UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Warburger Straße 100
33098 Paderborn

Sichere internetbasierte echtzeitfähige Robotersteuerung

Studienarbeit

Studiengang Ingenieurinformatik
Schwerpunkt Maschinenbau

von

Jens Twiefel

Habichtweg 1

33428 Harsewinkel

vorgelegt bei

Prof. Dr. Wolfram Hardt

im

Mai 2003

Inhaltsverzeichnis

1 Einführung **1**
1.1 Motivation . . . 1
1.2 Die Aufgabenstellung . . . 1

2 Technologie Überblick **3**
2.1 Darstellung des virtuellen Systems . . . 3
2.1.1 Virtual Reality Modeling Language . . . 3
2.1.2 Java3D . . . 6
2.2 Applet . . . 6
2.3 Servlet . . . 7
2.4 Applet - Servlet Kommunikation . . . 7
2.5 Java Native Interface . . . 10

3 Konzept zur Problemlösung **11**
3.1 Das Gesamtsystem . . . 11
3.2 Das vorhandene Teilsystem . . . 12
3.3 Die Schnittstelle . . . 13
3.4 Das neue Teilsystem . . . 14
3.4.1 Der Client . . . 15
3.4.2 Der Server . . . 16
3.4.3 Sicherheits-Strategie . . . 18
3.5 Das Lösungskonzept im Überblick . . . 18

4 Kollisionskontrolle **19**
4.1 Statische Bedingungen . . . 20
4.2 Dynamische Bedingungen . . . 21
4.3 Simulation . . . 27
4.4 Ergebnis . . . 27

5 Robotersteuerung **29**
5.1 Implementierung . . . 29
5.1.1 Die Client - Server Schnittstelle . . . 29
5.1.2 Der Server . . . 30
5.1.3 Der Client . . . 36
5.2 Inbetriebnahme und Test . . . 44

5.2.1 Systemvoraussetzungen 44
5.2.2 Inbetriebnahme des Webserver 45
5.2.3 Test der Software an einem „simulierten Roboter“ 45
5.2.4 Test der Software an dem Roboter 45
5.3 Zusammenfassung 46

6 Zusammenfassung und Ausblick 47
6.1 Zusammenfassung 47
6.2 Ausblick 49

A Kommandoreferenz Applet 51

B Starten des Roboters 53

C Beispiel Kollisionskontrolle 55

D Arbeitsbereich des Roboters 59

E Quellcode 61
E.1 Server und Client 61
E.1.1 roboPosition.java 61
E.2 Server 64
E.2.1 RoboServlet.java 64
E.2.2 HardwareInterface.java 66
E.2.3 HardwareInterface.h 69
E.2.4 collisionTest.java 70
E.2.5 roboPositionQueue.java 73
E.2.6 roboDemoQueue.java 75
E.3 Client 76
E.3.1 roboApplet.java 76
E.3.2 vrmlRobo.java 88
E.3.3 robo.jsp 92
E.3.4 HttpMessage.java 93
E.3.5 Base64Encoder.java 97
E.4 Excel- Makro 100

Literaturverzeichnis 101

Abbildungsverzeichnis

2.1 VRML Szenengraph 4
2.2 EAI Event- Kaskade 5
2.3 Zugriff auf eine VRML- Szene mit dem EAI 5
2.4 Remote Method Invocation 10

3.1 Schema Gesamtsystem 11
3.2 Roboter 13
3.3 Schema des neuen Teilsystems 14
3.4 Erster Prinzipentwurf der Benutzeroberfläche 16

4.1 Winkel am Roboter 19
4.2 Statische Endwerte für die Roboterachsen 20
4.3 Modell zum Kollisionstest (Simulation in Excel) 21
4.4 Längen am Roboter in mm 22
4.5 Skizze für p und α 23
4.6 Mindestabstände am Roboter in mm 24
4.7 Ungültiger gültiger Endpunkt 25
4.8 VRML-Modell und Kollisionstest Modell 26
4.9 Überprüfung der Kollisionsbedingungen in einer EXCEL-Tabelle 28

5.1 Klassendiagramm Server 31
5.2 Flussdiagramm Kollisionsvermeidung 34
5.3 Flussdiagramm Hardware Zugriff 36
5.4 Klassendiagramm Client 37
5.5 Screenshot Applet 39
5.6 Screenshot VRML 40
5.7 Sensorbereiche im VRML-Modell 41
5.8 Screenshot JSP-Seite 43

B.1 Home Position des Roboters 54
B.2 Frontansicht des Rabbit-Systems 54

C.1 Modell der Beispiel-Konfiguration 57

D.1 Arbeitsbereich des Roboters in der Ebene 59

1 Einführung

Ziel dieser Arbeit ist die Entwicklung einer Benutzerschnittstelle zur Steuerung eines technischen Systems. Diese Benutzerschnittstelle soll über das Internet mit dem zu steuernden, technischen System verbunden werden. Aufgrund der räumlichen Trennung zwischen der Bedienoberfläche und dem technischen System soll eine virtuelle dreidimensionale Darstellung dem Benutzer die Bedienung erleichtern. Diese Arbeit baut auf das Seminar „Steuerung technischer Systeme via Internet“ (SISI) an der Universität Paderborn aus dem Sommer-Semester 2002 auf.

1.1 Motivation

Das Medium Internet ist mittlerweile weit verbreitet und die zur Verfügung stehende Bandbreite wächst stetig. Es existiert eine Reihe von Technologien zur Programmierung von Internetanwendungen. Von denen sind einige bereits ausgereift und ermöglichen die Programmierung auf einem hohen Abstraktionsniveau. Die wohl am meisten verbreitete Programmiersprache für Internetanwendungen ist Java. Java bietet eine große Auswahl an Standard-Bibliotheken. Mit diesen ist Java für die verschiedensten Aufgaben gewappnet. Es ist beispielsweise mit Java relativ einfach eine (zweidimensionale) grafische Benutzeroberfläche zu implementieren. Daher kommt die Idee diese vorhanden Technologien für die Steuerung von technischen Systemen, zum Beispiel eingebettete Systeme, einzusetzen. Es gibt allerdings einen großen Unterschied zwischen den Internettechnologien und den eingebetteten Systemen. Während die eingebetteten Systeme in der Regel echtzeitfähig sind, fällt bei näherer Betrachtung der Internettechnologien auf, dass hier das Übertragungsprotokoll (TCP/IP, UDP,...) und viele andere Komponenten keine oder nur eingeschränkte Echtzeitfähigkeiten besitzen.

1.2 Die Aufgabenstellung

Ein vorhandenes mechatronisches System, ein Fünf-Achsen-Knickarmroboter, soll über das Internet kontrolliert und gesteuert werden. Das System besteht im wesentlichen aus den Komponenten Roboter und Rabbit. Der Roboter stellt das mechanische Grundsystem dar. Das Rabbit ist eine Rapid Prototyping Plattform für verteilte, mechatronische Systeme. Auf dieser Plattform werden die Regelalgorithmen für den Roboter ausge-

führt. Des Weiteren bietet das Rabbit ein Kommunikationssystem, mit dem es möglich ist, dass meherere Rabbits untereinander oder auch mit einem PC kommunizieren.

Um das eingebettete System über das Internet zu bedienen, ist es notwendig, das Echtzeitsystem Roboter mit der Internettechnologie zu verbinden.

Es soll ermöglicht werden, dass der Roboter sicher und zeitnah (weiche Echtzeit) gesteuert und visualisiert werden kann. Die Visualisierung des Roboters auf einem Client soll als dreidimensionale, graphische Oberfläche erfolgen. Über diese Oberfläche soll auch der Benutzer die Möglichkeit haben den Roboter zu steuern. Das Gegenstück zu dem Client stellt ein Server dar. Auf dem Server soll die Kommunikation mit dem Roboter erfolgen. An dieser Stelle ist es wichtig, dass das System gegen falsche Eingaben abgesichert wird. Das bedeutet, dass neue, durch den Benutzer vorgegebene, Soll-Werte auf Gültigkeit geprüft werden müssen. Durch diese Prüfung soll ausgeschlossen werden, das der Roboter an eine Position fährt, die zu einer Kollision mit sich selbst führt und garantiert werden, dass das System „Fail Safe“ ist. Da es nur einen Roboter, aber viele potentenzielle Nutzer gibt, soll eine Multi-User-Strategie erstellt und implementiert werden.

Die Arbeit gliedert sich in die folgenden Punkte:

- Auswahl der Technologien
- Erstellen eines Konzeptes
- Implementierung des Systems

Bei der Auswahl der Technologien sollen nur Technologien betrachtet werden, die für eine Implementierung in Frage kommen. Für die dreidimensionale Visualisierung stehen zwei verschiedene Technologien zur Auswahl: Virtual Reality Modeling Language (VRML) und Java3D. Weiterhin sollen die verschiedenen Möglichkeiten zur Verbindung von Server und Client untersucht werden.

Das Konzept soll die Funktionsweisen der einzelnen Komponenten, in denen die ausgewählten Technologien zum Einsatz kommen, detailliert beschreiben. Dazu gehören die Funktionsweise der Kollisionsvermeidung, des Servers, des Clients, aber auch ein Überblick über die Funktion des Gesamtsystem.

Der abschließende Teil der Arbeit befasst sich mit der Implementierung der Robotersteuerung und der Client-Server Kommunikation.

2 Technologie Überblick

Im Folgenden werden kurz die zur Verfügung stehenden Technologien, die zur Bearbeitung der Aufgabe geeignet scheinen vorgestellt.

2.1 Darstellung des virtuellen Systems

Das technische System soll zur besseren Bedienbarkeit mittels eines dreidimensionalen Modells repräsentiert werden. Für die Visualisierung stehen zwei Technologien zur Auswahl: Virtual Reality Modeling Language und als Alternative Java3D.

2.1.1 Virtual Reality Modeling Language

Die Virtual Reality Modeling Language (VRML) ist eine Beschreibungssprache für virtuelle dreidimensionale Modelle. Für die Darstellung einer 3D-Szene wird ein spezielles Programm, ein VRML-Browser, bzw. ein Plug-In für einen Webbrowser, benötigt. Die aktuelle Version heißt VRML97, und ist in einer ISO-Norm (ISO/IEC 14772) definiert, siehe [21]. Eine Nachfolgeversion ist in Arbeit, diese wird X3D heißen, siehe [20]. Die in einer Text-Datei gespeicherten Daten sind in einer Baumstruktur organisiert. Dieser Baum wird Szenengraph genannt. Die (graphischen) Elemente der Szene werden im Szenengraphen als Knoten und die Eigenschaften der Elemente als Blätter abgebildet, siehe Abbildung (2.1). In dem Szenengraphen gibt es beispielsweise den Knoten `trans1`, vom Typ Transform. Die Eigenschaft rotation ist als Blatt dargestellt.

Die dargestellten Grafiken werden aus den geometrischen Primitiven (Quader, Zylinder, Kegel und Kugel) zusammengesetzt. Dieser Umstand erschwert das manuelle erzeugen einer Szene. Um technische Systeme mit einem vertretbaren Aufwand mit VRML darzustellen werden Programme eingesetzt, die CAD (Computer Aided Design)-Daten in das VRML konvertieren können. Viele CAD-Programme wie beispielsweise SolidEdge oder Catia bringen eine VRML-Export Funktion mit.

Um eine interaktive bewegte Szene zu erstellen, können einige Eigenschaften der VRML-Elemente zur Laufzeit bzw. Darstellungszeit verändert werden. Um eine Eigenschaft, beispielsweise die Translation (Verschiebung) eines Transform-Knotens, zu verändern, muss ein Ereignis an diese Eigenschaft gesendet werden. Ein solches Ereignis kann auf drei verschiedene Arten erzeugt werden: Der Benutzer kann über Sensor-Knoten direkt

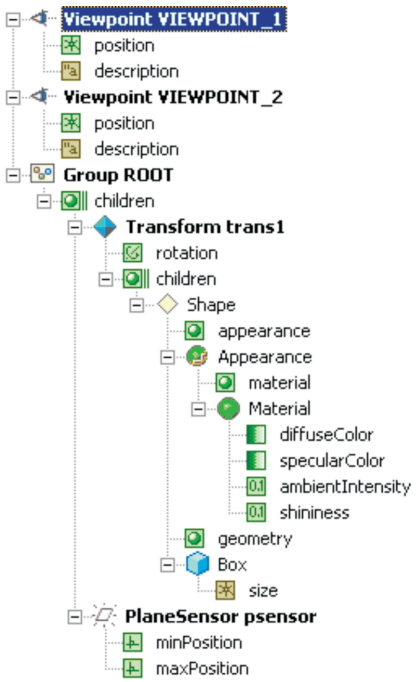

Abbildung 2.1: VRML Szenengraph

ein Ereignis provozieren. Mit Hilfe eines Interpolator-Knotens, den man sich als eine Art Signalgenerator vorstellen kann, kann eine fest definierte Folge von Ereignissen ausgelöst werden. Für die Anwendung in einer interaktiven Umgebung sind sie aufgrund fehlender Flexibilität ungeeignet. Der Script-Knoten reagiert auf ein eingehendes Ereignis und generiert, mit einem in die VRML-Daten eingebetteten Programm, ein neues ausgehendes Ereignis. Die Ereignisse werden über Routen zwischen den Eigenschaften der Knoten verschickt. Alle Ereignisse werden in einer Kaskade (siehe Abbildung (2.2)) abgearbeitet. Das heißt, wenn infolge eines Ereignisses 1 ein neues Ereignis 2 generiert wird, wird dieses vor einem anderem Ereignis 4 gesendet, welches z.B. vom Benutzer zeitgleich mit dem Ereignis 1 ausgelöst wurde.

Bisher wurden die Möglichkeiten beschrieben, um innerhalb eines VRML-Modells Benutzereingaben zu verarbeiten und Bewegungen zu erzeugen. Wenn ein reales, technisches System (echtzeitnah) dargestellt werden soll, muss von außen auf die Szene zugegriffen werden. Mit dem external authoring Interface (EAI) ist dieses auch möglich. Dieses Interface ist in dem VRML97 Standard programmiersprachenunabhängig definiert.

In der Regel ist das EAI als Java Interface implementiert, so kann von einem Java Applet im Webbrowser auf die VRML-Szene im VRML-Plug-In zugegriffen werden. Das external authoring interface bietet vier verschiedene Kommunikationsmöglichkeiten zwischen dem Szenengraph und der Applikation, siehe Abbildung (2.3). Von diesen sind hier nur zwei Möglichkeiten von Interesse: Zum einen das Senden von Ereignissen von der Applikation in die Szene, um dem Szenengraphen mitzuteilen, wie die aktuelle Po-

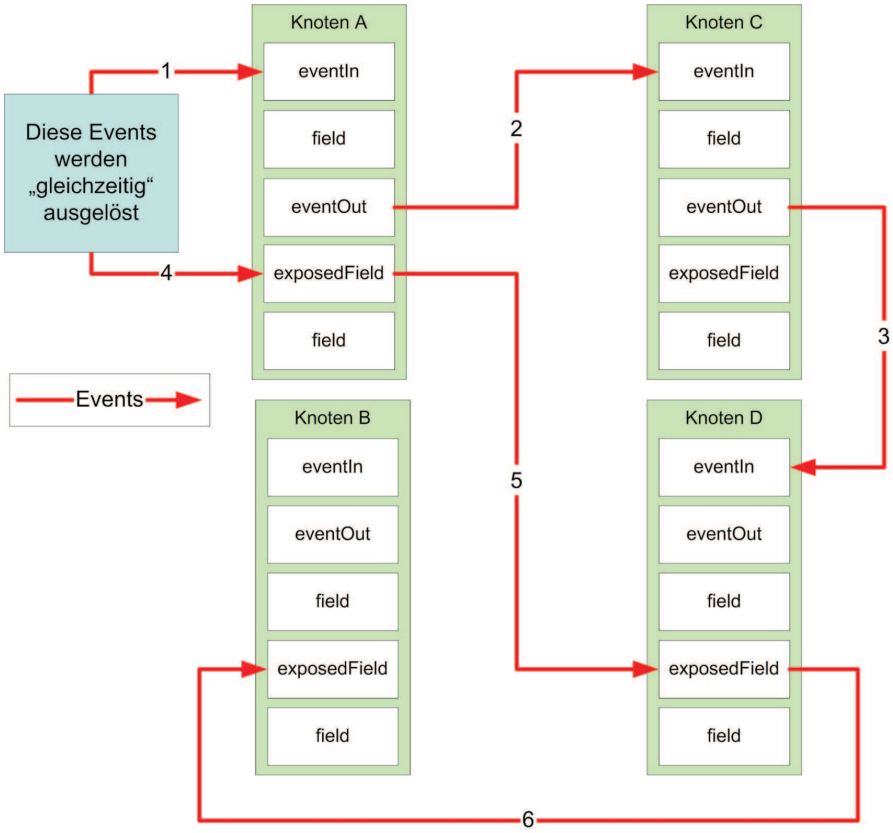

Abbildung 2.2: EAI Event- Kaskade

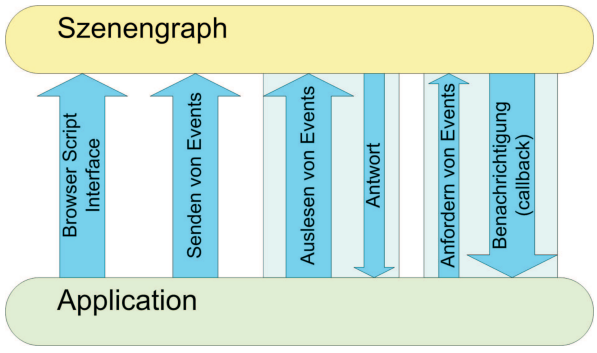

Abbildung 2.3: Zugriff auf eine VRML- Szene mit dem EAI

sition des technischen Systems ist. Zum anderen das callback, die Benachrichtigung der Anwendung, wenn sich in den Soll-Werten etwas ändert.

2.1.2 Java3D

Java3D ist ein API (application programming Interface) zur Erzeugung und Darstellung dreidimensionaler Szenen. Dieses API ist zurzeit noch kein Bestandteil der Java 2, Standard Edition (J2SE), sondern als „Optional Package“ Teil der Standard Erweiterungen, siehe [5]. Es muss daher zusätzlich installiert werden.

In Java3D werden die Szenen wie in VRML, in einer hierarchischen Struktur, einem Szenengraphen, verwaltet. Da Java eine objektorientierte Programmiersprache ist sind einzelne Knoten Objekte. Anders als bei VRML, wo immer nur eine Szene in einer Datei bereitgestellt werden kann, beinhaltet ein Java3D `VirtualUniverse` Objekt mehrere Szenen. Ein Java Programm kann auch mehrere `VirtualUniverse` Objekte haben. Dieses Objekt ist die Wurzel der 3D Struktur. Der die einzelnen Szenen des „Universums“ untergeordnet sind. Eine Szene beginnt immer mit einem `Local` Objekt. Dort hinein können verschiedene Knoten eingefügt werden, und es existiert eine große Menge von Knoten für den Aufbau einer 3D- Szene. Hier sei exemplarisch der `TransformGroup` Knoten herausgegriffen: Dieser Knoten bietet Eigenschaften für die Positionierung der im untergeordneten Teilgraphen angegebenen Szene. Weiter gibt es Blatt-Knoten, die keine weiteren untergeordneten Knoten haben dürfen. Ein Beispiel für einen Blatt-Knoten ist der `Shape3D` Knoten: Dieser kann eine geometrische Form enthalten. Im Gegensatz zu VRML gibt es aber keine vordefinierten geometrischen Formen. Zur Darstellung auf dem Bildschirm dient eine `ViewPlatform`. Zu einem solchen Objekt lassen sich eine oder mehrere Kameras hinzufügen. Das Bild einer Kamera muss noch eine Position auf dem Bildschirm zugewiesen bekommen, das geschieht mit einem `Canvas3D` Objekt, welches wiederum auf einem `Screen3d` Objekt dargestellt wird.

Die elementaren Objekte, die für die Darstellung notwendig sind, werden normalerweise in einem `SipmleUniverse` zusammengefasst. Diese Objekte sind: `VirtualUniverse`, `Locale` und eine `ViewPlatform` mit den dazugehörigen Objekten.

Um eine Java3D Szene zu animieren oder um in einer Szene zu interagieren existiert eine spezielle Gruppe von Knoten: Die `Behavior` Objekte. Es gibt ein paar vordefinierte `Behavior` Objekte um beispielsweise irgendetwas mit der Maus zu Steuern oder um eine vordefinierte Animation auf Basis einer Interpolation zu erstellen. Des Weiteren können auch neu programmierte `Behavior` Objekte eingebunden werden. Eine gute Einführung in Java3D ist „Getting Started with the Java 3D API“ [1].

2.2 Applet

Applets sind kleine Java Programme, die in Webseiten eingebettet werden können. Applets werden auf der Java Virtual Maschine des Rechners (Client) ausgeführt, der die Webseite anzeigt. Für Applet gibt es ein umfangreiches Sicherheitskonzept, das gewährleistet, dass das Applet keinen Zugriff auf die Systemressourcen des Clients bekommt.

Ein Applet kann alle Java Bibliotheken nutzen. Das hat den Vorteil, dass auch mit kleinen Dateien graphische Benutzeroberflächen erzeugt werden können. Applets haben keine `main` Funktion, da sie als Thread der Java Virtual Maschine geladen werden.

2.3 Servlet

Servlets sind Erweiterungen für Webserver, die in Java geschrieben sind. Diese Erweiterungen können von dem Webserver dynamisch in eine Java Virtual Maschine geladen werden. Es ist mit Servlets möglich den kompletten Umfang der Java Kernelemente zu nutzen: Netzwerkkommunikation, Datenbankzugriffe (JDBC), Multithreading, Objektserialisierung, Java Native Interface (JNI), entfernte Methodenaufrufe (RMI) und weitere. Es ist auch möglich die J2EE Plattform zu nutzen. Servlets bleiben wenn sie einmal geladen sind als Objektinstzanz im Speicher des Servers. So wird gewährleistet, dass Servlets fast unmittelbar mit der Abarbeitung von Anfragen beginnen können. Da ein Servlet dauerhaft im Webserver geladen ist kann es auch Verbindungen zu Datenbanken dauerhaft offen halten oder auch andere Funktionen ausführen. Servlets haben wie die Applets keine `main` Funktion.

Auf Servlets baut die Java-Server-Pages (JSP) Technologie auf. Im Wesentlichen ist JSP eine „Sprache“ zur Generierung von dynamischen Webseiten. Eine JSP-Datei gleicht im Aufbau einer HTML-Datei, mit dem Unterschied, dass zusätzlich Anweisungen für einen JSP-Parser enthalten sind. Der Parser ist in einem Servlet implementiert, welches mit den Anweisungen die Webseite erstellt und diese an den Client sendet.

2.4 Applet - Servlet Kommunikation

Es gibt mehrere Möglichkeiten der Kommunikation zwischen einem Applet und einem Servlet:

- Text über Http
- Text über sockets
- Objekte über Http
- Objekte über sockets
- Nutzung des Remote Method Invocation

Kommunikation über eine Http-Verbindung

Bei der Kommunikation über eine Http (hypertext transfer protocol)-Verbindung verhält sich das Applet wie ein Webbrowser. Es nutzt das standardisierte Http- Protokoll. Das Servlet fungiert als Webserver. Ein Nachteil dieses Verfahrens ist, dass keine dauerhaften Verbindungen hergestellt werden können. Für jede Nachrichtensendung muss ein neuer Kommunikationskanal angefordert werden, was zu Beeinträchtigungen der Geschwindigkeit führt. Ein weiterer Nachteil ist, dass eine neue Verbindung immer nur vom Applet aus erfolgen kann, woraus folgt, dass der Server nur auf Anfragen antworten, nicht aber von sich aus Nachrichten an das Applet schicken kann. Ein bedeutender Vorteil der Kommunikation über Http ist, das es auch auf Computern, die hinter einer Firewall liegen funktioniert, wenn Http Verbindungen zugelassen sind, was unerlässlich ist, wenn die Internetverbindung zum surfen genutzt werden soll. Es ist auch möglich die Verbindungen mit dem Https Protokoll zu verschlüsseln.

Kommunikation über eine socket-Verbindung

Die Kommunikation funktioniert in der der Übertragungstechnik nicht anders als die Kommunikation über Http, da die Kommunikation via Http eine Teilmenge der Kommunikation über sockets darstellt. Das Http-Protokoll, welches oben angesprochen wurde, wird hier nicht benutzt und muss stattdessen vom Programmierer selbst implementiert werden. Mit einem solchen Protokoll ist es dann auch möglich die Verbindungen länger als für eine Anfrage geöffnet zu halten. Des Weiteren ist es möglich eine bidirektionale Verbindung zwischen Servlet und Applet zu schaffen, was bedeutet, dass auf eine einmal geöffnete Verbindung beide Seiten schreiben dürfen. Mit einer solchen Verbindung ist die Übertragung der Nachrichten schneller als bei der Verbindung über Http möglich, da fast der gesamte overhead, der zum Verbindungsaufbau benötigt wird, entfällt. Aus Sicherheitsgründen muss auch hier immer das Applet die Kommunikation beginnen. Wenn hier der Server und/oder der Client hinter einer Firewall liegen ist es sehr wahrscheinlich, dass die Verbindung nicht aufgebaut werden kann, da die Firewall direkte Socket-Verbindungen über unbekannte Ports gewöhnlich blockiert. Nur wenn die Fierewall entsprechend eingerichtet ist, kann eine Kommunikation erfolgen. Der Programmieraufwand ist deutlich höher als bei der Http-Variante, da hier die meisten Module neu entwickelt werden müssen.

Für das Format der Nachrichten gibt es auch zwei Varianten: Text und Java Objekte.

Übertragen der Nachrichten in Text-Form

Hier werden die Nachrichten einfach als Klartext versand. Der Sender muss die Textnachricht erst generieren, worauf der Empfänger diese Nachricht parst, und die einzelnen Bestandteile in die entsprechenden Formate/Typen konvertiert.

Übertragen eines Java Objektes

In diesem Fall wird an Stelle des Textes ein Java Objekt versand. Das Versenden von Objekten ist allerdings nicht ohne weiteres möglich. Die Objekte müssen zunächst serialisiert und später wieder deserialisiert werden. Java stellt hierfür spezielle Methoden und Interfaces zur Verfügung. Beim Serialsieren wird das Objekt in einen Byte-Datenstrom umgewandelt und so ermöglicht, dass ein Objekt verschickt oder auch gespeichert werden kann. Die Übertragung von Objekten hat zwei Vorteile gegenüber der Übertragung von Text: Die Datentypen müssen nicht zwei mal umgewandelt werden und die Übermittlung findet nicht im Klartext statt. Um sowohl auf dem Server als auch auf dem Client das Objekt zu lesen bzw. zu erzeugen muss die Klasse, von der das Objekt abstammt, auf beiden Seiten exakt dieselbe Quelle haben. Wenn die Objekt-Serialisierung genutzt werden soll, muss die Java Virtual Machine (JVM) mindestens Version 1.1 sein.

