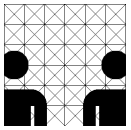
Diplomarbeit
im Studiengang Informatik

Simulation des parallelen Betriebs zweier MHI PA10-6C Roboterarme in RCCL

am Arbeitsbereich für
Technische Aspekte Multimodaler Systeme,
Universität Hamburg

vorgelegt von
Gunter Labes
Juni 2009



betreut von
Prof. Dr. Jianwei Zhang
Dr. Werner Hansmann

Abstract

A simulator is usually needed for the effective offline programming of robots, which tests the developed program before its online implementation. In this thesis, a simulator for the control of multiple MHI PA10-6C robot arms was implemented and integrated into RCCL. This allows the simulation of arbitrary applications for the apparatus, displayed either as text or in 3D by an external program connected via the network. The latter is included in this thesis as well. The implementation was achieved through an extension of the built in simulator (robotsim) and necessary adjustments to the PA10 support of RCCL.

With the simulator, it is now possible to simultaneously develop and test programs for the TASER service robot of the University of Hamburg's TAMS group. Furthermore the simulator can be used as supplement in lectures and corresponding exercises where PUMA robots are used at this time.

Kurzzusammenfassung

Zur effektiven Offline-Programmierung von Robotern wird üblicherweise ein Simulator verwendet, der idealerweise direkt vom entwickelten Programm aus gesteuert werden kann. In der vorliegenden Diplomarbeit wird ein Simulator zum Betrieb mehrerer MHI PA10-6C Roboterarme implementiert und in die Roboterprogrammierbibliothek RCCL integriert. Damit ist es möglich beliebige in RCCL geschriebene Anwendungen für einen MHI PA10-6C simulieren zu lassen. Der Simulationszustand kann optional in 3D Grafik mit einem weiteren per Netzwerk verbundenen Programm dargestellt oder in Textform ausgegeben werden. Die Implementierung erfolgte durch Erweiterung des vorhandenen RCCL-Simulators `Robotsim`, sowie notwendiger Anpassung der PA10 Unterstützung von RCCL.

Mit dem Simulator ist es nun möglich Programme für den TASER Serviceroboter vom Arbeitsbereich TAMS der Universität Hamburg zeitlich parallel zu entwickeln und insbesondere zu testen. Es ist geplant den Simulator in Lehrveranstaltungen sowie den zugehörigen Übungen dort einzusetzen, wo bis jetzt PUMA Modelle behandelt wurden, die am Fachbereich nicht zur Verfügung stehen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

1

Der Begriff „Roboter“ wurde zum erstenmal 1921 im Theaterstück „Rossums Universalroboter“ des tschechischen Schriftstellers Karel Čapeck verwendet, der es vom tschechischen Wort „robota“ für Frondienst ableitete. In diesem Theaterstück wird ein Roboter als jede automatisch tätige Maschine, die menschliche Arbeitskraft ersetzt, bezeichnet. Dabei muss diese Tätigkeit nicht in menschenähnlicher Art und Weise ausgeführt werden und die Maschine muss auch nicht zwangsweise menschenähnlich aussehen. Nicht sehr viel später, im Jahre 1954, wurde von Georg Devol ein Patent für einen programmierbaren Manipulator eingereicht und 2 Jahre darauf zusammen mit Joseph Engelberger auch praktisch realisiert. Der erste Roboterhersteller, Unimation, wurde daraufhin im Jahre 1958 gegründet. Dies stellt den Beginn des industriellen Einsatzes von Robotern dar.

Der Bewegungsgrad der Roboter stellt ein wichtiges Unterteilungskriterium in der Robotik dar. Zum einen gibt es die mobilen Roboter, die sich frei in ihrer Umwelt bewegen können und zum anderen die stationären Roboter, die schon von Anfang an in der industriellen Fertigung eingesetzt wurden und hauptsächlich aus einem fest montierten Manipulator mit zur Aufgabe passendem Endeffektor bestehen. Die vorliegende Diplomarbeit beschäftigt sich näher mit einem Roboterarm, der sich vor allem gut in mobilen Roboteranwendungen einsetzen läßt. Speziell geht es hier um einen Serviceroboter, der am Arbeitsbereich Technische Aspekte Multimodaler Systeme (TAMS) der Universität Hamburg eingesetzt wird.

Das Fraunhofer-Institut für Produktionstechnik und Automatisierung (IPA) definiert dabei Serviceroboter wie folgt: „Ein Serviceroboter ist eine frei programmierbare Bewegungseinrichtung, die teil- oder vollautomatisch Dienstleistungen verrichtet. Dienstleistungen sind dabei Tätigkeiten, die nicht der direkten industriellen Erzeugungen von Leistungen für Menschen oder Einrichtungen dienen.“

In dieser Arbeit wird ein Simulator für den Arm eines Serviceroboters entwickelt, was vor allem zur Unterstützung der freien Programmierbarkeit dient. Warum dies sinnvoll ist, soll im nun folgenden Kapitel ausgeführt werden.

1.1 Motivation

Aufgrund des kontinuierlichen technologischen Fortschritts wie Miniaturisierung, bessere Energieeffizienz bei mehr Rechenleistung usw., welcher zu kleineren und kostengünstigeren Geräten führt, wäre es anzunehmen, dass es nur eine Frage der Zeit ist, bis jegliche technischen Geräte allgegenwärtig verfügbar sind und deren Simulation überflüssig ist. Im Folgenden wird jedoch erläutert, dass, auch wenn die Annahme zur allgegenwärtigen Verbreitung technischer Geräte zutrifft, es immer gewisse Vorteile bringt, einen Simulator zur Verfügung zu haben.

Insbesondere in Hinblick auf die Offline-Softwareentwicklung für einen Roboter sind folgende Vorteile zu nennen:

- Der Simulator kann von beliebig vielen Personen gleichzeitig genutzt werden. Dabei ist es gleichgültig, wieviele Roboter tatsächlich vorhanden sind. Dies ist insbesondere bei Lehrveranstaltungen hilfreich, wenn mehreren unerfahrenen Personen ein Robotersystem nähergebracht werden soll. Es ist dann nicht notwendig, jedem Teilnehmer einen Roboter zur Verfügung zu stellen und ihn individuell zu betreuen, um zum Beispiel Unfälle zu vermeiden.
- Die Roboter können am Simulator für neue Aufgaben umprogrammiert und ohne Verzögerung eingesetzt werden.
- Während der Softwareentwicklung können auch kleine Änderungen sofort getestet werden, ohne auf den realen Roboter zuzugreifen und eventuell aufwendige Testbedingungen herstellen zu müssen. Des Weiteren ist die Fehlersuche einfacher, da zum Beispiel in Einzelschritten durch das Programm gegangen und der Zustand in jedem Schritt in Ruhe inspiziert werden kann.

  Dies ist am realen Gerät schon aus physikalischen Gründen gar nicht möglich.
- Es besteht keine Gefahr den Roboter oder dessen Umgebung zu beschädigen.
- Daraus folgt, dass das Testen potentiell gefährlicher Bewegungsabläufe, wie zum Beispiel relativ schnelle Bewegungen in restriktiver/begrenzter Umgebung, einfacher oder überhaupt erst möglich ist.
- Korrektheitstests von Programm(teil)en können automatisiert werden.

Ein weiterer wichtiger Vorteil in Hinblick auf diese Arbeit ist die Skalierbarkeit eines Simulators. Es besteht die Möglichkeit der Erweiterung der Simulation auf mehr als nur einen Roboterarm. Im Moment ist an der mobilen Plattform des Arbeitsbereichs TAMS nur einer der beiden MHI PA10-6C Roboterarme angebaut. In Zukunft soll aber auch der zweite Manipulator integriert werden. Dies führt zu der Frage, wie beide Arme am sinnvollsten zueinander platziert werden können. Zur Beantwortung dieser Frage ist eine Arbeitsraumuntersuchung mit beiden Armen im Hinblick auf gute Reichweite und vielfältige Interaktionsmöglichkeiten zwischen den Armen Voraussetzung. Um eine solche Evaluierung durchführen zu können, bevor ein Umbau

stattfindet, ist ein Simulator hilfreich, um verschiedene Konfigurationen austesten zu können. Dieser sollte dafür eine Schnittstelle zur programmatischen Steuerung beinhalten, damit der Arbeitsraum systematisch von einem oder mehreren Programmen erfasst werden kann. Das ist ein Grund dafür, dass ein zentraler Aspekt dieser Arbeit die Integration in RCCL[1] sein wird.

Mit funktionierender Simulation zweier Arme wird es dann möglich Interaktionen zwischen diesen schon auszuprobieren und zu untersuchen, bevor sie praktisch umsetzbar sind.

Zusammenfassend kann gesagt werden, dass ein in RCCL integrierter Simulator für den PA10-6C Roboterarm viele Möglichkeiten mit sich bringt, die bis jetzt nur für PUMA[2] Modelle vorhanden sind. Außerdem ist die Integration in RCCL eine wichtige Vorraussetzung für systematische Arbeitsraumuntersuchungen zur Integration eines zweiten Manipulators.

1.2 Zielsetzung

Aus den vorhergehenden Überlegungen ergibt sich das Thema dieser Arbeit: „Simulation des parallelen Betriebs zweier MHI[3] PA10-6C Roboterarme in RCCL“.

Über eine Netzwerkschnittstelle wird außerdem der Simulationszustand angeboten, der dann zur grafischen Ausgabe in 3D dienen kann.

Die Ansteuerung des Simulators soll über die in der Roboterprogrammierbibliothek RCCL[4] integrierte Methode zur Simulation erfolgen. Ein Programm, das RCCL benutzt, um einen Roboter zu steuern, soll dabei transparent, das heißt ohne Modifikation, auch auf dem Simulator ausführbar sein. Die Ausgabe geschieht entweder über die oben genannte Grafikschnittstelle oder es wird die Trajektorie der ausgeführten Bewegung in Textform ausgegeben, bzw. in eine Datei geschrieben.

RCCL ist eine Programmierbibliothek zur Robotersteuerung, die besonders im wissenschaftlichen Umfeld verbreitet ist. Sie wird am Arbeitsbereich TAMS zur Steuerung der vorhandenen PA10-6C Manipulatoren verwendet.

1.3 Aufbau der Arbeit

In der vorliegenden Arbeit wird die Erstellung eines in RCCL integrierten Simulators für den Roboterarm PA10-6C von Mitsubishi dargelegt.

[1] Robot Control C Library – siehe Abschnitt 2.3
[2] Programmable Universal Machine for Assembly
[3] Mitsubishi Heavy Industries® - http://www.robot-arm.com, http://www.sdia.or.jp/mhikobe-e/products/mechatronic/e_index.html
[4] Robot Control C Library, siehe Abschnitt 2.3

Die Arbeit ist wie folgt aufgebaut:

Im Kapitel 2 werden zuerst die Grundlagen zur Simulation und anschließend zur Roboterprogrammierung im Allgemeinen und mit RCCL im Speziellen erläutert. Außerdem wird der Serviceroboter des Arbeitsbereiches TAMS, der TASER, vorgestellt, von welchem der Roboterarm ein wesentlicher Bestandteil ist.

Im Kapitel „Stand der Forschung“ werden zu dieser Arbeit vergleichbare Projekte vorgestellt und deren Einfluss auf die Zielsetzung evaluiert.

Im Kapitel 4 und 5 wird die Implementierung des Simulators erläutert. Hierzu wird zuerst in Kapitel 4 die Methode der Integration in RCCL dargelegt, sowie die Arbeitsweise des derzeitig vorhandenen Simulators `Robotsim`. Sodann werden zwei mögliche Ansätze der Implementierung (Erweiterung des bestehenden Simulators vs. Neudesign) miteinander verglichen und der gewählte Ansatz im darauffolgenden Kapitel im Detail beschrieben. Außerdem wird die neue grafische Darstellung der Simulation vorgeführt und beschrieben.

Im Kapitel 6 wird zunächst die Benutzung des nun erweiterten `Robotsim` erklärt und dann ein Bewegungsablauf parallel auf dem Simulator und dem TASER demonstriert.

Im Kapitel 7 wird ausgewertet, inwiefern die Zielsetzung dieser Arbeit erfüllt werden konnte und welche Anwendungsmöglichkeiten sich daraus ergeben. Es folgt ein Ausblick auf sinnvolle Erweiterungen des Simulators.

In dieser Arbeit wird für einige Begriffe, die sich einerseits im deutschen Fachkollegium schon seit Jahren mit ihrer englischen Bezeichnung etabliert haben, oder für die keine präzise Übersetzung zu finden war, die englische Nomenklatur verwendet. Des Weiteren werden grundlegende mathematische und physikalische Kenntnisse der Robotik weitgehend als bekannt vorausgesetzt.

Grundlagen

2

In diesem Kapitel werden die dieser Arbeit zugrundeliegenden Gebiete der Computersimulation und Roboterprogrammierung kurz angesprochen, sowie die verwendete Roboterprogrammierbibliothek RCCL und deren Entstehung und Anwendung erläutert.

Danach wird der Serviceroboter TASER, die mobile Plattform des Arbeitsbereichs TAMS, vorgestellt. Ein wichtiger Bestandteil des TASER ist der PA10-6C Roboterarm von Mitsubishi Heavy Industries®, für den der Simulator in dieser Arbeit entwickelt wird.

2.1 Computersimulation

Unter Computersimulation bzw. Rechnersimulation ist die Modellierung realer Prozesse (zum Beispiel aus der Physik) und das Experimentieren mit dem erhaltenen Modell mit Hilfe eines Computers, genauer eines Computerprogramms, zu verstehen. Dieses Programm beschreibt bzw. definiert das Simulationsmodell. (KD88)

Traditionell geschieht die formale Modellierung eines Systems mittels mathematischer Modelle, welche eine analytische Lösung zu finden versuchen, die die Vorhersage des Systemverhaltens, basierend auf bestimmten Parametern und eines Initialzustandes, erlaubt.

Computersimulationen können einerseits ausschließlich Algorithmen benutzen, die ihre Grundlage in diesen mathematischen Modellen haben und andererseits diese Algorithmen auch mit aktuellen Ereignissen kombinieren. Zum Beispiel ist es möglich, nur die Eingabedaten zu simulieren und damit ein reales System auf Langzeitbenutzungsprobleme oder Limitierungen in Extremsituationen hin zu untersuchen. Andersherum können reale Eingabedaten dazu benutzt werden, ein simuliertes System zu steuern. Hierbei beinhaltet der Begriff Computersimulation als Spezialfall die Computermodellierung, womit im Allgemeinen gemeint ist, dass tatsächlich alle Aspekte des realen Systems im Computer repräsentiert werden.

Im Rahmen dieser Arbeit ist im Bereich der Computersimulation außerdem die Steuerbarkeit mittels eines Computerprogramms und dabei der transparente Zugriff

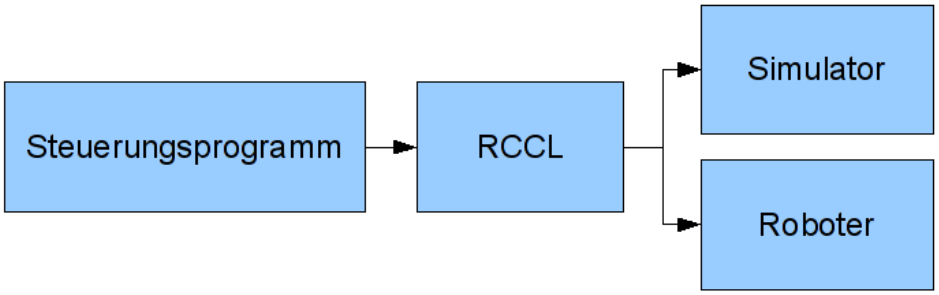


Abbildung 2.1: Transparenter Zugriff auf Roboter oder Simulator am Beispiel von RCCL

auf Simulator sowie realen Roboter wichtig. In Abbildung 2.1 wird der transparente Zugriff anhand eines Diagramms am Beispiel von RCCL verdeutlicht.

2.2 Roboterprogrammierung

2.2.1 Online-Programmierung durch Nachahmung

In der Vergangenheit sowie bis heute werden Roboter in der Industrie hauptsächlich mit Hilfe der Online-Programmierung gesteuert. Hierbei wird, in einem speziellen Modus (meist auch mit einer speziellen Version des Roboters, dem sogenannten „teach pendant“), dem Roboter eine Bewegung vorgeführt, die dieser dauerhaft speichert, um sie dann beliebig oft wieder abspielen zu können. Weitere Methoden zur Vorführung von Bewegungen sind unter anderem im „Handbook of Robotics“ (BCDS07) beschrieben. Hier werden auch einige Methoden zum Verallgemeinern der Bewegung über mehrfache Wiederholungen vorgestellt, wodurch zum Beispiel unvermeidliches Zittern/Rauschen herausgefiltert werden kann. In diesem Zusammenhang wird im Artikel „Imitation Learning of Dual-Arm Manipulation Tasks in Humanoid Robots“ (AAGD08) ein auf dem „Hidden Markov Model“ basierendes Lernverfahren auf Bewegungen zweier Arme angewendet.

Der große Vorteil der Online-Programmierung besteht in der Trivialität[1] des Lernprozesses sowie die klare Vorhersagbarkeit der Bewegung. Das ist gerade bei unüberwachter Arbeit sehr wichtig. Durch geeignete Verfahren kann auch das Beibringen „guter“ Bewegungen sehr einfach gestaltet werden. In der Abhandlung „Easy robot programming for industrial manipulators by manual volume sweeping“ (MUM08) wird so ein Verfahren vorgestellt.

[1] Auch wenn, wie an den zitierten Arbeiten zu sehen ist, es durch aus auch hochentwickelte Lernmethoden gibt.

Von Nachteil ist die Unflexibilität des einmal erlernten Arbeitsablaufes. Da einfach ein Bewegungsmuster stur abgearbeitet wird, muss zum Beispiel die Positionierung des zu bearbeitenden Werkstückes bei jedem Durchlauf sehr präzise erfolgen. Eine hohe Wiederholgenauigkeit der erlernten Bewegung kann bei den meisten Robotern allerdings problemlos erreicht werden.

2.2.2 Offline-Programmierung mit Hilfe eines Simulators

Die Alternative zur Online-Programmierung stellt die Offline-Programmierung dar. Dies bedeutet, dass separat (offline) ein Programm zur Lenkung des Roboters entwickelt und anschließend auf dem Roboter ausgeführt wird.

Vorteil hierbei ist die im Prinzip unbegrenzte Flexibilität der Reaktionen des Roboters. Mit geeigneter Programmierung und der nötigen Ausstattung können Sensordaten intelligent ausgewertet und auf beliebige Situationen kann entsprechend dynamisch reagiert werden. Dies ist eine Grundvoraussetzung für mobile Roboter, die autonom agieren sollen.

Im Allgemeinen ist dabei ein Simulator notwendig, um während des Entwicklungsprozesses das Programm einfach testen und auf Fehler überprüfen zu können. Eventuell ist zu dem Zeitpunkt das eigentliche Gerät noch in der Entwicklung oder zumindest lokal nicht verfügbar.

Ein wichtiger Aspekt in Hinblick auf die hierbei verwendeten Simulatoren ist die Möglichkeit, dasselbe Programm unmodifiziert sowohl auf dem Simulator als auch auf dem eigentlichen Roboter ablaufen lassen zu können. (Beispiel siehe Abbildung 2.1)

2.3 RCCL - Robot Control C Library

RCCL ist eine Sammlung von in der Programmiersprache C implementierten Funktionen zur Echtzeitsteuerung einer Reihe von Industrieroboterarmen, hauptsächlich PUMAs. Eine Übersicht des prinzipiellen Aufbaus von RCCL ist in Abbildung 2.2 zu sehen. Der Quellcode ist frei verfügbar. Dies ermöglicht Einsicht in die Funktionsweise, falls die Dokumentation zum Beispiel nicht ausreichend oder veraltet ist und vor allem eine beliebige Anpassung sowie Erweiterung, zum Beispiel um neue Robotertypen zu unterstützen. RCCL darf allerdings nicht kommerziell eingesetzt werden. Es wird daher weltweit vor allem in Forschungseinrichtungen verwendet und weiterentwickelt.

Im Arbeitsbereich TAMS der Universität Hamburg wird RCCL zur Steuerung von PA10-6C Roboterarmen eingesetzt. Torsten Scherer hat dazu im Rahmen seiner Dissertation (Sch04) RCCL um Unterstützung dieses Robotertyps erweitert.

