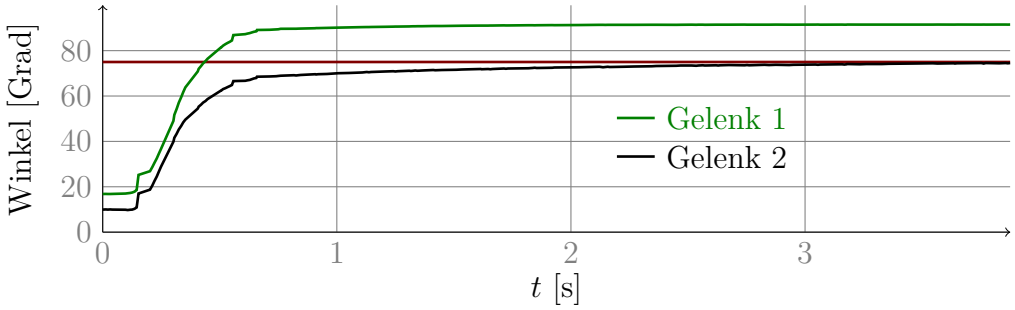


Abbildung 7.10: Bewegungskurve der äußeren beiden Gelenke des Mittelfingers bei einer Bewegung von 10 Grad bis 75 Grad

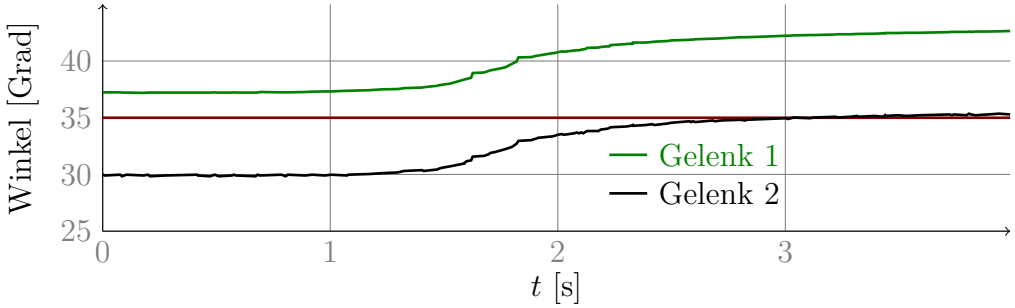


Abbildung 7.11: Bewegungskurve der äußeren beiden Gelenke des Mittelfingers bei einer Bewegung von 30 Grad bis 35 Grad

ist insofern nicht erstaunlich, da in diesem Fall die Regler stärker reagieren müssen. Allerdings ist die Bewegung im ersten Fall schon beinahe beim Zielwinkel angekommen, bevor im zweiten Fall überhaupt eine Bewegung beginnt. In beiden Fällen dauert die Bewegung etwa 2s bis zu einer Abweichung von unter zwei Grad. Dies ist wesentlich länger als in der Spezifikation der Hand [Sha08] dargestellt. Dort werden 0,2s angegeben.

In Abbildung 7.12 ist der Druck in den beiden Muskeln bei einer Bewegung von 30 Grad bis 35 Grad dargestellt. Hier sieht man, dass der Druck bereits vor dem Beginn der Bewegung aufgebaut wird. Während der Bewegung ändert sich der Druck in den Muskeln hingegen kaum noch. Die vermutliche Ursache für die deutliche Verzögerung bei den geringen Bewegungen ist daher die starke Reibung in den Gelenken. Dadurch muss erstmal eine große Kraft aufgebaut werden, bis die Haftreibung überwunden wird und die Bewegung beginnen kann. Möglicherweise kann hier eine Korrektur der Reglereinstellungen helfen, um dieses Verhalten zu verbessern.

Außerdem ist in den Bewegungskurven zu sehen, dass das Gelenk 1 immer etwas weiter geöffnet ist, als das Gelenk 2. Dieses Gelenk wird nicht geregelt, sondern wird bei einer Bewegung von Gelenk 2 mitbewegt. Daher ist ein abweichender Winkel nicht ungewöhnlich. Für eine genaue Simulation sollte dies aber auch berücksichtigt werden.

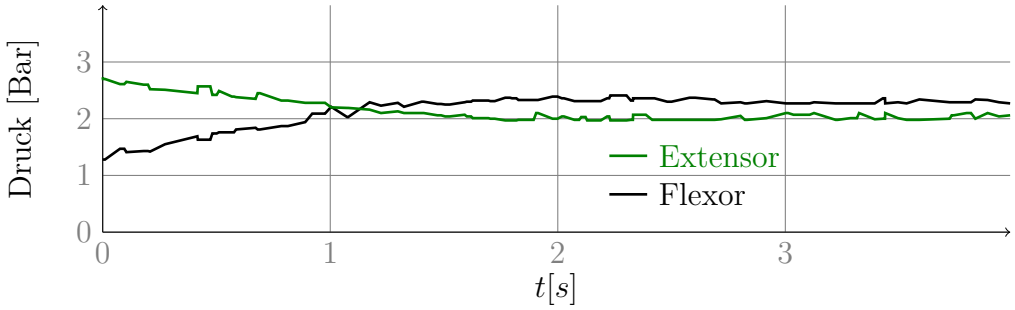


Abbildung 7.12: Druck in den Muskeln des Gelenks 2 des Mittelfingers bei einer Bewegung von 30 Grad bis 35 Grad