Remote Method Invocation

Die Remote Method Invocation (RMI) stellt eine weitere Möglichkeit zur Kommunikation zwischen Applet und Server dar. Für eine Kommunikation via RMI wird nicht unbedingt ein Servlet benötigt. Stattdessen wird ein spezieller RMI- Server und ein RMI-Registry-Server benötigt.

Remote Method Invocation ist ein verteiltes Objektsystem, bei dem der Sever Objekte für den Client bereitstellt. Der Client kann auf ein solches Objekt zugreifen, wie er auf ein lokales Objekt zugreifen würde. Das heißt der Aufruf von Objekt-Methoden erfolgt synchron also wartet der aufrufende Thread auf die Antwort. Der einzige Unterschied bei einem Zugriff auf ein entferntes Objekt besteht in einer erweiterten Ausnahmebehandlung. Objekte können aus Sicht des Programmierers mit dem RMI als Referenz übergeben werden. Um dies technisch zu ermöglichen werden spezielle Objekte automatisch während des Übersetzungsvorgangs erzeugt: das Rumpf- und das Gerüstobjekt. Das Rumpfobjekt wird dann an den Client übertragen und kommuniziert mit dem Gerüstobjekt des Servers. Für den Programmierer ist diese Kommunikation nicht sichtbar, siehe Abbildung (2.4) aus [16]. Der Entwickler muss nur den nicht grau hinterlegten Teil implementieren. Bei der Java 2 Plattform ist das Gerüstobjekt nicht mehr notwendig. Die Java 2 Rümpfe können direkt mit dem Server kommunizieren. Die Verbindung zwischen dem Server und dem Client läuft in der Regel über eine Socket Verbindung (der Standard Port für RMI ist 1099). Es gibt Ansätze in denen RMI über eine Http Verbindung kommuniziert, falls eine direkte Socket Verbindung nicht zustandekommt. Diese Ansätze funktionieren allerdings noch nicht mit einer akzeptablen Qualität. Für jede Client-Server Verbindung wird ein separater Thread auf dem Server erstellt, daher findet der Zugriff parallel statt. Wenn auf eine Resource nur exklusive Zugriffe erfolgen dürfen, muss der Programmier diese synchronisieren.

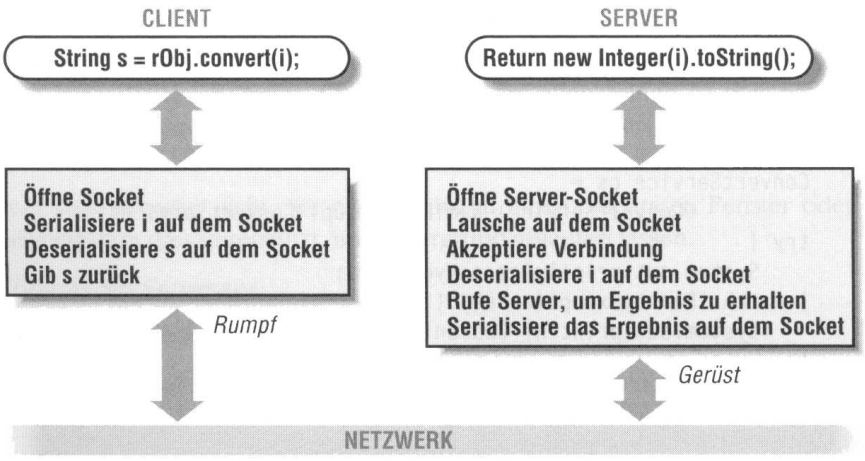

Abbildung 2.4: Remote Method Invocation

2.5 Java Native Interface

Das Java Native Interface (JNI) ist eine Schnittstelle zwischen Java und beispielsweise C oder C++. Inzwischen gibt es für diverse Hardwarekomponenten spezielle Java Pakete wie beispielswese das `javax.comm` Paket für den Zugriff auf die serielle Schnittstelle (RS232). Wenn aber nicht auf Standard-Hardware zugegriffen werden soll muss ein Treiber für diese Hardware geschrieben werden. Ein solcher Treiber wird in einer hardwarenahen Programmiersprache, beispielsweise C, geschrieben. Soll jetzt ein Java Programm mit diesem Treiber kommunizieren muss auf das Java Native Interface zurückgegriffen werden.

3 Konzept zur Problemlösung

In diesem Kapitel wird das Gesamtsystem vorgestellt. Zuerst werden die Systemteile erläutert, die in einer früheren Phase des Projektes fertiggestellt wurden. Eine Schnittstelle zwischen dem vorhandenem Teilsystem und dem neuen Teilsystem wird definiert. Im Anschluss werden die Komponenten vorgestellt, die im Rahmen dieser Studienarbeit erstellt werden.

3.1 Das Gesamtsystem

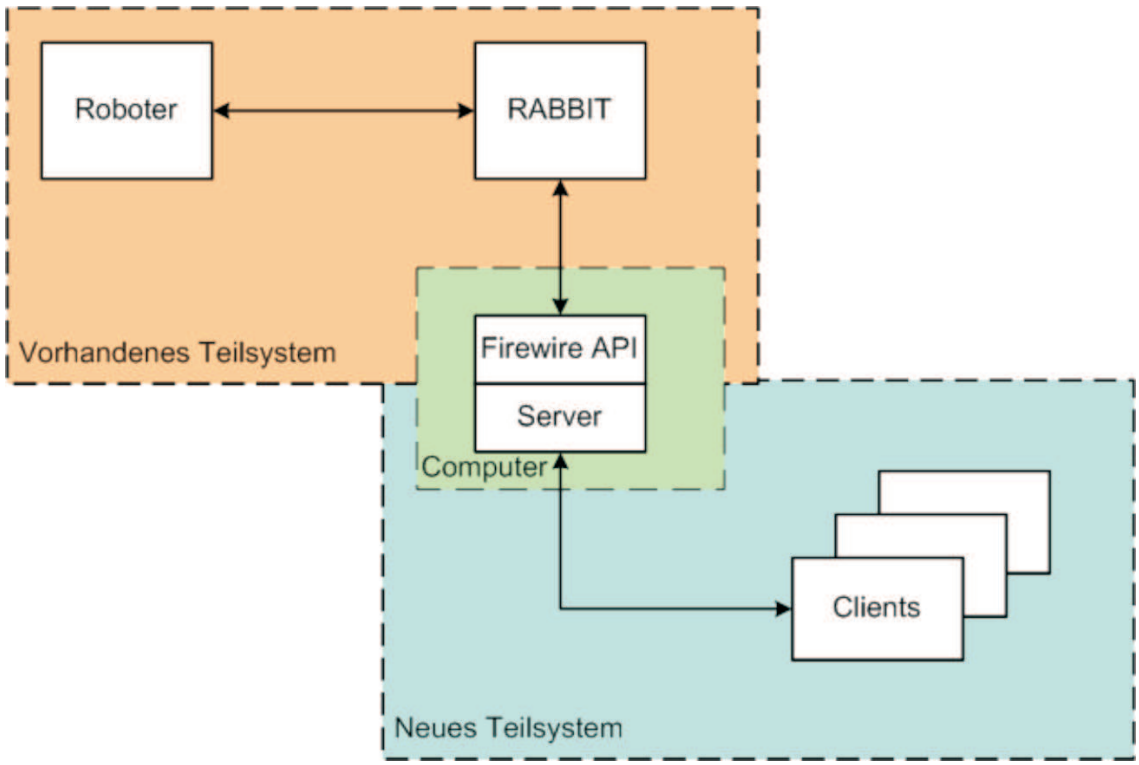

Abbildung 3.1: Schema Gesamtsystem

In der Abbildung (3.1) ist das komplette technische System als Schema dargestellt. Das technische System besteht aus den Komponenten Roboter, Rabbit, einem PC mit einer Kommunikations-API und einem Serverprogramm und (mehreren)Clients.

3.2 Das vorhandene Teilsystem

Das Vorhandene Teilsystem besteht aus dem Roboter, dem Rabbit System und einem PC mit Firewire Anschluss.

Der Roboter

Der Roboter (Abbildung (3.2)) hat fünf Freiheitsgerade für die Positionierung im Raum und einen Freiheitsgrad für die Öffnungsweite des Greifers. Die Achsen sollen im Folgendem von null bis fünf durchnummeriert werden, von null, was den Freiheitsgrad für die Drehung des gesamten Roboters um die Hochachse beschreibt, bis fünf, was die Öffnung des Greifers darstellt. Die Achsen null, eins, zwei und fünf werden jeweils mit einem eigenen unabhängigen Motor angetrieben. Die Achsen drei und vier teilen sich zwei Motoren. Eine Bewegung um Achse drei wird realisiert, indem beide Motoren gleich schnell in die gleiche Richtung drehen. Für die Bewegung um Achse vier müssen sich die beiden Motoren entgegengesetzt bewegen. Die Motoren an dem Roboter sind mit Sensoren ausgestattet, die es ermöglichen die Stellwinkel der Motoren mit einer hohen Genauigkeit zu erfassen.

Rabbit

Das RABBIT System ist eine vom „Mechatronik Laboratorium Paderborn“ (MLaP) und „Informatik und Prozesslabor“ (IPL) entwickelte modulare Rapid Prototyping Plattform für verteilte, mechatronische Systeme. Das Rabbit System ist modular aufgebaut und besteht in der Grundkonfiguration aus drei Modulen: einem Firewire (IEEE 1394) Modul, einem Microcontroller-Modul und einem FPGA (Field Programable Gate Array). Für den Roboter ist das Rabbit System um DC-Motor-Module und ein Adapter Board erweitert worden. Das Adapter Board verbindet einen KEL100 Bus mit einem VG64 Bus. Die Grundkomponenten kommunizieren über den KEL100 Bus. Die neuen DC-Motor-Module kommunizieren über den VG64 Bus. Diese Module können jeweils zwei Motoren des Roboters mit elektrischer Spannung versorgen und steuern so die Motoren an. Weiter besitzen die Module, pro Motor Eingänge für ein Encodersignal und zwei Endschalter. Maximal können drei DC-Motor-Module in einem Rabbit System arbeiten. Also können maximal sechs Motoren angesteuert werden. Die Regelalgorithmen, ein PID-Regler pro Motor, sind auf dem Rabbit Grundsystem implementiert. Die Regelparameter sind mit Hilfe eines mathematischen Modells ermittelt worden und manuell angepasst worden.

Kommunikationssystem

Für die Kommunikation zwischen eingebetteten Systemen ist eine Abstrakte Kommunikations-Schicht definiert worden. Der Vorteil einer solchen Schicht ist es, das der Anwen-

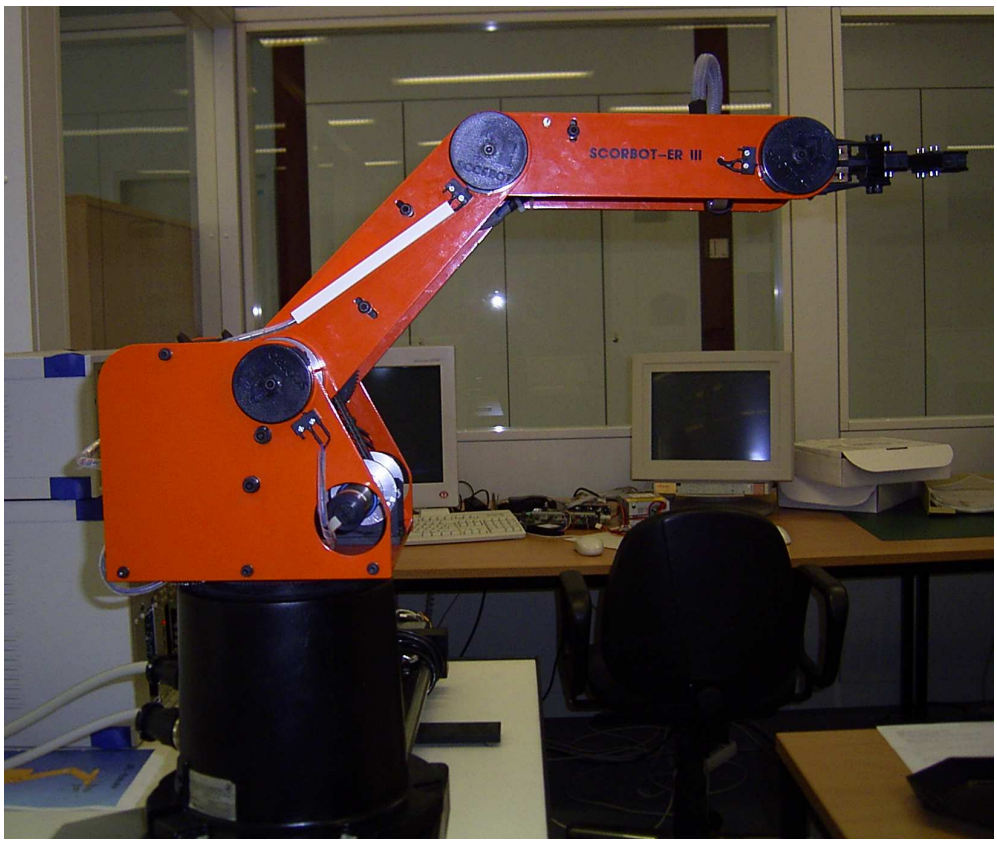

Abbildung 3.2: Roboter

dungsprogrammierer einfach einen Komunikationskanal benutzen kann, ohne sich darum zu kümmern wie genau die Kommunikation implementiert ist oder auch welche Schnittstelle genutzt wird. Das erstellte Kommunikationssystem kann über eine große Anzahl von Schnittstellen kommunizieren: RS-232, RS-485, CAN und Firewire (synchron und asynchron). Dieses Kommunikationssystem ist auf dem Rabbit implementiert. Für die Kommunikation zwischen Rabbit und Linux-PC über die Firewire Schnittstelle ist eine spezielle Kommunikations-API für den Roboter erstellt worden. Die Kommunikations-API bietet zwei Methoden: eine zum Auslesen der aktuellen Roboter-Position, und eine zum setzen neuer Soll-Werte.

3.3 Die Schnittstelle

Die Schnittstelle zwischen dem vorhandenem Teilsystem und dem neuen Teilsystem liegt in der Verbindung zwischen der Kommunikations-API und dem Servlet. Da das Kommunikations-API in C und das Servlet in Java geschrieben ist wird das Java Native Interface (JNI) zur Kommunikation eingesetzt.

3.4 Das neue Teilsystem

Die Hauptfunktionalität des bearbeiteten Teilsystems kann in zwei überlagerte Vorgänge aufgeteilt werden:

1. Visualisierung der Ist-Position in einer dreidimensionalen Benutzerschnittstelle.
2. Übermittlung von neuen Sollwerten an den Roboter.

Um die geforderten Funktionalitäten zu erbringen sind eine Reihe von Komponenten notwendig, die in der Abbildung (3.3) dagestellt sind.

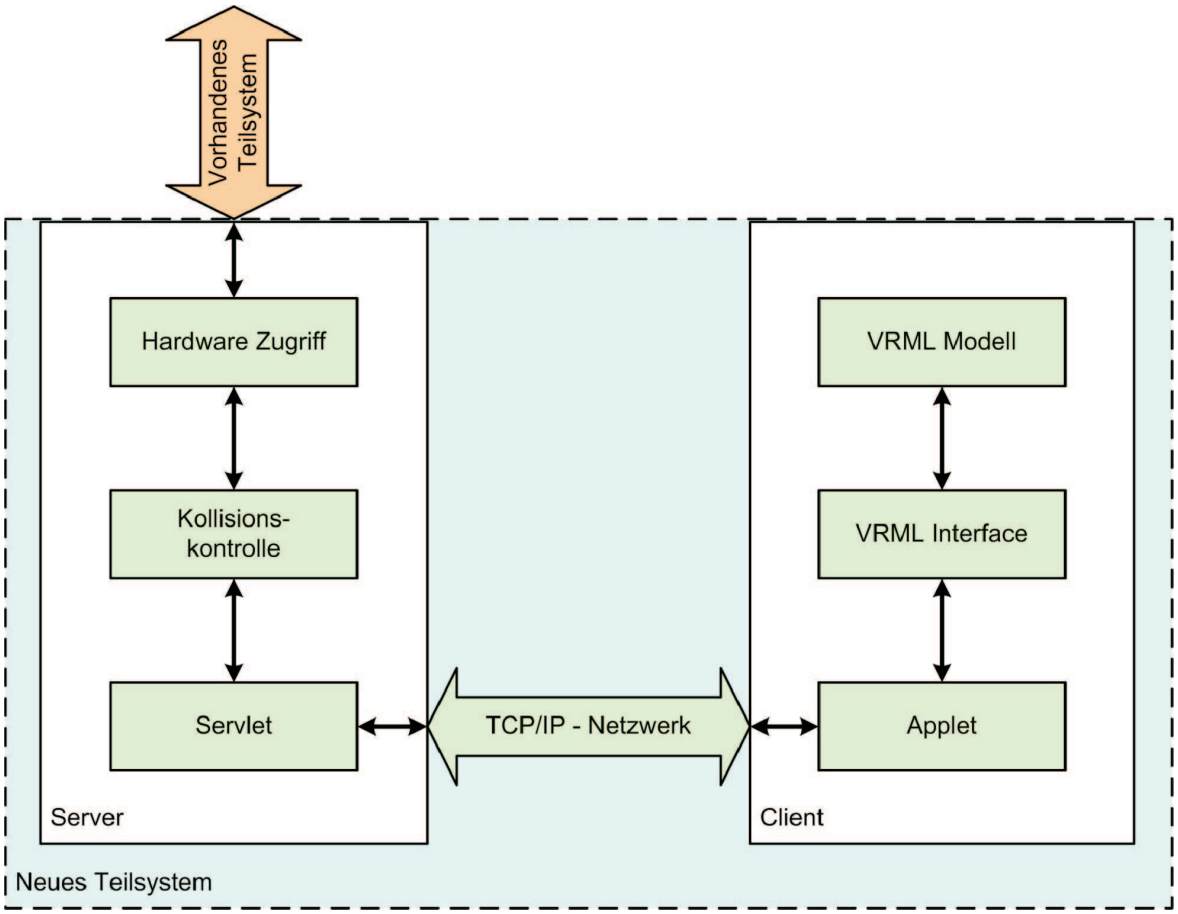

Abbildung 3.3: Schema des neuen Teilsystems

Das Neue Teilsystem ist im wesentlichen eine Client-Server Anwendung. Der Client dient zur Visualisierung des Roboters und als Eingabeoberfläche für den Benutzer. Der Client ist in zwei Hauptkomponenten: das VRML-Modell und das Applet unterteilt. Diese werden im Folgendem beschrieben. Der Server hat die Aufgabe zwischen dem Client und

dem technischen System Roboter zu vermitteln. Er überträgt auf Anfrage des Clients die aktuelle Roboterposition, des Weiteren übermittelt er neue Soll-Werte, nach einer Prüfung, an den Roboter.

3.4.1 Der Client

Der Client besteht aus einem Applet und einem 3D-Modell. Für das 3D-Modell wird VRML ausgewählt da dass bereits ein weitgehend automatisch generiertes VRML-Modell des Roboters vorliegt. Diese werden zusammen in eine Webseite eingebunden. Ein früher Entwurf des Designs ist in der Abbildung (3.4) dargestellt. Sowohl das Applet als auch das VRML-Modell bieten die Möglichkeit der Visualisierung der Position des Roboters als auch die Steuerung des Roboters, wobei beide Technologien miteinander synchronisiert werden.

VRML- Modell

Das VRML Modell ist in weiten Teilen vorhanden und wird nur in einigen Punkten angepasst. Es ist aus CAD-Daten erzeugt worden. In diesem Modell wird der Roboter auf einem Untergrund dargestellt. Zur besseren Nutzbarkeit des Systems sollen zwei Roboter in dem VRML-Browser, der in eine Internetseite eingebettet ist, angezeigt werden. Einer dient zur Visualisierung des Soll-Werts, der andere zur Visualisierung des Ist-Werts (siehe Abbildung (3.4) oben). Der Roboter, der für den Ist-Wert zuständig ist, ist mit graphischen Sensoren ausgestattet, um eine Steuerung durch den Benutzer zu ermöglichen. Die Sensoren reagieren in bestimmten Bereichen auf ein „klicken“ mit der Maus, siehe Abbildung (5.7).

Das Applet

Das Applet soll zwei Anwendungszenarien genügen: Zum einen sollen die im VRML-Modell vorgegebenen Werte ausgelesen und an den Server übermittelt werden, zum anderen soll das Applet auch ohne das VRML-Modell eine komplette Steuerung des Roboters ermöglichen. Das Applet soll, wenn es geladen wird, eine Benutzeroberfläche erzeugen. Der Benutzer kann wählen ob er das VRML-Modell aktivieren möchte. Bei der Aktivierung des VRML-Modells wird das callback initialisiert, damit Änderungen im VRML-Modell an das Applet gesendet werden. Zusätzlich zur Möglichkeit das VRML-Modell zu aktivieren soll die Benutzeroberfläche Elemente für die Vorgabe von Soll-Werten und zur Anzeige der Ist-Werte enthalten.

Für die Übermittlung der Soll-Daten an das Servlet soll es zwei Modi geben. Erstens der Folgemodus, in dem jede Änderung im Applet und/oder im VRML-Modell unverzüglich an den Server übermittelt wird und zweitens einen Modus, bei dem nur dann Daten

gesendet werden wenn, der Benutzer dies explizit wünscht. Dieser Modus dient dazu, eine genau definierte Position anzufahren.

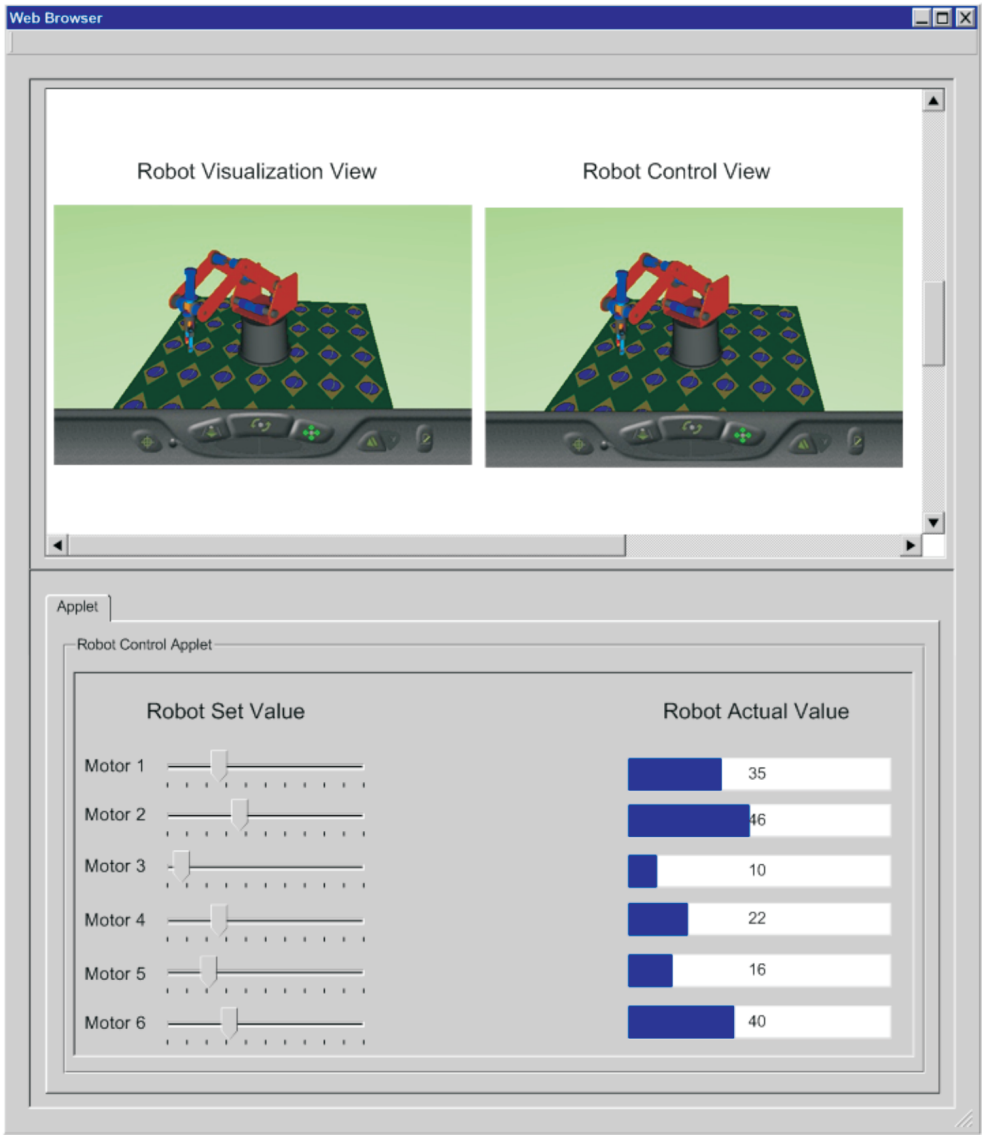

Abbildung 3.4: Erster Prinzipentwurf der Benutzeroberfläche

3.4.2 Der Server

Der Server übernimmt die Aufgabe zwischen dem Client und dem vorhandenem Teilsystem Daten auszutauschen. Das Serversystem besteht aus drei wichtigen Modulen: Das Servlet, der Kollisionskontrolle und dem Hardwarezugriff. Die Kollisionskontrolle wird im Kapitel (4) vorgestellt.

Das Servlet

Das Servlet übernimmt die Aufgabe die von dem Applet gesendeten Daten zu empfangen, die Daten auf ihre Gültigkeit prüfen zu lassen und gültige Daten an den Roboter zu senden. Des Weiteren wird die aktuelle Roboterposition auf Anfrage an das Applet gesendet.

Wenn das Servlet eine Positionsanfrage bekommt, sendet es die Position, welche stets im Server vorgehalten wird, an den anfragenden Client. Falls dem Servlet eine neue Soll-Position übermittelt wird, prüft das Servlet ob der Client zurzeit berechtigt ist Positionen vorzugeben. Sollte dies der Fall sein wird die neue Position an einen Prüfalgorithmus übergeben. Dieser Prüfalgorithmus testet, wie Kapitel (4) genauer erläutert, ob die neue Position gültig ist und schreibt nur gültige Werte in ein Modul zur Datenverwaltung der Soll-Positionen.

Der Hardwarezugriff

Zu den weiteren Aufgaben des Servers gehört es den Zugriff auf das vorhandene Teilsystem zu realisieren. Dieser erfolgt über das Firewire Kommunikationssystem und die dazugehörige API. Der Hardwarezugriff muss sequenziell erfolgen, damit sich Lese- und Schreibzugriffe nicht überschneiden. Der Zugriff auf die Hardware erfolgt in einer Endlosschleife: Die Roboter Position wird, wie später genauer erläutert, ausgelesen. Wenn eine neue Soll-Position vorhanden ist wird diese an die Roboter-Hardware übermittelt und der nächste Schleifendurchlauf wird gestartet.