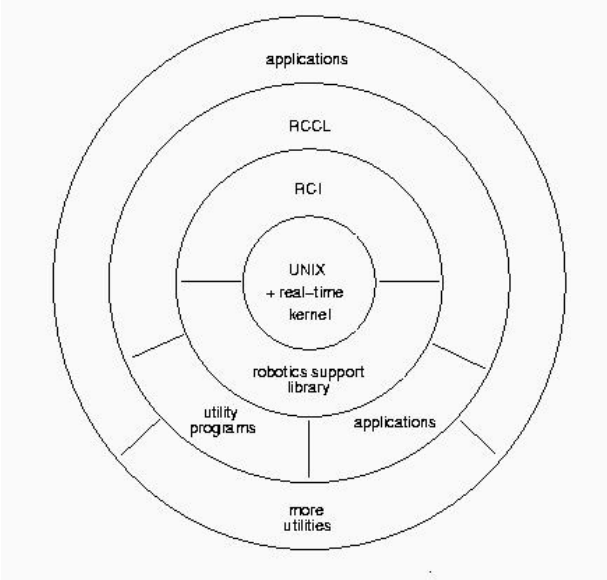


Abbildung 2.2: Der strukturelle Aufbau von RCCL-Programmen und deren unterliegenden Schichten.

2.3.1 Entstehung und Weiterentwicklung

- 1983 ursprünglich an der Purdue University von Vincent Hayward und Richard Paul entwickelt (HP84; HR87) – Version 1.0
- 1985 an der McGill University von Lloyd, Parker und McClain leicht erweitert – Version 2.0
- 1987-88 um Multiroboter- und Multiprozessorfähigkeiten an der McGill University und dem General Electric Advanced Technology Laboratory (New Jersey) im Auftrag des Jet Propulsion Laboratory (JPL) erweitert (LH93) – Version 3.0
- 1989 Fehlerbereinigung und Dokumentationsaufbesserung – Version 4.0
- 1996 Portierung auf Betriebssysteme (Sun Solaris, SGI IRIX) mit richtigen Echtzeitfähigkeiten, um die zugrundeliegende Implementierung zu vereinfachen; letzte offizielle Fassung (LH96) – Version 5.0
- 1999 an der Universität Bielefeld von Torsten Scherer um Unterstützung für den PA10 Roboterarm ergänzt – Version 5.1
- 2003 an der Universität Hamburg von Torsten Scherer die PA10-6C Roboterarmsteuerung ermöglicht – Version 5.1.4

Seit der Entwicklung der Version 5.0 1996 wurde die Dokumentation praktisch nicht weiter aktualisiert.

2.3.2 Funktionsunterstützung

Zur Steuerung von Robotern müssen häufig Positionen in kartesischen oder Gelenkkoordinaten berechnet werden. RCCL bietet hierzu nützliche Datentypen und Funktionen an.

Der wohl wichtigste Datentyp ist die homogene Transformation, die auch einfach als Transformation oder TRSF in RCCL bezeichnet wird. Eine Transformation spezifiziert ein Koordinatensystem und kann als absolute Pose (Position und Orientierung) oder relative Veränderung einer Pose (eine Bewegung) im Hinblick auf ein Referenzkoordinatensystem betrachtet werden. Wenn das Referenzkoordinatensystem der Ursprung ist, meinen die beiden Betrachtungsweisen dasselbe. Eine Transformation setzt sich somit aus einer Translation und einer Rotation zusammen. (siehe (Zha08))

Transformationen können einfach miteinander multipliziert werden, um eine kombinierte Pose/Bewegung als Aneinanderreihung einzelner Bewegungen darzustellen. RCCL erlaubt es, benutzerdefinierte Funktionen (control functions) an Transformationen zu binden. Mit Hilfe dieser Funktionen können neben den vordefinierten Bewegungstypen (zum Beispiel in kartesischen oder Gelenkwinkelkoordinaten interpolierte lineare Bewegung) beliebige neue Bewegungstypen realisiert werden.

Ein weiterer wichtiger Aspekt der Robotersteuerung ist die Kinematik eines Roboters. Die Vorwärtskinematik wird gebraucht, um aus den Gelenkwinkelstellungen eine kartesische Position zu erzeugen. Sie ist immer eindeutig, hat also genau eine Lösung. Entsprechend ist die inverse Kinematik dazu da, aus einer kartesischen Position die einzelnen Gelenkwinkelwerte zu berechnen. Die inverse Kinematik eines Roboters mit sechs Freiheitsgraden hat teilweise mehrere Lösungen, so dass es notwendig ist, weitere Kriterien festzulegen, um eine bevorzugte Lösung auszuwählen. Zum Beispiel ist es vorteilhaft, wenn sich der Ellenbogen des linken Armes immer nach links außen dreht und der des rechten entsprechend nach rechts außen, so dass sie sich gegenseitig nicht behindern. Hierfür bietet RCCL eine Bitmaske (configuration bitmask) zur Einstellung der Kinematikroutinen an.

Die Bibliothek beinhaltet außerdem `Robotsim`, einen Simulator, der an Stelle des tatsächlichen Roboters gesteuert werden kann. Die grafische Darstellung erfolgt im Drahtgittermodell, wie in Abbildung 2.3 gezeigt.

Die Funktionsweise von `Robotsim` wird in Kapitel 4.2 näher beschrieben.

Im nächsten Abschnitt soll anhand eines Beispielprogramms die Benutzung von RCCL verdeutlicht werden.

2.3.3 Beispielprogramm

Das im Folgenden vorgestellte Beispielprogramm bewegt den Manipulator zu einer geeigneten Anfangsposition, führt ihn dann in gerader Linie zu einem nahegelegenen

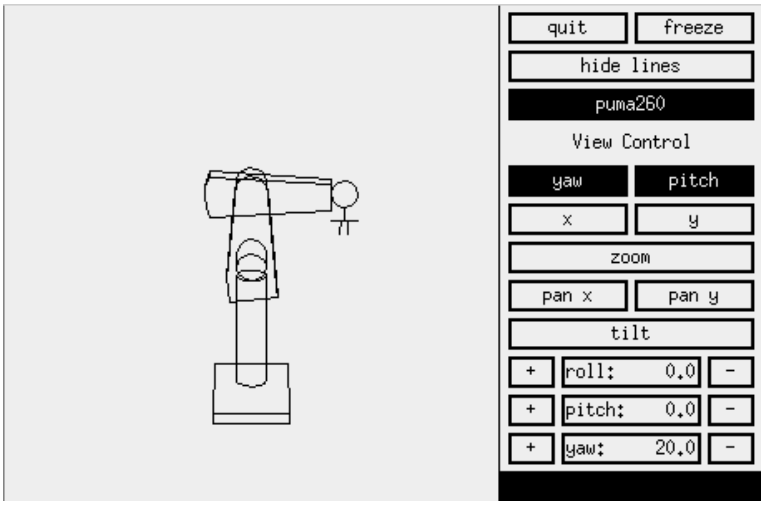


Abbildung 2.3: `Robotsim` GUI zeigt einen PUMA260 im Drahtgittermodell

Zielpunkt und schließlich wieder zurück zur Startposition. (siehe Abbildung 2.4) Das Program wird mit RCCL ausgeliefert[2] und ist im Verzeichnis *demo.rccl* unter dem Namen *simple.560.c* zu finden. Es ist auf einen PUMA 560 Roboterarm ausgelegt. Ein equivalentes Beispiel für den PUMA 260 ist im selben Verzeichnis in der Datei *simple.260.c* zu sehen. (Später wird es dort auch ein gleichartiges Beispiel für den PA10-6C in der Datei *simple.PA10-6.c* geben.)

Abbildung 2.4 zeigt die beiden Bewegungen, die ausgeführt werden nachdem der Manipulator die Ausgangsposition erreicht hat. Es werden zwei Koordinatensysteme[3] angedeutet. Zum Einen das Basiskoordinatensystem der sogenannten T6 Transformation[4] des Arms, welches an dessen Schulter ansetzt und das T6-Koordinatensystem, welches sich im letzten Gelenk befindet.

Der Quelltext sieht wie folgt aus:

```
#include <rccl.h>
#include "manex.560.h"

main()
{
        TRSF_PTR P, T;                                          /*#1*/
```

[2]Die hier gezeigte Variante ist etwas abgeändert, da in der Version 5.1.4 von RCCL einige Funktionen, vor allem im Bezug auf das Speichermanagement, vereinfacht wurden. Zum Beispiel muss Transformationen kein Name mehr zugewiesen werden und *rcclCreate()* braucht kein CPU-Argument. Die Beispielprogramme wurden bis jetzt jedoch noch nicht aktualisiert.

[3]in der Abbildung mit „frame“ bezeichnet

[4]Damit ist die Transformation gemeint, die von der Basis des Manipulators bis zu seinem letzten Gelenk führt. (Manipulatoren haben oft sechs Gelenke um die sechs notwendingen Freiheitsgrade für beliebige Bewegungen zu ermöglichen.)

```
        POS_PTR pos;                                                    /*#2*/
        MANIP *mnp;                                                     /*#3*/
        JNTS rcclpark;                                                  /*#4*/
        char *robotName;                                                /*#5*/

        rcclSetOptions (RCCL_ERROR_EXIT);                               /*#6*/
        robotName = getDefaultRobot();                                  /*#7*/
        if (!getRobotPosition (rcclpark.v, "rcclpark", robotName))
         { printf ("position 'rcclpark' not defined for robot\n");
           exit(-1);
         }
                                                                        /*#8*/
        T = allocTransXyz (-300.0, 0.0, 75.0);
        P = allocTransRot (P_X, P_Y, P_Z, xunit, 180.0);
        pos = makePosition ("pos", T6, EQ, P, T, NULL);                 /*#9*/

        mnp = rcclCreate (robotName);                                   /*#10*/
        rcclStart();

        movej (mnp, &rcclpark);                                         /*#11*/

        setMod (mnp, 'c');                                              /*#12*/
        move (mnp, pos);                                                /*#13*/
        stop (mnp, 1000.0);

        movej (mnp, &rcclpark);                                         /*#14*/
        stop (mnp, 1000.0);

        waitForCompleted (mnp);                                         /*#15*/
        rcclRelease (YES);                                              /*#16*/
}
```

Der Header *rccl.h* enthält notwendige Deklarationen von RCCL-Funktionen, Strukturen und Variablen. Es ist der typische Einstiegspunkt in RCCL ähnlich, wie zum Beispiel der *stdio.h* Systemheader. Die Datei *manex.560.h* beinhaltet Definitionen, die speziell auf den PUMA 560 ausgerichtet sind.

Das Programm braucht zwei 4x4 homogene Transformationen vom Typ „TRSF“. Dazu werden die Variablen **P** und **T** als Zeiger auf diese Transformationen mit Hilfe des Zeigertyps „TRSF_PTR“ deklariert (#1) (welcher equivalent zu „TRSF*“ ist). Der Endpunkt einer der beiden auszuführenden Bewegungen wird mittels einer Positionsgleichung vom Datentyp „POS“ beschrieben. Hierzu wird wieder ein Zeiger verwendet, der den Typ „POS_PTR“ besitzt (#2) (welcher equivalent zu „POS*“ ist). In RCCL wird ein Manipulator mittels der „MANIP“ Struktur angesprochen,

auf welche die Variable mnp zeigt (#3). Die bekannte Ausgangsposition (rcclpark) wird über eine Menge von Gelenkwinkeln vom Typ „JNTS“ festgelegt (#4). Als letztes brauchen wir noch eine Zeichenkette „robotName“, die den Namen des Roboters bezeichnet (#5).

Der erste Befehl ruft die Funktion *rcclSetOptions()* mit dem Wert RCCL_ERROR_EXIT auf (#6). Nach dem setzen dieser Option werden die meisten RCCL-Funktionen eine Diagnosenachricht ausgeben und das Programm benden sobald ein Laufzeitfehler entdeckt wird. Dadurch muss nicht jedesmal der Rückgabewert einer Funktion überprüft werden.

Als nächstes holt sich das Program den Namen des zu steuernden Roboters (#7) und die Ausgangsposition. *getDefaultRobot()* gibt den Namen des im System definierten Standardroboter zurück. Dieser ist in der Datei *defaultRobot.cfg* angegeben, die über die Umgebungsvariable `RCCL_PATH_<arch>`[5] gefunden wird. Mit Hilfe des Roboternamens kann dann die sogenannte „rcclpark“-Stellung abgefragt werden. Die Funktion *getRobotPosition()* schaut dazu in der Datei `<robotName>.pos` unter dem übergebenen Namen nach und liefert eine 1 zurück, falls eine Winkelstellung gefunden wurde, ansonsten ein 0. (Keine Position zu finden wird nicht als ein Fehler an sich interpretiert, weshalb der Rückgabewert explizit überprüft werden muss.) Die gefundenen Winkelwerte werden in das Datenfeld v von rcclpark eingelesen, welches ein Array von *floats* ist.

Anschließend werden die homogenen Transformationen alloziert und instanziiert (#8). Die Transformation **T** wird dazu benutzt eine Lage relativ zur Endeffektorposition in „rcclpark“ Stellung zu spezifizieren. Sie wird mit Hilfe der Funktion *allocTransXyz()* erstellt, die Speicher für den Datentyp „TRSF“ alloziert und ihr eine Translationskomponente von -300.0, 0.0 und 75.0 zuteilt. Der Rotationsanteil wird auf die Identitätsmatrix gesetzt. Die Transformation **P** wird genauso, wie die **T6** Transformation in der Anfangsstellung definiert. Es wäre ebenso möglich diesen Wert auszulesen, sobald der Manipulator diese Stellung einnimmt. Die Transformation wird diesmal mit der Funktion *allocTransRot()* erstellt, wobei die Translationskomponente durch die Konstanten P_X, P_Y und P_Z gegeben ist, die in der Datei *manex.560.h* festgelegt sind. Der Rotationsbeitrag besteht aus einer Drehung von 180 Grad um die X-Achse. (xunit ist eine vordefinierte Variable, die den Einheitsvektor in X-Richtung (1, 0, 0) darstellt.)

Die beiden Transformationen werden nun beim Aufruf von *makePosition()* benutzt (#9), um eine kinematische Positionsgleichung (wie in (Pau81) beschrieben) aufzustellen. *makePosition()* akzeptiert eine variable Anzahl von Parametern. Der Erste von ihnen gibt der Position einen Namen. Die übrigen Parameter bis zum speziellen Argument **EQ** bilden die linke Seite der Gleichung und alle folgenden die rechte Seite. Die Liste endet mit dem NULL-Zeiger. Die **T6** Transformation des Manipulators wird durch das spezielle Argument **T6** symbolisiert. Die im Funktionsaufruf

[5] arch – Architektur/Systemarchitektur; zum Beispiel „Linux“

beschriebene Positionsgleichung sieht demnach wie folgt aus:

$$\mathbf{T6} = \mathbf{P}\,\mathbf{T}$$

Als Bewegung interpretiert, bedeutet das, dass der Manipulator solange bewegt werden soll, bis seine **T6** Transformation die Gleichung erfüllt. In diesem Fall wird **P** (dem Wert von **T6** an der Anfangsposition) multipliziert mit **T** der **T6** Transformation gleichgesetzt.

Die nächsten beiden Funktionen initialisieren und starten den Trajektoriengenerator (#10). *rcclCreate()* alloziert Strukturen, die zum steuern eines bestimmten Roboters notwendig sind. Als Argument erwartet die Funktion den Namen des Roboters. Zurückgegeben wird ein Zeiger auf eine Datenstruktur des Typs „MANIP“. Über diese Struktur wird der Manipulator im Rest des Programms referenziert und gesteuert. *rcclStart()* startet den Trajektoriengenerator ohne den Bewegungsbefehle keinen Effekt haben.

Jetzt ist das Programm soweit den Roboter zu bewegen. Der erste Befehl wird mit *movej()* gegeben. Dieser besagt, dass der Manipulator sich in Stellung der Winkel von rcclpark begeben soll (#11). RCCL erlaubt es Befehle in Gelenkkoordinaten (*movej()*) sowie kartesischen Koordinaten (*move()*) zu geben. Bei Angabe von kartesischen Koordinaten besteht die Möglichkeit zu bestimmen, auf welche Weise die Trajektorie berechnet werden soll. Standardmäßig arbeitet RCCL gelenkinterpoliert. Durch den Aufruf von *setMod (mnp, 'c')* (#12) kann zur linearinterpolierten Methode gewechselt werden. In diesem Modus wird das Koordinatensystem **TOOL**[6], welches im Beispielprogramm der **T6** Transformation entspricht, entlang einer Geraden in kartesischen Koordinaten geführt. Das Problem dabei besteht darin, dass es unter Umständen nicht möglich ist in einer geraden Linie von A nach B zu kommen. Typischerweise, weil es den Roboter in eine unmögliche Position bringen würde. Was daher normalerweise, wie hier auch, gemacht wird, ist die Anfangsposition gelenkinterpoliert anzufahren. Darauffolgende, lokale Bewegungen im kartesischen Koordinatenraum gelingen dann mit einer höheren Wahrscheinlichkeit.

Der sich nun anschließende Funktionsaufruf von *move()* bringt den Roboter in die Lage, die durch auflösen der Positionsgleichung nach **T6** berechnet wurde(#13). Der konsekutive *stop()* Befehl veranlasst den Roboter dort für eine Sekunde zu verweilen.

Mit einem weiteren *movej()* Aufruf wird die Bewegung zurück zum Ausgangspunkt angefordert (#14). (Bewegungen per *movej()* Befehl werden unabhängig vom gesetzten Interpolationsmodus immer in Gelenkwinkelkoordinaten berechnet.)

Bewegungsanfragen sind nicht mit der eigentlichen Bewegung des Armes synchronisiert. Sie platzieren die Anforderung einfach in eine Warteschlange, wo sie darauf warten. so bald wie möglich, vom Trajektoriengenerator bearbeitet zu werden. Nachdem die *movej()* und *move()* Funktionsaufrufe im Beispielprogramm zurückgekehrt

[6]das Koordinatensystem, welches das Werkzeug am Ende des Manipulators beschreibt (tool – Werkzeug)

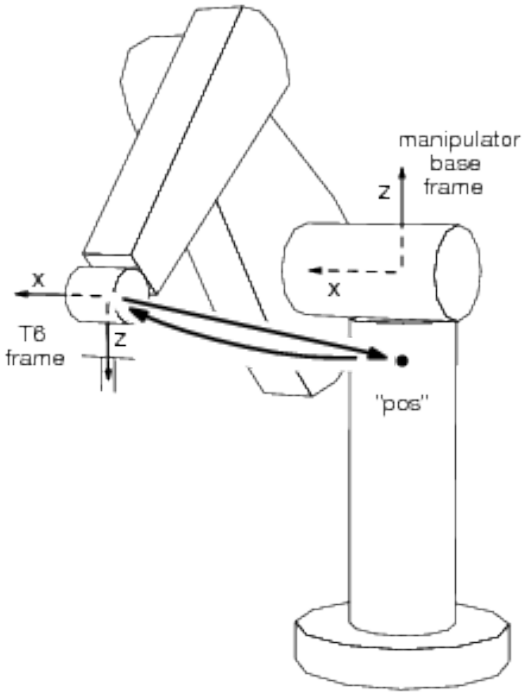


Abbildung 2.4: Beispielbewegung zum Punkt „pos“ in kartesischen Koordinaten und zurück mit Hilfe von Gelenkwinkeln.

sind, hat der Roboter wahrscheinlich noch nicht einmal den ersten *movej()* Befehl zu Ende ausgeführt. Um Programmausführung und Armbewegung zu synchronisieren, stehen verschiedene Vorgehensweisen zur Auswahl. Die einfachste, hier benutzte, Methode besteht darin, darauf zu warten, dass alle Bewegungsanforderungen vollständig abgearbeitet wurden. Dies wird durch das Makro *waitForCompleted(mnp)* erreicht (#15).

Zum Abschluß wird *rcclRelease()* aufgerufen, um den Trajektoriengenerator abzuschalten. Das zu übergebende Boole'sche Argument gibt an, ob auch die Stromzufuhr zum Arm abgeschaltet werden soll.

2.3.4 Anmerkungen

Das RCCL zugrundeliegende Design ist objektorientiert, die Implementierung erfolgte allerdings in der Programmiersprache C. Das führt dazu, dass es viele Strukturen (C-structs) mit Zeigern auf weitere Strukturen oder Funktionen gibt. Dieser Umstand macht es nicht einfach, dem Programm- und Datenfluss zu folgen. Ein eleganterer Aufbau mit sinnvoller Klassenhierarchie wäre bei Reimplementierung in einer objektorientierten Programmiersprache möglich.

Die Autoren merken an, dass eine Konvertierung nach C++[7] interessant wäre. An dieser Stelle sollte darauf hingewiesen werden, dass C++ 1983 erschienen ist. Ungefähr zur gleichen Zeit, zu der die erste Version von RCCL herausgegeben wurde. Es gab also zum Entwicklungsbeginn von RCCL praktisch keine alternative Programmiersprache, die einen objektorientierten Ansatz erlaubte, so dass es sehr einfach nachzuvollziehen ist, warum C gewählt wurde. Die gute Portierbarkeit von C-Programmen auf die verschiedenen unterstützten Plattformen war dabei sicher auch ein wichtiger Grund. Das die erste Version von C++ gerade erst herausgegeben wurde, heißt zu dem natürlich, dass die Sprache zu der Zeit noch lange nicht weit verbreitet oder standardisiert war und wohl noch keine Compiler frei verfügbar waren.