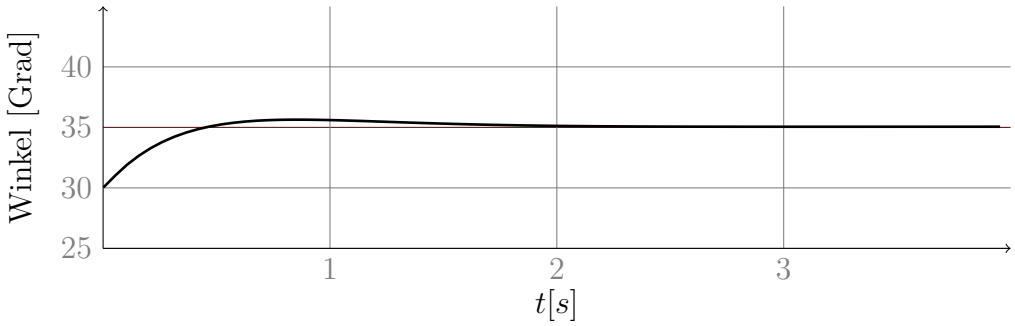


Abbildung 7.13: Bewegung des Fingers im Simulator von 30 Grad bis 35 Grad

In Abbildung 7.13 ist das Verhalten des Fingergelenks 2 des Mittelfingers im Simulator dargestellt. Auch hier wird der Winkel von 30 Grad auf 35 Grad erhöht. Gut sichtbar ist hier die deutlich schnellere Reaktion. Nach dem Erreichen des Zielwinkels kommt es zu einer leichten Überschwingung. Diese wird von der Gravitation hervorgerufen, die als Störgröße wirkt. Um das Verhalten der Shadow Hand nachzuahmen, sind zwei Änderungen am derzeitigen Regler notwendig. Es müsste eine Totzeit eingefügt werden, die abhängig von der gewünschten Winkeländerung ist und es müsste eine Kompensation des von der Gravitation ausgelösten Fehlers durchgeführt werden.

7.3 Bewegen von Gegenständen mit dem Simulator

Um die Funktionsfähigkeit des Simulators zu zeigen, wurden einige Experimente durchgeführt, bei denen Gegenstände mit dem simulierten Roboter bewegt wurden. Für alle diese Experimente wurde die Länge der Zeitschritte auf 1/500 s und die Anzahl an Iterationen des Constraintsolvers auf 80 festgelegt. Da dann nur etwa 100 bis 200 Zeitschritte pro Sekunde berechnet werden können, erfolgte diese Simulation nicht in Echtzeit, sondern mit etwa 1/3 der realen Geschwindigkeit. Diese Experimente zeigen die Funktionsfähigkeit des Simulators in realistischen Einsatz-

szenarien.

Verwendet wurden hier insbesondere Szenarien mit Mehrfingergriffen und Manipulationen, bei denen die Gegenstände in der Hand bewegt werden, da diese in anderen Simulatoren nicht darstellbar sind.

7.3.1 Greifen und Anheben der Gegenstände

Als erstes sollte versucht werden, vier Gegenstände, die in der Simulation vorhanden sind, zu greifen und jeweils ein Stück anzuheben. Zum Greifen der Gegenstände musste zuerst die Roboterhand in eine geeignete Position bewegt werden. Hierzu wurde eine Position ausgewählt, die auch ein Mensch beim Greifen verwenden würde. Danach wurden unter Verwendung des Steuerungsinterfaces die Finger langsam in eine Position gebracht, in der der Gegenstand gehalten werden konnte. Daraufhin wurde dann die Hand leicht angehoben und einige Zeit in diesem Zustand belassen um die Stabilität zu prüfen.

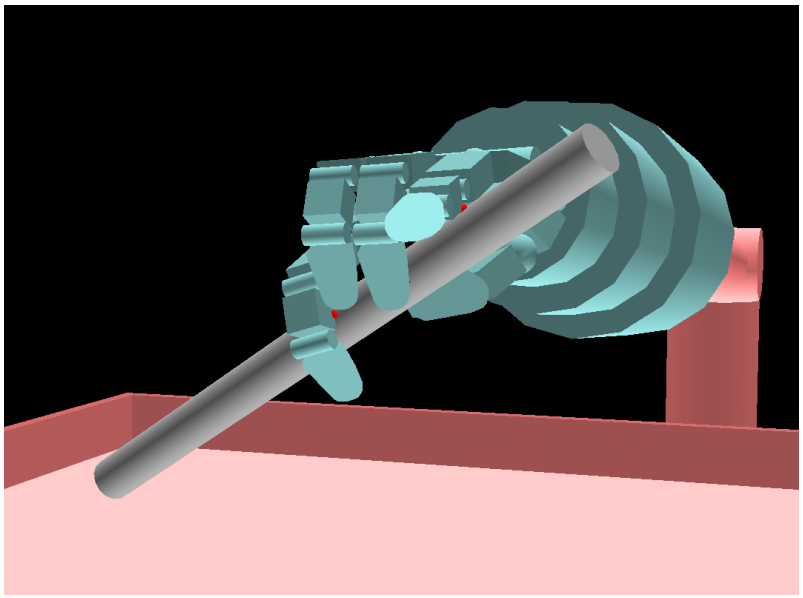


Abbildung 7.14: Stabiler Mehrfinger-Griff bei dem Zylinder

Alle Gegenstände bis auf den langen Zylinder konnten hierbei leicht stabil gehalten werden. Die Finger der Hand und auch die Objekte zitterten aber die ganze Zeit leicht. Dieses Verhalten war auch schon bei den vorhergehenden Messungen beobachtet worden und liegt an dem Simulationsverfahren. Wie sich bei den Messungen zeigte, ist dieser Fehler dabei sehr stark von der Länge der Zeitschritte abhängig. Durch das leichte Zittern rutschen einige Gegenstände langsam aus der Hand. Bei den meisten Gegenständen war diese Bewegung aber so langsam, dass die Gegenstände erst nach mehreren Minuten aus der Hand fallen würden. Wenn bei dem

langen Zylinder allerdings nur zwei oder drei Finger verwendet wurden, um diesen zu halten, war der Griff nicht stabil genug. Dadurch rutschte der Zylinder schon nach einigen Sekunden aus der Hand. Dies lag an den wenigen möglichen Kontaktpunkten, die durch die Form des Zylinders begründet sind. Wenn man aber einen Finger so bewegte, dass dieser unter den Zylinder griff, war es möglich, auch diesen stabil zu halten. Der erreichte stabile Griff ist in Abbildung 7.14 dargestellt. Hier sieht man auch die Kontakte der inneren Fingerglieder, die durch die gewählte Simulation problemlos möglich sind.

7.3.2 Stapeln der Gegenstände

Bei diesem Experiment sollte der Würfel auf den kleineren, aus Zylindern bestehenden Block gestellt werden. Hierbei sind verschiedene Teile der Simulation von Bedeutung. Es muss möglich sein, den Gegenstand stabil bei Bewegungen in der Hand zu halten, ohne dass dieser aus der Hand rutscht. Zusätzlich kann festgestellt werden, ob eine genaue Steuerung der Gelenke beim Absetzen des Gegenstandes möglich ist. Abbildung 7.15 zeigt einige Screenshots dieses Experimentes.