Die Multi-User-Strategie

Da die Anwendung eine Internet-Applikation ist, können potenziell mehrere Benutzer zur gleichen Zeit auf die Robotersteuerung zugreifen. Ein Servlet unterstützt grundsätzlich eine Multi-User-Nutzung. Ein technisches System bietet eine solche Unterstützung für die Steuerung nicht. Da die Resource Roboter nur einmal vorhanden ist muss gewährleistet werden, dass nicht mehr als ein Benutzer zur gleichen Zeit neue Soll-Werte an den Roboter senden kann. Um dies zu garantieren, wird eine Verwaltung der neuen Soll-Positionen implementiert. Hier werden die korrekten, aber noch nicht an den Roboter gesendeten, neuen Soll-Werte zwischengespeichert. Weiter wird über eine Sitzungsverfolgung festgehalten welcher Benutzer die Daten gesendet hat. Um die Steuerung (für einen kurzen Zeitrahmen) exclusive an einen Benutzer zu geben, kann nur derjenige neue Werte in die Datenverwaltung schreiben der auch die anderen Werte gesendet hat. Wenn keine neuen Soll-Positionen vorhanden sind darf jeder schreiben.

Für die Visualisierung des Systems ist ein Multi-User-Betrieb recht einfach zu realiesieren. Um nicht bei jeder Positionsanfrage eines Clients auch eine Anfrage an das Robotersystem stellen zu müssen, werden solange das Datenverwaltungsmodul keine neuen

Soll-Werte liefert alle ca. 100ms (jeder 100ster Schleifendurchlauf) Positions-Anfragen an die Roboter-Hardware gesendet. Und auf dem Server gespeichert. Bei einer Positionsanfrage des Clients wird der gespeicherte Wert zurückgegeben. Diese Abtastrate ist hoch genug da die Übermittlung über das Netzwerk zum Client länger dauert. Nach Übermittlung einer neuen Soll-Position wird eine Positionsabfrage forciert.

3.4.3 Sicherheits-Strategie

Eine geforderte Eigenschaft des Systems ist: Das System soll bezüglich falscher Eingaben sicher sein. Um eine solche Sicherheit zu gewährleisten, wird wie folgt vorgegangen:

- Der Server ist über eine exclusive Leitung an das Rabbit-System angeschlossen.
- Es wird garantiert das nur solche Daten an das Rabbit gesendet werden, die eine gültige Position beschreiben.

Das Kollisionstest-Modul nimmt die zentrale Rolle in der Sicherheits-Strategie ein. Es ist ausführlich in Kapitel (4) beschrieben. Dieses Modul ist auf dem Server untergebracht, denn nur an dieser Stelle kann sichergestellt werden, dass alle Daten geprüft werden bevor sie an den Roboter gesendet werden.

3.5 Das Lösungskonzept im Überblick

Es ist ein Software-System erstellt worden, welches die Hauptaufgabe hat ein Technisches System, hier ein fünf-Achsen Knickarmroboter, via Internet zu steuern. Der Benutzer kann den Roboter auf zwei verschieden Arten steuern: In einem dreidimensionalen VRML- Modell des Roboters und in einem klassischen Dialogfeld, dem Applet. Das Applet und VRML-Modell kommunizieren miteinander und synchronisieren die Ist- und Soll-Werte. Das Applet nimmt via Internet Kontakt zu dem Servlet auf und sendet neue Soll-Werte oder fordert neue Ist-Werte an. Damit das System sicher ist, überprüft das Servlet ankommende Soll-Werte auf Gültigkeit und leitet diese über eine definierte Schnittstelle (Kommunikations-API) an das vorhandene Teilsystem weiter. Auf Anfrage sendet das Servlet die Ist Werte an das Applet.

4 Kollisionskontrolle

Der Grundgedanke der Kollisionsvermeidung ist ungültige Roboterpositionen zu verhindern. Dazu kommt, das immer dann, wenn die an den Roboter gesendeten Daten eine gültige Endposition darstellen, auch ein gültiger Weg zu der Position ermöglicht wird. Daher muss der neue Soll-Wert auf Gültigkeit geprüft werden. Das Verfahren zur Berechnung wird in den folgenden Abschnitt erläutert. Ein komplettes Beispiel befindet sich in Anhang (C).

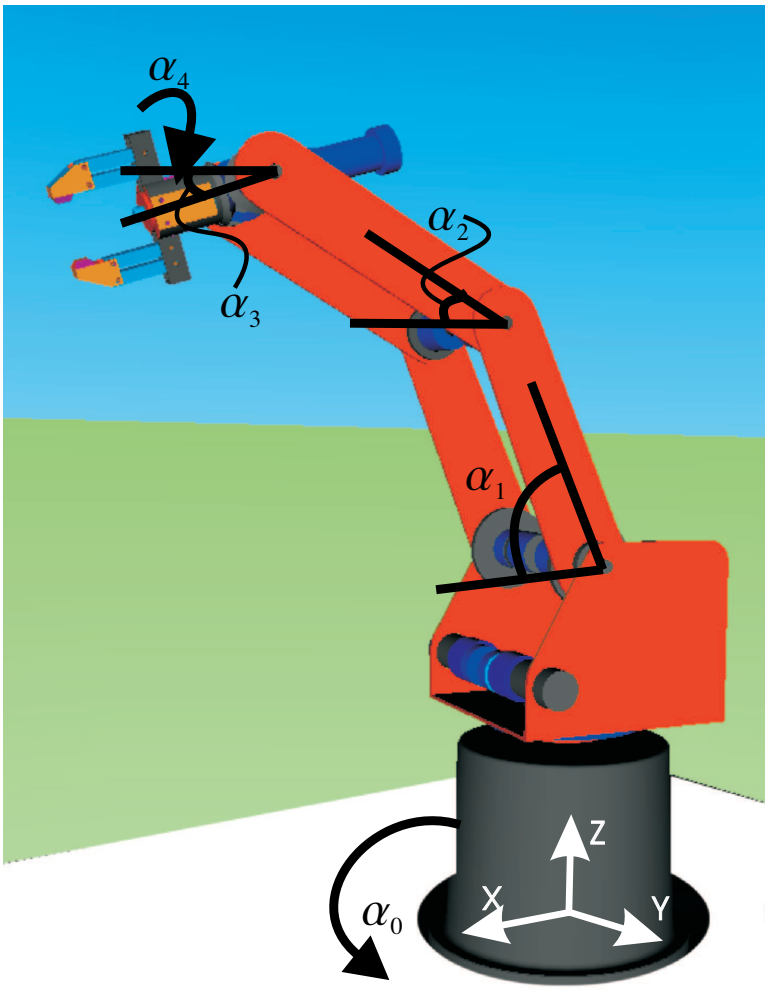

Abbildung 4.1: Winkel am Roboter

Vorbereitungen

Die Winkel für die mathematischen Bedingungen sind der Abbildung (4.1) zu entnehmen. Es handelt sich bei den Winkeln jeweils um den Winkel zwischen der X-Achse und der Mittellinie des entsprechenden Roboterarms:

α_0 beschreibt die Drehung um die Hochachse

α_1 ist der Winkel zwischen X-Achse und Arm 1

α_2 ist der Winkel zwischen X-Achse und Arm 2

α_3 ist der Winkel zwischen X-Achse und Arm 3

α_4 beschreibt die Rotation des Greifers

Es gibt zwei Arten von Bedingungen die erfüllt sein müssen: Statische und dynamische. Im Folgenden werden diese Bedingungen erläutert und beschrieben wann und wie sie abgeprüft werden müssen.

4.1 Statische Bedingungen

Diese Bedingungen sind im Verlauf der gesamten Anwendung fest. Es handelt sich hier um die Endlagen der einzelnen Arme/Achsen des Roboters. Die Endlagen des Roboters sind in der Abbildung (4.2) dargestellt. Diese Bedingungen sind mit mechanischen Endschaltern vergleichbar. Die Zahlenwerte müssen noch an die Endschalter angepasst werden. Diese sind im Roboter-Rabbit-System noch nicht integriert.

Diese Bedingungen können leicht geprüft werden. Es wird getestet, ob die neue Position innerhalb der Intervalle liegt.

	Minimalwert	Maximalwert
Winkel α_0	-150°	150°
Winkel α_1	-20°	180°
Winkel α_2	-180°	180°
Winkel α_3	-360°	360°
Winkel α_4	-180°	180°
Greiferöffnungsweite	0mm	50mm

Abbildung 4.2: Statische Endwerte für die Roboterachsen

4.2 Dynamische Bedingungen

Die dynamischen Bedingungen beschreiben die Möglichkeiten der Kollisionen die in den statischen Bedingungen nicht enthalten sind. Es ist möglich, dass der Roboter mit sich selbst kollidiert ohne, das eine der statischen Bedingungen eine Kollision beschreibt. Es währe zwar möglich, mit den statischen Bedingungen jegliche Kollision auszuschließen, aber damit würde der Bewegungsraum des Roboters stark eingeschränkt. Ein Beispiel hierfür wäre die Winkelkombination: $\alpha_2 = 130°$, $\alpha_3 = 180°$. Diese Kombination ist mit dem Winkel $\alpha_1 = 20°$ gültig und führt mit dem Winkel $\alpha_1 = -20°$ zu einer Kollision. Eine solche Kollision, bei der der Roboter mit sich selbst kollidiert wird im Folgenden Selbstkollision genannt. In dieser Arbeit wird eine Kollision mit der Umgebung nicht betrachtet, da in das vorhandene Teilsystem keine Sensoren zur Umgebungserfassung implementiert wurden.

Da nur die Selbstkollision vermieden werden soll ist es aus Gründen der Symmetrie nicht notwendig die Drehung um die Hochachse zu betrachten. Des Weiteren kann die Rotation des Greifers vernachlässigt werden, wenn der worst case angenommen wird. Für die dynamischen Kollisionsbedingungen genügt es also die Winkel $\alpha_1, \alpha_2, \alpha_3$ zu betrachten. Aus dieser Betrachtung folgt das in Abbildung (4.3) dargestellte Modell. Im Folgenden werden 11 Punkte eingeführt, die in den Berechnungen benötigt werden.

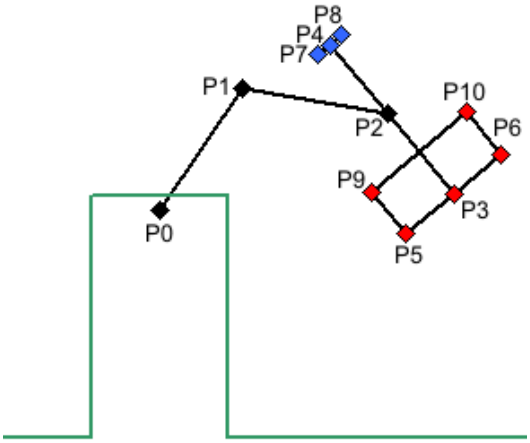

Abbildung 4.3: Modell zum Kollisionstest (Simulation in Excel)

Die im Modell dargestellten Punkte werden mathematisch wie folgt beschrieben:

$$
\begin{aligned}
P_0 &= (0, l_0)^T && (4.1)\\
P_1 &= P_0 + l_1 \left(\sin\left(\alpha_1\right), \cos\left(\alpha_1\right)\right)^T && (4.2)\\
P_2 &= P_1 + l_2 \left(\sin\left(\alpha_2\right), \cos\left(\alpha_2\right)\right)^T && (4.3)\\
P_3 &= P_2 + l_3 \left(\sin\left(\alpha_3\right), \cos\left(\alpha_3\right)\right)^T && (4.4)\\
P_4 &= P_2 - l_4 \left(\sin\left(\alpha_3\right), \cos\left(\alpha_3\right)\right)^T && (4.5)\\
P_5 &= P_3 + l_5 \left(\sin\left(\alpha_3 + 90°\right), \cos\left(\alpha_3 + 90°\right)\right)^T &&\\
\Rightarrow P_5 &= P_3 + l_5 \left(\cos\left(\alpha_3\right), -\sin\left(\alpha_3\right)\right)^T && (4.6)\\
P_6 &= P_3 - l_5 \left(\sin\left(\alpha_3 + 90°\right), \cos\left(\alpha_3 + 90°\right)\right)^T &&\\
\Rightarrow P_6 &= P_3 - l_5 \left(\cos\left(\alpha_3\right), -\sin\left(\alpha_3\right)\right)^T && (4.7)\\
P_7 &= P_4 + l_6 \left(\sin\left(\alpha_3 + 90°\right), \cos\left(\alpha_3 + 90°\right)\right)^T &&\\
\Rightarrow P_7 &= P_4 + l_6 \left(\cos\left(\alpha_3\right), -\sin\left(\alpha_3\right)\right)^T && (4.8)\\
P_8 &= P_4 - l_6 \left(\sin\left(\alpha_3 + 90°\right), \cos\left(\alpha_3 + 90°\right)\right)^T &&\\
\Rightarrow P_8 &= P_4 - l_6 \left(\cos\left(\alpha_3\right), -\sin\left(\alpha_3\right)\right)^T && (4.9)\\
P_9 &= P_5 - l_7 \left(\sin\left(\alpha_3\right), \cos\left(\alpha_3\right)\right)^T && (4.10)\\
P_{10} &= P_6 - l_7 \left(\sin\left(\alpha_3\right), \cos\left(\alpha_3\right)\right)^T && (4.11)
\end{aligned}
$$

Die Werte $\{l_0 \dots l_7\}$, siehe Abbildung (4.4) stammen aus der Roboter-Geometrie. Zur Bestimmung der Kollision soll der Abstand ausgewählter Punkte zu den Geraden durch die Drehpunkte der einzelnen Achsen berechnet werden. Die Gleichung für eine Gerade durch zwei Punkte $P_i = (x_i, y_i), P_j = (x_j, y_j)$ lautet:

$$\frac{y - y_i}{y_j - y_i} - \frac{x - x_i}{x_j - x_i} = 0 \tag{4.12}$$

Die Geradengleichung in der HESSEschen Normalform:

$$x \cos\left(\varphi\right) + y \sin\left(\varphi\right) - p = 0 \tag{4.13}$$

l_0	340
l_1	220
l_2	220
l_3	155
l_4	135
l_5	92.5
l_6	22.5
l_7	82

Abbildung 4.4: Längen am Roboter in mm

mit p als kürzester Abstand zwischen dem Ursprung und der Geraden (siehe Abbildung (4.5)), φ ist der Winkel zwischen der X-Achse und p . Aus der HESSEschen Normalform folgt der Abstand d zum Punkt $P_k = (x_k, y_k)$:

$$d = x_k \cos(\varphi) + y_k \sin(\varphi) - p \tag{4.14}$$

p bekommt man aus

$$p = a \sin(\alpha') \tag{4.15}$$

oder aus

$$p = b \cos(\alpha') \tag{4.16}$$

mit

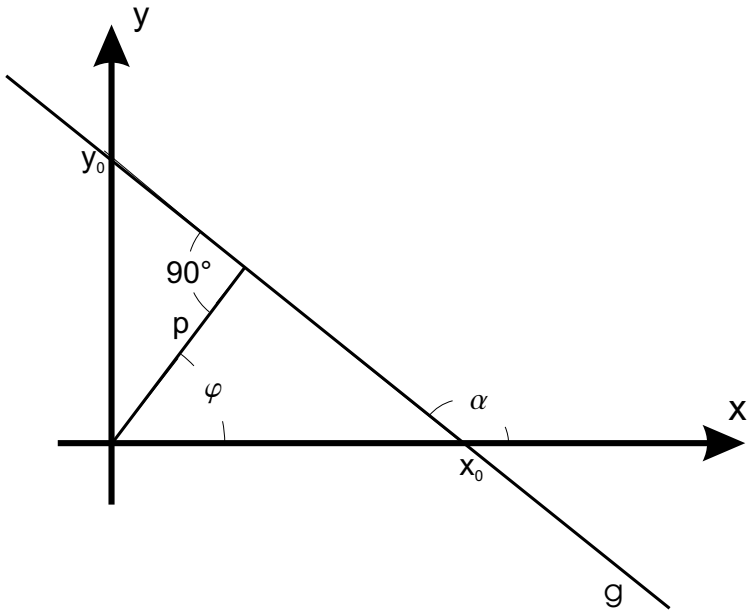

Abbildung 4.5: Skizze für p und α

$$\alpha' = \arctan\left(\frac{b}{a}\right) \tag{4.17}$$

und $a = x\,(y = 0)$ als Abstand zwischen dem Ursprung und dem Schnittpunkt der Geraden mit der X-Achse aus Gleichung (4.12)

$$a = \frac{x_j y_i - x_i y_j}{y_i - y_j} \tag{4.18}$$

bzw. $b = y\,(x = 0)$ als Abstand zwischen dem Ursprung und dem Schnittpunkt der Geraden mit der Y-Achse aus Gleichung (4.12)

$$b = \frac{x_j y_i - x_i y_j}{x_j - x_i} \tag{4.19}$$

Der Winkel φ :

$$\varphi = 90° - \alpha' \tag{4.20}$$

Um mit der Abstandsformel (4.14) eine Kollision zu ermitteln muss für jeden Arm ein Mindestabstand, wenigstens die halbe Breite des Arms, definiert werden. Die Werte für den Roboter sind der Tabelle in der Abbildung (4.6) zu entnehmen. Da mit der Formel (4.14) der Abstand zu einer Geraden ermittelt wird, muss der Gültigkeitsbereich für den Kollisionstest eingeschränkt werden. Der Punkt $P_k = (x_k, y_k)$, der auf Kollision getestet werden soll, muss in X- und in Y-Richtung zwischen den beiden Endpunkten $P_i = (x_i, y_i)\,, P_j = (x_j, y_j)$ des Arms liegen:

$$(x_i < x_k < x_j \vee x_i > x_k > x_j) \wedge (y_i < y_k < y_j \vee y_i > y_k > y_j) \Rightarrow \mathit{gueltig} \tag{4.21}$$

Eine Kollision besteht dann, wenn der Abstand $|d_{P_k}| < |d_i|$ und der Punkt $P_k = (x_k, y_k)$ im Gültigkeitsbereich ist.

Da in diesem konkreten Modell nicht die Endpunkte, sondern die Drehpunkte definiert sind, wird der Gültigkeitsbereich vergrößert. Für den Test mit Punkt P_5 auf Kollision mit Arm 1 folgt (P_0, P_1)

$$\begin{aligned}(x_0 - d_1 < x_5 < x_1 + d_1 \vee x_1 - d_1 < x_5 < x_0 + d_1)\\ \wedge (y_0 < y_5 < y_1 \vee y_1 < y_5 < y_0) \Rightarrow \text{gueltig}\end{aligned} \tag{4.22}$$

als Gütigkeitsbereich. Hier kann d_1 verwendet werden, da der Arm mit einem Radius um die Drehpunkte abgeschlossen wird.

Mit diesem Verfahren lässt sich jetzt feststellen ob ein Punkt mit einem Arm kollidiert, da der Roboter nicht aus einzelnen Punkten, sondern aus Elementen mit einer dreidimensionaler Ausdehnung besteht, von denen hier nur zwei berücksichtigt werden müssen. Es ist möglich, dass obwohl kein Punkt kollidiert, eine Kollision vorliegt. Genau dann, wenn eine Achse zwischen zwei Punkten hindurchpasst, siehe Abbildung (4.7). Dieses Problem ist hier nur deshalb relevant, weil nur eine Endpunktprüfung durchgeführt wird. Wenn jede Bewegung in kleine Segmente aufgeteilt werden würde, also die Bahn kontinuierlich getestet würde, könnte es zu einem solchen Fall nicht kommen.

Um diese Kollision auszuschließen kann man zwei Wege gehen: Die Gleichungen für Geraden durch (P_5, P_6) und (P_1, P_2) aufstellen, den Schnittpunkt errechnen und dann prüfen ob dieser zwischen P_{1x} und P_{2x} oder zwischen P_{1y} und P_{2y} liegt. Alternativ kann man in diesem Fall eine Grenze für den kleinsten Winkel zwischen Arm 2 und Arm 3

d_0	35
d_1	35

Abbildung 4.6: Mindestabstände am Roboter in mm

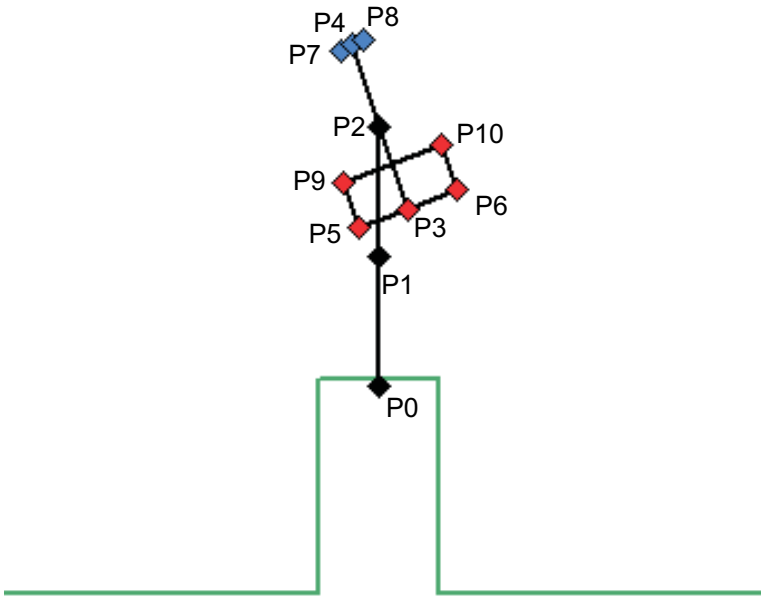

Abbildung 4.7: Ungültiger gültiger Endpunkt

angeben, laut Simulation liegt dieser bei 67°. Wegen der Symmetrie muss der kleinste eingeschlossene Winkel immer $\geq 67°$sein. Das Verfahren, welches hier zur Simulation eingesetzt wurde, ist das oben beschriebene. Die Formel für diesen Winkel lautet:

$$\min(|180° - (\alpha_2 - \alpha_3)|, |180° - (\alpha_3 - \alpha_2)|) \geq 67° \tag{4.23}$$

Dieses Verfahren kann hier nur deshalb angewendet werden, weil die beiden betroffenen Arme direkt hintereinander geschaltet sind. Wenn im allgemeinen Fall eine beliebige Strecke, die durch zwei Punkte definiert ist, auf Kollision mit einem Arm des Roboters getestet werden soll, muss eine Gerade durch die Endpunkte des Roboterarms gelegt werden. Allgemeine Geradengleichungen:

$$A_1 x + B_1 y + C_1 = 0 \tag{4.24}$$
$$A_2 x + B_2 y + C_2 = 0 \tag{4.25}$$

Der Schnittpunkt der beiden Geraden wird bestimmt:

$$x_0 = \frac{(B_1C_2 - C_1B_2)}{(A_1B_2 - B_1A_2)} \tag{4.26}$$

$$y_0 = \frac{(A_2C_1 - A_1C_2)}{(A_1B_2 - B_1A_2)} \tag{4.27}$$

Wenn sich der Punkt $P_0 = (x_0, y_0)^T$ auf dem Geradenabschnitt, der durch die Endpunkte des Roboterarms bestimmt wird, befindet, dann liegt ein Fall der Kollision vor.

Nun muss noch geprüft werden, ob eine Kollision mit dem Untergrund vorliegt. Dieser wird in dem Modell in Abbildung (4.3) durch die grüne Linie dargestellt:

$$f_{Untergrund}(x) = \begin{cases} 360 & : -110 \leq x \leq 110 \\ 0 & sonst \end{cases} \tag{4.28}$$

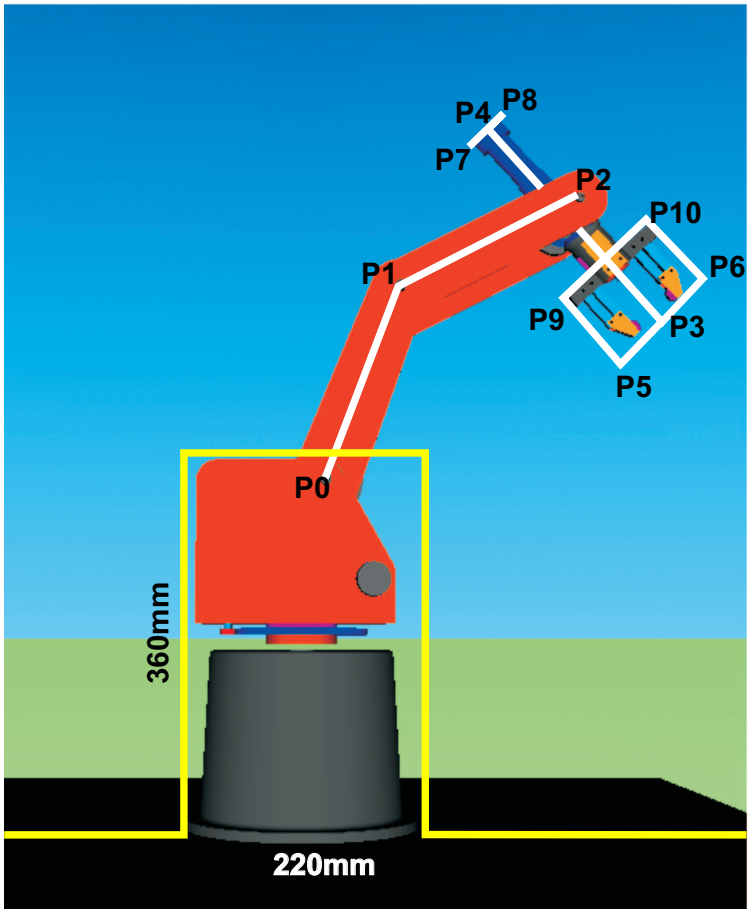

Abbildung 4.8: VRML-Modell und Kollisionstest Modell

In den Untergrund ist der Grundkörper des Roboters mit eingefasst. Zur Verdeutlichung ist in Abbildung (4.8) das Modell zum Kollisionstest über dem VRML-Modell dargestellt. In dieser Darstellung ist $f_{Untergrund}$ gelb gezeichnet. Eine Kollision von Punkt $P_k = (x_k, y_k)$ mit dem Untergrund liegt dann vor wenn:

$$f_{Untergrund}(x_k) > y_k \tag{4.29}$$

In dem erstellten Programm werden die Punkte $(P_2, ..., P_{10})$ auf Kollision mit Arm 1 (die Strecke zwischen P_0 und P_1), den eingeschlossenen Winkel zwischen den den Strecken (P_1, P_2) und (P_2, P_3) und die Punkte $(P_1, ..., P_{10})$ auf Kollision mit dem Untergrund getestet.

4.3 Simulation

Zum Testen der Kollisionsvermeidung ist in einer Excel Tabelle eine Simulation erstellt worden. Excel bot sich aus zwei Gründen an: Die Visualisierung der einzelnen Bedingungen ist sehr übersichtlich zu realisieren und man kann es mit wenig Aufwand variieren. Die Abbildung (4.9) zeigt eine Übersicht der einzelnen Kollisionsbedingungen. Hier ist leicht zu sehen welche der Bedingungen eine Kollision meldet. Um das Excel Modell mit der späteren Java Implementierung vergleichen zu können ist ein Makro (siehe Anhang (E.4)) erstellt worden mit dem die Winkel variiert werden können, und auf Wunsch eine Wertetabelle angelegt werden kann.