2.4 Der Serviceroboter - TASER

Der TASER ist ein Roboter, der möglichst unabhängig Dienstleistungen für den Menschen verrichten soll. Hierzu wurde er aus verschiedenen gebrauchsfertigen Standardkomponenten[8] zusammengestellt.

Der Arbeitsbereich TAMS besitzt seit Dezember 2003 eine modifizierte Version der mobilen Plattform MP-L655 von Neobotix[9] mit einem Roboterarm, einer Dreifingerhand und diversen Kamerasystemen. Als aktive Sichtsysteme stehen eine omnidirektionale Kamera, ein Stereo-Sicht-System, sowie eine an dem Manipulator installierbare Mikro-Kopf-Kamera zur Verfügung.

Die mobile Plattform ist weiterhin mit zwei gefederten Differentialgetrieben und einem Stützrad hinten sowie zwei Stützrädern vorn, einem Präzisionsgyroskop und je einem Raddrehsensor mit 4096 Schritten pro Motorumdrehung pro Antriebsrad, zwei Laserscannern vom Typ LS 200 der Firma SICK, einem Pentium™ IV 2.4 GHz Standardrechner und zwei Roboterarmen vom Typ PA10-6C ausgestattet. Informationen über das Robotersystem und dessen Steuerung können aus dem Paper „Distributed Applications for Robotic Systems using Roblet-Technology“ (WSZ06) entnommen werden. Abbildung 2.5 zeigt die Software-Architektur des Serviceroboters.

In Abbildung 2.6 ist der TASER mit beiden Armen montiert zusehen. Der momentane Aufbau beinhaltet allerdings nur den linken Arm, da sich bei der abgebildeten Anordnung die beiden Arme zu leicht stören und keine effektive Nutzung erlauben. Als Endeffektor und Greifwerkzeug ist an jedem Arm eine Dreifinger-Roboterhand von Barrett Technologies, Inc. mit vier Freiheitsgraden angebracht. Durch den TorqueSwitch™ Mechanismus werden zwei Gelenke eines Finger mit

[7] C++ unterstützt viele Programmierparadigmen, wie zum Beispiel die Objektorientierung.
[8] Einige dieser Komponenten, wie der PA10-6C, sind heutzutage nicht mehr erhältlich.
[9] http://www.neobotix.de

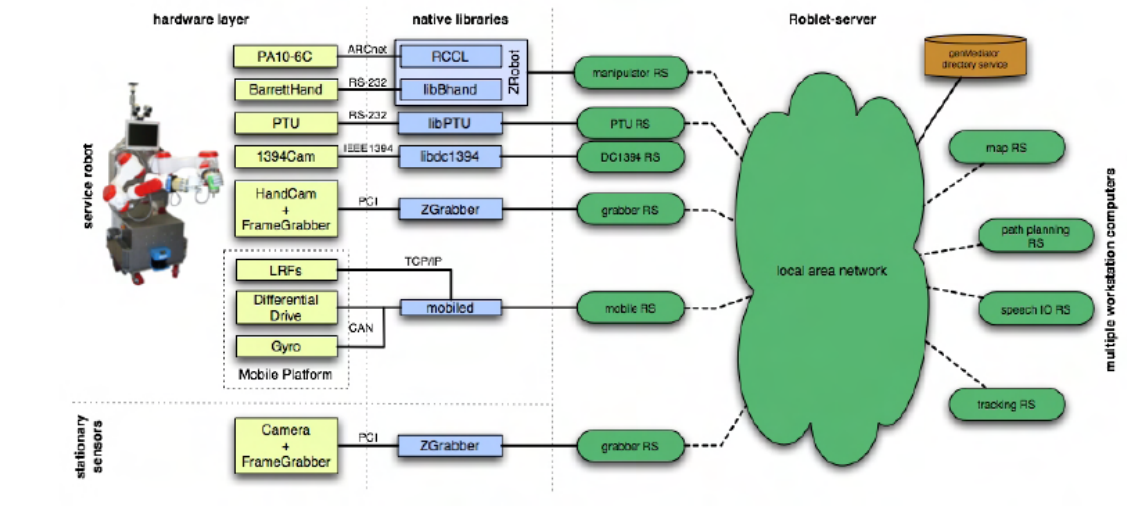


Abbildung 2.5: Die Software-Architektur des TASER Servicereoboters. Die Hardwarekomponenten sind gelb unterlegt, die zugehörigen C/C++ Bibliotheken blau und die sogenannten Roblet-Server grün. (WBLHZ06)

nur einem Motor gesteuert. So wird die Beweglichkeit menschlicher Finger mit ihren Sehnen nachgeahmt. Zwei der drei Finger sind über ein gemeinsames Winkelgelenk miteinander verbunden.

Für die visuelle Unterstützung gibt es ein omnidirektionales Sichtsystem bestehend aus einer Sony DFW-SX900 Kamera und einem Hyperbolspiegel sowie zwei Sony DFW-VL500 Kameras auf einer schwenk- und neigbaren Einrichtung, der sogenannten PTU[10].

Gesteuert wird das komplette System über einen Standardrechner, auf dem GNU/Linux mit einem 2.4er Kernel läuft.

2.4.1 Der PA10-6C Roboterarm

Der PA10-6C Roboterarm (siehe Abbildung 2.7) von Mitsubishi Heavy Industries® hat sechs Gelenke, die in ihrer Anordnung (nicht unbedingt im Aussehen) den Fähigkeiten eines menschlichen Armes ähneln. Aus der Gelenkanzahl ergeben sich die sechs vorhandenen Freiheitsgrade (DOF[11]), die für beliebige Bewegungen im Raum notwendig sind. Der Arm wiegt 38 Kilogramm, kann eine Last von 10 Kilogramm tragen und hat eine nutzbare Länge von 93 Zentimetern. Die Positioniergenauigkeit liegt bei +/- 0,1mm. (HKK+03)

[10]Pan Tilt Unit
[11]Degrees Of Freedom

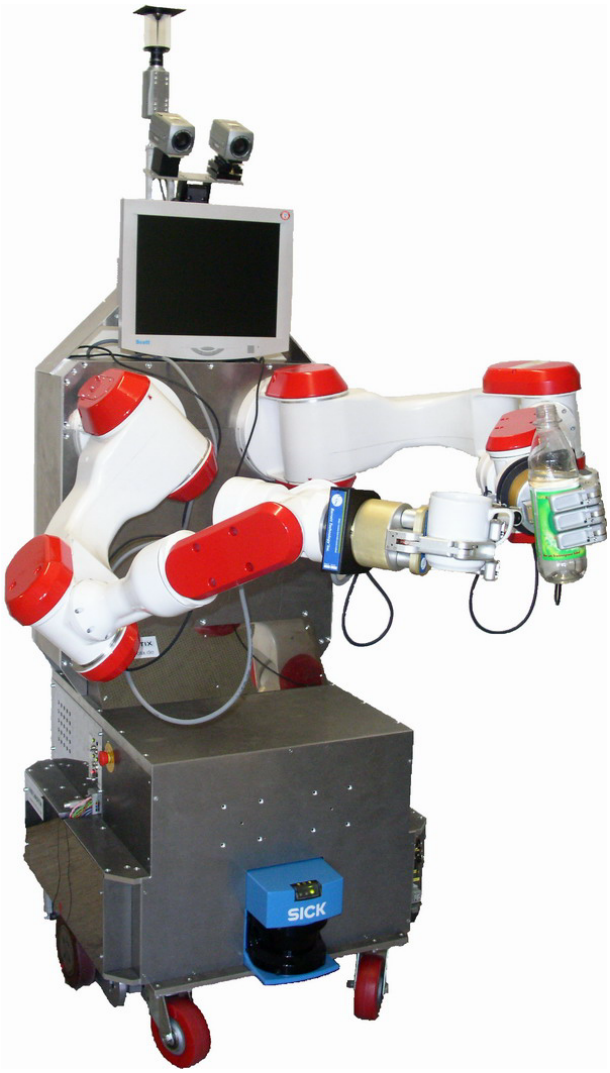


Abbildung 2.6: Der Serviceroboter des Arbeitsbereich TAMS in ursprünglich angestrebter Ausbaustufe mit zwei Armen.

Gesteuert wird der Arm über einen eigenen Controller, der über eine ARCNet-Schnittstelle mit dem PC verbunden werden kann. ARCNet ist vom Prinzip ein Token-Ring-Netzwerk. Dies bedeutet, dass eine Station immer nur dann senden darf, wenn sie den Token besitzt. Dieser wird mit festem Zeitverhalten reihum weitergereicht. Das Kommunikationsprotokoll ist weiterhin so ausgelegt, dass der Controller nie von sich aus Daten sendet. Daher kann es im Netzwerk nicht zu Kollisionen kommen und die Echtzeitkommunikation ist garantiert.

Das Echtzeitverhalten ist nötig, da die Gelenksteuerung alle 10 Millisekunden Befehlsdaten erwartet und es bei Nichteinhaltung zu ruckartigen Bewegungen kommt, die den Arm und den zu manipulierenden Gegenstand beschädigen können.

Die Motorsteuerung der einzelnen Gelenke kann in zwei verschiedenen Modi ablaufen:

1. Geschwindigkeitsmodus
2. Drehmomentmodus

Zu beachten ist hierbei das aufgrund der Gravitation ein Drehmoment von null nicht bedeutet, dass das Gelenk still steht. Verschiedene Gelenke können beliebig per Geschwindigkeits- oder Drehmomentkommando gesteuert werden. (Anhang B von (Sch04))

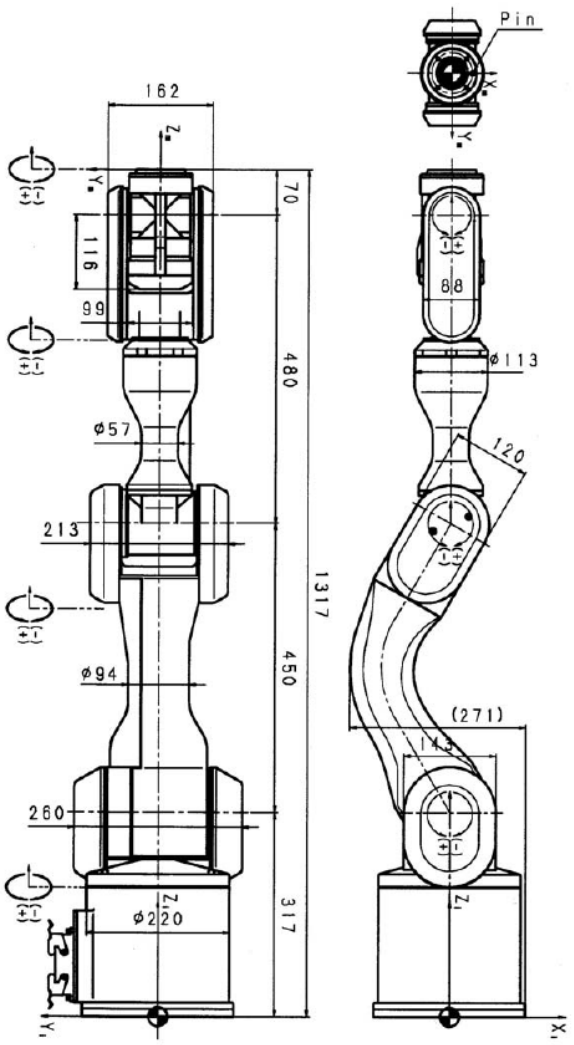


Abbildung 2.7: Schema des PA10-6C Manipulators, Längenangaben in Millimeter

Stand der Forschung

3

In diesem Kapitel werden mit der Zielstellung dieser Arbeit vergleichbare Forschungsprojekte vorgestellt und am Ende deren Auswirkung auf die vorliegende Arbeit ausgewertet.

Zuerst wird ein Projekt beschrieben, dass die grafische Ausgabe von `Robotsim` verbessert. Daraufhin wird QRobot – eine echtzeitfähige, PC-basierte Steuerung von PUMA Manipulatoren – präsentiert. Dieses Projekt hat die Entwicklung eines, im Vergleich zu RCCL ähnlichen, neuen, objektorientierten Robotersteuerungssystems angestoßen, dass auch einen 3D-Simulator enthalten wird. Als letztes wird ein Überblick über das Orocos Projekt gegeben, welches als eine moderne Alternative zu RCCL bezeichnet werden kann.

3.1 Modernisierung der grafischen Darstellung von `Robotsim`

Die Veröffentlichung „A New Graphics Simulator for RCCL and its use in Undergraduate Robotics Instruction“ von Matthew R. Stein und Shawn Falchetti (SF97) beschreibt die Verwendung von RCCL in einem Kurs zur Robotertechnik. Hierbei wird besonders der Nutzen eines Robotersimulators als Lehrmittel hervorgehoben, da Inhalte nur begrenzt durch einfache Vorführungen oder Online-Programmierung per „teach pendant“ vermittelt werden können.

Des Weiteren wird RCCL als sehr gut geeignete Roboterprogrammiersprache gelobt, da hier keine Spezialsprache erlernt wird, die später im Allgemeinen wenig von Nutzen sein wird. Stattdessen wird mit C gearbeitet, einer universalen und sehr weit verbreiteten Programmiersprache. Die Wahrscheinlichkeit ist dabei groß das einige Studenten C schon beherrschen und wenn nicht wird es ihnen in weiteren Projekten von Nutzen sein.

Im Anschluss werden die Entwicklung einer moderneren dreidimensionalen Darstellung des Modells eines PUMA 760 und dessen Umgebung dargelegt. Das Resultat ist in Abbildung 3.1 zu sehen. Für die Darstellung wurde Open Inventor von Silicon Graphics® verwendet. Inventor ist ein objektorientierter Werkzeugsatz in Form von C++ Bibliotheken, die auf SGI® Arbeitstationen in Zusammenarbeit mit XWindows und OpenGL zum Einsatz kommen.

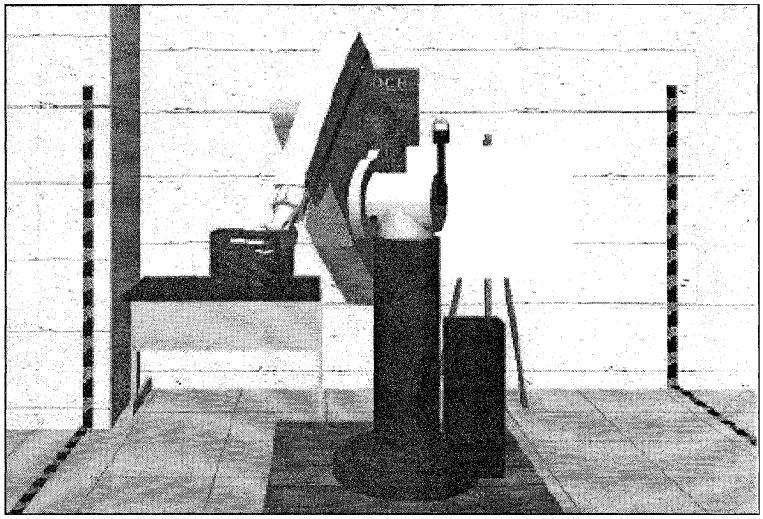


Abbildung 3.1: Modell des PUMA 760 in neuer Grafik

Mit Hilfe von Inventor werden Szenen aus sogenannten Szenenknoten hierarchisch zusammengesetzt. Szenenknoten können sich gegenseitig (aber nur von rechts nach links und von oben nach unten) beeinflussen, so dass kumulative geometrische Transformationen möglich sind. Daher wurden keine Kinematikberechnungen zur Translation der Modelle gebraucht. Die einzelnen Teile eines Roboters, die sich zusammen bewegen, werden durch Knotengruppen repräsentiert. Die Texturen für den Roboter und andere Objekte wurden mit einer CCD-Kamera aufgenommen und mit SGI® Annotator auf die Objekte abgebildet.

Im Verlauf des erwähnten Robotertechnikkurses wurde von den Studenten ein RCCL-Programm zur Steuerung eines Endeffektors, der Malutensilien aufnehmen, benutzen und wieder zurücklegen können soll, erstellt. Das Programm wurde in mehreren Schritten jeweils auf dem Simulator entwickelt und wenn es dort erfolgreich lief, auf dem tatsächlichen Roboter getestet.

3.2 QRobot – Implementierung einer PC-basierten Robotersteuereinheit

In diesem Projekt (NCD98) der Clemson Universität wurde eine echtzeitfähige, PC-basierte Steuerung eines PUMA Roboterarms entwickelt. Alle Berechnungen, die normalerweise die PUMA Steuereinheit durchführt, werden hierbei vom PC übernommen. Dafür wurde zuerst ein Treiber für den PUMA 560 implementiert und dann mittels RCCL sowie ARCL (CK93) angesprochen.

Die Anbindung an RCCL erfolgte per TCP/IP über eine Ethernetverbindung, um zu vermeiden, ein eigenes Kommunikationsprotokoll entwickeln zu müssen. Da der ein-

Abbildung 3.2: OpenGL basierter Simulator von QRobot

gebaute Simulator von RCCL per TCP/IP mit RCCL-Programmen kommuniziert, konnte er recht einfach zur Anbindung eingesetzt werden. Somit können RCCL-Programme so ausgeführt werden, dass sie den Simulator benutzen, aber in Wirklichkeit den Roboterarm steuern.

Da die Ansteuerung über RCCL oder ARCL keine vollständig zufriedenstellenden Lösungen ermöglichen, soll in der nächsten Phase ein neues, objektorientiertes Robotersteuerungssystem entworfen werden, dass in der Funktionalität RCCL und ARCL ähnlich ist. Das neue System wird auch einen 3D Simulator in OpenGL enthalten. Anfänge dazu sind schon gemacht, aber bis jetzt wird nur der PUMA 560 unter Verwendung des Microsoft Windows®Betriebssystems unterstützt. (siehe Abbildung 3.2)

3.3 Orocos

Orocos[1] (Bru01) ist ein Projekt mit dem Ziel, ein allgemein verwendbares, modulares Rahmenwerk zur Roboter- bzw. Maschinensteuerung zu entwickeln. Es wird in Form eines Open Source Projektes durchgeführt, was bedeutet, dass Quelltext und Dokumentation unter Freien Software Lizenzen stehen.

Gestartet wurde Orocos als ein EU-Projekt, an dem die K.U.Leuven in Belgien, das LAAS Toulouse in Frankreich und das KTH Stockholm in Schweden beteiligt waren. Aufgrund der guten Anwendbarkeit in der Industrie hat sich das Projekt in Richtung Maschinensteuerung weiterentwickelt. Gegenwärtig sponsert das Flanders Mechatronics Technology Centre die Weiterentwicklung von Orocos und koordiniert die Integration in Industriegeräte von Werkzeugmaschinenherstellern.

Orocos besteht aus vier C++ Bibliotheken:

[1]Open Robot Control Software – http://www.orocos.org/

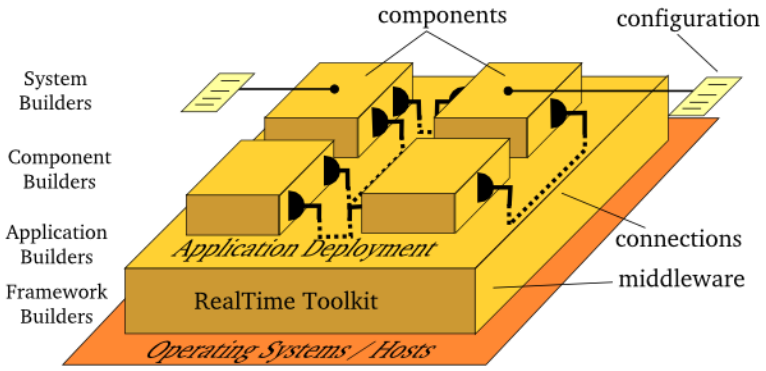


Abbildung 3.3: Aufbau von Orocos

- RealTime Toolkit (RTT)
- Kynematics and Dynamics Library (KDL)
- Bayesian Filtering Library (BFL)
- Orocos Component Library (OCL)

Das RTT (BSK03) bietet die nötige Infrastruktur und Funktionalität, um eine echtzeitorientierte, interaktive und objektorientierte Roboteranwendung in C++ zu entwickeln. Die KDL (Smi) unterstützt das Berechnen von kinematischen Ketten in Echtzeit. Die BFL (GLB05) ist ein anwendungsagnostisches Rahmenwerk zur Inferenz in dynamischen Bayesischen Netzen. Sie stellt rekursive Informationsverarbeitungs- und Abschätzalgorithmen basierend auf der Bayesregel, wie zum Beispiel dem (erweiterte) Kalmanfilter, Partikelfilter (sequentielle Monte Methoden) usw., zur Verfügung. Die OCL beinhaltet komplette Steuerungskomponenten. Die Bibliothek umfasst sowohl Komponentenmanagement als auch einzelne Komponenten zur Steuerung und zum Hardwarezugriff.