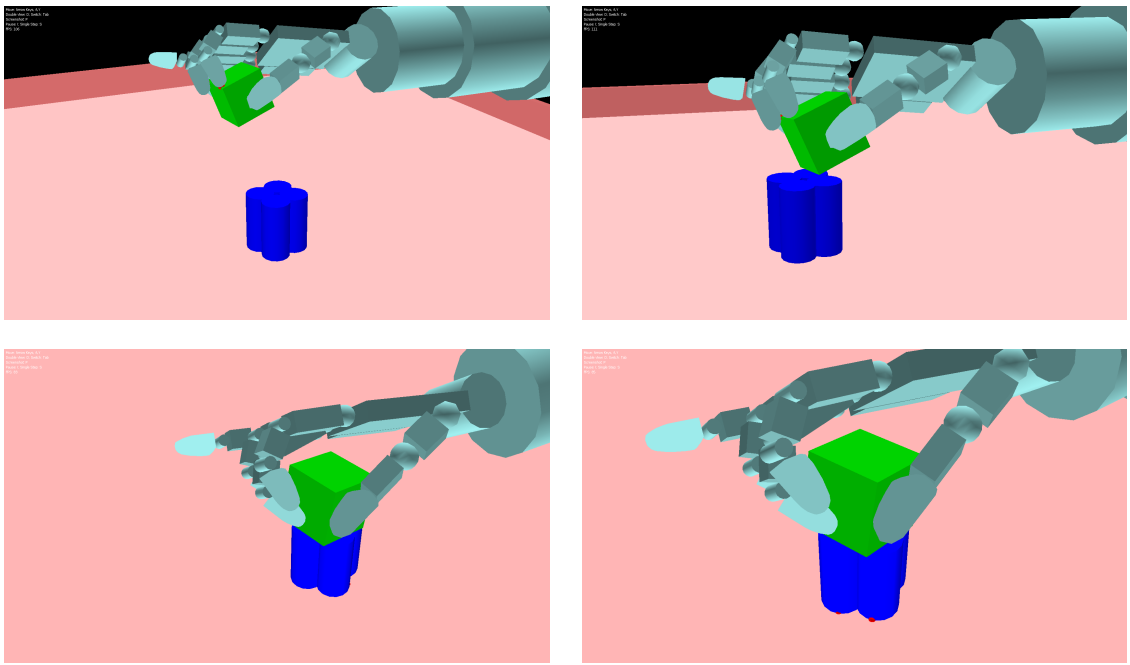


Abbildung 7.15: Stapeln zweier Gegenstände

7.3.3 Drehen eines Gegenstandes in der Hand

Hier sollte ein Gegenstand in den Fingern bewegt werden. Dieses Experiment wurde zuvor mit dem realen Roboter durchgeführt, weshalb es auch für die Simulation interessant war.

In Abbildung 7.16 sind einige Schritte der Bewegung dargestellt. Der Gegenstand wird zuerst von Mittelfinger und Daumen gehalten und befindet sich nahe der Handfläche. Dann wird durch Drücken mit dem Zeigefinger von der einen Seite der Gegenstand ein wenig gedreht und dann durch den Ringfinger weiter angehoben. Dadurch entsteht ein stabiler Griff, bei dem der Gegenstand gedreht und weiter von der Hand entfernt ist als vorher. Als Mensch führt man teilweise ähnliche Bewegungen wie diese durch, um einen Gegenstand in der Hand neu zu positionieren.

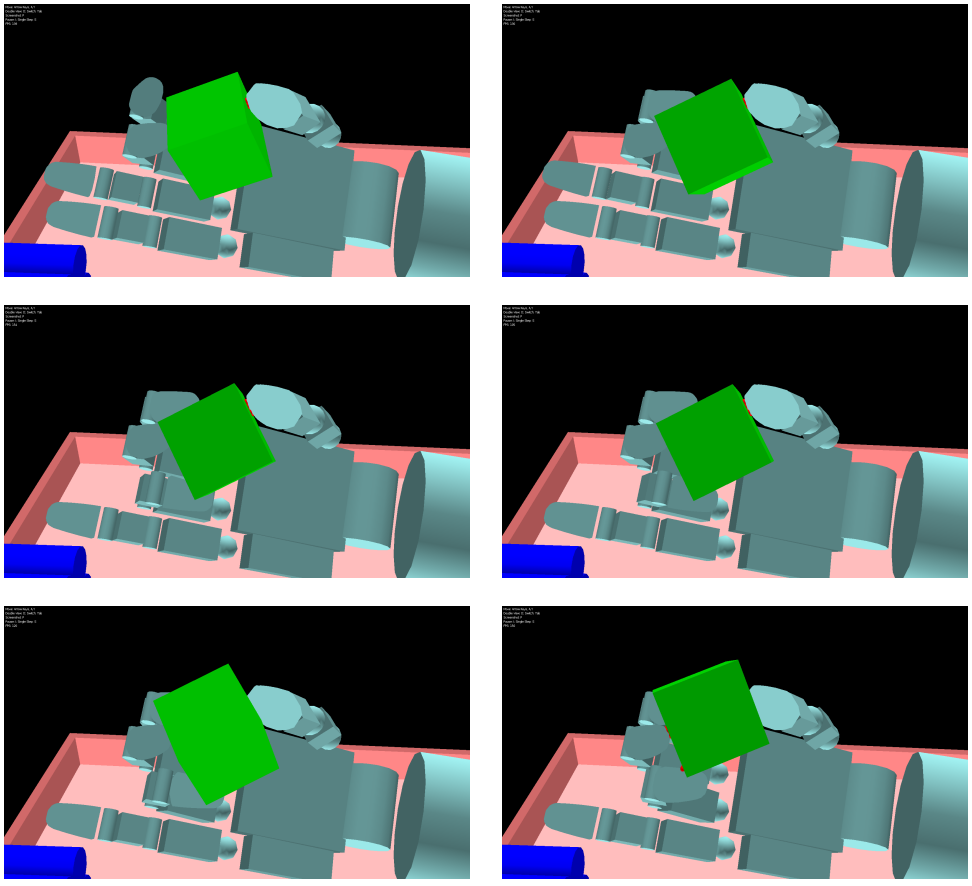


Abbildung 7.16: Drehung eines Gegenstandes in der Hand

Die Bewegung der Finger bei dieser Manipulation ist in Abbildung 7.17 dargestellt. Dort sieht man den Verlauf der Winkel der Gelenke 1,2 und 3 des Zeigefingers (rot), des Mittelfingers (grün) und des Ringfingers (blau). Die hellere Kurve zeigt jeweils das Gelenk 3 und die dunkleren Kurven die Gelenke 1 und 2. Die mit a bis f bezeichneten Zeitpunkte sind in den Screenshots abgebildet.