4.4 Ergebnis

In diesem Kapitel ist ein Verfahren beschrieben worden, mit dem unter bestimmten Voraussetzungen die Kollision von Roboterkomponenten vermieden werden kann. Es sind statische und dynamische Bedingungen für die Kollision aufgestellt worden. Die statischen Bedingungen repräsentieren die Endlagen bzw. Endanschläge der einzelnen Komponeten des Roboters. Mit den dynamischen Bedingungen wird geprüft, ob in der zu testenden Position eine Kollision besteht obwohl die statischen Bedingungen keine Kollision anzeigen. Um die Berechnungen der dynamischen Bedingungen in einer möglichst kurzen Zeit durchzuführen ist ein einfaches Modell des Roboters erstellt worden. Dieses Modell vernachlässigt die Drehung um die Hochachse (α_0) und die Rotation des Greifers (α_5). Daraus ergibt sich dann ein rein zweidimensionales Modell, in dem dann mit Hilfe einer Abstandsgleichungen getestet wird, ob die Menge zu testender Punkte eine Kollision erzeugt. Die Rotation des Greifers wird durch eine Worst-Case-Betrachtung abgeschätzt. Diese Kollisionsvermeidung kann im Bedarfsfall auf mehrere interagierende Roboter erweitert werden. Dafür müssen die Geradengleichungen ins Dreidimensionale erweitert werden. Das heist die Drehung um die Hochachse muss dann berücksichtigt werden.

Bedingungen für den Fall keiner Kollision

Statische Bedingungen

	Untergrenze	Obergrenze	Winkel	rel Winkel	Keine Kollision
Winkel alpha 1 in den Grenzen	-20	120	70		WAHR
Winkel alpha 2 in den Grenzen	-180	180	-70	140	WAHR
Winkel alpha 3 in den Grenzen	-180	180	-30	220	WAHR
					WAHR

Dynamische Bedingungen

	X-Bed.	Y-Bed.	Abstands Bed	Keine Kollision
Arm 1 P2	WAHR	FALSCH	WAHR	WAHR
Arm 1 P3	FALSCH	FALSCH	WAHR	WAHR
Arm 1 P4	WAHR	WAHR	FALSCH	FALSCH
Arm 1 P5	WAHR	FALSCH	WAHR	WAHR
Arm 1 P6	FALSCH	FALSCH	WAHR	WAHR
Arm 1 P7	WAHR	WAHR	FALSCH	FALSCH
Arm 1 P8	WAHR	WAHR	FALSCH	FALSCH
Arm 1 P9	WAHR	FALSCH	WAHR	WAHR
Arm 1 P10	FALSCH	FALSCH	WAHR	WAHR

	X-Bed.	Y-Bed.	Abstands Bed	Keine Kollision
Arm 2 P3	FALSCH	FALSCH	WAHR	WAHR
Arm 2 P5	WAHR	FALSCH	WAHR	WAHR
Arm 2 P6	FALSCH	FALSCH	WAHR	WAHR
Arm 2 P9	WAHR	FALSCH	WAHR	WAHR
Arm 2 P10	FALSCH	FALSCH	FALSCH	WAHR

	Y-Wert Punkt	Y-Wert Boden	Keine Kollision
Boden P1	425.24	0	WAHR
Boden P2	500.49	360	WAHR
Boden P3	630.39	360	WAHR
Boden P4	374.92	360	WAHR
Boden P5	660.39	360	WAHR
Boden P6	600.39	0	WAHR
Boden P7	386.17	360	WAHR
Boden P8	363.67	360	WAHR
Boden P9	589.38	360	WAHR
Boden P10	529.38	360	WAHR

Gesamt	FALSCH

Abbildung 4.9: Überprüfung der Kollisionsbedingungen in einer EXCEL-Tabelle

5 Robotersteuerung

In diesem Kapitel wird zunächst die Implementierung des Lösungskonzeptes vorgestellt. Im weiteren Verlauf wird dann die Inbetriebnahme und der Funktionstest erläutert

5.1 Implementierung

Die Implementierung gliedert sich in drei Teile: Als erstes wird die Schnittstelle zwischen dem Server und dem Client implementiert. Im Anschluss wird die Implementierung des Servers und dann die des Clients erläutert.

5.1.1 Die Client - Server Schnittstelle

Da das komplette Programm Design von der Kommunikation der einzelnen Komponenten abhängt wird hier mit der Applet-Servlet Kommunikation begonnen. Zur Auswahl stehen die folgenden fünf Verfahren:

- Text über Http
- Text über Socket
- Objekt über Http
- Objekt über Socket
- Remote Method Invocation

Bei diesen Verfahren scheidet die Remote Method Invocation direkt aus, da für die Applet-VRML Kommunikation ein VRML-Browser mit funktionierendem external authoring interface benötigt wird. Der VRML-Browser bietet diese Funktionalität nur in Zusammenarbeit mit der Microsoft Java Virtual Maschine (JVM). Die aktuellste MS JVM ist in etwa auf dem Stand der JVM 1.1 von Sun. Für die RMI ist die Version 1.3.1 Voraussetzung. Die direkte Socket-Kommunikation ist aufgrund der Firewall-Problematik ungünstig und bietet nur wenige Vorteile gegenüber der Kommunikation via Http. Daher fiel die Wahl auf die Http-Kommunikation, wobei sich jetzt noch die Frage stellt, ob die klassische Textversendung oder die Objektübertragung verwendet wird. Wegen der

höheren Transparenz der Schnittstelle wird die Variante „Objekt über Http“ ausgewählt. Die Daten die zwischen dem Servlet und dem Applet ausgetauscht werden müssen sind im Wesentlichen die Positionsdaten, zu denen kommen noch weiter Daten die nicht direkt der Hauptaufgabe zugeordnet werden: Eine Session ID, eine Fehler Nummer und einen Nachrichtentext. Jede Instanz einer Klasse stellt in Java ein Objekt dar. Wenn ein Objekt von einem Java Programm aus an eine, aus Sicht des Programms, externe Stelle geschrieben werden soll, muss das Objekt serialisiert, das heißt in einen Bytecode umgewandelt werden. Damit eine Java Klasse serialiesiert werden kann, muss sie das Interface Serializable implementieren. Ein so bearbeitetes Objekt kann nun gespeichert oder auch versendet werden. Der Kopf der Klasse sieht wie folgt aus:

```
import java.io.Serializable;
public class roboPosition implements Serializable{
  int          errorNo          =  0;
  float[]      radPosition      =  {0,0,0,0,0,0,0};
  int          sessionID        =  0;
  String       msgString        =  "";
  ...
  }
```

Weiter besitzt die Klasse einen Konstruktor und einige `set`, `get` und `toString` Methoden. Der kommplette Quelltext ist im Anhang (E) zu finden. Nachdem jetzt die Schnittstelle definiert, ist wird die Implementierung der Serverseite erläutert.

5.1.2 Der Server

Eine Hauptfunktion des Servers ist es die Anfragen des Clients zu bearbeiten und die Antworten an diesen zurückzusenden. Der Aufbau der Klassen sieht wie folgt aus (siehe Abbildung (5.1)): `RoboServlet` ist die Basisklasse. Hier wird die Kommunikation via Netzwerk realisiert. Die Klasse `HardwareInterface` übernimmt vielfältige Aufgaben. Die wichtigste ist die Kommunikation mit der Schnittstelle des vorhandenen Teilsystems. Sie hält immer eine aktuelle Position des Roboters vor und organisiert die Datenhaltung für neue Soll- Positionen. Die Kollisionsvermeidung ist ebenfalls in der Klasse `collisionTest` angesiedelt. Die Klassen `roboPositionQueue` und `roboDemoQueue` sind mit der Klasse `queueElement` für die Datenhaltung zuständig. Die Klasse `roboPosition` ist die Schnittstelle zum Client.

Die Internet-Kommunikation

Für die Internet-Kommunikation ist auf dem Server ein spezielles Java Programm, ein Servlet, zuständig. Damit eine Java Applikation zu einem Servlet wird, muss das Programm den folgenden Grundaufbau haben: Das Servlet muss von der Klasse `HttpServlet` erben:

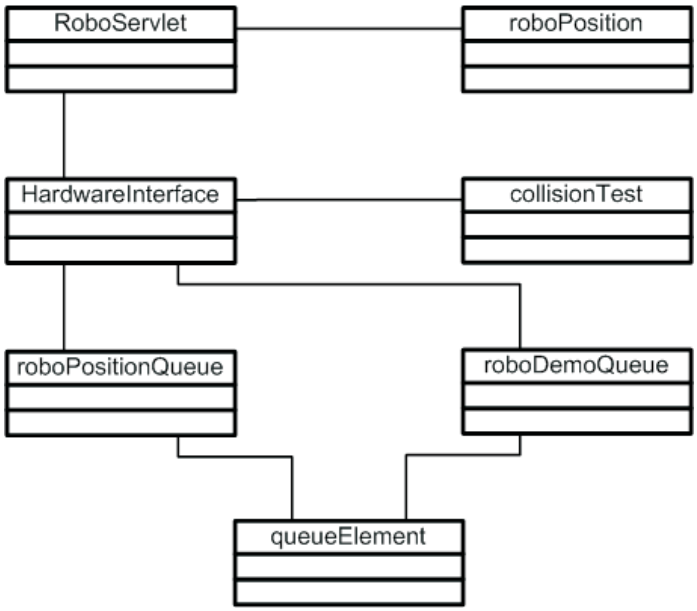

Abbildung 5.1: Klassendiagramm Server

```
public class RoboServlet extends HttpServlet
```

Damit diese Klasse und weitere benötigte Klassen zu Verfügung stehen werden die Pakete `javax.servlet` und `javax.servlet.http` eingebunden. Das Servlet benötigt für jeden Typ vom Http-Anfragen (`get`, `post`,...), auf die es antworten soll, eine eigene Methode. Die Namen der Http-Anfragen sind aus Sicht des Client zu verstehen und so in der Http-Spezifikation definiert. Bei der Robotersteuerung gibt es zwei Arten von Anfragen des Applets, die der Server bearbeiten muss: Zum einen die Anfrage nach der momentanen Position des Roboters, zum andern die Übermittlung einer neuen Soll-Position für den Roboters. Daher sind zwei Methoden implementiert worden: `doGet` und `doPost`.

doGet Die `doGet` Methode sendet, wenn sie aufgerufen wird, ein aktuelles `roboPosition` Objekt an den Client. Der Methodenkopf sieht wie folgt aus:

```
public void doGet(HttpServletRequest req, HttpServletResponse ↩
    res) throwsServletException, IOException
```

- Der Parameter `req` ist ein Objekt, welches die Anfrage des Clients beinhaltet, vom Typ `HttpServletRequest`.
- Der Parameter `res` ist ein Objekt, in das die Antwort des Servers geschrieben werden kann. Es ist vom Typ `HttpServletResponse`.

Da das Servlet, gleich bei welcher Get-Anfrage, immer ein Objekt mit der aktuellen Roboter-Position an den Client senden soll, kann der Inhalt des

`HttpServletRequest` Objekts ignoriert werden. Die Methode fragt also bei dem `HardwareInterface` nach der aktuellen Position und sendet diese als `roboPosition` Objekt, mittels des `res` Parameters, an den Client. Wenn der Zugriff auf das `HardwareInterface` fehlschlägt wird eine entsprechende Fehlernachricht in den `msgString` des `roboPosition` Objekts geschrieben. Um ein Objekt zu versenden muss ein Objekt-Output-Stream geöffnet werden. Die Methoden für die Arbeit mit Streams gehören zu dem Packet `java.io`, das auch importiert wird. Um in das `HttpServletRespose` Objekt schreiben zu können, stellt das Objekt eine Methode `getOutputStream()` bereit. Das Öffnen des Objekt Output Stream sieht dann so aus:

```
ObjectOutputStream out = new ObjectOutputStream (res. ↩
    getOutputStream());
```

Eine Kopie des aktuellen `roboPosition` Objekts wird, mit dem folgendem Aufruf, in den Objekt Output Stream geschrieben: `out.writeObject(newRoboPos)`. Die Kopie wird in diesem Schritt auch automatisch seriallisiert.

doPost Die `doPost` Methode ist für das Empfangen von neuen Soll Positionen zuständig. Der Methodenkopf ist fast identisch mit dem der `doGet` Methode:

```
public void doPost(HttpServletRequest req, HttpServletResponse ↩
    res) throws ServletException, IOException
```

Die übergebenden Objekte sind völlig analog zur `doGet` Methode. Hier muss als erstes die ankommende `roboPosition` eingelesen werden. Hierfür wird ein Objekt Input Stream geöffnet. Das `HttpServletRequest` Objekt hat eine Methode `getInputStream`:

```
ObjectInputStream in = new ObjectInputStream(req. ↩
    getInputStream());
Object obj = in.readObject();
```

Damit ist das empfangende Objekt eingelesen und deserialisiert. Das Objekt wird nun zwangskonvertiert, in ein `roboPosition` Objekt. Die Fehlernummer wird ausgelesen und ausgewertet da sie noch weitere Anweisungen für das Servlet enthalten kann. Dann werden die Daten mit der Methode `setRobotData` an die Hardware gesendet. Diese Methode gibt eine Fehlernummer zurück, die mittels eines Objekt Output Streams, der `doGet` Methode entsprechend, an das Applet übermittelt wird. Die Übergabe der neuen Soll Position an das `HardwareInterface` wird mit einem `synchronized(this){...}` Block synchronisiert. Wenn ein Stream nicht mehr benötigt wird, wird er mit der Methode `close` geschlossen. Bei einem Output Stream wird vorher mit der Methode `flush` dafür gesorgt das die gepufferten Daten in den Stream geschrieben werden.

Die Sitzungsverfolgung

Damit immer nur ein Benutzer zur gleichen Zeit Befehle an den Roboter geben kann, ist es notwendig, die verschiedenen Clients zu identifizieren. Da die Lebensdauer einer http-Verbindung auf eine einzige Anfrage-Antwort Kombination begrenzt ist, ist es nicht möglich den Client anhand der Verbindung zu identifizieren. Als Alternative kann der Client sich selbst identifizieren, indem er dem Server bei jedem Verbindungsaufbau eine zuvor vom Server vergebende Kennung übermittelt. Wenn ein Client, der noch keine Sitzungs-ID zugewiesen bekommen hat, eine neue Soll Position an den Roboter übermitteln möchte, wird eine Null als Sitzungs-ID übermittelt. Das Servlet vergibt dann die nächste freie ID an diesen Client. Als ID wird eine Zahl von Typ Integer genutzt. Aus Sicherheitsaspekten wäre es günstiger eine komplizierter zusammengesetzte ID zu wählen. Davon wurde hier abgesehen, da die Daten per Java-Objekt und nicht im Klartext übermittelt werden. Damit das Servlet, auch wenn mehrere Instanzen existieren, jede ID nur einmal vergibt ist die Variable lastSessionID als static deklariert, also eine Klassenvariable. Die Sitzungsverfolgung wird nur bei der Übertragung neuer Soll Koordinaten benutzt. Bei der Positionsabfrage ist es unwichtig welcher Client die Daten anfordert.

Die Verwaltung der Soll Positionen

Die Idee ist es, die Antwortzeit einer Anfrage zu minimieren, damit der Client nicht lange auf die Antwort warten muss. Daher bietet es sich an, die Daten auf dem Server vor dem Weiterleiten an den Roboter, in einer FIFO (first-in-first-out) Queue zwischenzuspeichern. Die Daten, die in den Elementen dieser Queue gespeichert werden, bestehen lediglich aus einem Array neuer Koordinaten und einer Referenz auf das nächste Element. Die Queue ist mit Hilfe der Klasse `roboPositionQueue` realisiert. Die Elemente sind vom Typ `queueElement`. Die Klasse `roboPositionQueue` bietet die Methoden: `hasElements`, `getPosition` und `addPosition`. Bevor ein Wert in die Queue geschrieben wird muss geprüft werden, ob der Client berechtigt ist in die Queue zu schreiben und ob die neue Sollposition zu keiner Kollision führt. Die Berechtigung in die Queue zu schreiben hat der Client genau dann, wenn die Queue leer ist, oder die Elemente in der nicht leeren Queue von ihm sind.

Die Kollisionsvermeidung

Bevor die neuen Positionsdaten in die Queue der zu sendenden Daten eingetragen werden können müssen sie auf Kollision getestet werden. Das Prinzip ist im Kapitel (4) beschrieben. Die Methode `test` ist in der `testCollision` Klasse angeordnet. Immer wenn eine neue Soll- Position mit der Methode `setRobotData` an das `HarwareInterface` Objekt gesendet wird, wird geprüft, ob der Sendende Client in die Queue schreiben darf. Wenn das der Fall ist wird die Position an die `test` Methode übergeben. Hier wird zunächst auf alle Werte 10^{-9} addiert, um eine mögliche Division durch Null zu vermeiden. Als

erstes wird auf ein Überschreiten der statischen Grenzen getestet. Sobald ein Test nicht „bestanden“ wird, bricht die Methode ab und gibt den Wahrheitswert „Falsch“ zurück. Im Anschluss wird getestet, ob der eingeschlossene Winkel zwischen den Armen 2 und 3 (siehe Abbildung (4.7)) den Anforderungen genügt. Damit sind alle Bedingungen, die mit sehr geringen Rechenaufwand zu testen sind, überprüft. Es folgt die Berechnung der Punkte mit dem vorgestellten Verfahren. Mit wenig Aufwand lässt sich nun eine Kollision mit der Bodenfunktion überprüfen. Bei bestandenem Test werden die Abstände der Punkte zur Geraden durch Arm 1 berechnet. Wenn die Punkte im Kollisionsraum sind wird geprüft ob der Abstand zu gering ist. Für die Kontrolle der Abstände zu Arm 2 wird analog vorgegangen. Der Ablauf des Tests ist in dem Flussdiagramm (5.2) visualisiert.

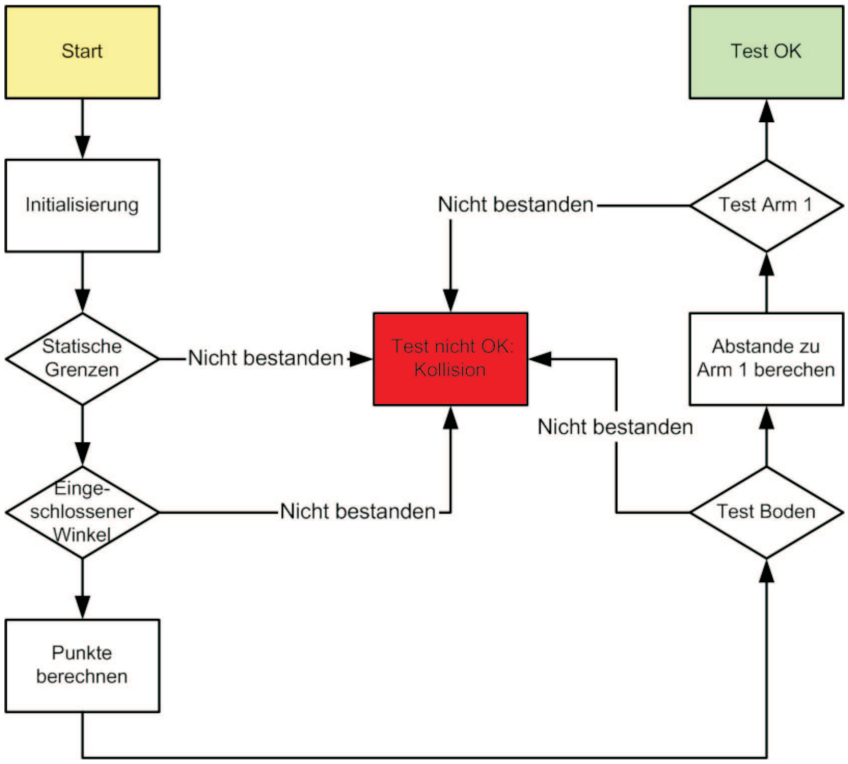

Abbildung 5.2: Flussdiagramm Kollisionsvermeidung

Die Alternative Bahnvorgabe

Alternativ zur Vorgabe einzelner Punkte durch den Client kann auch eine vordefinierte Folge von Punkten abgefahren werden. Dazu gibt es eine extra Klasse, `roboDemoQueue`, die zur Übersetzungszeit mit einer Punktfolge versehen wurde. Wenn der Konstuktor aufgerufen wird, erstellt sich ein Ring aus `queueElement` Objekten bei der das letzte Element auf das Erste zeigt. Bei der FIFO- Queue zeigt das letzte Element ins „Leere“.

Das Umschalten zwischen Demo-Modus und dem Standard Betriebszustand erfolgt indem ein Client bei einer Positionsübermittlung die Fehlernummer entsprechend setzt. Eine 0 verändert nichts, eine 1 schaltet die Demo ein und alle anderen Werte stoppen die Demo.

Der Zugriff auf die Hardware

Der Zugriff auf die Hardware bzw. auf die Kommunikations- API erfolgt sequenziell. Die Klasse `HardwareInterface` erzeugt, bei einem Aufruf des Konstruktors, einen eigenen Thread. In diesem wird dann der Zugriff auf die API realisiert.

Die Kommunikations-API besteht aus zwei Methoden die aus der Java-Anwendung über das Java Native Interface (JNI) angesprochen werden können. Um das in C programmierte API mit dem JNI anzusprechen, muss die Bibliothek geladen werden und die Methoden initialisiert werden:

```
private native double[] receiveRobotData();
private native int      sendRobotData(int channelNumber, ↩
   double[] motorData);
static {System.loadLibrary("HardwareInterface");}
```

Beim Übersetzungsvorgang muss zusätzlich ein Header-File erzeugt werden. Ein entsprechendes Tool ist im Java Software Development Kit beigelegt. Jetzt können die Methoden, genau wie gewöhnliche Java Methoden, genutzt werden. Genauere Informationen zum JNI sind in [12] zu finden. Die Methode `sendRobotData` sendet ein double-Array mit einer neuen Soll-Position an den Roboter und die Methode `receiveRobotData` liefert ein double-Array mit der aktuellen Soll-Position zurück.

Der gesamte Hardwarezugriff läuft in einer Endlosschleife ab, siehe Abbildung (5.3). In jedem 100sten Schleifendurchlauf wird die aktuelle Roboter Position erfragt und in die entsprechende Positions-Variable geschrieben. In jedem Durchlauf der Schleife wird geprüft ob der Demo-Modus oder der Normale Steuer-Modus aktiv ist. Wenn der Steuer-Modus Aktiv ist, wird geprüft, ob in der Queue ein Element vorhanden ist, vgl. Abbildung (5.3) „Neue Position?“. Dieses Element, falls vorhanden, wird an den Roboter gesendet und eine Positionsabfrage für den nächsten Schleifendurchlauf wird forciert. Wenn Demo Modus aktiv ist wird geprüft, ob die letzten Soll-Werte erreicht worden sind, und wenn das der Fall ist wird die nächste Position gesendet.

Eine Abfrage ob eine Position erreicht ist wird nur in dem Demo-Modus durchgeführt. Eine Solche Abfrage im Steuer-Modus würde das zeitnahe Ansteuern des Roboters deutlich erschweren. Dies ist in der beinahe kontinuierlichen Sendung neuer Soll-Positionen begründet. Da mit der „Position-Erreicht“-Abfrage jeder dieser Soll-Positionen genau angefahren würde, wäre die Bewegung des Roboters deutlich weniger flüssig.

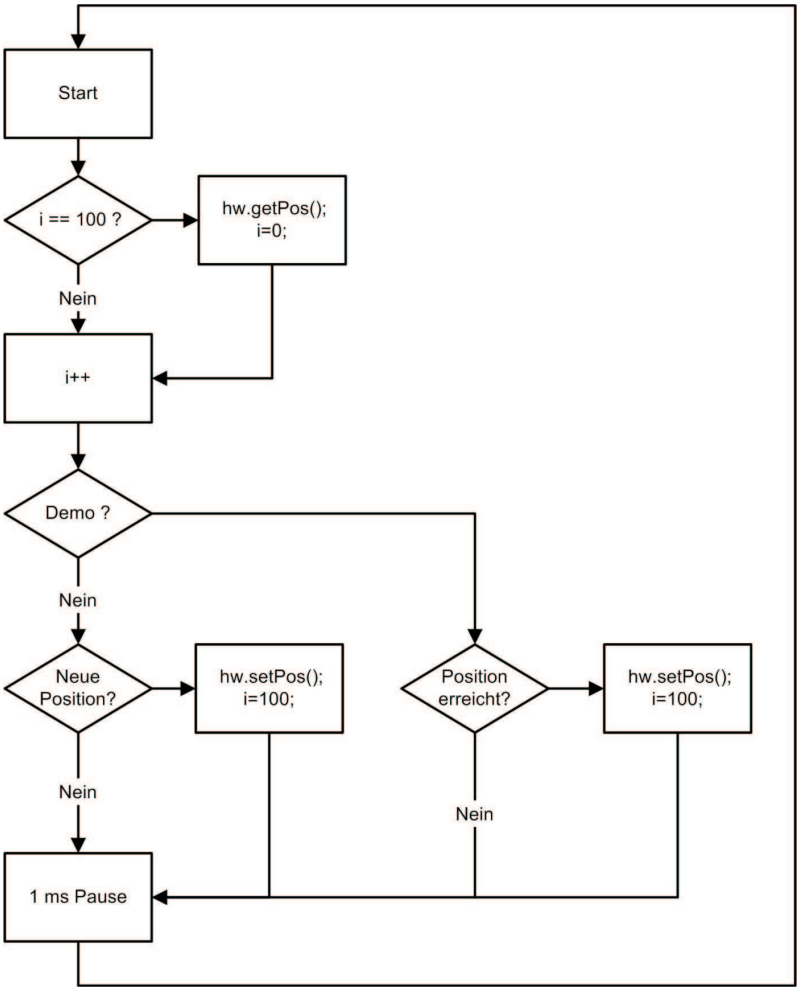

Abbildung 5.3: Flussdiagramm Hardware Zugriff

5.1.3 Der Client

Die Hauptfunktion des Clients ist es die Benutzereingaben an den Server zu senden und die aktuelle Roboterposition zu visualisieren. Für die Erfüllung dieser Aufgabe gibt es die Klassen: `RoboApplet`, `vrmlRobo`, `HttpMessage` und `Base64Encoder` siehe Abbildung (5.4). Die Klasse `roboPosition` ist die Schnittstelle zum Server.

Die Netzwerkkommunikation

Die Kommunikation unterscheidet sich nur unwesentlich von der Kommunikation des Servlets. Im Gegensatz zu den Servlets gibt es keine Standard Klasse, wie etwa `HttpApplet`, welche die Verbindungsmethoden implementiert. Dafür sorgen zwei Klassen aus dem Package `com.oreily.servlet`: `HttpMessage` und `Base64Encoder`.