Orocos richtet sich an 4 verschiedene Zielgruppen (siehe Abbildung 3.3):

1. Entwickler von Rahmenwerken: Auf Grund der großen Reichweite von Orocos sind Rahmenwerke als Grundlage anderer Komponenten sehr wichtig. RTT, KDL und BFL sind alles Rahmenwerke.

2. Komponentenentwickler: Komponenten bieten bestimmte Dienste innerhalb einer Anwendung an. Mittels der Infrastruktur der Rahmenwerke beschreiben Komponentenentwickler die Schnittstelle eines Dienstes und stellen eine oder mehrere Implementierungen bereit. Zum Beispiel kann eine Kinematikkomponente so entworfen werden, dass sie ihren Dienst für verschiedene Architekturen bereitstellt. Andere Beispiele sind Komponenten für bestimmte Geräte oder Diagnose-, Sicherheits- und Simulationskomponenten. Die OCL wird von Komponentenentwicklern erstellt.

3. Anwendungsentwickler: Diese Entwickler nutzen die von Orocos bereitgestellten Rahmenwerke ebenso wie Komponenten und integrieren sie in eine bestimmte Anwendung. Damit erstellen sie eine spezifische, anwendungsabhängige Architektur. Komponenten werden so verbunden und konfiguriert, dass sie eine Anwendung bilden.
4. Endbenutzer: Diese Gruppe benutzt die Produkte der Anwendungsentwickler, um spezielle Aufgaben zu lösen.

3.4 Auswertung

In diesem Kapitel wurden Rahmenwerke zur Roboterprogrammierung beschrieben, die Alternativen zu RCCL darstellen. Daraus ergibt sich die Frage, warum der Simulator gerade in RCCL integriert werden soll. Da im Moment am Arbeitsbereich TAMS aber nur RCCL zur Manipulatorsteuerung eingesetzt und auch in der Lehre hauptsächlich darauf eingegangen wird, konzentriert sich diese Diplomarbeit ebenfalls ausschließlich auf RCCL. Eine vorstellbare Umstellung auf ein anderes Robotersteuerungssystem würde sehr weitreichende Veränderungen notwendig machen. Diese wären sicher nicht allein aufgrund von Simulationsunterstützung zu rechtfertigen.

Wie weiterhin aus diesem Kapitel ersichtlich wird, gibt es im Zusammenhang mit RCCL nur wenige Arbeiten, die sich mit dem Simulator beschäftigen und ihn um neue Funktionen erweitern oder die grafische Darstellung verbessern. In den vorgestellten Projekten werden außerdem nur PUMA Robotermodelle betrachtet. Demzufolge ist auch nach Auswertung des aktuellen Forschungsstandes die Entwicklung eines Simulators für andere Manipulatoren, wie die bei TAMS benutzten PA10-6C Modelle, erforderlich. Damit muss sich also die Zielsetzung dieser Arbeit nicht ändern, sondern wird in ihrer Notwendigkeit erneut bestätigt.

Vorbereitung der Implementierung

4

Die grundlegende Vorüberlegung zur Implementierung des Simulators war es, einen Netzwerkserver zu erstellen, der die eigentlichen Simulationsberechnungen vornimmt und auf den von RCCL aus zugegriffen werden kann. Weiterhin soll es ein unabhängiges Programm für die Visualisierung des Simulationszustandes geben. Dieses erhält zur 3D Anzeige notwendige Informationen über die Netzwerkschnittstelle. Dabei kann der Netzwerkzugriff je nach Anforderung natürlich auch über das lokale „loopback“ Interface stattfinden. Die Netzwerkschnittstelle ist in erster Linie eine komfortable Abstraktion, die sehr viel Flexibilität bei der Implementierung der grafischen Darstellung erlaubt. Genau genommen gilt dies nicht nur für die grafische Darstellung, sondern beliebige Anwendungen, die den Simulationszustand als Eingabe brauchen. Über die Netzwerkschnittstelle wird im Prinzip nur der aktuelle Zustand der Roboter, in einem hoffentlich erweiterungssicheren Format, angeboten. Das benutzte Kommunikationsprotokoll wird im Abschnitt 5.3 vorgestellt.

Ein weiterer wichtiger Grund für die Abtrennung der Grafikausgabe ist der nicht unerhebliche Rechenaufwand, der im Allgemeinen zur 3D Anzeige nötig ist. Gerade im Hinblick auf eine Erweiterung, um Funktionen zur Kollisionserkennung/-vorhersage, wird diese Leistung vor allem bei Echtzeitanforderungen dringend benötigt. Durch die Entkopplung der Grafikausgabe über das Netzwerk kann mit geringem Leistungsverbrauch auch in leistungskritischen Szenarien eine grafische Ausgabe erfolgen, in dem die Berechnung der Anzeige einfach auf einem anderen Rechner stattfindet.

Im Folgenden werden die Simulatoranbindung von RCCL und die Arbeitsweise von `Robotsim` erläutert sowie zwei mögliche Ansätze zur Realisierung des Simulators vorgestellt und verglichen. Diese beiden Ansätze wurden anfänglich parallel verfolgt, bis genug positive Kriterien für einen der beiden Wege gefunden wurden, um sich auf diesen festzulegen.

4.1 Simulationsunterstützung von RCCL

In RCCL werden Befehle über mehrere Schichten an den entsprechenden Roboter weitergeleitet.

Roboterbeschreibung ⇒ Kommunikationsfunktionen ⇒
Ein-/Ausgabegerät(etreiber) ⇒ Roboter

In dieser Arbeit wird der Einfachheit halber immer von RCCL gesprochen, obwohl gerade die Gerätetreiberschicht eher zu RCI[1] gehört.

Zuerst muss die Roboterbeschreibung definiert werden. Dazu werden ein eindeutiger Name, die entsprechende Roboterklasse, Gelenkparameter und -kinematik und Funktionen zur Kommunikation mit dem eigentlichen Roboter benötigt. Diese Funktionen werden zum Starten und Freigeben des Roboters, sowie zur Ein- und Ausgabe verwendet. Die aktuellen Roboterbeschreibungen sind in der Datei *src/robots/robotDataTab.c* zu finden. Die Kinematik könnte generisch auf den Denavit-Hartenberg (DH) Parametern (DH55; DH64) aufbauend berechnet werden, aber in RCCL wird sie individuell für jede Roboterklasse definiert. Dies liegt daran, dass RCCL schon recht alt ist und diese Berechnungen früher so gut wie möglich optimiert werden mussten, um Rechenzeit zu sparen. Da Hersteller im Allgemeinen schon darauf achten, dass ihre Roboter einfache DH Parameter bekommen, ist dies normalerweise gut möglich.

Des Weiteren werden in RCCL sogenannte Ein-/Ausgabegerätetreiber definiert. Diese Treiber haben einen eindeutigen Namen und mehrere Funktionen, um Daten von dem zugehörigen Gerät auszulesen und Daten zu diesem Gerät zu schreiben. Die aktuellen Treiberdefinitionen sind in der Datei *src/drivers/IOdriverTable.c* aufgelistet.

Die Zuordnung von der Roboterbeschreibung zum Ein-/Ausgabegerät geschieht in der Konfigurationsdatei *robotIO.cfg*. (Diese Datei wird über eine vom Benutzer zu definierende, von der Systemarchitektur abhängige Umgebungsvariable gefunden. In der Hamburger Konfiguration befindet sie sich unter lib/generic/.) Der ursprüngliche Grund für diese Aufteilung liegt darin, dass einige Roboter zusammen mit einem Hardware-Controller ausgeliefert werden und dieser Controller dann die Funktion des Ein-/Ausgabegerätes in RCCL übernimmt und direkt von den Kommunikationsfunktionen angesprochen wird.

In jedem Steuerungszyklus (*setpCtrl()*) von RCCL werden zuerst Daten von allen Robotern eingelesen. Dies geschieht, in dem die Eingabefunktion (*input()*) vom jeweiligen Roboter ausgeführt wird und diese wiederum die Lesefunktion (*read()*) am Gerätetreiber aufruft und die erhaltenen Daten dekodiert und in entsprechende RCCL-eigene Datenstrukturen einträgt. Als Zweites werden die neuen Daten/Befehle an alle Roboter ausgegeben. Dabei wird die Ausgabefunktion (*output()*) des jeweiligen Roboters ausgeführt, welche die neuen Befehle/Daten aus den RCCL Datenstrukturen ausliest, überprüft ob sie zulässig sind und sie dann in korrektem Format an die Schreibfunktion (*write()*) vom Gerätetreiber übergibt.

Danach werden Funktionen zur Verarbeitung der Eingabedaten und zur Berechnung

[1] Robot Control Interface – der Teil von RCCL, der die direkte Hardwaresteuerung vornimmt.

der nächsten Ausgabebefehle abgehandelt. Es ist auch möglich, die berechneten Befehle erst nach Ausführung der Verarbeitungsfunktionen an den Roboter zu senden, um dadurch eine noch engere Steuerung (ohne einen Zyklus Verzögerung) zu ermöglichen. Allerdings geschehen dann die Berechnung der nächsten Ausgabebefehle und die eigentliche Ausführung nicht mehr parallel, was den gesamten Zyklus verlängert.

Die Simulatorunterstützung von RCCL setzt an der Ein-/Ausgabeschicht an, indem ein Simulatorgerät implementiert wird, das anstatt mit einem richtigen Roboter mit dem Programm `Robotsim` per TCP/IP[2] kommuniziert.

Es gibt verschiedene Wege ein RCCL-Programm in den Simulationsmodus zu versetzen. Das Programm kann explizit die RCCL-Option RCCL_SIMULATE setzen oder man kann Simulationsoptionen in der Datei *.rciparams*[3] festlegen. In beiden Fällen wird die Funktion *rcclSetSimulator()* aufgerufen, falls das nicht auch schon vorher explizit vom Programm getan wurde. Als Parameter bekommt *rcclSetSimulator()* eine Zeichenfolge der Form: <simulatorname>[@<hostname>]['<'<port>'>'] (Zeichenketten innerhalb von spitzen Klammern (<>) müssen durch die eigentlichen Werte ersetzt werden. Zeichenketten zwischen eckigen Klammern ([ ]) sind optional. Die gerade genannten Sonderzeichen verlieren ihre Bedeutung, wenn sie innerhalb von Hochkommata (' ') auftauchen.)

Der Simulatorname setzt sich im Allgemeinen aus dem Roboternamen, einem Doppelpunkt und dem Benutzernamen unter dem `Robotsim` läuft zusammen (<robotname>:<USER>). Der Benutzername wird aus der Umgebungsvariablen USER abgelesen, wenn diese nicht gesetzt ist, wird '0' benutzt. Dadurch ist ein Simulator standardmäßig an den aktuellen Benutzer gebunden, so dass mehrere Personen den Simulator parallel nutzen können, ohne sich zu stören. Wenn zum Simulator eines anderen Benutzers verbunden werden soll, muss der Simulatorname manuell angegeben werden. 'hostname' gibt die Internetadresse des Rechners, auf welchem sich der Simulator befindet, an und 'port' die Portnummer auf der `RCISimMuxd` läuft. `RCISimMuxd` ist eine Art Verzeichnisdienst für Simulatoren. Hier registriert `Robotsim` die Portnummer unter der ein simulierter Roboter zu finden ist über den entsprechenden Simulatornamen. `RCISimMuxd` wartet dann auf eingehende TCP/IP Verbindungen von RCCL-Programmen, die eine Anfrage für einem bestimmten Simulatornamen stellen und leitet sie an den entsprechenden Port weiter. Der Simulatorname kann explizit in der Datei *.rciparams* angegeben werden, wobei die optionalen Teile, bei nicht Vorhandensein, mit Standardwerten besetzt werden.

Standardwerte: hostname = '127.0.0.1', port = '-1' ('-1' bedeutet, dass der Standardport von `RCISimMuxd` benutzt wird, so wie er in /etc/services registriert ist.)

[2] Transmission Control Protocol/Internet Protocol

[3] Die Datei *.rciparams* befindet sich im selben Verzeichnis wie *robotIO.cfg*. Wie sie genau aufgebaut ist, wird in der Handbuchseite RCCL_params (man 5 RCCL_params) genau erklärt.

Beispiel: 'left:0@127.0.0.1'

Die mit RCCL mitgelieferten Hilfsprogramme wie `movej` bieten oftmals einen Kommandozeilenparameter '-sim' an, um komfortabel zur Laufzeit in den Simulationsmodus zu wechseln.

Wenn also ein RCCL-Programm einen oder mehrere Roboter steuert, funktioniert das im Simulationsmodus in gleicher Weise, nur dass RCCL bei Programmbeginn den Roboterbeschreibungen jeweils anstelle der Roboterklasse des entsprechenden Gerätetreibers den Simulatortreiber zuordnet.

Beim Initialisieren dieses virtuellen Simulatorgerätes wird zuerst getreu den Simulationsoptionen zu `RCISimMuxd` Verbindung aufgenommen und die Ports der zu jedem Roboter gehörenden Simulatoren werden erfragt. `RCISimMuxd` markiert hiernach die Portnummer als nicht länger verfügbar. Danach wird auf den erhaltenen Ports und dem in den Optionen festgelegten Rechnernamen/-adresse mit `Robotsim` Verbindung aufgenommen. Wenn dann Befehle ausgeführt werden sollen, werden diese anstatt an ein reales Gerät an `Robotsim` weitergeleitet. (siehe Abbildung 4.1)

Kurzzusammenfassung:

- RCCL ist in mehrere Schichten aufgeteilt: Roboterbeschreibung ⇒ Kommunikationsfunktionen ⇒ Ein-/Ausgabegerät(etreiber) ⇒ Roboter
- Der Simulator setzt an der Gerätetreiberschicht an.
- Das Simulatorgerät leitet die Befehle/Daten an `Robotsim` weiter.

Im nächsten Abschnitt wird unter anderem beschrieben, was `Robotsim` mit den erhaltenen Befehlen/Daten macht.

4.2 Arbeitsweise von Robotsim

Wenn `Robotsim` gestartet wird, erwartet es die Namen der zu simulierenden Roboter als Argumente auf der Kommandozeile. Über diese Namen baut es sich dann die Roboterbeschreibungen, die alle relevanten Daten zu einem Roboter speichern, ähnlich zu denen von RCCL auf (siehe vorherigen Abschnitt). Danach werden die zu simulierenden Roboter initialisiert und bei `RCISimMuxd` unter dem Namen „<robotname>:<USER>“ registriert. Falls `RCISimMuxd` noch nicht läuft, wird es von `Robotsim` gestartet. Die Portnummer ist systemweit durch die Datei */etc/services* bekannt. Danach fängt `Robotsim` selbst an, über den `select()` Systemaufruf (system call), auf eingehende Verbindungen zu warten (*pollForEvents()*). Die verschiedenen Netzwerkverbindungen sind in Abbildung 4.1 nocheinmal übersichtlich dargestellt. Bis auf die Verbindung zu `RCISimMuxd` (und später zur grafischen Ausgabe) werden alle verwendeten Ports zufällig vom Betriebssystem zugewiesen.

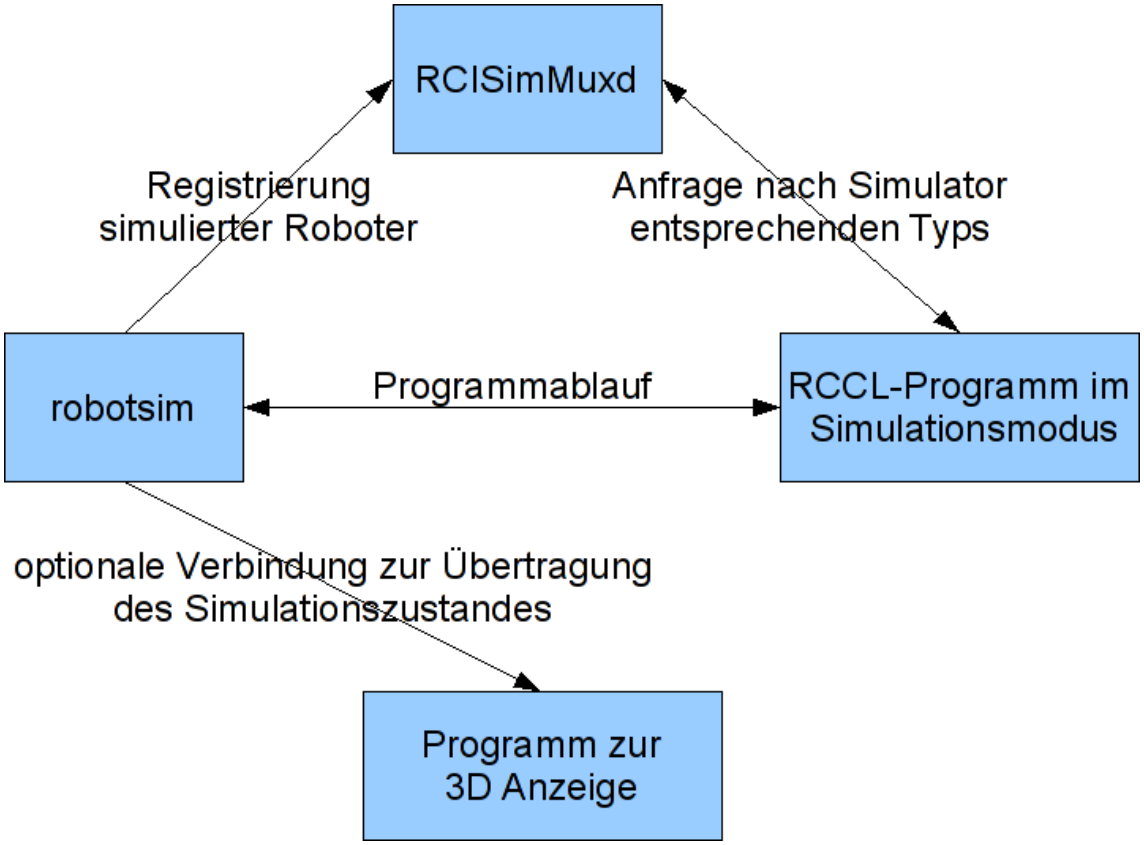


Abbildung 4.1: Diagramm der Netzwerkkommunikation von `Robotsim`, wobei schon die geplante Verbindung zur Grafikausgabe eingezeichnet ist. Da es möglich ist mehrere Roboter gleichzeitig zu simulieren, können mehrere Programmabläufe parallel stattfinden.

Sobald eine neue eingehende Verbindung ansteht, wird eine entsprechende Funktion aufgerufen, die die Verbindung akzeptiert und eine weitere Verbindung für band-externe Signale ausmacht. Außerdem wird der aktuelle Roboterzustand übergeben und der Roboter als aktiv markiert. Dann läuft der normale Simulationszyklus (*doCycle()*) ab, welcher wie folgt aussieht:

1. Einlesen der Daten/Befehle von allen Robotern. (*robotCycleBack()*)
2. Simulationsfunktionen ausführen.
3. Ausgabedaten (neuer Zustand) an alle Roboter schreiben. (*robotCycleFront()*)

Nach diesem Zyklus wird dann wieder auf eingehende Verbindungen gewartet und die Roboter, von denen Daten anliegen, werden als aktiv markiert. Sobald es Zeit für den nächsten Zyklus ist und alle Roboter aktiv sind, beginnt der Zyklus von vorne.

Falls ein Signal über den band-externen Kanal eingeht oder das Lesen/Schreiben von Daten vom/zum Roboter fehlschlägt, wird die Verbindung abgebrochen, der entsprechende Roboter wieder bei `RCISimMuxd` registriert und auf eingehende Verbindungen gewartet.

Hierbei wird der Roboterstatus nicht zurückgesetzt und bleibt über mehrere Programmabläufe konsistent.

Wie im vorhergehenden Kapitel beschrieben, schickt der Simulatortreiber die Daten genau so an den Simulator bzw. empfängt sie von dort, wie sie normalerweise an das tatsächliche Gerät weitergeleitet werden. `Robotsim` muss also dem simulierten Robotertyp entsprechend auf die Daten reagieren.

Bisher unterstützt `Robotsim` nur die PUMA Modelle über das sogenannte `moper` Programm, welches normalerweise auf dem Roboter-Controller für den PUMA läuft und in `Robotsim` teilweise reimplementiert ist.