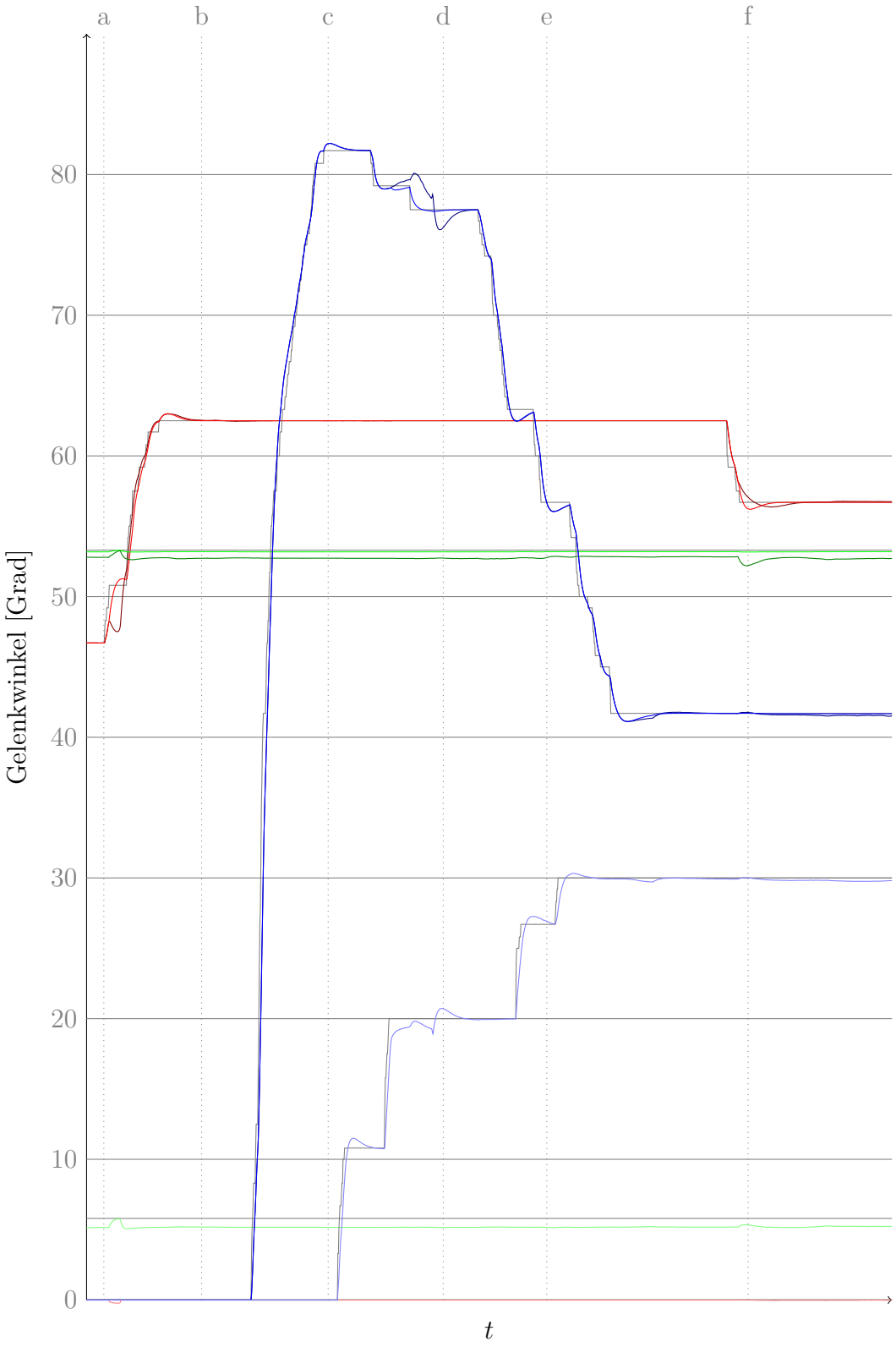


Abbildung 7.17: Verlauf der Gelenkwinkel bei der Drehung eines Gegenstandes

7.3.4 Bewegen des Würfels durch eine Öffnung

Als letztes Experiment wurde noch versucht, den Würfel durch eine senkrecht stehende Platte mit einem quadratischen Loch von 5 cm Kantenlänge hindurch zu befördern. Hierfür war es zuerst notwendig, den Würfel in der Hand zu bewegen, damit dieser die richtige Ausrichtung erhält. Bei dem Bewegen des Würfels durch das Loch, war es zusätzlich notwendig, einen nicht stabilen Griff zu verwenden. Screenshots hierzu sind in Abbildung 7.18 zu sehen.