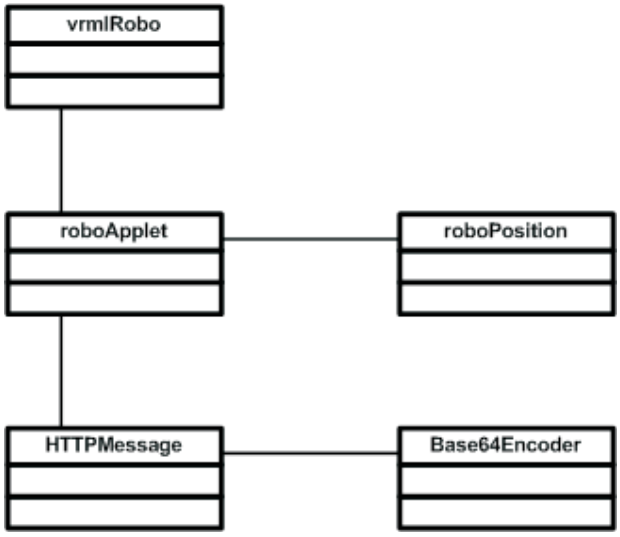

Abbildung 5.4: Klassendiagramm Client

Die Klasse `HttpMessage` implementiert Methoden zum Öffnen einer Http Verbindung: `sendGetMessage` und `sendPostMessage`. Die beiden Methoden sind in verschieden Varianten implementiert. Die Methode `sendGetMessage` existiert in zwei Varianten: Zum einen in einer parameterlosen Form und zum anderen mit der Möglichkeit eine Eigenschafts-Liste an die URL anzuhängen. Bei den `sendPostMessage` Methoden gibt es noch eine zusätzliche Variante um ein serialsiertes Objekt anzuhängen. Die Klasse `Base64Encoder` wird benötigt falls eine Https-Verbindung aufgebaut werden soll, was in dieser Arbeit aber nicht gemacht wird. Der Quellcode zu diesen Klassen ist im Anhang (E.3) zu finden. Diese Klassen sind nicht an Applets gebunden. Sie können aus jedem Java Programm genutzt werden. Um vom Applet aus, die aktuelle Roboter Position bei dem Server zu erfragen, wird ein URL Objekt erstellt, welches die Internetadresse des Servlets enthält.

```
URL url = new URL(getCodeBase(),"/servlet/RoboServlet");
```

Mit dem URL Objekt als Parameter wird ein neues Objekt vom Typ `HttpMessage` erstellt:

```
HttpMessage msg = new HttpMessage(url);
```

Dann wird ein Input- Stream für die Antwort des Servers geöffnet:

```
InputStream in = msg.sendGetMessage();
```

Die Antwort muss nun nur noch konvertiert werden und die neue Position im Applet verarbeitet werden. Um eine neue Soll-Position an den Server zu senden wird analog vorgegangen. Mit dem Unterschied, dass beim Öffnen des Input-Streams die Methode `sendPostMessage` verwendet wird. Das Objekt `localRoboPos` ist vom Typ `roboPosition` und enthält die neuen Sollwerte.

```
InputStream in = msg.sendPostMessage(localRoboPos);
```

Nach der Konvertierung werden die dann nicht mehr benötigten Streams mit der Methode `close` geschlossen.

Der Kommunikationsablauf

Beim starten des Applets wird ein Thread erzeugt, der ständig die Methode `getNextMessage` ausführt. Diese Methode ist für die Anforderung von neuen Ist-Positionen des Roboters zuständig. Durch einen `synchronized(this){...}` Block wird erreicht das maximal eine Abfrage zur Zeit von einem Applet an das Servlet gesendet wird. Das Senden einer neuen Soll-Position erfolgt immer infolge eines Events. Ein entsprechendes Event wird durch die Veränderung der lokalen Soll-Position durch den Benuzter ausgelöst.

Die Benutzerschnittstelle

Die Benutzerschnittstelle besteht aus 2 Elementen:

1. Applet
2. VRML- Modell

Abschließend werden Applet und VRML-Model in eine JSP-Seite eingebunden.

Applet Das Applet baut, sobald es initialisiert wird, eine Grafische Benutzeroberfläche auf, siehe Abbildung (5.5). Diese besteht aus einigen Bedien- und Anzeigeelementen. Für jeden Stellmotor am Roboter gibt es eine Scrollbar zum Einstellen der Soll-Werte und zwei Winkelanzeigen. Eine für den Soll- und eine für den Ist-Wert. Weiterhin einen Button um das Programm mit VRML Unterstützung zu starten und einen Button für den Start ohne VRML- Nutzung. Ein Button dient zur Übernahme der aktuellen Werte des Roboters (Synchronisation). Die anderen beiden Buttons sind zum ein- und ausschalten des Folgemodus und zum manuellen Senden der eingestellten Soll-Position.

Das große Textfeld ist für die Ausgabe von Statusmeldungen. Das kleine Textfeld stellt eine weitere Möglichkeit zur Befehlseingabe dar. Die möglichen Befehle sind im Anhang (A) beschrieben. Als exemplarisches Beispiel werden hier der Befehle `set1`...`set9` und `goto1`...`goto9` vorgestellt. Wenn der Benutzer Text in die Befehlszeile eingibt und mit Enter bzw. Return abschließt, wird der eingegebene Befehl an die Methode `parseInput` übergeben. Diese prüft mit Hilfe der String Methode `startsWith` ob die Eingabe einer der möglichen Befehle ist und führt diesen aus. Im Fall von `set1` wird die aktuell

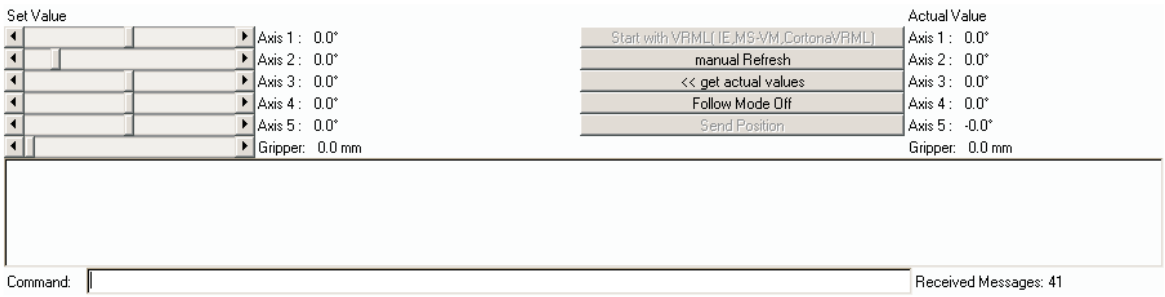

Abbildung 5.5: Screenshot Applet

dargestellte Ist-Position auf die Soll-Positionen übertragen und dann in einem Float-Array gespeichert. Mit Hilfe des Befehls `goto1` kann der Benutzer bewirken, dass die Soll Position auf den im Client gespeicherten Wert gesetzt wird. Durch Hinzufügen der Option `-f`, wird die gespeicherte Position direkt an den Server gesendet. Das Speichern der Positionen wird lokal auf dem Client vorgenommen. Jeder Benutzer kann also eigene Werte speichern. Der Benutzer hat die Möglichkeit neun solcher Punkte zu definieren, wobei noch erwähnt werden muss, dass diese Positionen nur temporär, das heißt bis zu Sitzungsende, gespeichert werden.

VRML-Modell In dem angepassten VRML-Modell sind zwei Roboter visualisiert, siehe Abbildung (5.6): Einer zeigt die aktuelle Position des Roboters und der andere Roboter symbolisiert die vom Benutzer eingestellte Soll-Position. Das Robotermodell für die Soll-Position ist mit Sensoren ausgestattet. So kann der Benutzer direkt im VRML-Browser mit der Maus eine Soll-Position einstellen. Die Bereiche in denen die Sensoren auf eine Benutzer-Eingabe reagieren sind in der Abbildung (5.7) markiert. Die sensiblen Bereiche des Roboters sind die einzelnen Roboterarme, der Grundkörper und der Greifer. Der Greifer kann in dem VRML-Modell nur gedreht, nicht aber geschlossen werden.

Die Kommunikation zwischen dem Applet und der VRML-Szene ist mit Hilfe des external authoring interface (EAI) implementiert. Da das Applet auch ohne VRML-Browser funktionsfähig sein soll, ist der Zugriff auf den Szenengrafen in der Extra-Klasse `vrmlRobo` realisiert. Der Konstruktor dieser Klasse initialisiert das Callback, also die Funktion des VRML-Browsers das Applet zu benachrichtigen, wenn sich an den „beobachteten“ VRML-Knoten etwas ändert. Das folgende Listing zeigt den Quellcode, der zu inititalisirung des Callback's benötigt wird. Das Applet soll benachrichtigt werden, wenn sich die Rotations-Eigenschaft des Knotens `Arm1` ändert. Im ersten Schritt werden die Variablen definiert:

```
roboApplet           myApplet;
Browser              browser;
EventOutSFRotation   callbackSetAxis1 = null;
```

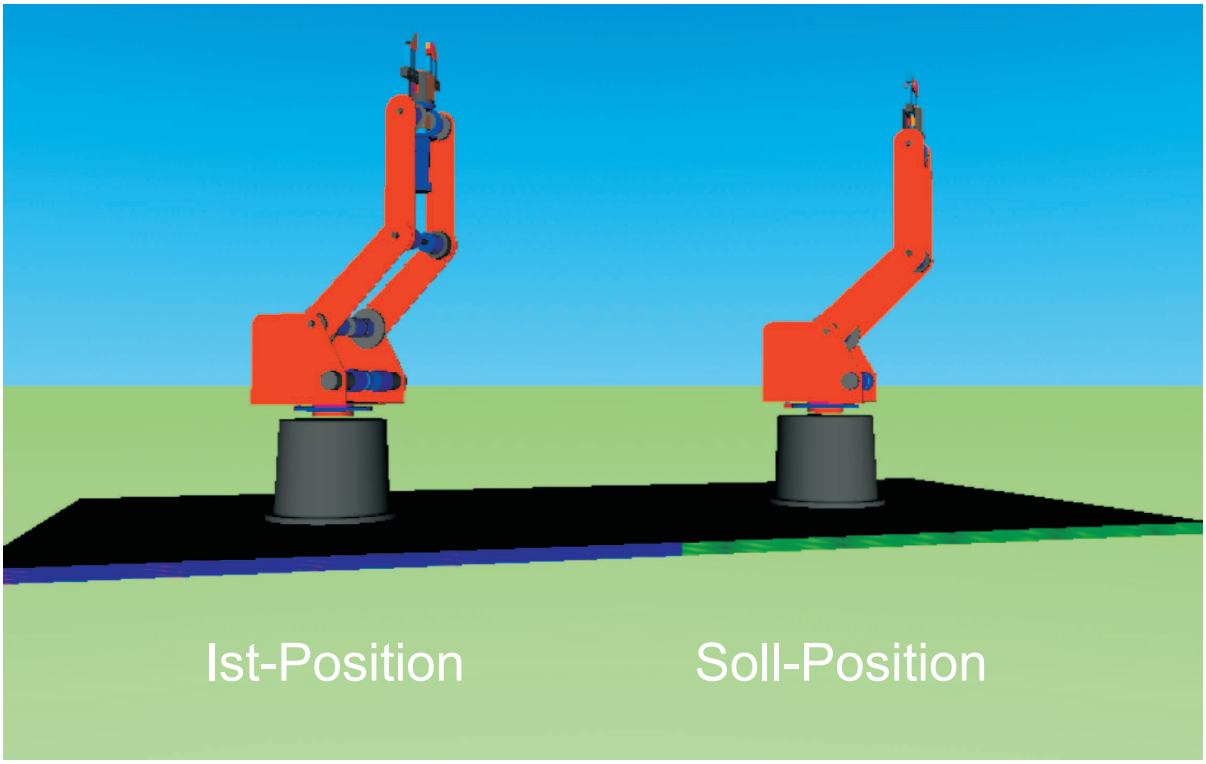

Abbildung 5.6: Screenshot VRML

In dem Konstruktor werden die Knoten initialisiert:

```
Node SetAxis1Node = browser.getNode("Arm1");
```

Die Variable `callbackSetAxis1` wird an das Event gebunden, dass beobachtet werden soll:

```
callbackSetAxis1 = (EventOutSFRotation) SetAxis1Node. ↩
    getEventOut("rotation");
```

Die Beobachtung wird gestartet:

```
callbackSetAxis1.advise(myApplet, null);
```

Die Benachrichtigung wird direkt an das Applet gesendet. Im Applet existiert einen Methode `callback(EventOut eventOut, double timestamp, Object obj)`. Diese wird über das EAI vom VRML-Browser aufgerufen. Da das Applet, also nicht das Objekt `vrmlRobo` benachrichtigt wird, muss das Applet `EventOutObserver` implementieren und die Pakete `vrml.external.*` und `vrml.external.field.*` importieren. Die beiden Pakete müssen auch in der Klasse `vrmlRobo` importiert werden. Mit dem folgenden Befehl kann der Wert der Rotations-Eigenschaft ausgelesen werden:

```
callbackSetAxis1.getValue();
```

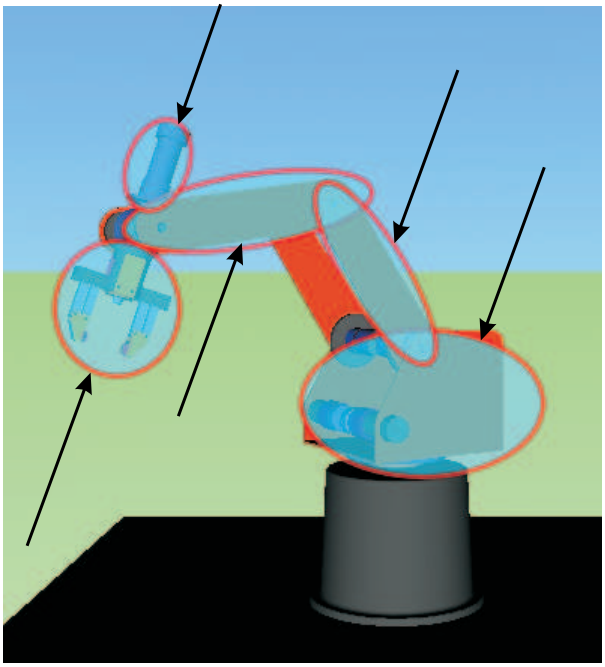

Abbildung 5.7: Sensorbereiche im VRML-Modell

Des Weiteren werden die VRML-Knoten initialisiert, an die Daten gesendet werden sollen. Die Klasse `vrmlRobo` implementiert Methoden um den beiden Robotern Koordinaten zu senden. Um einen Wert an einen VRML-Knoten zu senden wird der folgende Aufruf gemacht:

```
rotArm1In.setValue(actualAngelAxis1);
```

Weitere Informationen zur Programmierung mit dem EAI finden sich in „Interaktive Animationen auf Basis von VRML“ [19]

Der reale Roboter und das Robotermodell in der VRML-Szene unterscheiden sich in der Winkeldefinition maßgeblich: Das reale System ist so konstruiert, dass alle Winkel der Achsen, die eine Schwenkbewegung ausführen, immer zwischen Boden und Roboterarm gemessen werden. Im hierarchisch aufgebauten VRML-Modell sind die Winkel immer zwischen den einzelnen Elementen definiert. Das bedeutet, wenn ein Arm geschwenkt wird, werden auch alle untergeordneten Arme mitgeschwenkt. Dieser Unterschied zwischen den Modellen macht eine Umrechnung notwendig. Diesesw sieht wie folgt aus:

$$\alpha_{0VRML} = -\alpha_{0Roboter} \tag{5.1}$$
$$\alpha_{1VRML} = -\alpha_{1Roboter} \tag{5.2}$$
$$\alpha_{2VRML} = \alpha_{1Roboter} - \alpha_{2Roboter} \tag{5.3}$$
$$\alpha_{3VRML} = \alpha_{2Roboter} - \alpha_{3Roboter} \tag{5.4}$$

$$\alpha_{4VRML} = \alpha_{4Roboter} \tag{5.5}$$

Diese Umrechnung ist in der Klasse `vrmlRobo` implementiert.

JSP-Seite Das Applet und das VRML-Modell sind in eine Web-Seite eingebettet. Die ganze Mächtigkeit von JSP ist in dieser Anwendung nicht genutzt worden, lediglich die Funktion:

```
<jsp:include page="/Contents.jsp"/>
```

Diese Funktion bindet eine andere Datei mit in die Web-Seite ein. In diesem Fall handelt es sich um die Datei, die das Menü beinhaltet. Das VRML-Modell wird mit der folgenden Code-Zeile eingebunden:

```
<embed src="./5dof.wrl" width="1200" height="500">
```

Bei dem Applet sieht das dann so aus:

```
<APPLET align="baseline" code="roboApplet.class" archive=" ↩
    ../../classes/" height="225" width="900" mayscript></APPLET ↩
    >
```

Der Parameter `archive="../../classes/"` ist notwendig damit auf einem System ohne Cortona VRML-Browser die VRML-Java-Klassen zur Verfügung stehen. Die JSP-Seite ist in der Abbildung (5.8) dargestellt.

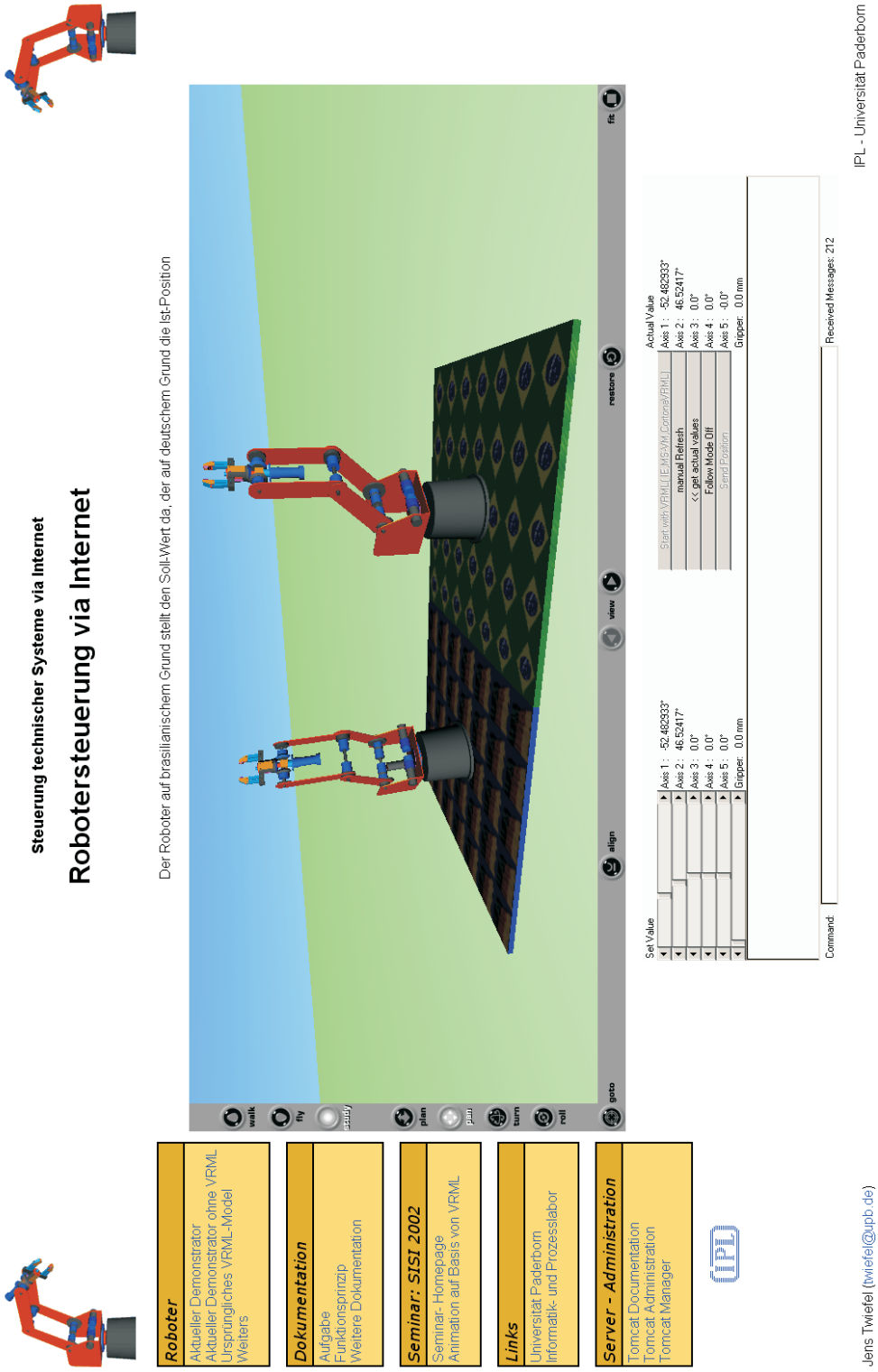

Abbildung 5.8: Screenshot JSP-Seite

5.2 Inbetriebnahme und Test

In diesem Abschnitt werden die Systemvoraussetzungen beschrieben, die sich aus der Implementierung ergeben haben.

5.2.1 Systemvoraussetzungen

Server

- Linux oder Windows Betriebssystem
- Firewire Karte
- Kommunikations API (bisher noch nicht für Windows verfügbar)
- Tomcat Webserver

Client mit VRML

- Windows Betriebssystem
- Internet Explorer ab Version 5
- Cortona VRML Player
- MS Java Virtual Maschine
- Schnelle Netzwerkanbindung an den Server
- Prozessor Takt $>$ 500MHz
- RAM $>$ 64MB

Client ohne VRML

- Java fähiger Webbrowser
- SUN Java ab Version 1.3.1
- Schnelle Netzanbindung

5.2.2 Inbetriebnahme des Webserver

Nach der Installation des Tomcat Webserver müssen noch einige wenige Anpassungen in der Konfiguration vorgenommen werden: Das Verzeichnis für die html und jsp Dateien ist `/$TomcatInstallDir$/webapps/ROOT/` und das Verzeichnis für die Klassendateien des Servlets lautet:`/$TomcatInstallDir$/webapps/ROOT/WEB-INF/classes/` Der Webserver ist unter der Adresse `http://rechnerName:8080/` oder `http://rechnerIP:8080/` zu erreichen.

Damit das Java Native Interface benutzt werden kann, muss unter Windows die .dll-Datei im `/$windowsHome$/System32` Verzeichnis sein. Unter Linux muss der LD-LIBRARY PATH auf den Pfad gesetzt werden, welcher die .so-Datei enthält. Für das Setzen des Pfades benutzt man den Befehl: `export LD_LIBRARY_PATH="/opt/jakarta/clib"` dieser wird in die Datei
`/$TomcatInstallDir$/bin/startup.sh`
vor dem Befehl:
`$BASEDIR/catalina.sh start "$@"`
eingetragen. Der Pfad muss der lokalen Installation angepasst werden.

5.2.3 Test der Software an einem „simulierten Roboter“

Während der Entwicklung der Software ist für den Test ein „simulierter Roboter“ zum Einsatz gekommen. Dafür ist die Kommunikations API durch eine „Roboter Simulations-API“ mit identischen Methoden-Deklarationen ersetzt worden, wobei die neue API keine Hardware Zugriffe macht sondern nur den als Soll-Wert eingegebenen Wert als Ist-Position ausgibt.

5.2.4 Test der Software an dem Roboter

Um die Implementierung in der Zielumgebung abschließend zu testen, ist das ganze System aufgebaut worden und nach der Checkliste in Anhang (B) gestartet worden.

Bei den Tests sind die folgenden (bleibende) Fehler aufgetreten:

Fehlerliste Hardware

- Arm 1 lässt sich nur von 0° bis ca. 104° bewegen
- Die Winkelwerte, die für die Achse 3 und 4 vom RABBIT System geliefert werden, sind nicht korrekt. Das ist aber in der Klasse `HardwareInterface` kompensiert worden es ergibt sich:

$$\alpha_{3Servlet} = \frac{\alpha_{3Rabbit} + \alpha_{4Rabbit}}{2} \tag{5.6}$$

$$\alpha_{4Servlet} = \frac{\alpha_{3Rabbit} - \alpha_{4Rabbit}}{2} \tag{5.7}$$

5.3 Zusammenfassung

Im ersten Abschnitt wurde für die Applet-Servlet Kommunikation die Variante „Objekt über Http“ ausgewählt. Für diese Variante wurde das Java-Objekt für die Kommunikation erstellt, das `roboPositon`-Objekt. Für den Server ist die Implementierung der Komponenten: Netzwerk-Kommunikation, Sitzungsverfolgung, Kollisionsvermeidung, Alternative-Bahnvorgabe und Hardware-Zugriff erläutert. Danach ist für den Client die Implementierung der Netzwerk-Kommunikation und der Benutzerschnittstelle vorgestellt worden. Zum Abschluss des Kapitels ist kurz die Inbetriebnahme erläutert worden.

6 Zusammenfassung und Ausblick

Dieses Kapitel schließt die Arbeit mit einer Zusammenfassung und einem Ausblick ab.

6.1 Zusammenfassung

Das technische System, ein fünf-Achsen-Knickarmroboter, soll über das Internet gesteuert werden.

Das bestehende System

Das Robotersystem besteht aus 3 Hauptkomponenten:

- Der Roboter
- Das Rabbit-System
- Das Kommunikationssystem

Der Roboter hat fünf Freiheitsgerade für die Positionierung im Raum. Die Motoren des Roboters sind mit Sensoren ausgestattet.

Das Rabbit-System ist eine modulare Rapid Prototyping Plattform für verteilte mechatronische Systeme. Es besteht in der hier genutzten Ausführung aus den Komponenten: Das Firewire-Modul (IEEE 1394) für die Kommunikation. Ein Microcontroller-Modul und ein FPGA-Modul führen die Regelalgorithmen des mechatonischen Systems aus. Drei DC-Motor-Module dienen zur Ansteuerung der Motoren des Roboters und zum Auslesen der Sensordaten.

Das Kommunikationssystem ist auf dem Rabbit-System implementiert. Es ermöglicht eine Kommunikation zwischen dem Rabbit und einem PC. Diese Kommunikation wird mit einer Firewire-Verbindung realisiert. Weiter existiert für einen Linux-PC eine API zur Kommunikation mit dem Roboter. Diese API implementiert zwei Methode: Lesen der aktuellen Ist-Position des Roboters und das Senden einer neuen Soll-Position an den Roboter.

Für die Robotersteuerung gibt es die folgenden Hauptforderungen:

- Die Steuerung soll webbasiert sein
- Dreidimensionale Visualisierung der Soll- und Ist-Position
- Steuerung in weicher Echtzeit
- Das System soll sicher sein

Im Folgenden wird kurz erläutert, wie die einzelnen Problemstellungen bearbeitet wurden.

Webbasierte Steuerung

Für eine Web-Anwendung wird ein Client und ein Server benötigt. Der Server ist als Java-Servlet implementiert worden. Der Client ist ein Java-Applet. Das Servlet und das Applet kommunizieren miteinander. Als Übertragungsprotokoll wurde Http ausgewählt. Über dieses Protokoll werden serialisierte Java Objekte ausgetauscht. Damit ist die Schnittstelle zwischen Servlet und Applet definiert.