Kurzzusammenfassung (siehe Abbildung 4.1)**:**

- `Robotsim` registriert zu simulierende Roboter bei `RCISimMuxd` und wartet auf eingehende Verbindungen.
- Wenn sich ein RCCL-Programm verbindet, startet ein Simulationsablauf mit Zyklen der Form: 1. Daten einlesen 2. Simulationsfunktionen ausführen 3. Daten ausgeben
- Nach Ende eines Ablaufs wird die Verbindung mit dem RCCL-Programm beendet und der entsprechende Roboter wieder bei `RCISimMuxd` registriert.

Zur Implementierung eines Simulators für den PA10-6C Roboterarm bieten sich generell zwei Wege an, die im Folgenden beschrieben werden sollen. Die eventuell notwendige Anpassung der Simulatorunterstützung seitens RCCL ist dabei invariant.

4.3 Neuentwicklung des Simulators

Die erste Möglichkeit besteht darin `Robotsim` mit ähnlicher Funktionalität neu zu schreiben.

Als Programmiersprache wäre dabei C++ eine gute Wahl, da die Integration in ein C-Projekt einfach möglich ist und unterstützte Programmierparadigmen, wie Objektorientierung, generell einfacher durchschaubaren Quelltext liefern sollten, als es mit C möglich ist.

Bei diesem Ansatz sollten auch weiterhin dieselben Mechanismen von RCCL zur Simulationsunterstützung benutzt werden, da es wenig Sinn macht, mehrere solcher Mechanismen parallel zu unterstützen und zu pflegen. Dass bedeutet zum Beispiel, weiterhin `RCISimMuxd` zur Verbindungsaufnahme zu benutzen. Prinzipiell ist dadurch auch jede Programmiersprache vorstellbar, da die Kommunikation mit dem Simulatortreiber schön gekapselt per TCP/IP funktioniert. Hierbei darf man allerdings nicht außer Acht lassen, dass es ein nicht unerheblicher Aufwand ist, ein Roboterbeschreibungssystem mit ähnlicher Mächtigkeit wie bei RCCL, zu implementieren. Alternativ sind dann wieder die Integrationsmöglichkeiten mit einem C-Projekt zu betrachten, wobei C++, aufgrund der Kompatibilität zu C, gegenüber anderen Programmiersprachen klar im Vorteil ist.

Im Rahmen dieser Arbeit würden erstmal nur PA10-6C oder auch allgemein PA10 Modelle unterstützt werden können. Für PUMA Modelle kann weiterhin `Robotsim` benutzt werden. Der Nachteil dabei ist dann natürlich, dass die beiden Robotertypen nicht in einem Simulator kombiniert werden können, was aber in der Praxis wahrscheinlich selten relevant ist. Ein weiterer wichtiger Nachteil ist der nicht zu unterschätzende Aufwand zur Reimplementierung (und zum gründlichen Testen!) des gesamten "Rahmenwerks„ um den Simulator herum. Das Simulatorrahmenwerk beinhaltet zum Beispiel: Kommandozeilenoptionen von `Robotsim`, Konfigurationsdateiunterstützung, Loggingausgaben, interaktive Kommandozeile im Simulator zum schrittweisen Durchspielen von Programmabläufen, usw. Alle diese Hilfs-/ Debug-/ Komfortfunktionen sind in `Robotsim` integriert und funktionieren.

Dafür würde man allerdings bei Reimplementierung moderneren, objektorientierten Quelltext erhalten, der im Allgemeinen deutlich lesbarer und einfacher nachzuvollziehen sein sollte. Zum Beispiel könnten die verschiedenen Roboterklassen, die der Simulator unterstützt, ideal auf Klassen im programmiertechnischen Sinne abgebildet werden. Das würde heißen, dass nicht in allen Funktionen nach der Roboterklasse unterschieden werden muss, sondern für jeden zu simulierenden Roboter ein Objekt von der entsprechenden Klasse erzeugt wird. Diese Klassen enthalten dann die für die jeweilige Roboterklasse entsprechend angepassten Funktionen. Des Weiteren können sehr ähnliche Robotertypen, wie PA10 und PA10-6C Funktionalität von einander oder einer abstrakten Oberklasse erben.

Verbesserte Lesbarkeit des Quelltextes bedeutet vor allem leichtere Pflege und gerin-

gere Fehlerrate, sowie einfachere Erweiterbarkeit. Dies sind gerade für Programmierbibliotheken, die die Grundlage für viele weitere Programme bilden, sehr wichtige Eigenschaften. Es ist aber auch zu bedenken, daß ein Großteil des Quelltextes von RCCL und somit `Robotsim` schon seit Jahren nicht veränderlich wurde und somit schon sehr lange und sicherlich auf den verschiedensten Systemkonfigurationen getestet wurde. Bei einer Reimplementierung wäre es naiv zu erwarten, dass keine neuen Fehler eingebaut werden, so dass zumindest anfänglich die Fehlerrate im Vergleich zur jetzigen Implementierung zunächst schlecht abschneiden würde.

Vorteile der Neuentwicklung:

- Erhalt von modernem, objektorientiertem Quelltext, der zu einer besserer Les-, Wart- und Erweiterbarkeit führt
- Relativ geringer Integrationsaufwand, wenn C++ als Programmiersprache gewählt wird

Nachteile der Neuentwicklung:

- Neuentwicklung bestehender, funktionierender Konzepte/Funktionalität, die für alle Robotertypen gleich sind
- (Vorerst) Keine Unterstützung für die in `Robotsim` zur Zeit funktionierenden PUMA Modelle

Als Fazit ergibt sich, dass das Verhältnis von Aufwand zu Nutzen einer Reimplementierung wohl zu gering ist.

4.4 Erweiterung von Robotsim

Die zweite Variante der Implementierung ist die Erweiterung von `Robotsim` um die Unterstützung von PA10-6C Robotermodellen.

Es ist zu erwarten, dass dabei weniger Implementierungsaufwand erforderlich ist, wenn das Design von `Robotsim` die Berücksichtigung der Besonderheiten der PA10 Reihe im Vergleich zu den traditionellen PUMA Modellen erlaubt. Zu den Besonderheiten gehört zum Beispiel die Steuerung per Gelenkwinkelgeschwindigkeiten anstelle von Positionen in kartesischen Koordinaten. Im Allgemeinen kann Robotern die anzusteuernde Position direkt in einem geeigneten Koordinatensystem übergeben werden. Dabei muss sichergestellt werden, dass diese Position bei gegebener Masse, Geschwindigkeit und Beschleunigungsvermögen des Manipulators auch in einem Steuerungszyklus erreichbar ist. Der Vorteil ist dann, dass der Manipulator/Controller genau weiß, “wo es hin geht„. Dadurch kann er selbst dafür sorgen, dass etwaige Abweichungen durch Steuerungsungenauigkeiten, äußere Einflüsse oder ähnliches, ausgeglichen werden. Bei den PA10 Modellen ist dies nicht möglich. Stattdessen muss RCCL selbst dafür sorgen, dass die angestrebte Position auch tatsächlich erreicht wird. Das wird durch eine zweite Besonderheit erschwert,

nämlich dass die Rücklieferung der aktuellen Position bzw. Winkelstellungen erst nachdem ein neuer Befehl gesendet wurde erfolgt.

Wie schon im vorhergehenden Abschnitt angedeutet, kann bei der Erweiterung von `Robotsim` der Großteil des bestehenden, langjährig getesteten Quelltext beibehalten werden. Dadurch bleibt zum Beispiel die bereits etablierte Benutzerschnittstelle über die Kommandozeilenoptionen von `Robotsim` weitgehend gleich. Auch besteht die Möglichkeit, dass schon unterstützte Robotertypen von der modernisierten Grafik profitieren, wenn von ihnen bereits ein 3D Modell vorhanden ist.

Vorteile der Erweiterung von `Robotsim`:

- Erfordert per Definition keine Neuentwicklung bestehender Funktionalität
- die bestehende 3D Grafik der PUMA Modelle (siehe Abbildung 2.3) kann mit neuen 3D Modellen modernisiert werden

Nachteile der Erweiterung von `Robotsim`:

- bestehendes Design von `Robotsim` könnte eine Erweiterung schwierig machen

Nach Abwägung der Vor- und Nachteile, sowie einer weiteren Untersuchung der invarianten Problemteile, wird im nächsten Kapitel der bis zum Schluss durchgeführte Implementierungsansatz dargestellt, wobei unter anderem die neue Art der Grafikausgabe erläutert wird.

Implementierung

5

Im Folgenden wird der Einfachheit halber oft der PA10 Manipulator anstelle des PA10-6C Manipulators genannt, da beide sehr ähnliche Treiberimplementierungen besitzen und über dieselbe Schnittstelle (ARCNet) auf das Gerät zugreifen.

Wie sich herausstellte, ist der kompliziertere Teil der Implementierung die notwendige Anpassung der PA10-spezifischen Kommunikationsfunktionen, damit die Simulatorunterstützung von RCCL korrekt funktioniert. Parallel dazu muss der PA10 Treiber in gleichem Maße verändert werden, damit der reale Roboter weiterhin ordnungsgemäß angesteuert wird. Hierbei stellt sich die Frage der Reimplementierung der RCCL-Seite nicht, da das darauf hinauslaufen würde, große Teile von RCCL neu oder umzuschreiben. Dabei wäre vor allem die Gerätetreiberschicht betroffen, wo es geradezu unmöglich ist, Korrektheit auf den verschiedenen unterstützten Plattformen zu garantieren oder zu testen. Es muss also versucht werden, das bestehende Zusammenspiel von Kommunikationsfunktionen und Treiber auf sinnvolle Art und Weise zu verändern.

Der PA10 Treiber kapselt die im vorhergehenden Abschnitt beschriebenen Eigenheiten der PA10 Steuerung von den Kommunikationsfunktionen ab, so dass der Simulator diese nicht berücksichtigen muss. Nach näherer Untersuchung des Quelltextes von `Robotsim` ließ sich erkennen, dass das bestehende Design zwar nicht ideal für eine Erweiterung auf andere Roboterklassen ausgelegt war, eine Anpassung aber durchaus möglich ist.

Aufgrund dieser Erkenntnisse wurde dann die Variante der Erweiterung von `Robotsim` gewählt und nicht die der Neuimplementierung. Die weiteren Teilschritte der Implementierung werden in den folgenden Abschnitten näher erläutert.

5.1 Anpassung der PA10 Kommunikationsfunktionen

Der PA10 Treiber (*PA10Read/Write()*) tauscht die sogenannte HOW-Struktur mit den Roboterkommunikationsfunktionen (*pa10_input/output()*) aus. Die HOW-Struktur ist ein allgemeiner Container für alle möglichen Informationen, die den Roboter betreffen. Sie enthält zum Beispiel Laufzeitinformationen über den Robo-

terzustand, wie Position, Geschwindigkeit und Drehmoment der Gelenke des Manipulators. Auch beliebige Daten von eventuell vorhandenen Sensoren bzw. von der Art der Sensoren, die überhaupt verfügbar sind, wird dort gespeichert und in jedem RCCL-Zyklus von einem Roboter-Controller oder Gerätetreiber aktualisiert. Außerdem werden in der HOW-Struktur die Befehle an den Roboter zwischengespeichert. All diese Informationen können vom Steuerungsprogramm zu jeder Zeit ausgelesen werden, um zum Beispiel die nächsten Befehle für den Roboter zu berechnen. Durch die generische Natur der HOW-Struktur müssen nicht alle ihre Parameter für einen bestimmten Robotertyp implementiert sein.

Einige Teile der HOW-Struktur sind hauptsächlich für die Ausführung der Kommunikationsfunktionen vorgesehen. Der PA10 Treiber benutzt aber nur die Winkel der einzelnen Gelenke, aus denen er letztendlich die Befehle an den Roboter berechnet. Daher konnten die Funktionen so geändert werden, dass sie nur eine Reihe von Gelenkwinkeln und nicht die gesamte HOW-Struktur übergeben. Dies macht die Anpassung der Simulatorfunktionen sehr viel einfacher, da sie an vielen Stellen sparsam kalkulierte Puffergrößen verwenden. Bei Austausch der gesamten HOW-Struktur würden diese leicht überschritten werden und es würde nicht die gesamte Struktur übertragen werden. Des Weiteren garantiert diese Änderung im Hinblick auf die Übertragung der Daten über ein Netzwerk eine effizientere Nutzung der Bandbreite.

Nach Anpassung der Kommunikations- und Treiberschicht hinsichtlich der Anforderungen des Simulatortreibers können jetzt theoretisch[1] Befehle von RCCL empfangen und Zustandsdaten zurückgesendet werden. Als nächsten Schritt gilt es `Robotsim` korrekt auf die PA10-spezifischen Daten reagieren zu lassen.

5.2 Erweiterung von `Robotsim` um die PA10 Unterstützung

Zur Implementierung der PA10 Unterstützung wurden im Quelltext von `Robotsim` zunächst die PUMA/moper-spezifischen Funktionen identifiziert, um sie auf notwendige Anpassungen für den PA10 hin zu untersuchen. Der Großteil dieser Funktionen befindet sich in der Datei *rbtContrl.c* (in src/robotsim/), wo die eingehenden Daten dekodiert und ausgehende Daten kodiert werden.

Die beiden wichtigsten Funktionen sind *rbtGather()*, welche aktuelle Roboterinformationen zusammenträgt und sie in einen Puffer schreibt und *rbtCommands()*, welche die eingehenden Befehle für den Roboter dekodiert und ausführt.

Die Funktionen wurden so modifiziert, dass im Allgemeinen im ersten Teil nach der Roboterklasse gefragt wird und je nachdem entweder ein PUMA- oder PA10-spezifischer Teil ausgeführt wird. Es ist so möglich, PA10-spezifische Befehle in eini-

[1] Noch stürzt `Robotsim` bei Angabe eines PA10 Roboters ab, da einige für die PA10 Reihe nicht relevante Funktionen, die nicht implementiert sind (Nullpointer), aufgerufen werden.

ge Funktionen einzubauen, bzw. die Ausführung von für den PA10 unsinnigen oder nicht unterstützten Funktionen zu verhindern, in dem nach der korrespondierenden Roboterklasse (RobotDesc.robotClass) unterschieden wird. Diese steht normalerweise überall zur Verfügung. Nach dem gleichen Prinzip wurden die anderen Funktionen zum Initialisieren, Einrichten, Starten des Roboters usw. falls notwendig individuell adaptiert.

Mit diesen Modifikationen steht dann die grundsätzliche Funktionsweise von `Robotsim` auch für PA10 Modelle zur Verfügung. Das heisst `Robotsim` kann mit Hilfe des "-ng„ Parameters, der die Grafikausgabe unterdrückt, bei Angabe eines bestimmten PA10 Roboters erfolgreich gestartet werden. (`robotsim -ng left`; Eine Einführung in die Programmoptionen von `Robotsim` erfolgt am Anfang von Kapitel 6, bzw. ist in der Handbuch-Seite zu finden.) Durch die schon vorgenommene Anpassung der Kommunikationsschicht funktioniert auch die Verbindung zum Simulationstreiber von RCCL, so dass nach Start des Simulators beliebige RCCL-Programme auf diesem Ablaufen können.

Nun fehlt im Prinzip nur die grafische Visualisierung des Simulationszustandes, der bisher ausschließlich in Form von Zahlenreihen ausgegeben wird.

5.3 Grafische Ausgabe

Zum Zweck der grafischen Ausgabe wurde `Robotsim` so erweitert, dass es auf einem bestimmten Port die Zustandsinformationen der simulierten Roboter anbietet. Wenn ein Client zu diesem Port Verbindung aufnimmt, werden unter Einsatz eines einfach strukturierten Textprotokolls die Zustandsinformationen übermittelt. Dies geschieht einmal direkt nach der Verbindungsaufnahme und außerdem in jedem Simulationszyklus bei laufender Steuerung. (Verbindung kann jederzeit aufgenommen werden, also auch wenn aktuell kein RCCL-Programm den Simulator benutzt.)

Zur grafischen Anzeige wurde die `ClientGUI` des Telemanipulations-Client verwendet, der von Jan Bruder im Laufe seiner Diplomarbeit (Bru09) entwickelt wurde. Hierbei handelt es sich um ein Java-Programm, welches ein 3D Modell der TASER-Plattform anzeigt und die Steuerung des PA10-6C Arms (inklusive der Barretthand) mit einem speziellen Eingabegerät ermöglicht. Hierbei wird schön die Flexibilität des verteilten Konzepts des Simulators deutlich. Durch Auslagerung der Grafikdarstellung über die Netzwerkschnittstelle ist eine von `Robotsim` weitgehend unabhängige Implementierung möglich. Dies hat den Vorteil, dass zum Beispiel eine völlig andere Programmiersprache benutzt und/oder schon vorhandene Funktionalität, wie der Telemanipulations-Client, relativ leicht angepasst und mit `Robotsim` verwendet werden kann.

Zur Kommunikation zwischen `ClientGUI` und `Robotsim` wurde ein Java-Programm (`SimulatorClient`) geschrieben, welches Verbindung mit `Robotsim` aufnimmt und die von dort gesendeten Gelenkwinkel mit Hilfe der `ClientGUI` anzeigt.

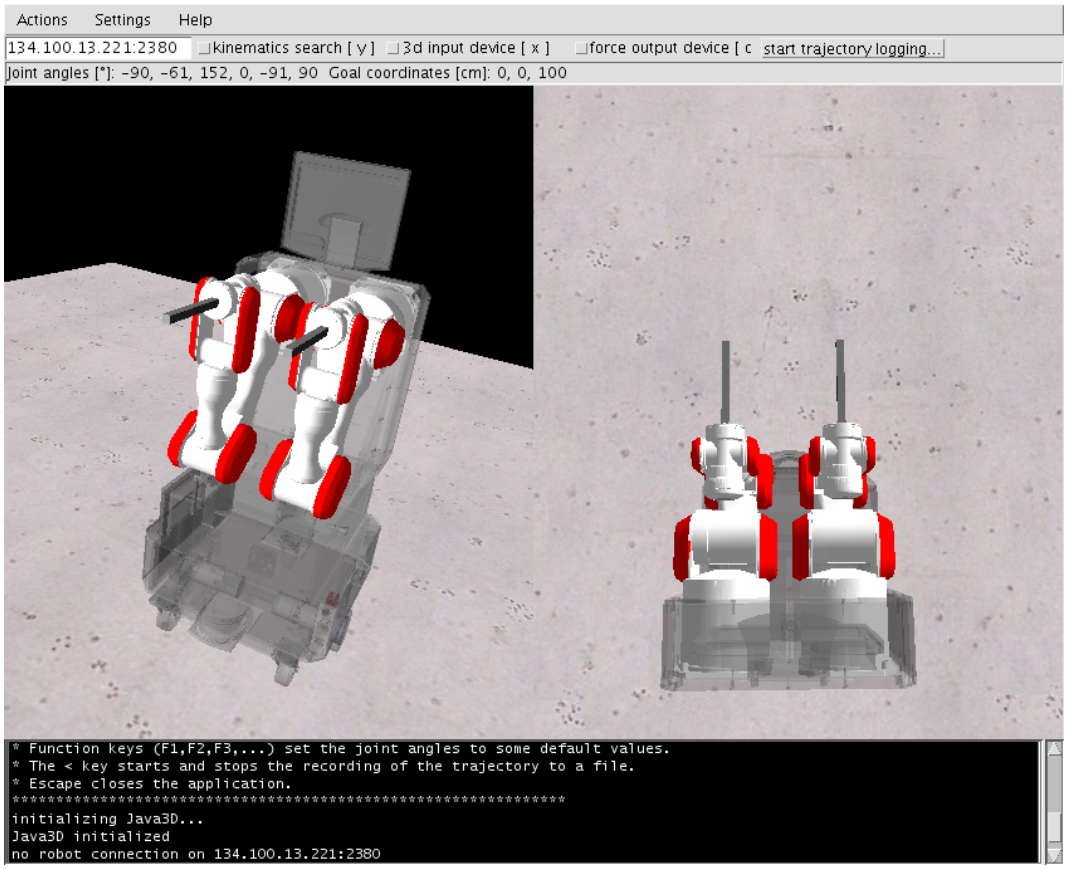


Abbildung 5.1: Darstellung der mobilen Plattform mit der `ClientGUI`, wobei hier schon der rechte Arm angebracht ist. Beide Manipulatoren sind in der sogenannten “rcclpark„ Stellung.

Das benutzte Kommunikationsprotokoll wird im Klartextformat übertragen und sieht wie folgt aus:

`<Robotername> <Roboterklasse> <Gelenkwinkel 1> <Gelenkwinkel 2> ...`

Zeichenketten zwischen spitzen Klammern (`<>`) sind durch die eigentlichen Werte zu ersetzen. Das Element '`...`' deutet beliebige Wiederholungen des vorherigen Elements an. Winkelwerte sind im Bogenmaß (rad) angegeben. Die Anzahl der Winkelwerte hängt von der Roboterklasse ab.