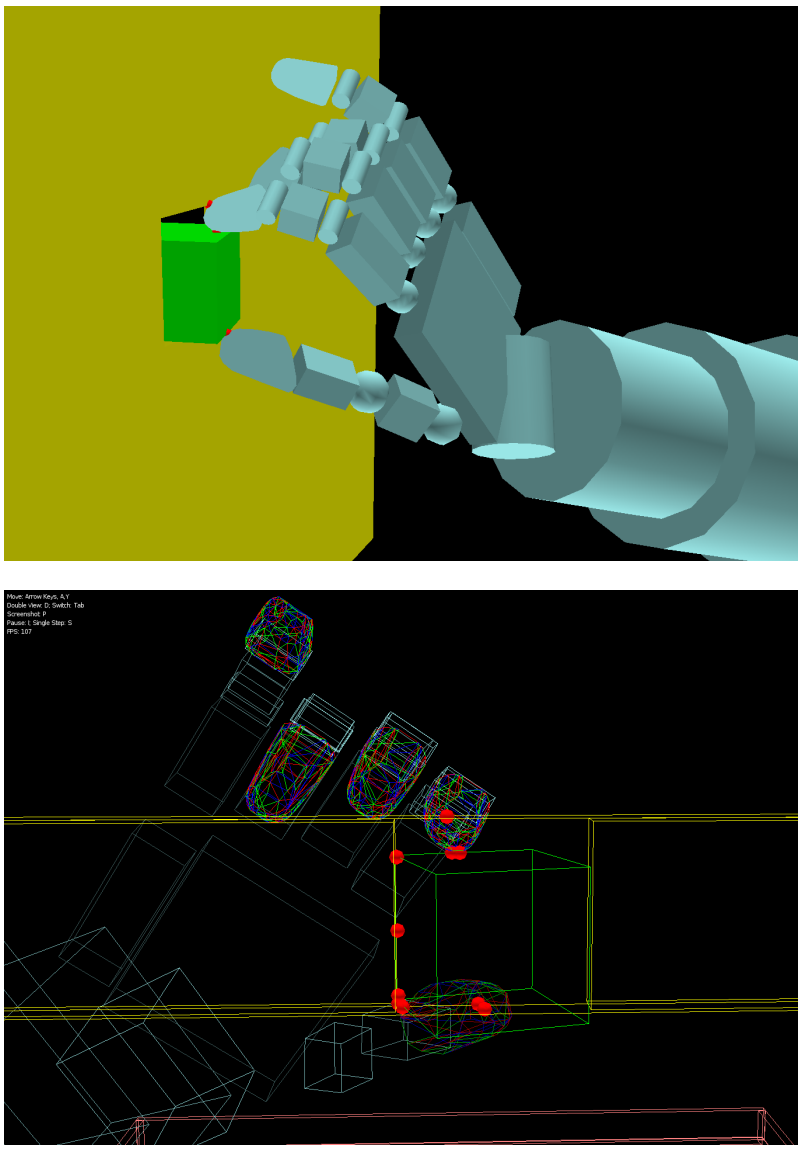


Abbildung 7.18: Bewegen des Würfels durch eine Öffnung

7.3.5 Zusammenfassung der Experimente

Alle hier aufgelisteten Experimente waren mit dem Simulator durchführbar. Bei einigen Experimenten waren allerdings mehrere Anläufe notwendig, bis das Ziel erreicht wurde. Die Ursache dafür war größtenteils, dass die Steuerung über die einzelnen Gelenkwinkel zu kompliziert war. Hierbei wäre es sinnvoll, inverse Kinematik einzusetzen. Damit wäre dann eine Steuerung mit kartesischen Koordinaten möglich.

Insgesamt zeigte sich, dass die Simulation sehr stabil und vorhersagbar funktioniert. Die Gegenstände in der simulierten Hand verhalten sich etwa so, wie man es erwarten würde. Ein merkbarer Unterschied ist allerdings, dass sich Gegenstände, während sie gehalten werden, manchmal leicht drehen. Dieses Verhalten wurde aber erwartet, da durch die Punktkontakte der starren Körper das Reibungsmoment nicht richtig simuliert werden kann.

8 Fazit

Diese Arbeit dokumentiert die Entwicklung eines Simulators, der auf die Simulation von Mehrfinger-Roboterhänden mit vielen Freiheitsgraden spezialisiert ist. Für diese Roboterhände kann eine realitätsnähere Simulation geboten werden, als es mit ähnlichen Robotersimulatoren der Fall wäre. Möglich wurde dies durch die Modellierung der ganzen Hand als dynamisches System. Dadurch können auch Kräfte zurück von einem Gegenstand an den Finger übertragen werden. Ohne diese Rückübertragung der Kräfte könnten Finger mit mehreren Gelenken nicht gut simuliert werden. Insbesondere bei der Verwendung von pneumatischen Muskeln, die eine hohe Elastizität an den Fingergelenken bieten, haben die rückübertragenen Kräfte deutliche Auswirkungen auf die Bewegungen der Finger.

Für die Modellierung war es notwendig, auf einer gut entwickelten Physiksimulation aufzubauen. JBullet bietet hierbei eine überzeugende Funktionalität. Insbesondere die Unterstützung von mehrfachen Objektkontakten und die performante Berechnung der Kollision von konkaven Objekten waren hier interessant. Aber auch die Stabilität bei der Berechnung von Kollisionen und den resultierenden Kräften konnte überzeugen. Es gibt allerdings auch viele Möglichkeiten, die Genauigkeit der Simulation noch weiter zu erhöhen. Hierzu werden im nächsten Kapitel einige Vorschläge gemacht.

Zusätzlich bietet der Simulator eine gute Performance, die auch einen Echtzeiteinsatz auf leistungsfähigen Computern nicht ausschließt. Dadurch kann der Simulator in Verbindung mit dem realen Roboter verwendet werden. Man kann Vorhersagen über die Auswirkungen möglicher Aktionen treffen, indem man diese vorher in der Simulation ausprobiert. Danach wählt man die am meisten erfolgversprechende Variante aus und führt diese mit dem Roboter durch.

Mit den durchgeführten Experimenten konnten die zu erwartenden Fehler bei verschiedenen Einstellungen des Simulators überprüft werden. Dadurch kann man bei einer Verwendung des Simulators entscheiden, welche Einstellungen der Parameter für einen speziellen Einsatzzweck sinnvoll sind. Man sieht dann auch welche Berechnungsgeschwindigkeit man durch die gewählten Parameter zu erwarten hat.

Durch den gewählten Aufbau des Simulators sind auch Modifikationen an dem simulierten Roboter einfach möglich. So kann der Einsatz anderer Sensoren, die Auswirkung einer anderen Platzierung des Roboters oder auch der Austausch des Roboterarmes ohne viel Aufwand modelliert und getestet werden. Auch können sehr schnell neue Gegenstände zur Simulation hinzugefügt werden.

9 Ausblick

Aufbauend auf dieser Diplomarbeit eröffnet sich eine Reihe von interessanten Themen. Diese lassen sich grob in drei Kategorien einteilen. Zum einen wäre dies eine Verbesserung der Funktionalität der Physikbibliothek selbst, insbesondere im Bereich der Genauigkeit und der Geschwindigkeit, aber auch um bisher nicht modellierbare Objekte zu unterstützen. Als weiteres könnte man den Simulator noch genauer an die Realität anpassen, um die Qualität der Vorhersagen, die durch die Simulation gemacht werden zu erhöhen. Und dann gibt es noch die Möglichkeit, auf einer höheren Abstraktionsebene mithilfe der Simulation Lernverfahren und Planungsverfahren zu implementieren und zu testen.