Visualisierung des Roboter

Dies Aufgabe übernimmt der Client, also das Applet in Verbindung mit einem VRML-Browser. Während das Applet eine klassische Benutzeroberfläche generiert, werden im VRML-Browser zwei dreidimensionale Roboter abgebildet. Einer diese Roboter visualisiert die Soll-Position und der andere die Ist-Position des technischen Systems. Der Roboter für die Soll-Psition ist mit Sensoren ausgestattet, die es dem Benutzer ermöglichen die Soll-Position zu verändern. Das Applet und der VRML-Browser kommunizieren über das external authoring interface.

Steuerung in weicher Echtzeit

Die Vorgabe eines neuen Soll-Werts wird nach einem bestimmten Schema durchgeführt:

1. Der Benutzer gibt die neue Position im VRML-Model oder im Applet vor
2. Bei der Änderung der Soll-Position wird im Applet ein Event ausgelöst
3. Das Event bewirkt, dass ein Java-Objekt an das Servlet gesendet wird
4. Das Servlet prüft die Gültigkeit der Position und meldet dem Applet den Erfolg bzw. Misserfolg

5. Die gültige Position wird an den Roboter gesendet
6. Der Roboter fährt die neue Soll-Position an

Ein Thread des Applets fragt beim Server stetig die aktuelle Ist-Position des Roboters ab und gewährleistet damit eine zeitnahe Visualisierung.

Systemsicherheit

Die Sicherheit des Systems gegen falsche Eingaben wird durch eine Prüfung der Soll-Werte gewährleistet. Mit dieser Prüfung wird getestet, ob der neue Soll-Wert eine Position beschreibt, an der es zu keiner Kollision mit dem Roboter selbst oder mit dem Boden kommt. Der Prüfalgorithmus wird auf dem Server audgeführt. So wird garantiert, dass alle an den Server gesendeten Daten geprüft werden bevor sie an den Roboter gesendet werden.

6.2 Ausblick

Weitere Aufgaben, die sich an diese Arbeit anschließen können, werden in diesem Abschnitt erläutert.

- Automatische Initialisierung wenn die Hardware-Endschalter im Rabbitsystem und im Roboter implementiert sind.
- Den zweiten Roboter in das Szenario integrieren und die Kollisionsvermeidung so anpassen, das eine Kollision zwischen den Beiden vermieden wird. Dafür ist es notwendig die Geraden aus der Ebene in den Raum zu exportieren und einige zusätzliche Punkte zu prüfen.
- Portierung des VRML-Models nach Java3D um eine größere Benutzergruppe ansprechen zu können.
- Einen Modus hinzufügen, mit dem der Benutzer ein Programm eingeben kann
- Einen Modus hinzufügen, in dem der Benutzer dem Roboter eine Bahn beibringen kann
- Eine Erfassung der Umgebung des Roboters

A Kommandoreferenz Applet

clear löscht das Textfenster

goto1..9 (-f) fährt vorher definierte Punkte an, die Standard Einstellung ist die Home Position. Die Option `-f` bewirkt das die Position auch direkt an den Server übermittelt wird.

gotoHP (-f) fährt die Home Position an. Die Option `-f` bewirkt das die Position auch direkt an den Server übermittelt wird.

help gibt eine kurze Hilfe aus

send gibt die Anzahl der gesendeten Soll Positionen aus.

set1..9 speichert die momentane Position des Roboters.

startdemo statet den Bahnmodus.

stopdemo beendet den Bahnmodus.

wehreR gibt die Position in Radiant aus.

whereG gibt die Position in Grad aus.

B Starten des Roboters

Wenn der Webserver eingerichtet ist wird das Gesamtsystem wie folgt in Betrieb genommen.

1. Den Roboter an das Rabbitsystem anschließen.
2. Das Programmiergerät an das Rabbitsystem und einen PC mit dem Metrowerks CodeWarrior anschließen.
3. Den Roboter von Hand auf die Home Position bewegen (siehe Abbildung(B.1)). Die Hochachsen, die in der Abbildung schwer zu erkennen sind müssen in der Mittelstellung sein und der Greifer muss geschlossen sein.
4. Am Rabbitsystem den Notausschalter und die Schalter für die einzelnen Motoren aus schalten.
5. Den Hauptschalter des Rabbitsystems einschalten.
6. Mit dem CodeWarrior die Software in das Rabbitsystem laden.
7. Am Rabbitsystem den Notausschalter und die Schalter für die einzelnen Motoren ein schalten.
8. Das Program auf dem Rabbitsystem mit dem CodeWarrior starten.
9. Den Webserver starten.
10. Mit dem Firewire Kabel das Rabbitsystem und den Webserver verbinden.
11. Jetzt ist das System einsatzbereit.

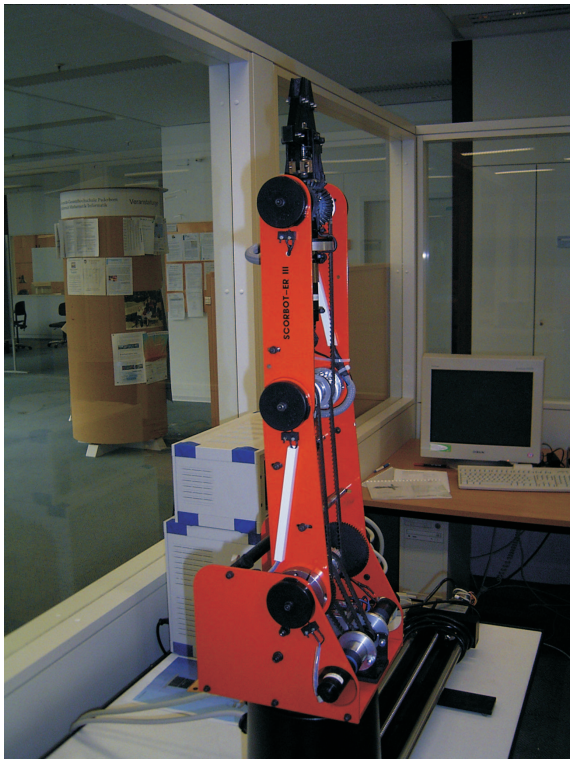

Abbildung B.1: Home Position des Roboters

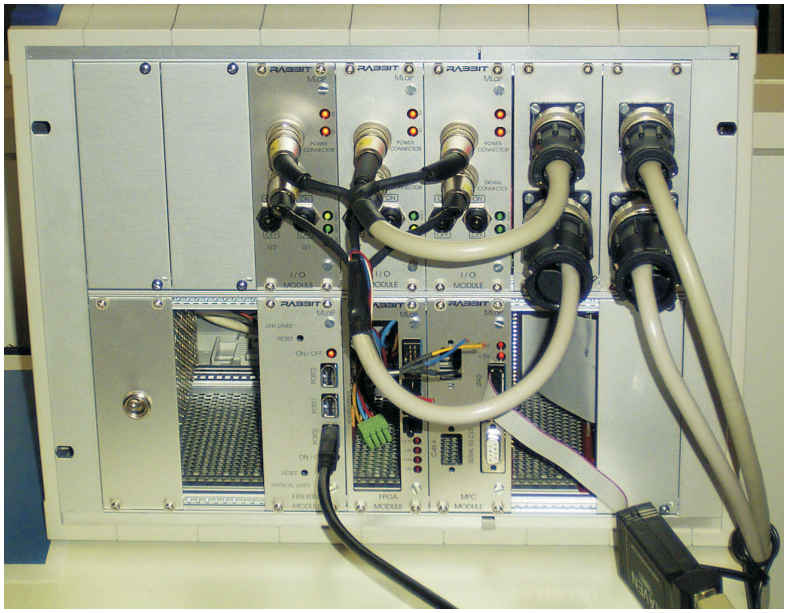

Abbildung B.2: Frontansicht des Rabbit-Systems

C Beispiel Kollisionskontrolle

In diesem Anhang wird exemplarisch eine neue Soll-Position, für den Roboter, auf Kollision getestet. Es wird das in Kapitel (4) beschriebene Verfahren angewendet. Die Reihenfolge der Berechnungen entspricht der Reihenfolge im Programm.

Neue Soll-Werte

$$\begin{aligned} \alpha_0 &= 20^\circ && \text{(C.1)} \\ \alpha_1 &= 70^\circ && \text{(C.2)} \\ \alpha_2 &= -70^\circ && \text{(C.3)} \\ \alpha_3 &= 20^\circ && \text{(C.4)} \\ \alpha_4 &= 20^\circ && \text{(C.5)} \\ Greifer &= 25mm && \text{(C.6)} \end{aligned}$$

Prüfen der Statischen Grenzen

Die Bedingungen lauten:

$$\begin{aligned} -150^\circ \leq \quad & \alpha_0 \quad && \leq 150^\circ && \text{(C.7)} \\ -20^\circ \leq \quad & \alpha_1 \quad && \leq 180^\circ && \text{(C.8)} \\ -180^\circ \leq \quad & \alpha_2 \quad && \leq 180^\circ && \text{(C.9)} \\ -180^\circ \leq \quad & \alpha_3 \quad && \leq 180^\circ && \text{(C.10)} \\ -180^\circ \leq \quad & \alpha_4 \quad && \leq 180^\circ && \text{(C.11)} \\ 0mm \leq \quad & Greifer \quad && \leq 50mm && \text{(C.12)} \end{aligned}$$

Diese sind erfüllt, daher ist noch keine Kollision festgestellt.

Prüfung des eingeschlossenen Winkels

Diese Bedingung

$$\min(|180^\circ - (\alpha_2 - \alpha_3)|, |180^\circ - (\alpha_3 - \alpha_2)|) \geq 67^\circ \quad \text{(C.13)}$$
$$\Rightarrow 140^\circ \geq 67^\circ \quad \text{(C.14)}$$

ist ebenfalls erfüllt.

Berechnung der Punkte $(P_0 \dots P_{10})$

Für die Längen $\{l_0 \dots l_7\}$ siehe Abbildung(4.4).

$$P_i = (x_i, y_i)^T \quad \text{(C.15)}$$
$$P_0 = (0, l_0)^T \quad \text{(C.16)}$$
$$\Rightarrow P_0 = (0, 350mm)^T \quad \text{(C.17)}$$
$$P_1 = P_0 + l_1 (\sin(\alpha_1), \cos(\alpha_1))^T \quad \text{(C.18)}$$
$$\Rightarrow P_1 = (206.732mm, 425.244mm)^T \quad \text{(C.19)}$$
$$P_2 = P_1 + l_2 (\sin(\alpha_2), \cos(\alpha_2))^T \quad \text{(C.20)}$$
$$\Rightarrow P_2 = (0mm, 500.489mm)^T \quad \text{(C.21)}$$
$$P_3 = P_2 + l_3 (\sin(\alpha_3), \cos(\alpha_3))^T \quad \text{(C.22)}$$
$$\Rightarrow P_3 = (-75mm, 630.393mm)^T \quad \text{(C.23)}$$
$$P_4 = P_2 - l_4 (\sin(\alpha_3), \cos(\alpha_3))^T \quad \text{(C.24)}$$
$$\Rightarrow P_4 = (72.5mm, 374.915mm)^T \quad \text{(C.25)}$$
$$P_5 = P_3 + l_5 (\cos(\alpha_3), -\sin(\alpha_3))^T \quad \text{(C.26)}$$
$$\Rightarrow P_5 = (-23.039mm, 660.393mm)^T \quad \text{(C.27)}$$
$$P_6 = P_3 - l_5 (\cos(\alpha_3), -\sin(\alpha_3))^T \quad \text{(C.28)}$$
$$\Rightarrow P_6 = (-126.962mm, 600.393mm)^T \quad \text{(C.29)}$$
$$P_7 = P_4 + l_6 (\cos(\alpha_3), -\sin(\alpha_3))^T \quad \text{(C.30)}$$
$$\Rightarrow P_7 = (91.986mm, 386.165mm)^T \quad \text{(C.31)}$$
$$P_8 = P_4 - l_6 (\cos(\alpha_3), -\sin(\alpha_3))^T \quad \text{(C.32)}$$
$$\Rightarrow P_8 = (53.014mm, 363.665mm)^T \quad \text{(C.33)}$$
$$P_9 = P_5 - l_7 (\sin(\alpha_3), \cos(\alpha_3))^T \quad \text{(C.34)}$$
$$\Rightarrow P_9 = (17.961mm, 589.379mm)^T \quad \text{(C.35)}$$
$$P_{10} = P_6 - l_7 (\sin(\alpha_3), \cos(\alpha_3))^T \quad \text{(C.36)}$$
$$\Rightarrow P_{10} = (-85.962mm, 529.379mm)^T \quad \text{(C.37)}$$

Mit diesen werten kann das Grafische Modell erstellt werden, siehe Abbildung (C.1).

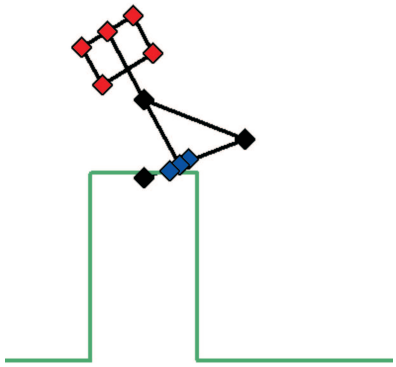

Abbildung C.1: Modell der Beispiel-Konfiguration

Test auf eine Kollision mit dem Untergrund

Für den Untergrund ist die folgende Funktion aufgestellt worden:

$$f_{Untergrund}(x) = \begin{cases} 360 & : -110 \leq x \leq 110 \\ 0 & sonst \end{cases} \tag{C.38}$$

Die Bedingung für keine Kollision ist:

$$f_{Untergrund}(x_i) < y_i \tag{C.39}$$

Diese Bedingung ist für $i = 1 \ldots 10$ erfüllt.

Berechnung der Abstände zur Graden durch Arm1

Zur Berechnung der Abstände müssen zunächst einige vorbereitende Schritte durchgeführt werden:

$$a = \frac{x_2 y_1 - x_1 y_2}{y_1 - y_2} \tag{C.40}$$

$$\Rightarrow a = -350mm \tag{C.41}$$

$$b = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1} \tag{C.42}$$

$$\Rightarrow b = 350mm \tag{C.43}$$

$$\alpha_1' = \arctan\left(\frac{b}{a}\right) \tag{C.44}$$

$$\Rightarrow \alpha_1' = -45° \tag{C.45}$$

$$\varphi_1 = 90° - \alpha_1' \tag{C.46}$$

$$\Rightarrow \varphi_1 = 135^\circ \tag{C.47}$$
$$p = a \sin(\alpha') \tag{C.48}$$
$$\Rightarrow p = 328.892 \tag{C.49}$$

Daraus folgt:

$$d_1 = x_i \cos(\varphi_1) + y_i \sin(\varphi_1) - p \tag{C.50}$$

Es ergibt sich:

$$d_{11} = 0mm \tag{C.51}$$
$$d_{12} = 141.413mm \tag{C.52}$$
$$d_{13} = 289.134mm \tag{C.53}$$
$$d_{14} = -1.384mm \tag{C.54}$$
$$d_{15} = 299.553mm \tag{C.55}$$
$$d_{16} = 278.716mm \tag{C.56}$$
$$d_{17} = 2.523mm \tag{C.57}$$
$$d_{18} = -5.291mm \tag{C.58}$$
$$d_{19} = 218.799mm \tag{C.59}$$
$$d_{110} = 197.961mm \tag{C.60}$$

Test der Punkte $(P_2 \dots P_{10})$ auf Kollision mit Arm1

Jetzt wird geprüft welche der Punkte $(P_2 \dots P_{10})$ in dem Bereich:

$$(x_0 - d_1 < x_i < x_1 + d_1 \vee x_1 - d_1 < x_i < x_0 + d_1) \wedge (y_0 < y_i < y_1 \vee y_1 < y_i < y_0) \tag{C.61}$$

liegen.

Die Punkte (P_4, P_7, P_8) sind in dem Bereich. Bei diesen Punkten muss der Betrag des Abstands (d_{1i}) größer als $d_1 = 35mm$, aus Abbildung (4.6) sein, damit keine Kollision vorliegt.

$$d_{14} \ngeq d_1 \tag{C.62}$$
$$d_{17} \ngeq d_1 \tag{C.63}$$
$$d_{18} \ngeq d_1 \tag{C.64}$$

Damit ist an drei Stellen eine Kollision festgestellt.

D Arbeitsbereich des Roboters

Eine Simulation der Kollisionskontrolle über die Winkel $\alpha_1, \alpha_2, \alpha_3$, mit einer Schrittweite von jeweils 1° ergibt ca. 5 Millionen gültige Positionen. In der Abbildung (D.1) sind die Positionen des Punktes P_3, dieser beschreibt die Greifermitte. Diese Simulation ist mit einem Java Programm, welches eine leicht abgeänderte Version der Klasse `collisionTest` nutzt, durchgeführt worden. Die Visualisierung ist mit Matlab durchgeführt worden.

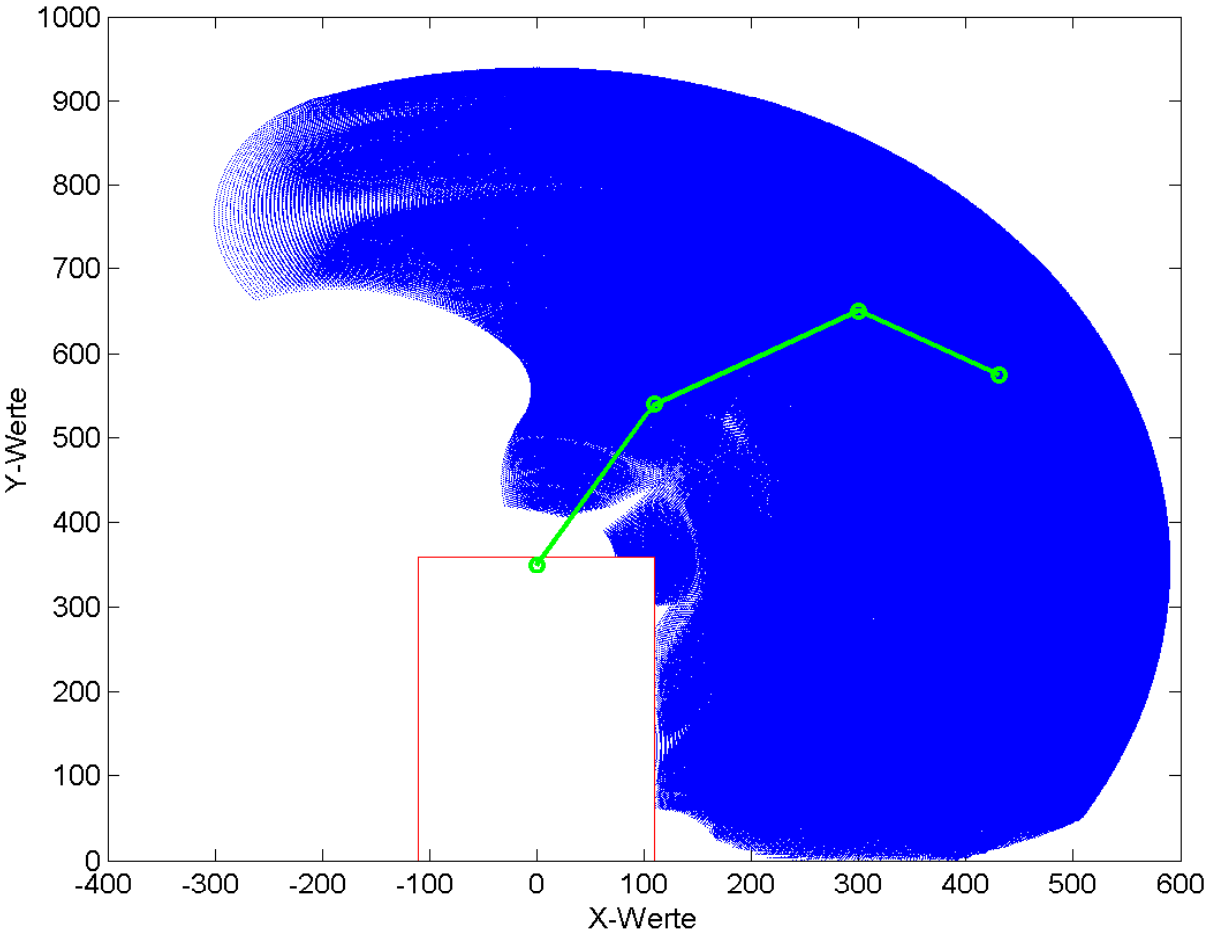

Abbildung D.1: Arbeitsbereich des Roboters in der Ebene

E Quellcode

E.1 Server und Client

E.1.1 roboPosition.java

```
1  import java.io.Serializable;
2  /**
3     *Class roboPosition
4     *@author Jens Twiefel
5     *@version 1.0
6     */
7  public class roboPosition implements Serializable{
8    int        errorNo       =  0;
9    float[]    radPosition   =  {0,0,0,0,0,0,0};
10   int        sessionID     =  0;
11   String     msgString     =  "";
12   /**
13    *Standard Constructer
14    *the Standard Position is 0 Rad for all axis
15    *
16    */
17   public roboPosition(){
18   }
19   /**
20    *Constructor
21    *@param startPos  The Start- Position of the Robot in rad
22    */
23   public roboPosition(float[] startPos){
24     this.radPosition=startPos;
25   }
26   //---Set Functions
27   /**
28    *Position set function
29    *@param newPos[] float-array for the new Position in rad
30    */
31   public void setRadPosition(float[] newPos){
32     this.radPosition=newPos;
33   }
34   /**
35    *Position set function
36    *@param newPos[] double-array for the new Position in rad
37    */
38   public void setRadPosition(double[] newPos){
39     this.radPosition[0]=(float)newPos[0];
40     this.radPosition[1]=(float)newPos[1];
41     this.radPosition[2]=(float)newPos[2];
42     this.radPosition[3]=(float)newPos[3];
43     this.radPosition[4]=(float)newPos[4];
44     this.radPosition[5]=(float)newPos[5];
45   }
46   /**
47    *Position set function for one Axis
```

```
48      *@param newPos[] float-array for the new Position in rad
49      *@param Axis int number of the Axis (0-5)
50      */
51    public void setRadPosition(float radValue,int Axis){
52      this.radPosition[Axis]=radValue;
53    }
54    /**
55     *Set the Message String
56     *@param str Message String
57     */
58    public void setString(String str){
59      this.msgString=str;
60    }
61    /**
62     *Set the Error- Number
63     *@param no int Error Number
64     */
65    public void setErrorNo(int no){
66      this.errorNo=no;
67    }
68    /**
69     *Set the Session ID Number
70     *@param no int Session ID
71     */
72    public void setSessionID(int no){
73      this.sessionID=no;
74    }
75    //---Get Functions
76    /**
77     *Get the Position Array in rad
78     *@return Float- Array
79     */
80    public float[] getRadPosition(){
81      return this.radPosition;
82    }
83    /**
84     *Get the Position Array in rad
85     *@return Double- Array
86     */
87    public double[] getDoubleRadPosition(){
88      double[] result={0.0,0.0,0.0,0.0,0.0,0.0,0.0};
89      for(int i=0;i<6;i++) result[i]=(double)this.radPosition[i];
90      return result;
91    }
92    /**
93     *Get the Position of a spezified Axis
94     *@param Axis Integer Axis Number
95     *@return Float- Value
96     */
97    public float getRadPosition(int Axis){
98      return this.radPosition[Axis];
99    }
100   /**
101    *Get the Error Number
102    *@return Iteger- Value
103    */
104   public int getErrorNo(){
105     return this.errorNo;
106   }
107   /**
108    *Get the Session ID
109    *@return Intger- Value
110    */
111   public int getSessionID(){
112     return this.sessionID;
113   }
```

```
114   /**
115    *Get the message String
116    *@return String
117    */
118   public String getMsgStr(){
119     return this.msgString;
120   }
121   //---To String Methoden
122   /**
123    *Get a Position String in radiant
124    *@return String
125    */
126   public String toString(){
127     String result="";
128     for(int i=0;i<6;i++) result+="Axis "+(i+1)+": "+this.radPosition[i]+" ,";
129     return result;
130   }
131   /**
132    *Get a Position String in grad
133    *@return String
134    */
135   public String toStringGrad(){
136     String result="";
137     for(int i=0;i<6;i++)
138       result+="Axis "+(i+1)+" :"+(this.radPosition[i]*20*3.14159265)+"° ,";
139     return result;
140   }
141 }
```

E.2 Server

E.2.1 RoboServlet.java

```
1  import java.io.*;
2  import javax.servlet.*;
3  import javax.servlet.http.*;
4   /**
5     *Class RoboServlet
6     *@author Jens Twiefel
7     *@version 1.0
8     */
9  public class RoboServlet extends HttpServlet{
10   static int                counter       = 0;
11   static int                lastSessionID = 0;
12   static roboPosition       roboPos       = new roboPosition();
13          roboPosition       newRoboPos    = new roboPosition();
14          HardwareInterface  hw            = null;
15          boolean            hw_test       = false;
16   final  boolean            Debug         = false;
17   final  boolean            Debug2        = false;
18   //Initialisierung
19   /**
20    *initialisation Method
21    **/
22   public void init() throws ServletException{
23     if (Debug) System.out.println("start init,,,\n");
24     try{
25       hw = new HardwareInterface();
26       hw_test=true;
27     }catch(Exception e){System.out.println("no init()");}
28     counter++;
29     System.out.println("RoboServlet "+counter+" started");
30     if (Debug) System.out.println("end init \n");
31   }
32   //---Beenden des Servlets
33   /**
34    *destroy Method
35    **/
36   public void destroy(){
37     counter--;
38     if (counter==0) hw = null;
39    System.out.println("RoboServlet "+(counter+1)+" stopted");
40   }
41   //---Roboter Position an den Client senden
42   /**
43    *  Answer on Get Request
44    *@param req Request Object
45    *@param res Response Object
46    **/
47   public void doGet(HttpServletRequest req, HttpServletResponse res)
48                     throws ServletException, IOException {
49
50     float[] helpfloat=roboPos.getRadPosition();
51     if (Debug) System.out.println("get HW start ");
52     if (!hw_test){roboPos.setString ("Hardware Interface Libary not found");}
53     else {
54       try{
55       roboPos.setRadPosition(hw.getRobotData());
56       }catch(Exception e){
57         roboPos.setRadPosition(helpfloat);
58         roboPos.setString ("Hardware Interface Libary Error");
59       }
60     }
```

```
61      ObjectOutputStream out = new ObjectOutputStream(res.getOutputStream());
62      out.writeObject(roboPos);
63      out.flush();
64      out.close();
65      if (Debug) System.out.println("get HW stop \n");
66      roboPos.setString("");
67    }
68    //---Position vom Client an den Roboter senden
69    /**
70     *  Answer on Post Request
71     *@param req Request Object
72     *@param res Response Object
73     **/
74    public void doPost(HttpServletRequest req, HttpServletResponse res)
75                        throws ServletException, IOException {
76
77      try{
78        ObjectInputStream in = new ObjectInputStream(req.getInputStream());
79        Object obj = in.readObject();
80        synchronized(this){
81          newRoboPos = (roboPosition)obj;
82          if (newRoboPos.getSessionID()==0){
83           newRoboPos.setSessionID(lastSessionID++);
84          }
85          if (Debug) System.out.println("set Robo "+newRoboPos.toString()+" \n");
86          if (!hw_test){
87           newRoboPos.setErrorNo(4);
88           roboPos.setString ("Hardware Interface Libary not found");
89          }
90          else{
91            if (Debug) System.out.println("set HW \n");
92            if (newRoboPos.getErrorNo()!=0){
93              if(newRoboPos.getErrorNo()==1)
94                newRoboPos.setErrorNo(hw.startDemo(newRoboPos.getSessionID()));
95              else {hw.stopDemo();newRoboPos.setErrorNo(0);}
96            }
97            else
98              newRoboPos.setErrorNo(
99                hw.setRobotData(1,
100                 newRoboPos.getDoubleRadPosition(),newRoboPos.getSessionID()));
101         }
102       }
103     }catch(Exception e){
104       newRoboPos.setErrorNo(5);System.out.println("Error set Robo \n");
105     }
106     ObjectOutputStream out = new ObjectOutputStream(res.getOutputStream());
107     out.writeObject(newRoboPos);
108     out.flush();
109     out.close();
110   }
111 }
```