Beispiel: `left 7 10 20 30 40 50 60` (7 ist die Nummer der PA10-6C Roboterklasse, siehe *src/include/robotClass.h*)

Der `SimulatorClient` erkennt an dem ersten Wort im Protokoll für welchen Roboter, bzw. hier für welchen Manipulator die Winkeldaten bestimmt sind und aktualisiert demzufolge die Winkel in der `ClientGUI`. Danach veranlasst er eine Aktualisierung der 3D-Anzeige, so dass der neue Zustand angezeigt wird.

Hierbei musste die `ClientGUI` noch um den rechten Arm erweitert werden. Da der rechte Arm vom exakt gleichen Typ, wie der Linke ist, kann der linke Arm kopiert und ohne etwaige Spiegelung per Translation an den "richtigen„ Platz verschoben werden. Richtig steht in Anführungszeichen, da die ursprünglich vorgesehene Anbaustelle im Hinblick auf Interaktion der Manipulatoren nicht unbedingt gut geeignet ist. In so einer Konfiguration überschneidet sich der von beiden Armen erreichbare Raum stark. Das erhöht die Chance von Kollisionen und macht eine Zusammenarbeit der Manipulatoren schwierig. Das ist der Grund, warum am TASER im Moment nur ein Arm montiert ist.

Im Simulator sind Kollisionen kein großes Problem und daher wurde der rechte Arm erstmal am ursprünglich vorgesehenen Ort platziert. Wie später im Kapitel 7 angedeutet wird, können nun mit Hilfe des Simulators relativ einfach günstigere Konfigurationen der Arme untersucht werden.

Experimentelle Auswertung

6

In diesem Kapitel wird die Benutzung von `Robotsim` mit seinen teilweise neuen Eigenschaften beschrieben und die implementierte Funktionalität des Simulators an einigen Beispielen getestet und vorgeführt.

6.1 Benutzung von `Robotsim`

In diesem Abschnitt wird zuerst die Benutzung von `Robotsim` anhand von Beispielen verdeutlicht. Die Gesamtheit von `Robotsim` unterstützter Parameter können aus dessen Handbuch-Seite (man page) entnommen werden. Dort wird auch die Konfigurationsdatei *.robotsim* erklärt, die aber hauptsächlich für Optionen bezüglich der alten, eingebauten Grafikdarstellung enthält, die im Rahmen dieser Arbeit nicht relevant ist. Die Handbuch-Seite wird mit RCCL ausgeliefert und die im Rahmen dieser Arbeit erweiterte RCCL Version enthält dann auch Beschreibungen der hinzugefügten Optionen. Diese neuen, sowie einige wichtige bestehende Optionen sollen hier der Vollständigkeit halber auch erklärt werden.

Mit Hilfe der Konfigurationsdatei *.rciparams* können globale Parameter für alle RCCL-Programme gesetzt werden. Hier kann zum Beispiel unabhängig von expliziter Unterstützung des jeweiligen Programms die Ausführung auf dem Simulator festgelegt werden. Dazu dient der “simulate„ Parameter, der wie folgt aufgebaut ist: `simulate=<simulatorname>[@<hostname>]['<'<port>'>']` Weitere Erklärungen zu den möglichen Parametern sind in der Handbuchseite RCCL_params[1] zu finden.

Wird ein Roboter vom Typ PA10 oder PA10-6 angegeben, wird automatisch der eingebaute Grafikmodus ausgeschaltet und stattdessen die Netzwerkschnittstelle aktiviert. Dies geschieht parallel zur etwaigen Simulation, so dass beim Simulieren von PA10 Modellen auf die Grafikausgabe verzichtet werden kann, indem einfach der im vorigen Abschnitt beschriebene `SimulatorClient` nicht startet wird. Allgemein kann die Grafikausgabe mit dem Parameter '`-ng`'ganz abgeschaltet werden.

Der Port zur Verbindung mit dem `SimulatorClient` kann mit dem Parameter '`-gp <port>`' oder '`--graphics-port <port>`' eingestellt werden. Standardmäßig wird

[1] man 5 RCCL_params

Port '5557' benutzt. Wenn Parameter mehrfach angegeben werden, gilt der zuletzt angegebene Wert.

Nach der Angabe eines Roboternamens kann man noch einige für diesen Roboter spezifische Parameter angeben. Diese gelten immer für den zuletzt auf der Kommandozeile angegebenen Roboter. Der Parameter '`-p <position>`' gibt eine bestimmte Startposition per Namen vor. Beliebige Positionen können in der Datei *<robotname>.pos* definiert werden, um sie später einfach mit Namen angeben zu können.

Beispielhafte Abfolge von Befehlen zum Starten der verschiedenen zur Simulation erforderlichen Programme:

- `Robotsim` mit linkem und rechtem Arm des TASER starten: "`robotsim left right`„
- `SimulatorClient` zur grafischen Anzeige starten: "`java SimulatorClient`„
- RCCL Programm(e) starten, z. B.: "`movej left to rcclpark`„

6.2 Vergleich mit dem realen Roboter

In diesem Abschnitt wird zur Demonstration beispielhaft eine bestimmte Bewegungsabfolge des PA10-6C auf dem Simulator und parallel dazu auf dem TASER vorgeführt.

In der Ausgangsposition dieser Bewegung befindet sich der Manipulator in der sogenannten "rcclpark„ Stellung. Diese wird im Allgemeinen eingenommen, wenn der Roboter nicht in Benutzung ist. Ziel der Bewegung ist die sogenannte Nullstellung bei der sich alle Gelenkwinkel des Roboterarms auf null Grad einstellen. Die Bewegung wurde von dem RCCL-eigenen Hilfsprogramm `movej`[2] über den Befehl "`movej left to zero`„ gesteuert. Mit dem Roboternamen "left„ wird der linke Arm des TASER spezifiziert und "zero„ ist der Name einer in der Datei *left.pos* vordeklarierten Stellung, die alle Gelenkwinkel auf null setzt. Das Wörtchen "to„ gibt an, dass es sich um eine absolute Position handelt im Gegensatz zu einer relativen Positionsänderung, die mittels "by„ angezeigt werden kann. Voraussetzung für die korrekte Ausführung der gewünschten Bewegung ist, dass der Manipulator, wie oben erwähnt, vor dem Befehl in der "rcclpark„ Stellung steht.

In den Abbildungen 6.1 bis 6.4 ist der Bewegungsablauf im Simulator illustriert. Die Grundstellung ist in Abbildung 5.1 vom vorherigen Kapitel zu sehen.

Derselbe Bewegungsablauf am TASER ist in den Abbildungen 6.5 bis 6.8 gezeigt. (allerdings aus einer anderen Perspektive)

[2] move – bewegen; j (joint) – Gelenk

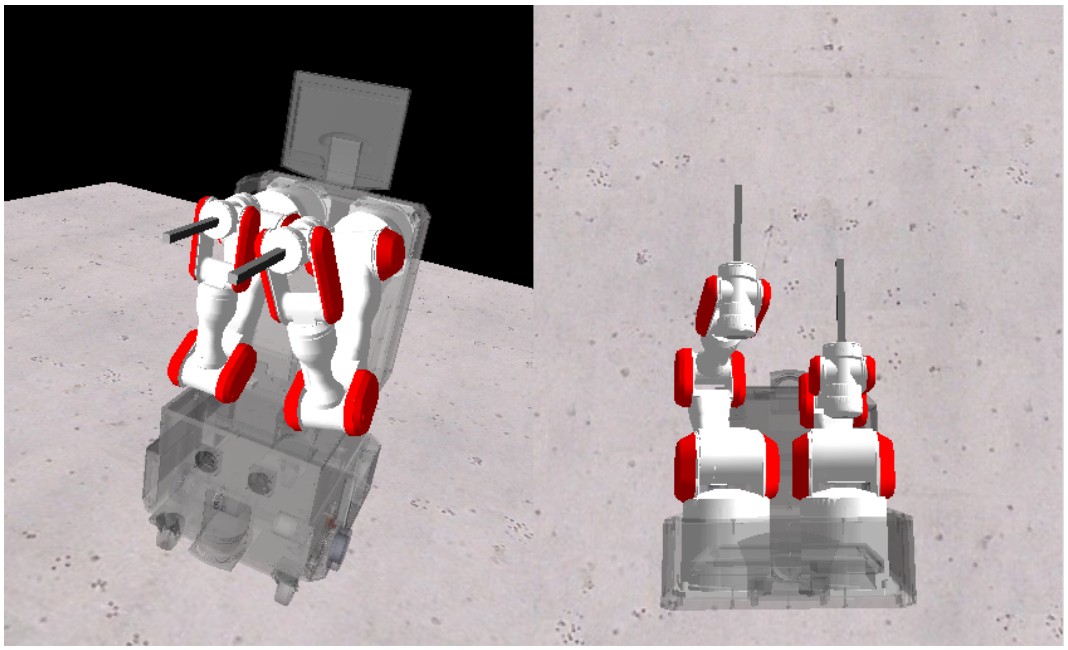


Abbildung 6.1: Beginn der Bewegung zur Nullstellung (alle Gelenkwinkel betragen null Grad)

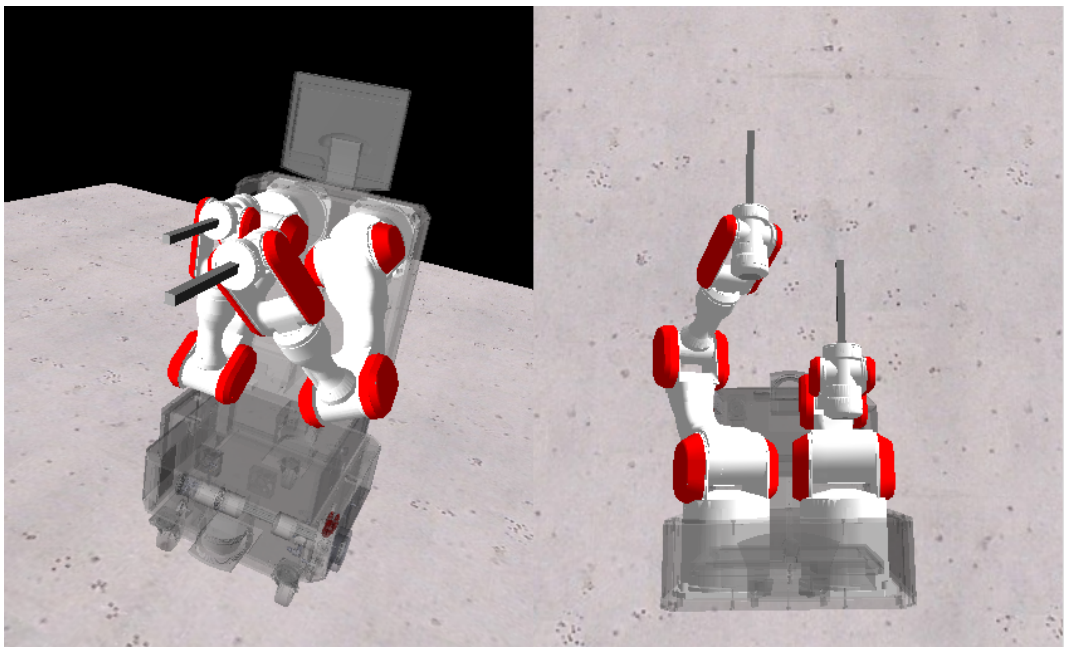


Abbildung 6.2: linker Arm des simulierten TASER auf dem Weg zur Nullstellung

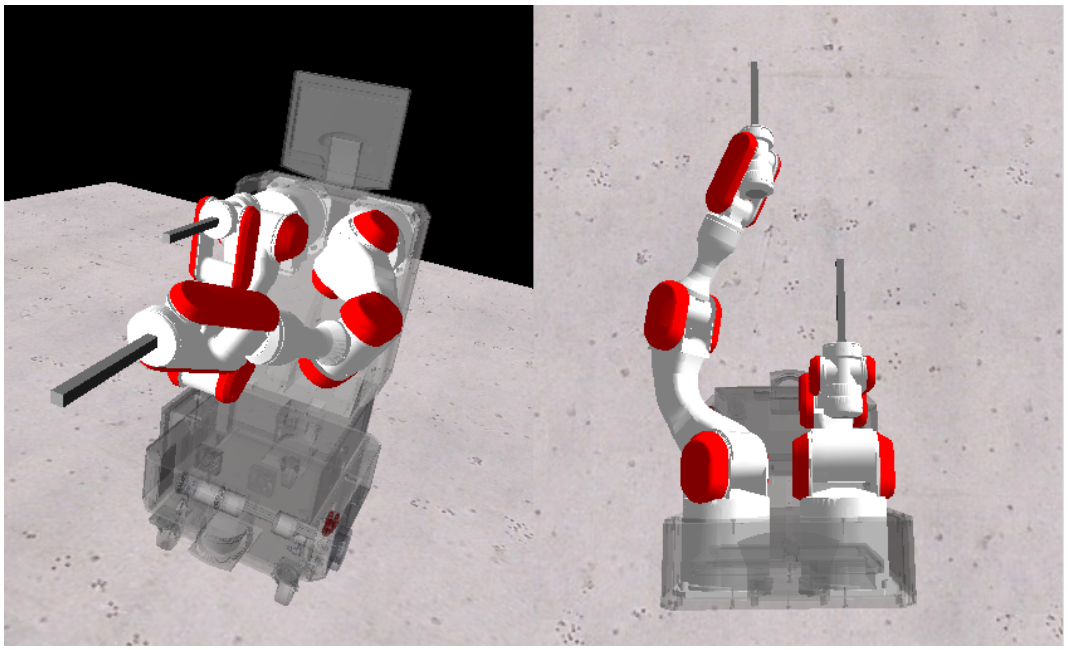


Abbildung 6.3: linker Arm des simulierten TASER kurz vor der Nullstellung

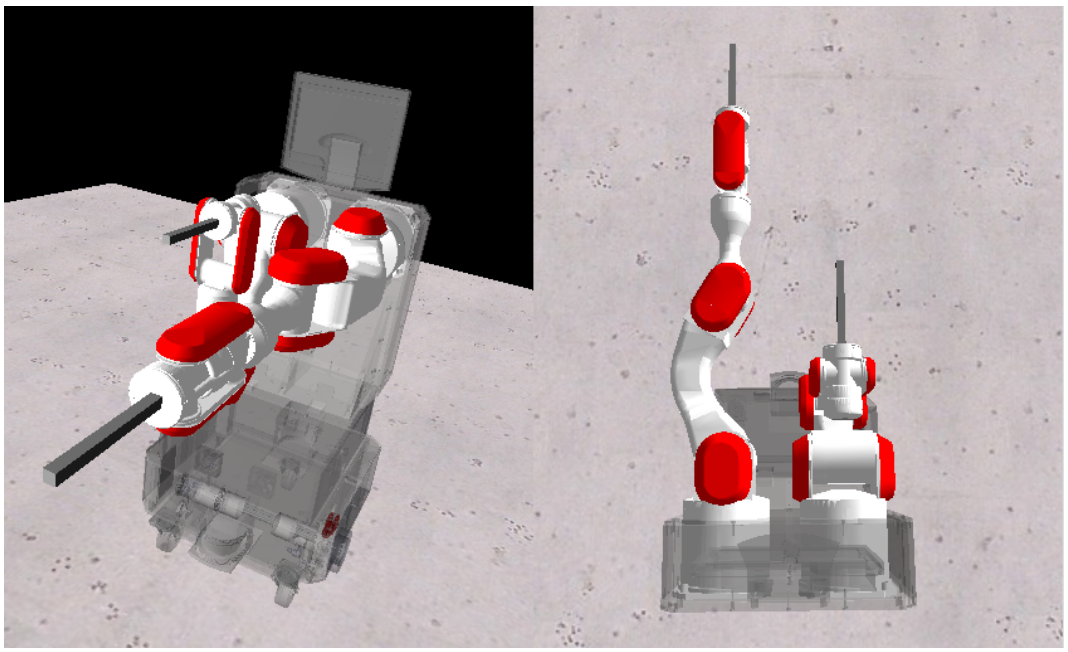


Abbildung 6.4: linker Arm des simulierten TASER in Nullstellung

Abbildung 6.5: TASER in Parkposition

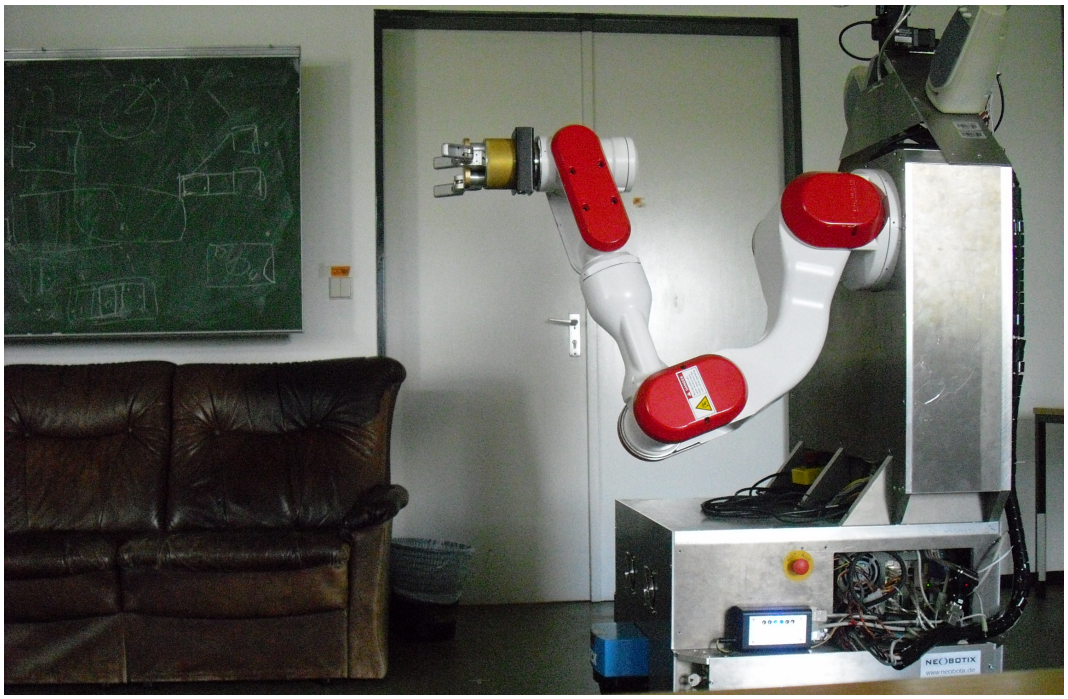


Abbildung 6.6: Beginn der Bewegung zur Nullstellung (alle Gelenkwinkel betragen null Grad)