Zur Verbesserung der Physikbibliothek gibt es einige interessante Ansätze, für die teilweise schon experimentelle Umsetzungen existieren:

- Die Modellierung von weichen Oberflächen würde eine realistischere Simulation des Reibungsmoments ermöglichen. Ein Verfahren dazu ist im Artikel [CLA07] vorgestellt. Hier werden anhand der Geometrie um die Kontaktpunkte weitere Kontaktpunkte erzeugt. Die Normalkraft des ursprünglichen Kontaktpunktes wird dann auf die neuen Kontakte verteilt. Durch die Vielzahl von Kontaktpunkten kann dann das Reibungsmoment mit geringen Fehlern simuliert werden.

- In [NHK+07] und [YN08] wurden Constraintsolver entwickelt, die eine höhere Stabilität und eine schnellere Konvergenz bieten sollen, als der bisher verwendete LCP-Constraintsolver. Es wäre denkbar, die Möglichkeiten dieser Solver im Rahmen der Simulation zu überprüfen.

- In vielen Physiksimulationen für Computerspiele werden derzeit Ansätze für verformbare Körper ausprobiert. Bei diesen Körpern sind die einzelnen Polygon-Eckpunkte nicht fest an einer Stelle, sondern sie lassen sich durch Kräfte auch verschieben, werden aber durch ihre benachbarten Punkte in einer stabilen Lage gehalten [TPBF87]. Man könnte diesen Ansatz zum einen für eine leichte Verformbarkeit der Handoberfläche selbst verwenden, um die Kontaktflächen zu vergrößern, oder auch um das Greifen von weichen Objekten zu simulieren. Interessant wird hierbei insbesondere sein, ob mit einem derartigen Verfahren auch weiche Gegenstände wie Schwämme oder Stoffe realitätsnah simuliert werden können, und wo es möglicherweise größere Abweichungen gibt. Allerdings wird das Verwenden von derartigen Objekten vermutlich die Geschwindigkeit der Simulation weiter verringern.

- Derzeit existiert bei der Bullet-Physikbibliothek nur ein einziger Wert für die Reibung, der eine Kombination aus Haftreibung und Gleitreibung darstellt. Eine mögliche Erweiterung wäre also, diesen Wert aufzuteilen und vielleicht auch das derzeit erzwungene proportionale Verhältnis zwischen Normalkraft und Reibung aufzuheben. In der Realität ist dieses Verhältnis insbesondere bei der Verwendung von Kunststoffen und anderen organischen Materialien oftmals nicht proportional. In Gelenken ist derzeit noch überhaupt keine Simulation von Reibung vorgesehen. Auch hier wäre also eine Erweiterung denkbar.

Zur Anpassung des Simulators gibt es auch einige Möglichkeiten, die insbesondere aus Zeitgründen nicht mehr betrachtet wurden:

- Bisher ist es nur möglich, lineare Regler einzusetzen. Insbesondere für die Simulation der durch die pneumatischen Muskeln entstehenden Kräfte wäre vermutlich aber die Verwendung eines nichtlinearen Reglers sinnvoll. Diesen müsste man dann unter Verwendung der Messdaten des Roboters auf das reale Verhalten einstellen. In [SEKD03] und [TL00] werden Regler für die Simulation von pneumatischen Muskeln vorgestellt.

- Bei der Verwendung eines stabileren Constraintsolvers wäre es möglicherweise sinnvoll, die Bewegungen des Roboterarmes auch durch die Physikbibliothek selbst zu berechnen und auf eine kinematische Kontrolle zu verzichten. Dadurch würde sich insbesondere bei Kollisionen des Armes mit der Umgebung die Qualität der Simulation erhöhen

- Das Handmodell entspricht zwar von den Gelenken und den Positionen der einzelnen Handglieder der realen Shadow Hand, es werden aber derzeit stark vereinfachte Formen für die einzelnen Fingerglieder und die Handoberfläche eingesetzt, um die Simulationsgeschwindigkeit zu erhöhen. Hier könnte man ermitteln, in welchen Bereichen der Hand noch genauere Modelle eingesetzt werden sollten.

An Lernverfahren wäre insbesondere Bestärkendes Lernen (Reinforcement Learning) interessant, da man hierfür durch die Simulation vergleichsweise einfach Belohnungen für Aktionen bestimmen kann. In der realen Welt ist das Bewerten von Aktionen oft schwierig, da dort über den Zustand der Welt nur wenig bekannt ist. Außerdem führen Lernverfahren oftmals zu unvorhergesehenen Aktionen, die in der Realität den Roboter beschädigen könnten. Möglich wäre es hier auch, ein Lernen in der Simulation zu beginnen und sobald es zu einigermaßen stabilen Ergebnissen führt, diese dann in der Realität zu verfeinern.

Literaturverzeichnis

[ASMC06] Ansuini, C. ; Santello, M. ; Massaccesi, S. ; Castiello, U.: Effects of end-goal on hand shaping. In: *Journal of neurophysiology* 95 (2006), Nr. 4, S. 2456. – ISSN 0022–3077

[BB07] Boeing, Adrian ; Bräunl, Thomas: Evaluation of real-time physics simulation systems. In: *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. New York, NY, USA : ACM, 2007 (GRAPHITE '07). – ISBN 978–1–59593–912–8, 281–288

[BGZ02] Bungartz, H.J. ; Griebel, M. ; Zenger, C.: *Einführung in die Computergraphik: Grundlagen, Geometrische Modellierung, Algorithmen*. Vieweg Teubner Verlag, 2002. – ISBN 3528167696

[Bic02] Bicchi, A.: Hands for dexterous manipulation and robust grasping: A difficult road toward simplicity. In: *Robotics and Automation, IEEE Transactions on* 16 (2002), Nr. 6, S. 652–662. – ISSN 1042–296X

[BL08] Baier-Löwenstein, T.: *Lernen der Handhabung von Alltagsgegenständen im Kontext eines Service-Roboters*, Universität Hamburg, Diss., 2008

[BLMV08] Biagiotti, L. ; Lotti, F. ; Melchiorri, C. ; Vassura, G.: How Far Is the Human Hand? A Review on Anthropomorphic Robotic End-effectors. (2008)