E.2.2 HardwareInterface.java

```
1  /**
2    *Class HardwareInterface
3    *@author Jens Twiefel
4    *@version 1.0
5    */
6
7  public class HardwareInterface implements Runnable{
8    private          int                channelNumber;
9                    int                errorLevel=0;
10                   int                i=100;
11                   int                ID=0;
12   private  final  int                defaultChannel = 1;
13   private  native  double[]          receiveRobotData();
14   private  native  int               sendRobotData(int channelNumber,
15                                                    double[] motorData);
16   static           double[]          receiveData = {0,0,0,0,0,0};
17   static           double[]          sendData;
18   static           double[]          thisData = {0,0,0,0,0,0};
19                   Thread             thread;
20   static           boolean           Demo=false;
21   final            boolean           Debug=false;
22   final            boolean           Debug1=false;
23                   roboPositionQueue  queue;
24                   roboDemoQueue      demoQueue;
25                   collisionTest       collision;
26   static { System.loadLibrary("HardwareInterface"); }
27
28   /**
29    *Standard Constructer
30    */
31   public HardwareInterface(){
32       collision = new collisionTest();
33     queue = new roboPositionQueue();
34     demoQueue = new roboDemoQueue();
35     channelNumber = defaultChannel;
36     receiveData = receiveRobotData();
37     this.start();
38   }
39   /**
40    *start Method
41    */
42   private void start() {
43     thread = new Thread(this);
44     thread.start();
45   }
46   /**
47    *run Method
48    */
49   public void run() {
50     while (true) {
51       if (i==100){
52         if (Debug1)System.out.print("get...");
53         receiveData = receiveRobotData();
54         i=0;
55         if (Debug1)System.out.println("...OK");
56       }
57       i++;
58       if (Demo){
59         double[] help={receiveData[0],receiveData[1],receiveData[2],
60                        (receiveData[3]+receiveData[4])/2,
61                       -(receiveData[4]-receiveData[3])/2,receiveData[5]};
62         //ohne HW-Fehler:
63         //if (pointReached(receiveData,thisData,0.12)){//0.04rad = 2.3°
64         if (pointReached(help,thisData,0.08)){//0.04rad = 2.3°
```

```
65            thisData=demoQueue.getPosition();
66            errorLevel=sendRobotData(channelNumber,thisData);
67            i=100;
68          }
69          else{
70            //System.out.println(receiveData[0]+""+receiveData[1]+""+receiveData ↩
                  [2]+""+receiveData[3]+""+receiveData[4]+""+receiveData[5]+"");
71          }
72        }
73        else{
74          if (queue.hasElements()){
75            if (Debug1) System.out.print("set...");
76            thisData=queue.getPosition();
77            errorLevel=sendRobotData(channelNumber,thisData);
78            if (Debug1)System.out.println("...OK");
79            i=100;
80          }
81        }
82        try { Thread.sleep(1); } catch (InterruptedException ignored) {}
83      }
84    }
85    /**
86     *stop Method
87     */
88    private void stop() {
89      thread.stop();
90      thread = null;
91    }
92    /**
93     *compare actual value and desired value
94     *@param x array of actual values
95     *@param y array of desired values
96     *@param tol absulute tolerance
97     *@return a boolean-value
98     */
99    private boolean pointReached(double[] x,double[] y,double tol){
100     for(int i=0;i<(x.length-1);i++){
101       if (Math.abs(x[i]-y[i])>Math.abs(tol)){
102         System.out.println("test..."+i+"  "+x[i]+" "+y[i]);
103         return false;
104       }
105     }
106     if (Math.abs(x[x.length-1]-y[x.length-1])>Math.abs(tol*20)){
107       System.out.println("test..."+i+"  "+x[x.length-1]+" "+y[x.length-1]);
108       return false;
109     }
110     return true;
111   }
112   /**
113    *add a new Position to the Position-Queue
114    *@param notUsed not Used
115    *@param motorData the new Position-Array
116    *@param sessionID the client session ID
117    *@retun a error-No.
118    */
119   public int setRobotData(int notUsed, double[] motorData,int sessionID){
120     if(ID==-1) return 3;
121     if (queue.hasElements()){
122       if (ID!=sessionID) return 2;
123     }
124     else ID=sessionID;
125     if (Debug)System.out.println("incoming data...");
126     if (collision.test(motorData)) queue.addPosition(motorData);
127     else{
128       if (Debug)System.out.println("Ilegal Point");
129       return 1;
```

```
130       }
131       if (Debug)System.out.println("Point_OK");
132       return 0;
133     }
134     /**
135      *@return the actual Robot Position
136      */
137     public double[] getRobotData(){
138       // Start Fehlerbereinigung HW...
139       double[] help={receiveData[0],receiveData[1],receiveData[2],
140                      (receiveData[3]+receiveData[4])/2,
141                     -(receiveData[4]-receiveData[3])/2,receiveData[5]};
142       return help;
143       // Stop Fehlerbereinigung HW...
144       // Ohne Fehler:
145       //return receiveData;
146     }
147     /**
148      *start the demo mode
149      *@param sessionID the client session ID
150      *@param return a error-No.
151      */
152     public int startDemo(int sessionID){
153       if (queue.hasElements()) if (ID!=sessionID) return 2;
154       Demo=true;
155       thisData=demoQueue.getPosition();
156        errorLevel=sendRobotData(channelNumber,thisData);
157       ID=-1;
158       return 0;
159     }
160     /**
161      *stop the demo mode
162      */
163     public void stopDemo(){
164       Demo=false;
165       ID=0;
166     }
167 }
```

E.2.3 HardwareInterface.h

```
1 /* DO NOT EDIT THIS FILE - it is machine generated */
2 #include <jni.h>
3 /* Header for class HardwareInterface */
4
5 #ifndef _Included_HardwareInterface
6 #define _Included_HardwareInterface
7 #ifdef __cplusplus
8 extern "C" {
9 #endif
10 /* Inaccessible static: receiveData */
11 /* Inaccessible static: sendData */
12 /* Inaccessible static: thisData */
13 /* Inaccessible static: Demo */
14 /*
15  * Class:     HardwareInterface
16  * Method:    receiveRobotData
17  * Signature: ()[D
18  */
19 JNIEXPORT jdoubleArray JNICALL Java_HardwareInterface_receiveRobotData
20   (JNIEnv *, jobject);
21
22 /*
23  * Class:     HardwareInterface
24  * Method:    sendRobotData
25  * Signature: (I[D)I
26  */
27 JNIEXPORT jint JNICALL Java_HardwareInterface_sendRobotData
28   (JNIEnv *, jobject, jint, jdoubleArray);
29
30 #ifdef __cplusplus
31 }
32 #endif
33 #endif
```

E.2.4 collisionTest.java

```
1  /**
2    *Class collisionTest
3    *@author Jens Twiefel
4    *@version 1.0
5    */
6  public class collisionTest{
7
8    final              boolean              Debug=false;
9    final              boolean              Debug1=false;
10
11   /**
12    * standard constuctor
13    */
14   public collisionTest(){
15   }
16   /**
17    *ground function(x)
18    *@param  x
19    *@return the function value at x
20    */
21   private double groundFunction(double x){
22     if((x>-110)&&(x<110)) return 360;
23     return 0;
24   }
25   /**
26    *validate a Position Array
27    */
28   public boolean test(double[] newData){
29     double[] newPos     = {newData[0]+0.000000001,newData[1]+0.000000001,
30                            newData[2]+0.000000001,newData[3]+0.000000001,
31                            newData[4]+0.000000001,newData[5]+0.000000001};
32     if (Debug) System.out.println("␣"+newPos[0]+"␣,␣"+newPos[1]+"␣,␣"+
33                                        newPos[2]+"␣,␣"+newPos[3]+"␣,␣"+
34                                        newPos[4]+"␣,␣"+newPos[5]);
35     double[] lowLimit  = {-2.618,-0.3665191,-Math.PI,-2*Math.PI,-Math.PI,0 };
36     double[] highLimit = {2.618 ,2.094395,Math.PI,2*Math.PI, Math.PI,50};
37     double[] length    = {350,220,220,150,145,60,22.5,82};
38     double[] distance  = {35.0,35.0};
39     double[] pointsX   = new double[11];
40     double[] pointsY   = new double[11];
41     double[] phi       = new double[2];
42     double[] distanceA = new double[2];
43     double[] distanceB = new double[2];
44     double[] distanceP = new double[2];
45     double[] alpha     = new double[3];
46     if (Debug) System.out.println("init␣Test");
47
48     //prüfen der Statischen Grenzen
49     for(int i=0;i<6;i++){
50       if (newPos[i]<lowLimit[i]) return false;
51       if (newPos[i]>highLimit[i]) return false;
52     }
53     if (Debug) System.out.println("staticTest");
54
55     //prüfen des eingeschlossen Winkels
56     if (Math.min(Math.abs(Math.PI-(-newPos[3]+newPos[2]))
57                  ,Math.abs(Math.PI-(newPos[3]-newPos[2])))<1.1694) return false;
58     if (Debug) System.out.println("staticTest␣II");
59
60     //Berrechnen der benötigten Punkte
61     pointsX[0]    =    0;
62     pointsX[1]    =    pointsX[0]+length[1]*Math.sin(newPos[1]);
63     pointsX[2]    =    pointsX[1]+length[2]*Math.sin(newPos[2]);
64     pointsX[3]    =    pointsX[2]+length[3]*Math.sin(newPos[3]);
```

```
65      pointsX[4]    =   pointsX[2]-length[4]*Math.sin(newPos[3]);
66      pointsX[5]    =   pointsX[3]+length[5]*Math.cos(newPos[3]);
67      pointsX[6]    =   pointsX[3]-length[5]*Math.cos(newPos[3]);
68      pointsX[7]    =   pointsX[4]+length[6]*Math.cos(newPos[3]);
69      pointsX[8]    =   pointsX[4]-length[6]*Math.cos(newPos[3]);
70      pointsX[9]    =   pointsX[5]-length[7]*Math.sin(newPos[3]);
71      pointsX[10]   =   pointsX[6]-length[7]*Math.sin(newPos[3]);
72      pointsY[0]    =   length[0];
73      pointsY[1]    =   pointsY[0]+length[1]*Math.cos(newPos[1]);
74      pointsY[2]    =   pointsY[1]+length[2]*Math.cos(newPos[2]);
75      pointsY[3]    =   pointsY[2]+length[3]*Math.cos(newPos[3]);
76      pointsY[4]    =   pointsY[2]-length[4]*Math.cos(newPos[3]);
77      pointsY[5]    =   pointsY[3]-length[5]*Math.sin(newPos[3]);
78      pointsY[6]    =   pointsY[3]+length[5]*Math.sin(newPos[3]);
79      pointsY[7]    =   pointsY[4]-length[6]*Math.sin(newPos[3]);
80      pointsY[8]    =   pointsY[4]+length[6]*Math.sin(newPos[3]);
81      pointsY[9]    =   pointsY[5]-length[7]*Math.cos(newPos[3]);
82      pointsY[10]   =   pointsY[6]-length[7]*Math.cos(newPos[3]);
83      if (Debug) System.out.println("calculate Points");
84
85      //test auf Kollision mit dem Boden
86      for(int j=1;j<11;j++){
87        if (pointsY[j] < groundFunction(pointsX[j]))
88          {if (Debug) System.out.println("--"+j+" "+pointsY[j]) ; return false;}
89      }
90      if (Debug) System.out.println("gound test");
91
92      //Berrechnen der Parameter für Arm 1
93      alpha[0] =Math.abs(newPos[1]);
94      if (alpha[0]<(Math.PI/2)) alpha[0]=alpha[0];
95      else if(alpha[0]<(Math.PI))alpha[0]=alpha[0]-(Math.PI/2);
96      else alpha[0]=alpha[0]-(Math.PI);
97
98      distanceA[0]=(pointsX[1]*pointsY[0]-pointsX[0]*pointsY[1])/
99                      ((pointsY[0]-pointsY[1])+0.00000001);
100
101     distanceB[0]=(pointsX[1]*pointsY[0]-pointsX[0]*pointsY[1])/
102                     ((pointsX[1]-pointsX[0])+0.00000001);
103
104     alpha[0] =Math.atan(distanceB[0]/(distanceA[0]+0.00000001));
105     distanceP[0]=distanceB[0]*Math.cos(alpha[0]);
106     phi[0]=(Math.PI/2)-alpha[0];
107     if (Debug)
108       System.out.println("phi 1= "+phi[0]+" da="+distanceA[0]+
109                          " db="+distanceB[0]+" dp="+distanceP[0]);
110
111     //Berechnen des Abstandes zu Arm 1
112     double[] dArm1={1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d};
113     for(int i=0;i<11;i++){
114       dArm1[i]= pointsX[i]*Math.cos(phi[0])+
115                 pointsY[i]*Math.sin(phi[0])-distanceP[0];
116
117       if (Debug)
118         System.out.println("Punkt: "+i+", Abstand zu Arm1: "+dArm1[i]+
119                            " X: "+pointsX[i]+" Y: "+pointsY[i]);
120     }
121     //Test der Punkte P2..P10 auf Kollision mit dem Arm 1
122     for(int i=2;i<11;i++){
123       if(((pointsX[0]-distance[0]<pointsX[i])&&
124           (pointsX[1]+distance[0]>pointsX[i]))||
125          ((pointsX[0]+distance[0]>pointsX[i])&&
126           (pointsX[1]-distance[0]<pointsX[i]))){
127
128         if( ((pointsY[0]<pointsY[i]) && (pointsY[1]>pointsY[i]))||
129             ((pointsY[0]>pointsY[i]) && (pointsY[1]<pointsY[i]))){
130
```

```
131             if(Math.abs(dArm1[i]) < distance[0])return false;
132           }
133         }
134       }
135       return true;
136     }
137 }
```

E.2.5 roboPositionQueue.java

```
1   /**
2    *Class roboPositionQueue
3    *This class creates a Queue of Position Elements
4    *@author Jens Twiefel
5    *@version 1.0
6    */
7  public class roboPositionQueue{
8    queueElement first=null;
9    queueElement last=null;
10   final boolean Debug=false;
11   int noQueueElements=0;
12
13   /**
14    *Standard Constructer
15    */
16   public roboPositionQueue(){
17   }
18   /**
19    *Test for Elements in the Queue
20    */
21   public boolean hasElements(){
22     if(first!=null) return true;
23     return false;
24   }
25   /**
26    *Get the Number of Elements in the Queue
27    *@return Integer Value
28    */
29   public int getNoQueueElements(){
30     return noQueueElements;
31   }
32   /**
33    *Get the Position of the first Elenemt
34    *@return double Array
35    */
36   public double[] getPosition(){
37     if (Debug) System.out.println("get Element from Queue");
38     noQueueElements--;
39     double[] out = first.getPosition();
40     first=first.getNext();
41     return out;
42   }
43   /**
44    *Add new Element into the Queue
45    */
46   public void addPosition(double[] newPos){
47     if (Debug) System.out.println("add Element to Queue");
48     noQueueElements++;
49     queueElement help = new queueElement(newPos);
50     if (last!=null)  last.setNext(help);
51     if (first==null) first=help;
52     last=help;
53   }
54
55 }
56 /**
57  *This class creates a Position Elements
58  *@author Jens Twiefel
59  *@version 1.0
60  */
61 class queueElement{
62   double[] position;
63   queueElement  next=null;
64   /**
```

```
65      *Constructor for a Queue Element
66      *@param Double Array with
67      */
68    public queueElement(double[] newPos){
69      this.position=newPos;
70    }
71    /**
72     *Set a Poiter on the next Queue Element
73     *
74     */
75    public void setNext(queueElement newElem){
76      this.next=newElem;
77    }
78    /**
79     *Get the next Queue Element
80     *@return queueElement
81     */
82    public queueElement getNext(){
83      return this.next;
84    }
85    /**
86     *Get the Position Array of the Queue Element
87     *@return Double Array
88     */
89    public double[] getPosition(){
90      return this.position;
91    }
92  }
```

E.2.6 roboDemoQueue.java

```
1  /**
2     *Class roboDemoQueue
3     *This class creates a Ring-Queue of Position Elements
4     *@author Jens Twiefel
5     *@version 1.0
6     */
7  public class roboDemoQueue{
8    queueElement first=null;
9    queueElement next=null;
10   queueElement last=null;
11   final boolean Debug=false;
12
13   /**
14    *Standard Constructer
15    */
16   public roboDemoQueue(){
17     double[] helpD1={0,0,0,0,0,0};
18     this.addPosition(helpD1);
19     double[] helpD2={0,0.8,1.57,1.52,0,0};
20     this.addPosition(helpD2);
21     double[] helpD3={1.77,1.63,1.57,3,1.38,0};
22     this.addPosition(helpD3);
23     double[] helpD4={-1.77,0,0.5*Math.PI,0,0,0};
24     this.addPosition(helpD4);
25     double[] helpD5={0.25*Math.PI,1.8,1.73,Math.PI,0.5*Math.PI,0};
26     this.addPosition(helpD5);
27     last.setNext(first);
28     next=first;
29   }
30   /**
31    *Test for Elements in the Queue
32    */
33   public boolean hasElements(){
34     if(first!=null) return true;
35     return false;
36   }
37   /**
38    *Get the Position of the first Elenemt
39    *@return double Array
40    */
41   public double[] getPosition(){
42     double[] out = next.getPosition();
43     next=next.getNext();
44     return out;
45   }
46   /**
47    *Add new Element into the Queue
48    */
49   private void addPosition(double[] newPos){
50     queueElement help = new queueElement(newPos);
51     if (last!=null)  last.setNext(help);
52     if (first==null) first=help;
53     last=help;
54   }
55
56 }
```