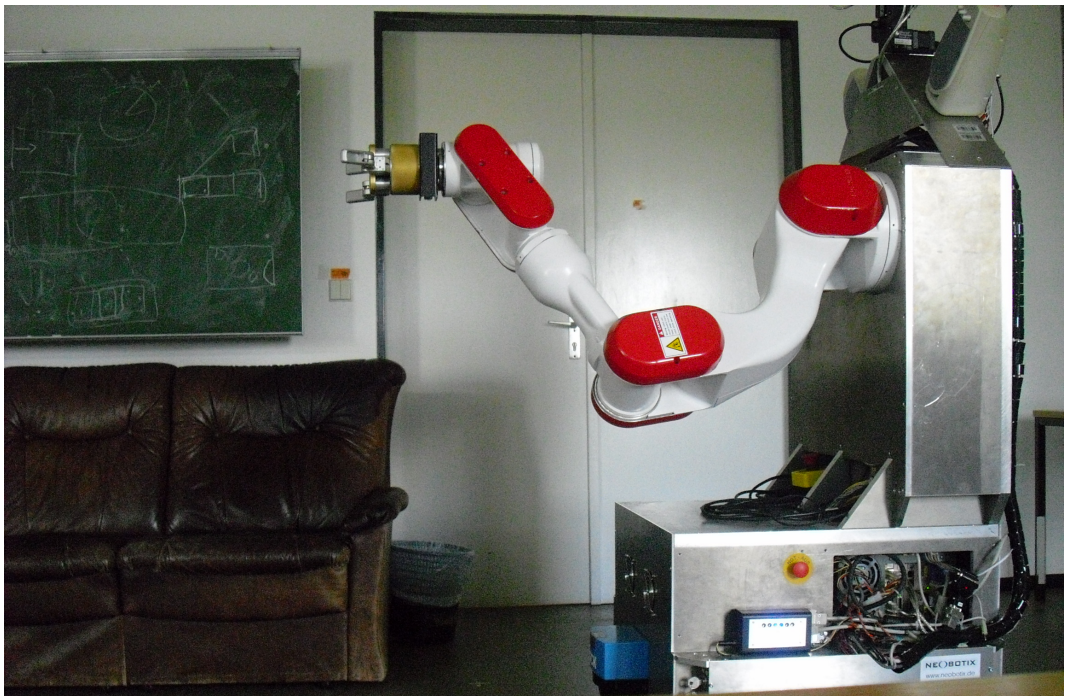


Abbildung 6.7: linker Arm des TASER auf dem Weg zur Nullstellung

Abbildung 6.8: linker Arm des TASER kurz vor der Nullstellung

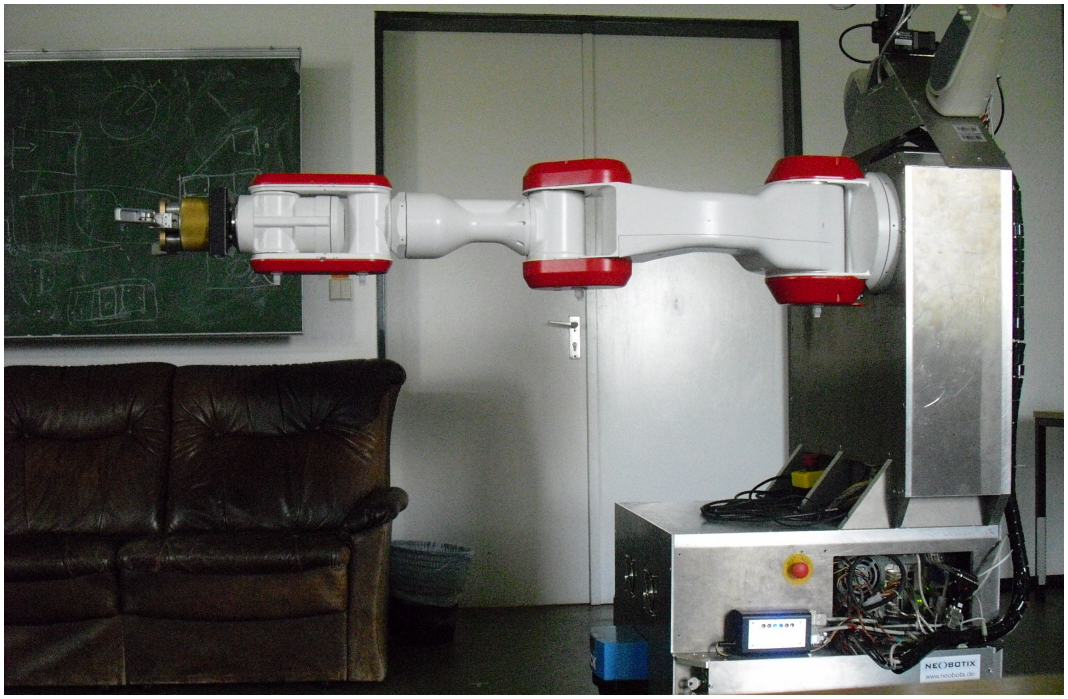


Abbildung 6.9: linker Arm des TASER in Nullstellung

Es sollte zu erkennen sein, dass in groben Zügen tatsächlich dieselbe Bewegung ausgeführt wird. Um genauer zu untersuchen, ob die Bewegung im Simulator der des realen Roboters gut entspricht, wurden die Gelenkwinkelwerte für beide Abläufe mit Zeitstempel protokolliert. Die Grafen in Abbildung 6.10 bis 6.15 zeigen die Bewegung der Gelenke 1 bis 6 jeweils im Vergleich zwischen Simulator (rot) und TASER (blau).

Zu erkennen ist, dass der zeitliche Verlauf nicht genau übereinstimmt. Der Simulator braucht im Durchschnitt 10,5 Millisekunden pro Zyklus, wohingegen der TASER ziemlich genau bei 10 Millisekunden liegt. Beide Bewegungen wurden mit derselben Intervaleinstellung (10 Millisekunden) von RCCL durchgeführt.[3] Allerdings wurden unterschiedliche Zeitplanungsmethoden (scheduling) verwendet. Beim Simulieren wird das Zeitverhalten über das Unix Systemsignal SIGALRM gesteuert. Für den TASER wurde dagegen die sogenannte “ON_TRIGGER„ Methode benutzt. Der wesentliche Unterschied ist, dass die Roboter den Trajektoriengenerator beim “ON_TRIGGER„ Verfahren selbst aufwecken. Dadurch verschwendet der Generator beim Warten auf die Roboter keine Zeit. Die “ON_TRIGGER„ Methode kann offensichtlich nicht auf dem Simulator eingesetzt werden.

Dieser Unterschied in den Zeitplanungsmethoden erklärt sehr wahrscheinlich das abweichende Zeitverhalten des Simulators im Vergleich zum tatsächlichen Gerät. Der

[3]Dazu gibt es in RCCL eine Datei für generelle Parameter: *.rciparams* – siehe “man 5 RCCL_params„

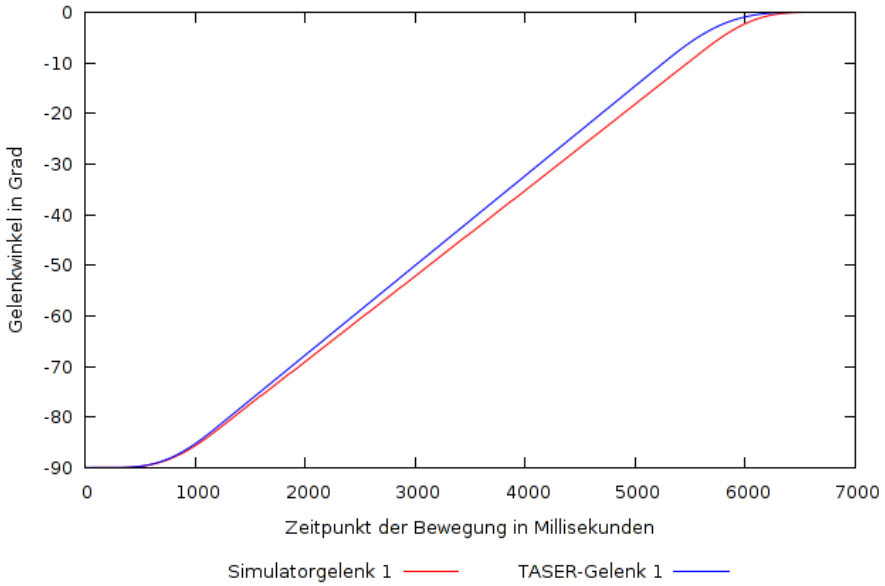


Abbildung 6.10: Vergleich des Bewegungsablaufs des ersten Gelenks zwischen Simulator und TASER

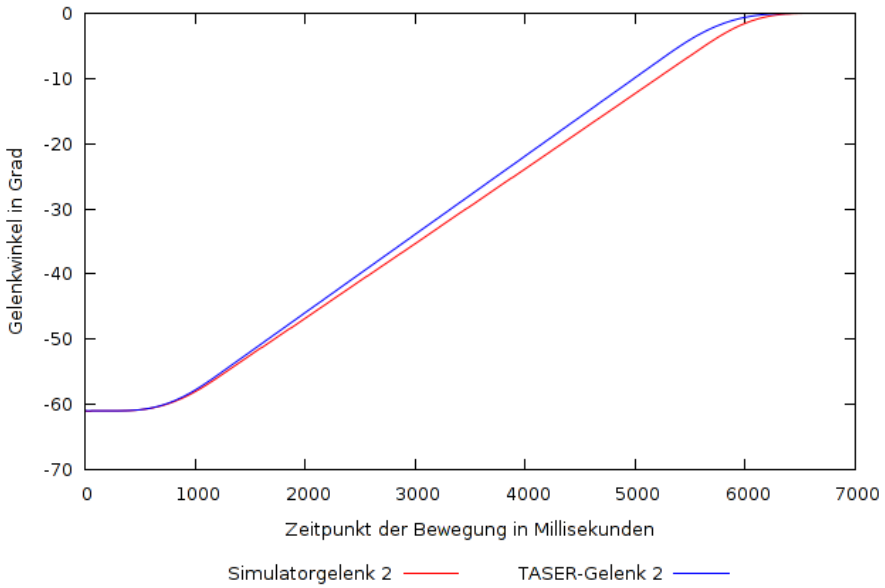


Abbildung 6.11: Vergleich des Bewegungsablaufs des zweiten Gelenks zwischen Simulator und TASER

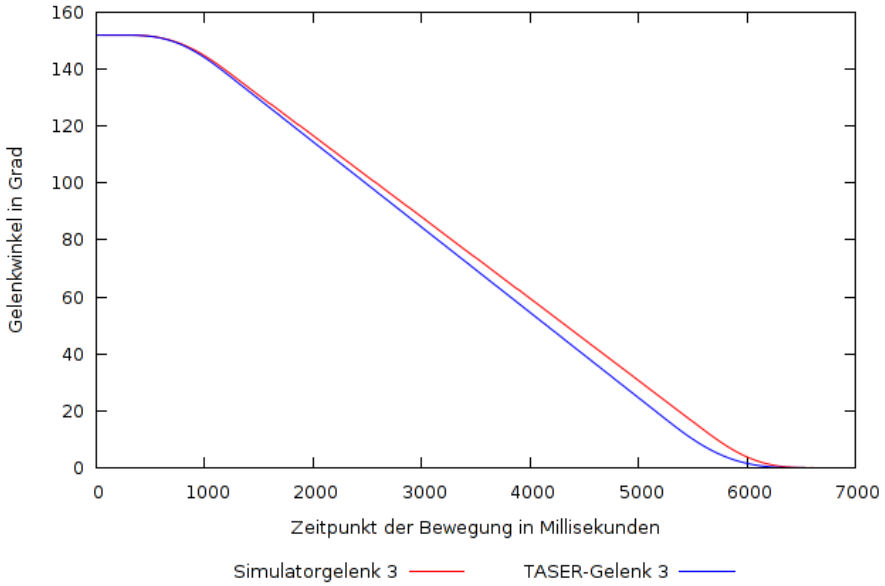


Abbildung 6.12: Vergleich des Bewegungsablaufs des dritten Gelenks zwischen Simulator und TASER

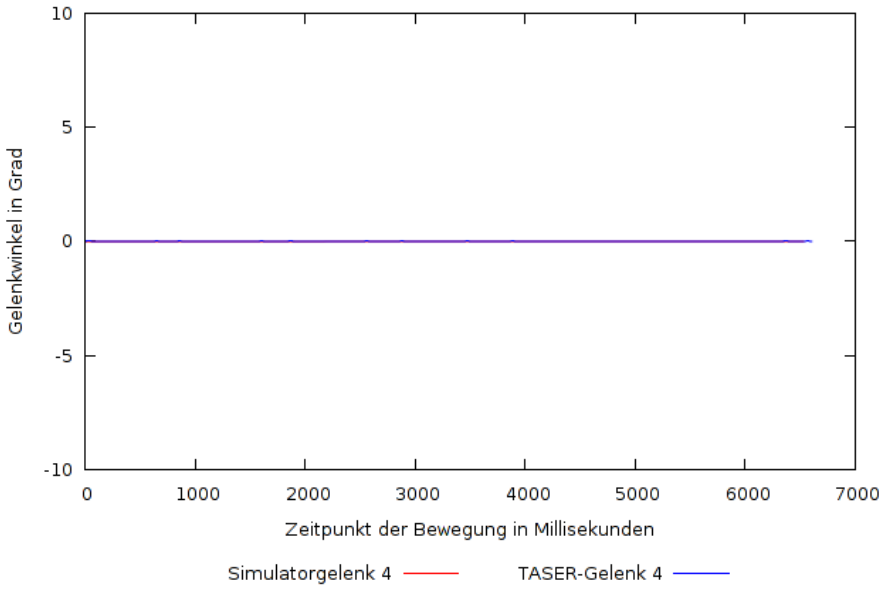


Abbildung 6.13: Vergleich des Bewegungsablaufs des vierten Gelenks zwischen Simulator und TASER

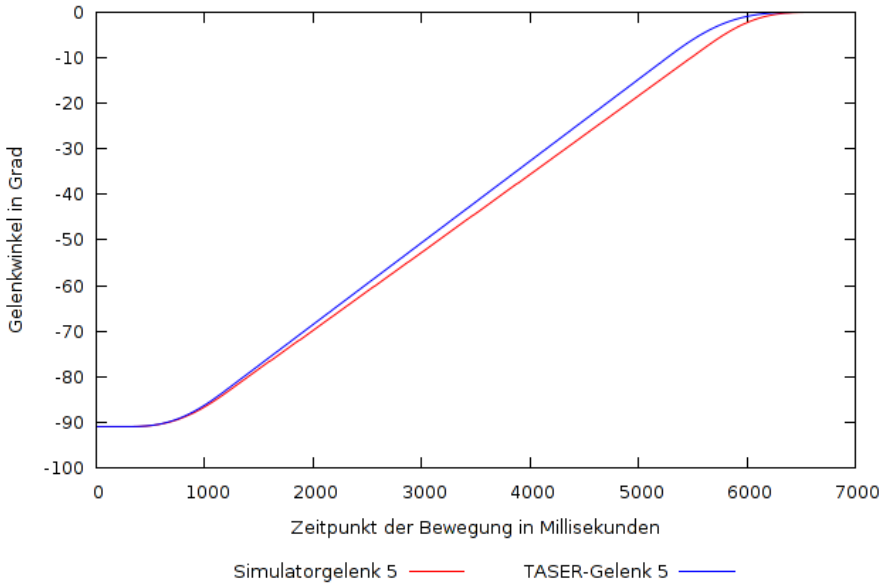


Abbildung 6.14: Vergleich des Bewegungsablaufs des fünften Gelenks zwischen Simulator und TASER

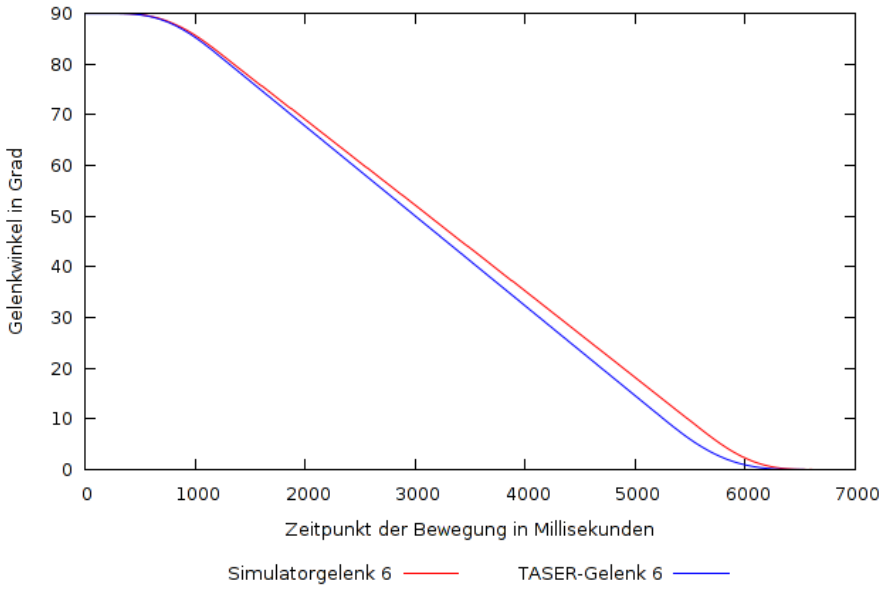


Abbildung 6.15: Vergleich des Bewegungsablaufs des sechsten Gelenks zwischen Simulator und TASER

Simulationslauf wurde außerdem auf einem normalen Desktopsystem durchgeführt, dass nicht, wie der Computer des TASER, auf Echtzeitverhalten konfiguriert ist. Dadurch kann das Zeitverhalten des Betriebsystems (processor scheduling) eine weitere Quelle von Ungenauigkeiten darstellen, da der SIGALRM Interrupt eventuell nicht immer sofort vom Program abgearbeitet werden kann.

Es ist anzunehmen, dass im Allgemeinen perfekte Taktung beim Simulieren nicht übermäßig wichtig ist, da zum Beispiel Synchronität innerhalb der Simulation weiterhin gegeben ist. Für die abschließende Feineinstellung einer Roboteranwendung steht immer der Test auf dem realen Gerät an.

6.3 Steuerung von zwei Manipulatoren

Als letztes soll die Steuerung des rechten und linken Arms des TASER im Simulator gezeigt werden. Hierzu wird wieder die Bewegung des Manipulators von Park- in Nullstellung absolviert, die schon im vorherigen Abschnitt beschrieben wurde. Diesmal sollen jedoch beide Arme zeitgleich gesteuert werden. Dazu werden zwei *movej* Befehle gestartet: `movej -sim left to zero & movej -sim right to zero`

Der parallele Bewegungsablauf ist in den Abbildungen 6.16 bis 6.19 dargestellt.

Wie zu erwarten, ist die Bahn beider Arme identisch. Wie schon im Abschnitt 2.3.2 angedeutet, wäre es daher sinnvoll die Konfigurationsbitmaske von RCCL für den rechten Arm so anzupassen, dass er sich auch nach außen dreht und aus der identischen ein spiegelsymmetrische Bewegung wird. Damit kann die gegenseitige Behinderung der beiden Manipulatoren sinnvoll eingeschränkt werden.

Nach erfolreichem Test des parallelen Betriebs beider Arme des TASER über zwei RCCL-Programme wurde versucht dies von einem Programm aus zu wiederholen. Dabei stellte sich heraus, dass der Simulatorregistrierungsdienst `RCISimMuxd` nicht korrekt funktioniert. Bei aufeinanderfolgenden Anfragen für verschiedene Simulatoren wird manchmal immer wieder derselbe, falsche Port zurückgeliefert, so dass keine Verbindung zu Stande kommen kann. Da auch nach eingehender Untersuchung des Programms der Defekt nicht identifiziert werden konnte und Debugging durch das sporadisches Auftreten dieses Fehler schwer war, wurde es in C++ neu geschrieben.