[BPSM+08] Bray, Tim ; Paoli, Jean ; Sperberg-McQueen, C. M. ; Maler, Eve ; Yergeau, François: *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. World Wide Web Consortium, Recommendation. http://www.w3.org/TR/REC-xml/. Version: November 2008

[BUL06] *Bullet Physics Library*. http://bulletphysics.org/. Version: 2006

[CA09] Ciocarlie, M.T. ; Allen, P.K.: Hand posture subspaces for dexterous robotic grasping. In: *The International Journal of Robotics Research* 28 (2009), Nr. 7, S. 851. – ISSN 0278–3649

[Cat05] Catto, Erin: Iterative Dynamics with Temporal Coherence. In: *Game Developers Conference*, 2005 (GDC 2005)

[Cel91] Cellier, F.E.: *Continuous system modeling*. Springer, 1991. – ISBN 0387975020

[CLA07] Ciocarlie, M. ; Lackner, C. ; Allen, P.: Soft finger model with adaptive contact geometry for grasping and manipulation tasks. In: *EuroHaptics Conference, 2007 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems. World Haptics 2007. Second Joint* IEEE, 2007. – ISBN 0769527388, S. 219–224

[Cou10] Coumans, Erwin: *Bullet 2.76 Physics SDK Manual*. http://www.bulletphysics.com, 2010. http://www.bulletphysics.com

[Cut02] Cutkosky, M.R.: On grasp choice, grasp models, and the design of hands for manufacturing tasks. In: *Robotics and Automation, IEEE Transactions on* 5 (2002), Nr. 3, S. 269–279. – ISSN 1042–296X

[Dia10] Diankov, Rosen: *Automated Construction of Robotic Manipulation Programs*, Carnegie Mellon University, Robotics Institute, Diss., August 2010. http://www.programmingvision.com/rosen_diankov_thesis.pdf

[Ebe01] Eberly, D.H.: *3D game engine design*. Bd. 40. Morgan Kaufmann, 2001

[EKS08] El-Khoury, S. ; Sahbani, A.: Handling objects by their handles. In: *IROS-2008 Workshop on Grasp and Task Learning by Imitation*, 2008

[GRA08] GRASP: *Grasp - Proposal of Work*. http://www.csc.kth.se/grasp/GRASP.pdf. Version: 2008

[Gro97] Group, Khronos: *OpenGL Software Development Kit: Documentation, Sample Code, Libraries, and Tools for creating OpenGL-based Applications*. http://www.opengl.org/sdk/docs/. Version: 1997

[GRS96] Gilks, W.R. ; Richardson, S. ; Spiegelhalter, D.J.: Introducing markov chain monte carlo. In: *Markov chain Monte Carlo in practice* (1996), S. 1–19

[HAN08] *Handle Project*. www.handle-project.eu/. Version: 2008

[HAN09] HANDLE: *Deliverable 4: Protocol for the corpus of sensed grasp and handling data*. 2009

[HD04] Horn, M. ; Dourdoumas, N.: *Regelungstechnik*. Pearson Studium, 2004. – ISBN 3827370590

[HP86] Hayward, V. ; Paul, R.P.: Robot manipulator control under Unix rccl: A robot control C library. In: *The International Journal of Robotics Research* 5 (1986), Nr. 4, 94. http://www.cs.ubc.ca/~lloyd/rccl.html. – ISSN 0278–3649

[IV] *Open Inventor File Format*. http://web.mit.edu/ivlib/www/iv.html

[JAV] *Java3D*. https://java3d.dev.java.net/

[JBU08] *JBullet*. http://jbullet.advel.cz/. Version: 2008

[JS] Jerez, Julio ; Suero, Alain: *Newton Game Dynamics*. http://newtondynamics.com/

[JV79] Johansson, R.S. ; Vallbo, AB: Tactile sensibility in the human hand: relative and absolute densities of four types of mechanoreceptive units in glabrous skin. In: *The Journal of physiology* 286 (1979), Nr. 1, S. 283. – ISSN 0022–3751

[KH05] Koenig, N. ; Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on* Bd. 3 IEEE, 2005. – ISBN 0780384636, S. 2149–2154

[KHW95] Kessler, G.D. ; Hodges, L.F. ; Walker, N.: Evaluation of the CyberGlove as a whole-hand input device. In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 2 (1995), Nr. 4, S. 263–283. – ISSN 1073–0516

[Koe97] Koecher, M.: *Lineare Algebra und analytische Geometrie*. Springer, 1997. – ISBN 3540629033

[Lia99] Liang, Sheng: *Java Native Interface: Programmer's Guide and Specification*. Sun Microsystems, Inc., 1999 http://java.sun.com/docs/books/jni/

[LM99] Lynch, K.M. ; Mason, M.T.: Dynamic nonprehensile manipulation: Controllability, planning, and experiments. In: *The International Journal of Robotics Research* 18 (1999), Nr. 1, S. 64. – ISSN 0278–3649

[LUD+10] León, Beatriz ; Ulbrich, Stefan ; Diankov, Rosen ; Puche, Gustavo ; Przybylski, Markus ; Morales, Antonio ; Asfour, Tamim ; Moisio, Sami ; Bohg, Jeannette ; Kuffner, James: OpenGRASP: A Toolkit for Robot Grasping Simulation. In: *Simulation, Modeling, and Programming for Autonomous Robots*, 2010 (SIMPAR 2010), S. 109–121

[LWJ] *LWJGL - Lightweight Java Game Library.* http://www.lwjgl.org/

[Mas01] Mason, M.T.: *Mechanics of robotic manipulation.* The MIT Press, 2001. – ISBN 0262133962

[MC03] Miller, A.T. ; Christensen, H.I.: Implementation of multi-rigid-body dynamics within a robotic grasping simulator. In: *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on* Bd. 2, 2003. – ISSN 1050–4729, S. 2262 – 2268 vol.2

[Mir98] Mirtich, Brian: Rigid Body Contact: Collision Detection to Force Computation. In: *IEEE International Conference on Robotics and Automation*, 1998

[MKCA03] Miller, A.T. ; Knoop, S. ; Christensen, H.I. ; Allen, P.K.: Automatic grasp planning using shape primitives. In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on* Bd. 2 IEEE, 2003. – ISBN 0780377362, S. 1824–1829