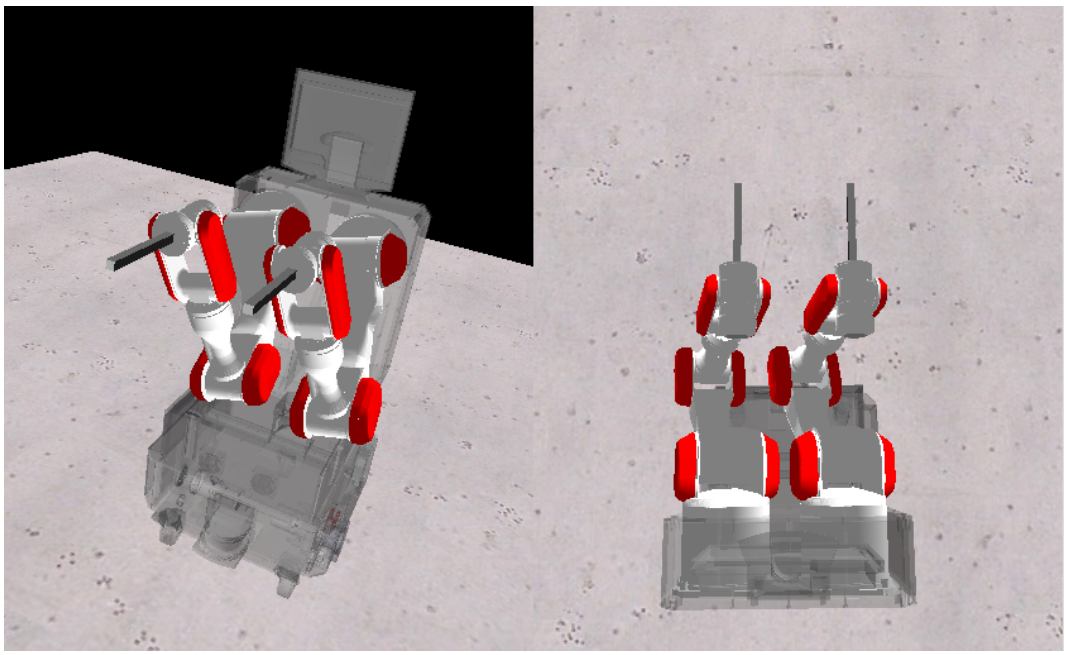


Abbildung 6.16: Beginn der Bewegung beider Arme zur Nullstellung

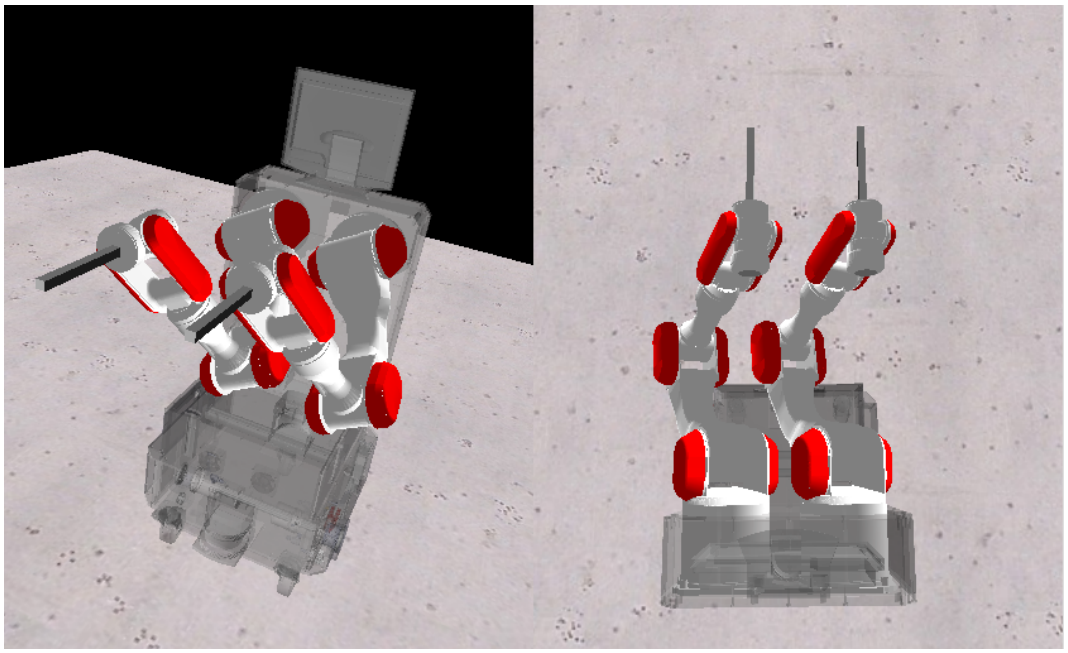


Abbildung 6.17: beide Arme des simulierten TASER auf dem Weg zur Nullstellung

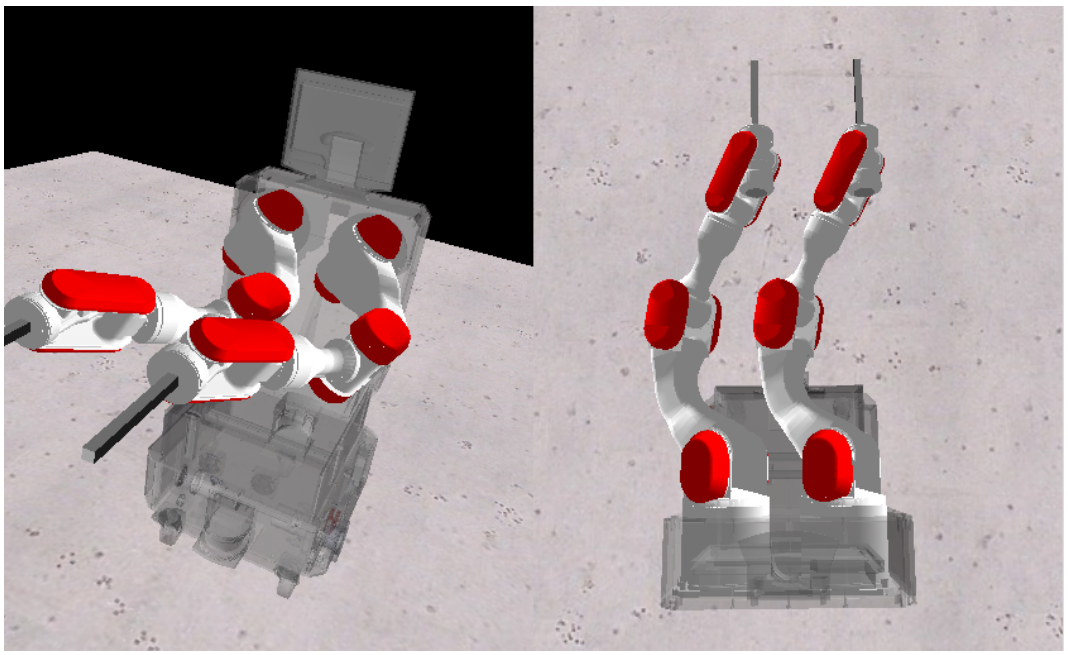


Abbildung 6.18: beide Arme des simulierten TASER kurz vor der Nullstellung

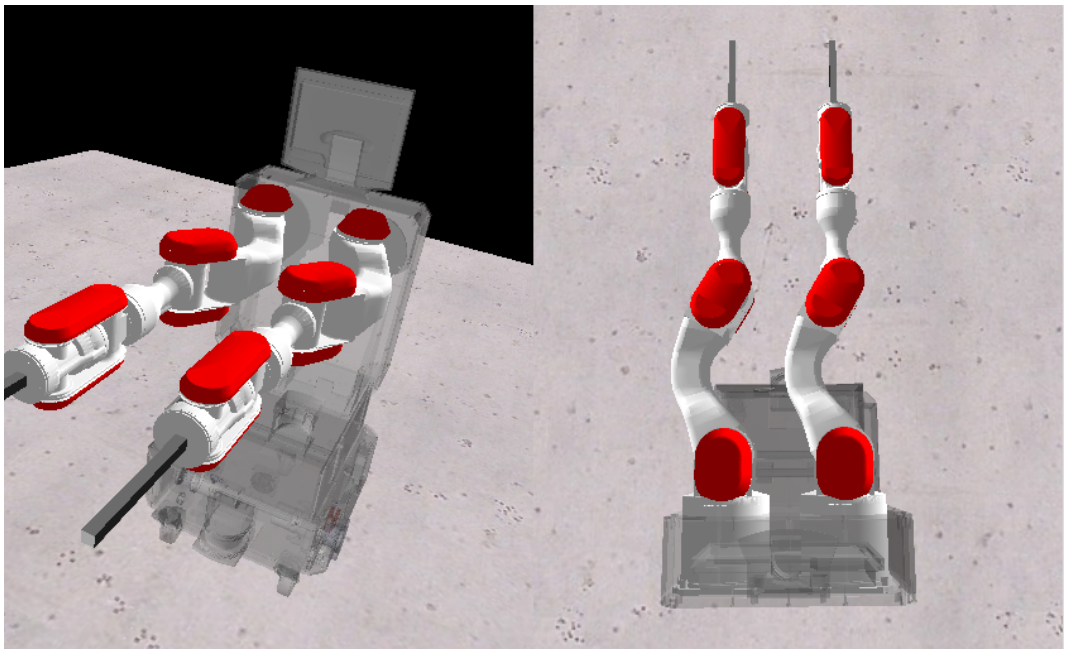


Abbildung 6.19: beide Arme des simulierten TASER in Nullstellung

Fazit und Ausblick

7

In der vorliegenden Arbeit wurde ein in RCCL integrierter Simulator für den Parallelbetrieb zweier MHI PA10-6C Roboterarme entwickelt. Das wurde durch Erweiterung des vorhandenen Simulators `Robotsim` und Modifikation von RCCL erreicht. Mit Hilfe dieses Programms ist es nun möglich, Manipulatoren des Typs PA10[1] und PA10-6C zu simulieren und somit RCCL-Programme für diese Modelle unabhängig vom physikalischen Gerät auszuführen. Am Arbeitsbereich TAMS der Universität Hamburg ist es dadurch nun möglich, dass gleichzeitig mehrere Personen Programme für den PA10-6C Manipulator des TASER Serviceroboters parallel entwickeln und testen können.

Der Simulationszustand kann grafisch mit Hilfe der von Jan Bruder in seiner Diplomarbeit entwickelten `ClientGUI`[2] dreidimensional dargestellt werden. Der Vorteil der Entkopplung von Programmentwicklung und Nutzung des realen Roboterarms sollte sich vor allem beim Einsatz in Lehrveranstaltungen zeigen, wo vielen unerfahrenen Teilnehmern die Steuerung eines Manipulators nähergebracht werden soll. Da am Arbeitsbereich TAMS der PA10-6C Roboterarm zur Verfügung steht, wäre es sinnvoll, in Vorlesungen bzw. den zugehörigen Übungen, auch diesen Manipulator anstelle eines PUMA Modells zu verwenden. Günstig ist bei der Umstellung, dass der vorhandene Simulator `Robotsim` so erweitert wurde, dass die Kommandozeilenoptionen im Hinblick auf das neu unterstützte PA10-6C Modell im Vergleich zum PUMA Modell nur minimal verändert wurden. Im Prinzip muss im Normalfall nur ein zusätzliches Programm für die neue 3D Anzeige gestartet werden.

In Kurzfassung geschah die Implementierung so, dass die PA10-spezifischen Kommunikationsfunktionen von RCCL so verändert wurden, dass die Simulationsunterstützung von RCCL korrekt funktioniert und mit `Robotsim` effektiv kommunizieren kann. Korrespondierend dazu wurde die PA10 Treiberschicht, die von den Kommunikationsfunktionen aufgerufen wird, so angepasst, dass die Steuerung des realen Manipulators weiterhin funktioniert. Daraufhin wurde `Robotsim` an entsprechenden

[1] Es gibt im Moment keine grafische Ausgabe für den PA10, da nur für den am Arbeitsbereich TAMS verwendeten PA10-6C ein 3D Modell vorhanden ist. Sobald verfügbar, kann das 3D Modell des PA10-6C aber in der `ClientGUI` durch ein entsprechendes Modell des PA10 ersetzt werden, um eine grafische Ausgabe auch für den PA10 zu ermöglichen.

[2] GUI – Graphical User Interface – grafische Benutzeroberfläche

Stellen so adaptiert, dass es sich beim Dekodieren der Befehle an den Simulator und beim Kodieren der Daten vom Simulator der Roboterklasse gemäß verhält. Damit wurde erreicht, dass `Robotsim` sowohl PUMA als auch PA10 Modelle korrekt und wenn gewünscht gleichzeitig simulieren kann.

`Robotsim` bietet außerdem den Simulationszustand über eine Netzwerkschnittstelle an, die auf einem einfachen Textprotokoll basiert. Diese Schnittstelle kann von dem entwickelten Java-Programm `SimulatorClient`, welches die `ClientGUI` verwendet, genutzt werden, um den Simulationszustand mit Hilfe des 3D Modelles des TASER anzuzeigen. Die `ClientGUI` wurde um den rechten Arm erweitert, da der Simulator mehrere Manipulatoren unterstützen kann und diese, soweit sinnvoll, auch angezeigt werden sollen.

Zur Zeit ist am TASER nur der linke Arm installiert. Es ist aber bereits ein zweiter Arm am Arbeitsbereich TAMS vorhanden und dieser soll auch angebaut werden. Voraussetzung dafür ist es, eine günstige Position für den rechten Arm zu finden, die eine gute Zusammenarbeit beider Manipulatoren ermöglicht. Mit Hilfe des in dieser Arbeit entwickelten Simulators können in Zukunft Arbeitsraumuntersuchungen für den Parallelbetrieb beider Arme des TASER durchgeführt werden. Es können dabei leicht verschiedene Stellungen der Arme zueinander ausprobiert werden, um eine optimale Position zu finden, in der sich die Arme gegenseitig minimal stören, aber immer noch sinnvoll miteinander agieren können.

Damit verbunden ist es auch vorstellbar, weitgreifender die Möglichkeiten von Zweiarminteraktionen zu untersuchen und dafür Testprogramme in RCCL zu schreiben. Bisher wurde die Steuerung beider Arme mit Hilfe von zwei unabhängigen Programmen getestet. Aber es ist mit RCCL auch möglich und sicher das Ziel des parallelen Betriebs zweier Arme, beide von demselben RCCL-Programm aus zu steuern.

Vorausgesetzt der Simulator arbeitet schnell genug, ist es vorstellbar, ihn für Echtzeitsicherheitsüberprüfungen zu benutzen, in dem er zwischen das Robotersteuerprogramm und den Roboter geschaltet wird und Befehle zuerst simuliert und auf Probleme wie Kollisionen untersucht, bevor sie zum Roboter weitergeleitet werden.

Durch Erweiterung der Simulation um ein Physikmodell könnten nicht nur Roboter, sondern beliebige Objekte sinnvoll in die Simulation integriert werden. Als Physikengine würde sich ODE (Smi07; Dem07) anbieten. Dann wäre es interessant Interaktionen (wahrscheinlich mittels harter Kollision) zwischen diesen Objekten und Robotern zu simulieren. Ein gutes Beispiel dafür wäre das Greifen von Objekten mit einem am Roboterarm befestigten Endeffektor. Das vorgestellte Protokoll der Netzwerkschnittstelle[3] von `Robotsim` sollte dabei gut auf beliebige Objekte erweitert werden können. Zum Beispiel indem Robotername und -klasse allgemeiner als Objektname und -klasse angesehen werden und die darauf folgenden Gelenkwinkelwerte allgemein als Zustandsparameter des Objektes interpretiert werden.

[3]Das Kommunikationsprotokoll ist im Abschnitt **??** spezifiziert.

Abschliessend ist zu sagen, dass die Zielsetzung der vorliegenden Arbeit erfüllt werden konnte und damit die Vorraussetzung für weitere Untersuchungen zum Parallelbetrieb zweier Arme und der Kollisionsvermeidung geschaffen worden ist.

Danksagung

8

An dieser Stelle möchte ich den Menschen danken, die mich bei dieser Diplomarbeit unterstützt haben, beziehungsweise ohne die sie gar nicht erst möglich gewesen wäre.

Als erstes bedanke ich mich bei den beiden Betreuern dieser Arbeit Prof. Dr. Janwei Zhang und Dr. Werner Hansmann für ihr entgegengebrachtes Vertrauen sowie das Ermöglichen eines reibungslosen Ablaufs im Hinblick auf organisatorische Hürden.

Des Weiteren möchte ich mich besonders bei Denis Klimentjew bedanken, der jederzeit als Ansprechpartner für inhaltliche und konzeptionelle Fragestellungen diente und mir bei allen technischen und organisatorischen Details der Diplomarbeit mit Rat und Tat zur Seite stand. Ihm und Hannes Bistry gilt auch Dank für wiederholtes kritisches Korrekturlesen, welches diese Arbeit ein kohärenteres Ganzes ergeben ließ.

Jan Bruder danke ich für seine Unterstützung bei der Integration der von ihm im Laufe seiner Diplomarbeit erstellten ClientGUI. Für organisatorische und technische Hilfe möchte ich mich bei den TAMS Mitarbeitern Tatjana Tetsis und Dr. Andreas Mäder bedanken.

Ganz herzlich möchte ich mich bei meiner Mutter für die unermüdliche moralische Unterstützung bedanken ohne die ich diese Arbeit sicher nicht hätte bewältigen können. Außerdem war ihre extensive Hilfe beim Korrekturlesen von einer nicht weiter mit dem Thema vertrauten Sichtweise sehr nützlich. Darüber hinaus bedanke ich mich bei meinen Eltern für ihre finanzielle Unterstützung, die es mir ermöglicht hat mich primär auf das Studium zu konzentrieren.

Literaturverzeichnis

[AAGD08] Asfour, T. ; Azad, P. ; Gyarfas, F. ; Dillmann, R.: Imitation Learning of Dual-Arm Manipulation Tasks in Humanoid Robots. In: *International Journal of Humanoid Robotics (IJHR)* (2008)

[BCDS07] *Kapitel* 59. In: Billard, A. ; Calinon, S. ; Dillmann, R. ; Schaal: *Handbook of Robotics*. MIT Press, 2007

[Bru01] Bruyninckx, Herman: Open Robot Control Software: the OROCOS project. In: *ICRA*, IEEE, 2001. – ISBN 0–7803–6578–X, 2523-2528

[Bru09] Bruder, Jan: *Praktische Realisierung einer haptischen Telerobotik-Steuerung für eine interaktive Nutzung*, Universität Hamburg, Diplomarbeit, 2009

[BSK03] Bruyninckx, Herman ; Soetens, Peter ; Koninckx, Bob: The Real-Time Motion Control Core of the Orocos Project. In: *IEEE International Conference on Robotics and Automation*, 2003, S. 2766–2771

[CK93] Corke, Peter ; Kirkham, Robin: The ARCL Robot Programming System. In: *Robots for Competitive Industries*, Publications, 1993, S. 484–493

[Dem07] Demura, Kosei: *Robot Simulation - Robot Programming with Open Dynamics Engine*. Tokyo : Morikita Publishing Co. Ltd., 2007. – ISBN 978–4627846913

[DH55] Denavit, J. ; Hartenberg, R. S.: A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. In: *Journal of Applied Mechanics* (1955), S. 215–221

[DH64] Denavit, J. ; Hartenberg, R. S.: *Kinematic Synthesis of Linkages*. New York : McGraw-Hill, 1964

[GLB05] Gadeyne, Klaas ; Lefebvre, Tine ; Bruyninckx, Herman: Bayesian Hybrid Model-State Estimation applied to Simultaneous Contact Formation Recognition and Geometrical Parameter Estimation. In: *The International Journal of Robotics Research* 24 (2005), Nr. 8, S. 615–630

[HKK+03] Higuchi, Masaru ; Kawamura, Takeya ; Kaikogi, Takaaki ; Murata, Tadashi ; Kawaguchi, Masataka: Mitsubishi Clean Room Robot / Mitsubishi Heavy Industries. 2003. – Forschungsbericht

[HP84] Hayward, Vincent ; Paul, Richard P.: Introduction to RCCL: a robot control 'C' library. In: *International Conference on Robotics*. Atlanta, GA, USA, 1984, S. ix+622, 293–7. 13–15

[HR87] Hayward, Vincent ; Richard, Paul P.: Robot manipulator control under Unix RCCL: A robot control C library. In: *International Journal of Robotics Research* 5 (1987), Nr. 4, S. 94–111. `http://dx.doi.org/http://dx.doi.org/10.1177/027836498600500407`. – DOI http://dx.doi.org/10.1177/027836498600500407. – ISSN 0278–3649

[KD88] Kheir, N. A. (Hrsg.) ; Dekker, M. (Hrsg.): *Systems modeling and computer simulation*. New York, NY, USA : Marcel Dekker, Inc., 1988. – ISBN 0–824–77812–X

[LH93] Lloyd, John ; Hayward, Vincent: Real-time trajectory generation in Multi-RCCL. In: *Journal of Robotic Systems* 10 (1993), Nr. 3, S. 369–390

[LH96] Lloyd, John ; Hayward, Vincent: *Multi-RCCL User's Guide*, 1996. `http://www.cs.ubc.ca/spider/lloyd/rccl/rcclGuide.ps.Z`

[MUM08] Maeda, Yusuke ; Ushioda, Tatsuya ; Makita, Satoshi: Easy robot programming for industrial manipulators by manual volume sweeping. In: *ICRA*, IEEE, 2008, S. 2234–2239

[NCD98] N. Costescu, E. Z. M. Loffler L. M. Loffler ; Dawson, D.: QRobot – A Multitasking PC Based Robot Control System. In: *Conference on Control Applications*. Trieste, Italy, 1998

[Pau81] Paul, R P.: *Robot Manipulators: Mathematics, Programming and Control*. 1981

[Sch04] Scherer, Torsten: *A Mobile Service Robot for Automisation of Sample Taking and Sample Management in a Biotechnological Pilot Laboratory*, Universität Hamburg, Diss., 2004

[SF97] Stein, Matthew R. ; Falchetti, Shawn: A New Graphics Simulator for RCCL and its use in Undergraduate Robotics Instruction. In: *Advanced Robotics, ICAR '97. Proceedings., 8th International Conference on Robotics*, 1997

[Smi] Smits, R.: *KDL: Kinematics and Dynamics Library*. `http://www.orocos.org/kdl`,

[Smi07] Smith, Russell L.: *The Open Dynamics Engine*. `http://ode.org`. Version: 2007

[WBLHZ06] Westhoff, Daniel ; Baier-Löwenstein, Tim ; Hüser, Markus ; Zhang, Jianw: A flexible software architecture for multi-modal service robots. In: *Proceedings of the Multiconference on Computational Engineering in Systems Applications (CESA 2006)*. Beijing, China, October 2006

[WSZ06] Westhoff, Daniel ; Stanek, Hagen ; Zhang, Jianwei: Distributed Applications for Robotic Systems using Roblet-Technology. In: *Proceedings of the 37th International Symposium on Robotics and Deutsche Fachtagung Robotik*. Munich, Germany, Mai 2006

[Zha08] Zhang, Jianwei: *Vorlesung: Einführung in die Robotik*. 2008

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ich bin mit einer Einstellung meiner Diplomarbeit in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den _____ Unterschrift: _____