[MLSS94] Murray, R.M. ; Li, Z. ; Sastry, S. ; Sastry, S.S.: *A mathematical introduction to robotic manipulation.* CRC, 1994. – ISBN 0849379814

[MM04] Miller, Andrew T. ; Miller, Andrew T.: Graspit!: A versatile simulator for robotic grasping. In: *IEEE Robotics and Automation Magazine* 11 (2004), S. 110–122

[MNP90] Markenscoff, Xanthippi ; Ni, Luqun ; Papadimitriou, Christos H.: The Geometry of grasping. In: *The International Journal of Robotics Research* 9 (1990), January, S. 61–74. – ISSN 0278–3649

[MY88] Murty, K. G. ; Yu, F. T.: *Linear Complementarity, Linear and Nonlinear Programming.* Helderman-Verlag, 1988 http://ioe.engin.umich.edu/people/fac/books/murty/linear_complementarity_webbook/

[Ngu86] Nguyen, V.D.: Constructing stable force-closure grasps. In: *Proceedings of 1986 ACM Fall joint computer conference* IEEE Computer Society Press, 1986. – ISBN 0818647434, S. 137

[NHK+07] Nakaoka, S. ; Hattori, S. ; Kanehiro, F. ; Kajita, S. ; Hirukawa, H.: Constraint-based dynamics simulator for humanoid robots with shock absorbing mechanisms. In: *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, 2007, S. 3641 – 3647

[OBJ] *Wavefront Object File Format.* http://www.fileformat.info/format/wavefrontobj/egff.html

[ODE01] *Open Dynamics Engine*. http://www.ode.org/. Version: 2001

[OFF] *Object File Format*. http://shape.cs.princeton.edu/benchmark/documentation/off_format.html

[OSC02] Okamura, A.M. ; Smaby, N. ; Cutkosky, M.R.: An overview of dexterous manipulation. In: *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on* Bd. 1 IEEE, 2002. – ISBN 0780358864, S. 255–262

[PA00] *PA 10 Series: Programming Manual*. Mitsubishi Heavy Industries, Ltd., 2000

[Pia00] Pias, C.: *Computer Spiel Welten*, Universitätsbibliothek Bauhaus Universität Weimar, Diss., 2000. – 154–183 S.

[Reg09] *Einfacher Regelkreis*. http://de.wikipedia.org/w/index.php?title=Datei:Einfacher_regelkreis.gif&filetimestamp=20091106112141. Version: 2009

[RWV00] Råde, L. ; Westergren, B. ; Vachenauer, P.: *Springers Mathematische Formeln: Taschenbuch für Ingenieure, Naturwissenschaftler, Informatiker, Wirtschaftswissenschaftler*. Springer, 2000. – ISBN 3540675051

[SEKD03] Schroder, J. ; Erol, D. ; Kawamura, K. ; Dillman, R.: Dynamic pneumatic actuator model for a model-based torque controller. In: *Computational Intelligence in Robotics and Automation, 2003. Proceedings. 2003 IEEE International Symposium on* Bd. 1, 2003, S. 342 – 347 vol.1

[SFS98] Santello, M. ; Flanders, M. ; Soechting, J.F.: Postural hand synergies for tool use. In: *Journal of Neuroscience* 18 (1998), Nr. 23, S. 10105

[Sha08] *Shadow Dexterous Hand C5: Technical Specification*. Shadow Robot Company, 2008

[SK08] *Kapitel* 15. In: Siciliano, Bruno ; Khatib, Oussama: *Handbook of Robotics*. Springer, 2008. – ISBN 978–3–540–23957–4, S. 345–360

[SMKF05] Shilane, P. ; Min, P. ; Kazhdan, M. ; Funkhouser, T.: The princeton shape benchmark. In: *Shape Modeling Applications, 2004. Proceedings* IEEE, 2005. – ISBN 0769520758, S. 167–178

[SRD00] Sowizral, Henry ; Rushforth, Kevin ; Deering, Michael: *The Java 3d API Specification*. 2nd. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2000. – ISBN 0201710412

[SSK08] Shin, D. ; Sardellitti, I. ; Khatib, O.: A hybrid actuation approach for human-friendly robot design. In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on* IEEE, 2008. – ISSN 1050–4729, S. 1747–1752

[Str87] Straumann, N.: *Klassische Mechanik: Grundkurs über Systeme endlich vieler Freiheitsgrade*. Springer-Verlag, 1987. – ISBN 0387185275

[TL00] Tondu, B. ; Lopez, P.: Modeling and control of McKibben artificial muscle robot actuators. In: *Control Systems Magazine, IEEE* 20 (2000), April, Nr. 2, S. 15 –38. – ISSN 0272–1708

[TPBF87] Terzopoulos, Demetri ; Platt, John ; Barr, Alan ; Fleischer, Kurt: Elastically deformable models. In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. New York, NY, USA : ACM, 1987 (SIGGRAPH '87). – ISBN 0–89791–227–6, 205–214

[TW05] Tegin, J. ; Wikander, J.: Tactile sensing in intelligent robotic manipulation – a review. In: *Industrial Robot: An International Journal* 32 (2005), Nr. 1, S. 64–70. – ISSN 0143–991X

[Unb92] Unbehauen, H.: *Regelungstechnik*. Vieweg Braunschweig, 1992. – ISBN 3528064692

[Vir98] Virtual Technologies. Inc.: *CyberGlove® Reference Manual*, 1998

[WB97] Witkin, Andrew ; Baraff, David: *Physically Based Modeling: Principles and Practice*. http://www.cs.cmu.edu/~baraff/sigcourse/index.html. Version: 1997

[WP80] Wu, Chi haur ; Paul, Richard P.: Manipulator compliance based on joint torque control. In: *Decision and Control including the Symposium on Adaptive Processes, 1980 19th IEEE Conference on* Bd. 19, 1980, S. 88 –94

[YN08] Yamane, K. ; Nakamura, Y.: A Numerically Robust LCP Solver for Simulating Articulated Rigid Bodies in Contact. In: *Proceedings of Robotics Science and Systems IV Zurich Switzerland* 4 (2008), S. 89–96

[Zä09] Zäschke, Tilmann: *ODE for Java*. http://ode4j.sourceforge.net/. Version: 2009

Erklärung

Ich, Hanno Scharfe, versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereichs Informatik einverstanden.

Hamburg, den 8.12.2010