E.3 Client

E.3.1 roboApplet.java

```
1  import java.awt.*;
2  import java.applet.*;
3  import vrml.external.*;
4  import vrml.external.field.*;
5  import java.io.*;
6  import java.net.*;
7  import java.util.*;
8   /**
9     *Class roboApplet
10    *@author Jens Twiefel
11    *@version 1.0
12    */
13 public class roboApplet extends Applet implements Runnable,EventOutObserver{
14                 boolean       followMode          = true;
15                 boolean       VRML                = false;
16                 boolean       started             = false;
17           final boolean       Debug               = false;
18           final boolean       Debug2              = false;
19           final boolean       Debug3              = false;
20           final boolean       Debug4              = false;
21                 int           received            = 0;
22                 int           send                = 0;
23                 int           sessionID           = 0;
24                 Thread        thread;
25                 TextArea      text                = new TextArea(2,60);
26                 TextField     input               = new TextField();
27                 Panel         panel1              = new Panel();
28                 Panel         panel2              = new Panel();
29                 Panel         panel3              = new Panel();
30                 Panel         panel4              = new Panel();
31                 Panel         panel5              = new Panel();
32                 Button        buttonGetActual     = new Button("<< get actual "+
33                                                       "values");
34                 Button        buttonStart         = new Button("Start without "+
35                                                       "VRML (other)");
36                 Button        buttonStartVrml     = new Button("Start with VRML"+
37                                                       "( IE,MS-VM,CortonaVRML)");
38                 Button        buttonFollow        = new Button("Follow Mode Off");
39                 Button        buttonPos           = new Button("Send Position");
40                 BorderLayout  borderLayout1       = new BorderLayout();
41                 BorderLayout  borderLayout2       = new BorderLayout();
42                 BorderLayout  borderLayout3       = new BorderLayout();
43                 GridLayout    gridLayout1         = new GridLayout(7,1);
44                 GridLayout    gridLayout2         = new GridLayout(7,1);
45                 GridLayout    gridLayout3         = new GridLayout(7,2);
46                 Scrollbar     scrollbar1          = new Scrollbar();
47                 Scrollbar     scrollbar2          = new Scrollbar();
48                 Scrollbar     scrollbar3          = new Scrollbar();
49                 Scrollbar     scrollbar4          = new Scrollbar();
50                 Scrollbar     scrollbar5          = new Scrollbar();
51                 Scrollbar     scrollbar6          = new Scrollbar();
52                 Label         lchat               = new Label("Command:");
53                 Label         lMessageRecived     = new Label("Received "+
54                                                       "Messages : 0            "+
55                                                       "           ");
56                 Label         label1              = new Label();
57                 Label         label2              = new Label();
58                 Label         label3              = new Label();
59                 Label         label4              = new Label();
60                 Label         labelSetAxis1       = new Label();
```

```
61                Label          labelSetAxis2      = new Label();
62                Label          labelSetAxis3      = new Label();
63                Label          labelSetAxis4      = new Label();
64                Label          labelSetAxis5      = new Label();
65                Label          labelSetAxis6      = new Label();
66                Label          labelActualAxis1   = new Label();
67                Label          labelActualAxis2   = new Label();
68                Label          labelActualAxis3   = new Label();
69                Label          labelActualAxis4   = new Label();
70                Label          labelActualAxis5   = new Label();
71                Label          labelActualAxis6   = new Label();
72  static final  float          pi                 = 3.141592653f;
73                float[]        nullPosFloat       = {0f,0f,0f,0f,0f,0f};
74                float[]        Pos1Float          = {0f,0f,0f,0f,0f,0f};
75                float[]        Pos2Float          = {0f,0f,0f,0f,0f,0f};
76                float[]        Pos3Float          = {0f,0f,0f,0f,0f,0f};
77                float[]        Pos4Float          = {0f,0f,0f,0f,0f,0f};
78                float[]        Pos5Float          = {0f,0f,0f,0f,0f,0f};
79                float[]        Pos6Float          = {0f,0f,0f,0f,0f,0f};
80                float[]        Pos7Float          = {0f,0f,0f,0f,0f,0f};
81                float[]        Pos8Float          = {0f,0f,0f,0f,0f,0f};
82                float[]        Pos9Float          = {0f,0f,0f,0f,0f,0f};
83                vrmlRobo       vR                 = null;
84                roboPosition   roboPos            = null;
85                roboPosition   resultPos          = null;
86                roboPosition   localRoboPos       = new roboPosition();
87  static final  Color          buttonColor        = Color.white;
88  static final  Color          backgroundColor    = new Color(252, 252, 252);
89  static final  String[]       errorStr           = {"No Error",
90                                                     "illegal Position",
91                                                     "Robo in use","",
92                                                     "internal Server Error"};
93
94  protected int getBrowserIndexParameter(){
95    try{
96      return Integer.parseInt(getParameter("BrowserIndex"));
97    }catch (NumberFormatException e){
98      return 0;
99    }
100 }
101 /**
102  *start Method
103  */
104 public void start() {
105   thread = new Thread(this);
106   thread.start();
107 }
108 /**
109  *run Method
110  */
111 public void run() {
112   while (true) {
113     text.appendText(getRoboterPosition());
114   }
115 }
116 /**
117  *stop Method
118  */
119 public void stop() {
120   thread.stop();
121   thread = null;
122 }
123  /**
124   *initialisation Method
125   */
126 public void init(){
```

```
127     //die GUI aufgauen...
128      URL codebase = getCodeBase();
129      setBackground(backgroundColor);
130      this.setLayout(borderLayout1);
131      panel4.setLayout(borderLayout2);
132      panel5.setLayout(borderLayout3);
133      panel1.setLayout(gridLayout1);
134      panel2.setLayout(gridLayout2);
135      panel3.setLayout(gridLayout3);
136      scrollbar1.setMaximum(3141);
137      scrollbar1.setMinimum(-3141);
138      scrollbar1.setOrientation(0);
139      scrollbar1.setPageIncrement(1);
140      scrollbar1.setBlockIncrement(1);
141      scrollbar2.setMaximum(2269);
142      scrollbar2.setMinimum(-349);
143      scrollbar2.setOrientation(0);
144      scrollbar2.setPageIncrement(1);
145      scrollbar2.setBlockIncrement(1);
146      scrollbar3.setMaximum(3141);
147      scrollbar3.setMinimum(-3141);
148      scrollbar3.setOrientation(0);
149      scrollbar3.setPageIncrement(1);
150      scrollbar3.setBlockIncrement(1);
151      scrollbar4.setMaximum(3141*2);
152      scrollbar4.setMinimum(-3141*2);
153      scrollbar4.setOrientation(0);
154      scrollbar4.setPageIncrement(1);
155      scrollbar4.setBlockIncrement(1);
156      scrollbar5.setMaximum(3141);
157      scrollbar5.setMinimum(-3141);
158      scrollbar5.setOrientation(0);
159      scrollbar5.setPageIncrement(1);
160      scrollbar5.setBlockIncrement(1);
161      scrollbar6.setMaximum(49000);
162      scrollbar6.setMinimum(0);
163      scrollbar6.setOrientation(0);
164      scrollbar6.setPageIncrement(1);
165      scrollbar6.setBlockIncrement(1);
166      label1.setText("Set Value                                                    ");
167      label2.setText("Actual Value                                                 ");
168      label3.setText("                                                             ");
169      label4.setText("                                                             ");
170      labelSetAxis1.setText("Axis 1 :   "+(scrollbar1.getValue()*2*pi/1000)+"°");
171      labelSetAxis2.setText("Axis 2 :   "+(scrollbar2.getValue()*2*pi/1000)+"°");
172      labelSetAxis3.setText("Axis 3 :   "+(scrollbar3.getValue()*2*pi/1000)+"°");
173      labelSetAxis4.setText("Axis 4 :   "+(scrollbar4.getValue()*2*pi/1000)+"°");
174      labelSetAxis5.setText("Axis 5 :   "+(scrollbar5.getValue()*2*pi/1000)+"°");
175      labelSetAxis6.setText("Gripper:   "+(scrollbar6.getValue()/1000)+" mm");
176      buttonPos.setEnabled(false);
177      text.setEditable(false);
178      input.setEditable(true);
179      text.setRows(5);
180      this.add(panel1              , BorderLayout.EAST   );
181      panel1.add(label2            , null                );
182      panel1.add(labelActualAxis1  , null                );
183      panel1.add(labelActualAxis2  , null                );
184      panel1.add(labelActualAxis3  , null                );
185      panel1.add(labelActualAxis4  , null                );
186      panel1.add(labelActualAxis5  , null                );
187      panel1.add(labelActualAxis6  , null                );
188      this.add(panel3              , BorderLayout.CENTER );
189      panel3.add(label3            , null                );
190      panel3.add(label4            , null                );
191      panel3.add(labelSetAxis1     , null                );
192      panel3.add(buttonStartVrml   , null                );
```

```
193     panel3.add(labelSetAxis2     , null                );
194     panel3.add(buttonStart       , null                );
195     panel3.add(labelSetAxis3     , null                );
196     panel3.add(buttonGetActual   , null                );
197     panel3.add(labelSetAxis4     , null                );
198     panel3.add(buttonFollow      , null                );
199     panel3.add(labelSetAxis5     , null                );
200     panel3.add(buttonPos         , null                );
201     panel3.add(labelSetAxis6     , null                );
202     this.add(panel2              , BorderLayout.WEST   );
203     panel2.add(label1            , null                );
204     panel2.add(scrollbar1                              );
205     panel2.add(scrollbar2                              );
206     panel2.add(scrollbar3                              );
207     panel2.add(scrollbar4                              );
208     panel2.add(scrollbar5                              );
209     panel2.add(scrollbar6                              );
210     this.add(panel4              , BorderLayout.SOUTH  );
211     panel4.add(text              , BorderLayout.CENTER );
212     panel4.add(panel5            , BorderLayout.SOUTH  );
213     panel5.add("West"            , lchat               );
214     panel5.add("East"            , lMessageRecived     );
215     panel5.add("Center"          , input               );
216   }
217   /**
218    *request the actual Robo Position from the Server
219    */
220   private String getRoboterPosition() {
221    //Neue Soll Positionen vom Server hohlen
222     synchronized(this){
223       try{
224         if (Debug) text.appendText("<-get ");
225         roboPos = null;
226         while (roboPos == null) {
227           try {
228             URL url = new URL(getCodeBase(), "/servlet/RoboServlet");
229             HttpMessage msg = new HttpMessage(url);
230             InputStream in = msg.sendGetMessage();
231             ObjectInputStream result = new ObjectInputStream(in);
232             Object obj = result.readObject();
233             roboPos = (roboPosition)obj;
234             in.close();
235           }catch (SocketException e) {
236             try{ Thread.sleep(50); } catch (InterruptedException ignored){}
237           }catch (FileNotFoundException e) {
238             try { Thread.sleep(50); } catch (InterruptedException ignored){}
239           }catch (Exception e) {
240             try { Thread.sleep(10); } catch (InterruptedException ignored){}
241           }
242         }
243         if (VRML) vR.setVRMLactual(roboPos.getRadPosition());
244         labelActualAxis1.setText("Axis 1 :    "+
245                                  ((roboPos.getRadPosition(0)*180)/pi)+"°");
246         labelActualAxis2.setText("Axis 2 :    "+
247                                  ((roboPos.getRadPosition(1)*180)/pi)+"°");
248         labelActualAxis3.setText("Axis 3 :    "+
249                                  ((roboPos.getRadPosition(2)*180)/pi)+"°");
250         labelActualAxis4.setText("Axis 4 :    "+
251                                  ((roboPos.getRadPosition(3)*180)/pi)+"°");
252         labelActualAxis5.setText("Axis 5 :    "+
253                                  ((roboPos.getRadPosition(4)*180)/pi)+"°");
254         labelActualAxis6.setText("Gripper:    "+
255                                  (roboPos.getRadPosition(5))+" mm ");
256         received++;
257         lMessageRecived.setText("Received Messages: "+received);
258         try {Thread.sleep(500);} catch (InterruptedException ignored){}
```

```
259         if (Debug) text.appendText("␣message->\n");
260         if (roboPos.getMsgStr()!=null) return roboPos.getMsgStr();
261         return "";
262       }catch(Exception ignored){}
263       return "";
264     }
265   }
266   /**
267    *send a new position to server
268    */
269   private void setRoboterPosition() {
270    //neue Position an den Server senden
271     try {
272       if (Debug4) text.appendText("<-broad");
273       URL url = new URL(getCodeBase(), "/servlet/RoboServlet");
274       HttpMessage msg = new HttpMessage(url);
275       localRoboPos.setSessionID(sessionID);
276       InputStream in = msg.sendPostMessage(localRoboPos);
277       ObjectInputStream result = new ObjectInputStream(in);
278       Object obj = result.readObject();
279       resultPos = (roboPosition)obj;
280       in.close();
281       sessionID=resultPos.getSessionID();
282       if(resultPos.getErrorNo()!=0)
283         text.appendText("Error:␣"+errorStr[resultPos.getErrorNo()]+"␣\n");
284       if (Debug4) text.appendText("cast->\n");
285       localRoboPos.setErrorNo(0);
286     }
287     catch (SocketException ignored) {}
288     catch (FileNotFoundException ignored) {}
289     catch (Exception ignored) {}
290     send++;
291     if (Debug4) text.appendText(send+ ".␣Position␣send␣\n");
292   }
293   /**
294    *this method is automatical called from the EAI
295    *@param EventOut the callback event
296    *@param timestamp a timestam
297    *@param obj not used
298    */
299   public void callback(EventOut eventOut, double timestamp, Object obj){
300     //Methode wird vom EAI aufgerufeb
301     vR.Fcallback();
302     vrmlChange(vR.getVRML());
303     sendPos();
304   }
305   /**
306    *this method refresh the GUI
307    *@param newPos the new Position
308    */
309   private void vrmlChange(float[] newPos){
310    //Die GUI wird mit neuen Werten versorgt
311     try{
312       scrollbar1.setValue((int)((newPos[0])*1000));
313       scrollbar2.setValue((int)((newPos[1])*1000));
314       scrollbar3.setValue((int)((newPos[2])*1000));
315       scrollbar4.setValue((int)((newPos[3])*1000));
316       scrollbar5.setValue((int)((newPos[4])*1000));
317       localRoboPos.setRadPosition(newPos[0],0);
318       localRoboPos.setRadPosition(newPos[1],1);
319       localRoboPos.setRadPosition(newPos[2],2);
320       localRoboPos.setRadPosition(newPos[3],3);
321       localRoboPos.setRadPosition(newPos[4],4);
322       labelSetAxis1.setText("Axis␣1␣:␣␣␣"+((newPos[0]*180)/pi)+"°");
323       labelSetAxis2.setText("Axis␣2␣:␣␣␣"+((newPos[1]*180)/pi)+"°");
324       labelSetAxis3.setText("Axis␣3␣:␣␣␣"+((newPos[2]*180)/pi)+"°");
```

```
325       labelSetAxis4.setText("Axis 4 : "+((newPos[3]*180)/pi)+"°");
326       labelSetAxis5.setText("Axis 5 : "+((newPos[4]*180)/pi)+"°");
327       labelSetAxis6.setText("Gripper: "+
328                             (localRoboPos.getRadPosition(5))+" mm ");
329     }catch(Exception ignored){}
330   }
331   /**
332    *set the desired values to the actual values
333    */
334   private void getActual2Model(){
335    //syncronisieren des Soll-Wertes mit dem Ist-Wert
336     boolean run=true;
337     while(run){
338       try{
339         scrollbar1.setValue((int)(roboPos.getRadPosition(0)*1000));
340         scrollbar2.setValue((int)(roboPos.getRadPosition(1)*1000));
341         scrollbar3.setValue((int)(roboPos.getRadPosition(2)*1000));
342         scrollbar4.setValue((int)(roboPos.getRadPosition(3)*1000));
343         scrollbar5.setValue((int)(roboPos.getRadPosition(4)*1000));
344         scrollbar6.setValue((int)(roboPos.getRadPosition(5)*1000));
345         labelSetAxis1.setText("Axis 1 : "+
346                               ((roboPos.getRadPosition(0)*180)/pi)+"°");
347         labelSetAxis2.setText("Axis 2 : "+
348                               ((roboPos.getRadPosition(1)*180)/pi)+"°");
349         labelSetAxis3.setText("Axis 3 : "+
350                               ((roboPos.getRadPosition(2)*180)/pi)+"°");
351         labelSetAxis4.setText("Axis 4 : "+
352                               ((roboPos.getRadPosition(3)*180)/pi)+"°");
353         labelSetAxis5.setText("Axis 5 : "+
354                               ((roboPos.getRadPosition(4)*180)/pi)+"°");
355         labelSetAxis6.setText("Gripper: "+
356                               (roboPos.getRadPosition(5))+" mm ");
357         //um eine "Rückkopplung zu vermeiden
358         if (VRML){
359           boolean help=followMode;
360           followMode=false;
361           vR.setVRMLset(roboPos.getRadPosition());
362           followMode=help;
363         }
364         run=false;
365       }catch(Exception ignored){}
366     }
367   }
368   /**
369    *send a new Position to the Server
370    */
371   private void sendPos(){
372    //falls der follow Mode aktiv ist wird eine neue Soll-Position gesendet
373     if (followMode) setRoboterPosition();
374   }
375   /**
376    *parse the input String
377    *@param in input String
378    */
379   private void parseInput(String in){
380    //Die Befehlseingabe auswerten
381     if(in.startsWith("gotoHP")){
382       scrollbar6.setValue((int)((nullPosFloat[5])*1000));
383       localRoboPos.setRadPosition(nullPosFloat[5],5);
384       vrmlChange(nullPosFloat);
385       String help="goto ";
386       for (int j=0;j<6;j++) help+="  "+j+":  "+nullPosFloat[j];
387       text.appendText(help+"\n");
388       if(in.indexOf("-f")>0){
389         setRoboterPosition();
390         text.appendText(getRoboterPosition());
```

```
391         }
392         return;
393       }
394       if(in.startsWith("goto1")){
395         scrollbar6.setValue((int)((Pos1Float[5])*1000));
396         localRoboPos.setRadPosition(Pos1Float[5],5);
397         vrmlChange(Pos1Float);
398         String help="goto␣";
399         for (int j=0;j<6;j++) help+="␣␣"+j+":␣␣"+Pos1Float[j];
400         text.appendText(help+"\n");
401         if(in.indexOf("-f")>0){
402           setRoboterPosition();
403           text.appendText(getRoboterPosition());
404         }
405         return;
406       }
407       if(in.startsWith("goto2")){
408         scrollbar6.setValue((int)((Pos2Float[5])*1000));
409         localRoboPos.setRadPosition(Pos2Float[5],5);
410         vrmlChange(Pos2Float);
411         String help="goto␣";
412         for (int j=0;j<6;j++) help+="␣␣"+j+":␣␣"+Pos2Float[j];
413         text.appendText(help+"\n");
414         if(in.indexOf("-f")>0){
415           setRoboterPosition();
416           text.appendText(getRoboterPosition());
417         }
418         return;
419       }
420       if(in.startsWith("goto3")){
421         scrollbar6.setValue((int)((Pos3Float[5])*1000));
422         localRoboPos.setRadPosition(Pos3Float[5],5);
423         vrmlChange(Pos3Float);
424         String help="goto␣";
425         for (int j=0;j<6;j++) help+="␣␣"+j+":␣␣"+Pos3Float[j];
426         text.appendText(help+"\n");
427         if(in.indexOf("-f")>0){
428           setRoboterPosition();
429           text.appendText(getRoboterPosition());
430         }
431         return;
432       }
433       if(in.startsWith("goto4")){
434         scrollbar6.setValue((int)((Pos3Float[5])*1000));
435         localRoboPos.setRadPosition(Pos3Float[5],5);
436         vrmlChange(Pos4Float);
437         String help="goto␣";
438         for (int j=0;j<6;j++) help+="␣␣"+j+":␣␣"+Pos4Float[j];
439         text.appendText(help+"\n");
440         if(in.indexOf("-f")>0){
441           setRoboterPosition();
442           text.appendText(getRoboterPosition());
443         }
444         return;
445       }
446       if(in.startsWith("goto5")){
447         scrollbar6.setValue((int)((Pos5Float[5])*1000));
448         localRoboPos.setRadPosition(Pos5Float[5],5);
449         vrmlChange(Pos5Float);
450         String help="goto␣";
451         for (int j=0;j<6;j++) help+="␣␣"+j+":␣␣"+Pos5Float[j];
452         text.appendText(help+"\n");
453         if(in.indexOf("-f")>0){
454           setRoboterPosition();
455           text.appendText(getRoboterPosition());
456         }
```

```
457       return;
458     }
459     if(in.startsWith("goto6")){
460       scrollbar6.setValue((int)((Pos6Float[5])*1000));
461       localRoboPos.setRadPosition(Pos6Float[5],5);
462       vrmlChange(Pos6Float);
463       String help="goto ";
464       for (int j=0;j<6;j++) help+="  "+j+": "+Pos6Float[j];
465       text.appendText(help+"\n");
466       if(in.indexOf("-f")>0){
467         setRoboterPosition();
468         text.appendText(getRoboterPosition());
469       }
470       return;
471     }
472     if(in.startsWith("goto7")){
473       scrollbar6.setValue((int)((Pos7Float[5])*1000));
474       localRoboPos.setRadPosition(Pos7Float[5],5);
475       vrmlChange(Pos7Float);
476       String help="goto ";
477       for (int j=0;j<6;j++) help+="  "+j+": "+Pos7Float[j];
478       text.appendText(help+"\n");
479       if(in.indexOf("-f")>0){
480         setRoboterPosition();
481         text.appendText(getRoboterPosition());
482       }
483       return;
484     }
485     if(in.startsWith("goto8")){
486       scrollbar6.setValue((int)((Pos8Float[5])*1000));
487       localRoboPos.setRadPosition(Pos8Float[5],5);
488       vrmlChange(Pos8Float);
489       String help="goto ";
490       for (int j=0;j<6;j++) help+="  "+j+": "+Pos8Float[j];
491       text.appendText(help+"\n");
492       if(in.indexOf("-f")>0){
493         setRoboterPosition();
494         text.appendText(getRoboterPosition());
495       }
496       return;
497     }
498     if(in.startsWith("goto9")){
499       scrollbar6.setValue((int)((Pos9Float[5])*1000));
500       localRoboPos.setRadPosition(Pos9Float[5],5);
501       vrmlChange(Pos9Float);
502       String help="goto ";
503       for (int j=0;j<6;j++) help+="  "+j+": "+Pos9Float[j];
504       text.appendText(help+"\n");
505       if(in.indexOf("-f")>0){
506         setRoboterPosition();
507         text.appendText(getRoboterPosition());
508       }
509       return;
510     }
511
512     if(in.startsWith("set1")){
513       text.appendText(getRoboterPosition());
514       getActual2Model();
515       String help="new Position 1:";
516       for (int j=0;j<6;j++){
517         Pos1Float[j]=localRoboPos.getRadPosition(j);
518         help+="  "+j+": "+Pos1Float[j];
519       }
520       text.appendText(help+"\n");
521       return;
522     }
```

```
523     if(in.startsWith("set2")){
524       text.appendText(getRoboterPosition());
525       getActual2Model();
526       String help="new Position 2:";
527       for (int j=0;j<6;j++){
528         Pos2Float[j]=localRoboPos.getRadPosition(j);
529         help+="  "+j+":  "+Pos2Float[j];
530       }
531       text.appendText(help+"\n");
532       return;
533     }
534     if(in.startsWith("set3")){
535       text.appendText(getRoboterPosition());
536       getActual2Model();
537       String help="new Position 3:";
538       for (int j=0;j<6;j++){
539         Pos3Float[j]=localRoboPos.getRadPosition(j);
540         help+="  "+j+":  "+Pos3Float[j];
541       }
542       text.appendText(help+"\n");
543       return;
544     }
545     if(in.startsWith("set4")){
546       text.appendText(getRoboterPosition());
547       getActual2Model();
548       String help="new Position 4:";
549       for (int j=0;j<6;j++){
550         Pos4Float[j]=localRoboPos.getRadPosition(j);
551         help+="  "+j+":  "+Pos4Float[j];
552       }
553       text.appendText(help+"\n");
554       return;
555     }
556     if(in.startsWith("set5")){
557       text.appendText(getRoboterPosition());
558       getActual2Model();
559       String help="new Position 5:";
560       for (int j=0;j<6;j++){
561         Pos5Float[j]=localRoboPos.getRadPosition(j);
562         help+="  "+j+":  "+Pos5Float[j];
563       }
564       text.appendText(help+"\n");
565       return;
566     }
567     if(in.startsWith("set6")){
568       text.appendText(getRoboterPosition());
569       getActual2Model();
570       String help="new Position 6:";
571       for (int j=0;j<6;j++){
572         Pos6Float[j]=localRoboPos.getRadPosition(j);
573         help+="  "+j+":  "+Pos6Float[j];
574       }
575       text.appendText(help+"\n");
576       return;
577     }
578     if(in.startsWith("set7")){
579       text.appendText(getRoboterPosition());
580       getActual2Model();
581       String help="new Position 7:";
582       for (int j=0;j<6;j++){
583         Pos7Float[j]=localRoboPos.getRadPosition(j);
584         help+="  "+j+":  "+Pos7Float[j];
585       }
586       text.appendText(help+"\n");
587       return;
588     }
```

```
589     if(in.startsWith("set8")){
590       text.appendText(getRoboterPosition());
591       getActual2Model();
592       String help="new Position 8:";
593       for (int j=0;j<6;j++){
594         Pos8Float[j]=localRoboPos.getRadPosition(j);
595         help+="  "+j+":  "+Pos8Float[j];
596       }
597       text.appendText(help+"\n");
598       return;
599     }
600     if(in.startsWith("set9")){
601       text.appendText(getRoboterPosition());
602       getActual2Model();
603       String help="new Position 9:";
604       for (int j=0;j<6;j++){
605         Pos9Float[j]=localRoboPos.getRadPosition(j);
606         help+="  "+j+":  "+Pos9Float[j];
607       }
608       text.appendText(help+"\n");
609       return;
610     }
611     if(in.startsWith("startdemo")){
612       localRoboPos.setErrorNo(1);
613       setRoboterPosition();
614       text.setText("Demo started");
615       return;
616     }
617     if(in.startsWith("stopdemo")){
618       localRoboPos.setErrorNo(2);
619       setRoboterPosition();
620       text.setText("Demo stoped");
621       return;
622     }
623     if(in.startsWith("whereG")){
624       text.appendText(localRoboPos.toStringGrad()+"\n");
625       return;
626     }
627     if(in.startsWith("whereR")){
628       text.appendText(localRoboPos.toString()+"\n");
629       return;
630     }
631     if(in.startsWith("about")){
632       text.appendText("Jens Twiefel, jens@twiefel.de\n");
633       return;
634     }
635     if(in.startsWith("help")){
636       text.appendText("clear          the textarea\n"
637                       +"goto1..9 (-f) move the roboter to a defined ↩
                            position,\n"
638                       +"              the parameter f transmit the ↩
                            position "
639                       +"directly to the server\n"
640                       +"gotoHP (-f)   move the roboter to the home ↩
                            position"
641                       +"set1..9       save a position"
642                       +"startdemo     start the demo-mode"
643                       +"stopdemo      stop the demo-mode");
644       return;
645     }
646     if(in.startsWith("send")){
647       text.appendText(send+" send Messages\n");
648       return;
649     }
650     if(in.startsWith("received")){
651       text.appendText(received+" received Messages\n");
```

```
652         return;
653       }
654       if(in.startsWith("clear")){
655         text.setText("");
656         return;
657       }
658       if(in.startsWith("fmode")){
659         if (followMode){text.appendText("Follow Mode on! \n");return;}
660         text.appendText("Follow Mode off! \n");return;
661       }
662       if(in.startsWith("dmode")){
663         text.appendText("Debug Mode"+Debug+" , "+Debug2+"\n");
664         return;
665       }
666       text.appendText("Invalid Command\n");
667     }
668     /**
669      *Event handeling
670      *@param event the incomming event
671      */
672     public boolean handleEvent(Event event) {
673      //Event Verarbeitung
674       switch (event.id) {
675         case Event.ACTION_EVENT:
676           if (event.target == input) {
677             parseInput(input.getText());
678             input.setText("");
679             return true;
680           }
681           if (event.target == buttonGetActual) {
682             getActual2Model();
683             return true;
684           }
685           if (event.target == buttonStart) {
686             buttonStartVrml.setLabel("Start VRML (IE, MS-VM, CortonaVRML)");
687             buttonStart.setLabel("manual Refresh");
688             text.appendText(getRoboterPosition());
689             getActual2Model();
690             return true;
691           }
692           if (event.target == buttonStartVrml) {
693             try{
694               vR = new vrmlRobo(this,getBrowserIndexParameter());
695               VRML=true;
696             }catch(Exception e){text.appendText("VRML not startet !! \n");}
697             buttonStart.setLabel("manual Refresh");
698             buttonStartVrml.setEnabled(false);
699             text.appendText(getRoboterPosition());
700             getActual2Model();
701             return true;
702           }
703           if (event.target == buttonFollow) {
704             if (followMode){
705               buttonFollow.setLabel("Follow Mode On");
706               buttonPos.setEnabled(true);
707               followMode=false;
708               return true;
709             }
710             buttonFollow.setLabel("Follow Mode Off");
711             buttonPos.setEnabled(false);
712             followMode=true;
713             return true;
714          }
715           if (event.target == buttonPos) {
716             setRoboterPosition();
717             return true;
```

```
718           }
719         case 602:
720         case 601:
721         case 605:
722         {
723           localRoboPos.setRadPosition((((float)scrollbar1.getValue())/1000),0);
724           localRoboPos.setRadPosition((((float)scrollbar2.getValue())/1000),1);
725           localRoboPos.setRadPosition((((float)scrollbar3.getValue())/1000),2);
726           localRoboPos.setRadPosition((((float)scrollbar4.getValue())/1000),3);
727           localRoboPos.setRadPosition((((float)scrollbar5.getValue())/1000),4);
728           localRoboPos.setRadPosition((((float)scrollbar6.getValue())/1000),5);
729           labelSetAxis1.setText("Axis 1 :   "+
730                                 ((localRoboPos.getRadPosition(0)*180)/pi)+"°");
731           labelSetAxis2.setText("Axis 2 :   "+
732                                 ((localRoboPos.getRadPosition(1)*180)/pi)+"°");
733           labelSetAxis3.setText("Axis 3 :   "+
734                                 ((localRoboPos.getRadPosition(2)*180)/pi)+"°");
735           labelSetAxis4.setText("Axis 4 :   "+
736                                 ((localRoboPos.getRadPosition(3)*180)/pi)+"°");
737           labelSetAxis5.setText("Axis 5 :   "+
738                                 ((localRoboPos.getRadPosition(4)*180)/pi)+"°");
739           labelSetAxis6.setText("Gripper:   "+
740                                 (localRoboPos.getRadPosition(5))+" mm ");
741           if (Debug3)text.appendText(localRoboPos.toString());
742           if (VRML) vR.setVRMLset(localRoboPos.getRadPosition());
743           sendPos();
744           return true;
745         }
746       }
747       if (Debug3) text.appendText(event.id+"\n");
748       return false;
749     }
750 }
```

E.3.2 vrmlRobo.java

```
1  import java.awt.*;
2  import java.applet.*;
3  import vrml.external.*;
4  import vrml.external.field.*;
5  import java.io.*;
6  import java.net.*;
7  import java.util.*;
8    /**
9     *Class vrmlRobo
10    *@author Jens Twiefel
11    *@version 1.0
12    */
13 public class vrmlRobo{
14
15   //Deklerationen
16   roboApplet          myApplet;
17   Browser             browser;
18   EventOutSFRotation  callbackSetAxis1     = null;
19   EventOutSFRotation  callbackSetAxis2     = null;
20   EventOutSFRotation  callbackSetAxis3     = null;
21   EventOutSFRotation  callbackSetAxis4     = null;
22   EventOutSFRotation  callbackSetAxis5     = null;
23   EventInSFRotation   rotArm1In            = null;
24   EventInSFRotation   rotArm2In            = null;
25   EventInSFRotation   rotArm3In            = null;
26   EventInSFRotation   rotArm4In            = null;
27   EventInSFRotation   rotArm5In            = null;
28   EventInSFRotation   rotArm1aIn           = null;
29   EventInSFRotation   rotArm2aIn           = null;
30   EventInSFRotation   rotArm3aIn           = null;
31   EventInSFRotation   rotArm4aIn           = null;
32   EventInSFRotation   rotArm5aIn           = null;
33   float[]             setAngelAxis1;
34   float[]             setAngelAxis2;
35   float[]             setAngelAxis3;
36   float[]             setAngelAxis4;
37   float[]             setAngelAxis5;
38   float[]             actualAngelAxis1;
39   float[]             actualAngelAxis2;
40   float[]             actualAngelAxis3;
41   float[]             actualAngelAxis4;
42   float[]             actualAngelAxis5;
43   float[]             helpAxis1;
44   float[]             helpAxis2;
45   float[]             helpAxis3;
46   float[]             helpAxis4;
47   float[]             helpAxis5;
48   float[]             actualAngel          = {0,0,0,0,0,0};
49
50   /**
51    *Constructer
52    *@param newApplet
53    */
54   public vrmlRobo(roboApplet newApplet,int newIndex){
55     myApplet=newApplet;
56     int index = newIndex;
57     //Browser init
58     for (int i = 0; i < 20; ++i){
59       browser = Browser.getBrowser(myApplet, "", index);
60       if (browser != null)
61       break;
62       try {Thread.sleep(200);}
63       catch (InterruptedException e){break;}
64     }
```

```
65      if (browser!= null){
66        //Knoten initialisieren
67        Node ActualAxis1Node = browser.getNode("Arm1a");
68        Node ActualAxis2Node = browser.getNode("Arm2a");
69        Node ActualAxis3Node = browser.getNode("Arm3a");
70        Node ActualAxis4Node = browser.getNode("Arm4a");
71        Node ActualAxis5Node = browser.getNode("Arm5a");
72        Node SetAxis1Node = browser.getNode("Arm1");
73        Node SetAxis2Node = browser.getNode("Arm2");
74        Node SetAxis3Node = browser.getNode("Arm3");
75        Node SetAxis4Node = browser.getNode("Arm4");
76        Node SetAxis5Node = browser.getNode("Arm5");
77        //callback initialisieren
78        callbackSetAxis1 =
79              (EventOutSFRotation) SetAxis1Node.getEventOut("rotation");
80        callbackSetAxis2 =
81              (EventOutSFRotation) SetAxis2Node.getEventOut("rotation");
82        callbackSetAxis3 =
83              (EventOutSFRotation) SetAxis3Node.getEventOut("rotation");
84        callbackSetAxis4 =
85              (EventOutSFRotation) SetAxis4Node.getEventOut("rotation");
86        callbackSetAxis5 =
87              (EventOutSFRotation) SetAxis5Node.getEventOut("rotation");
88        callbackSetAxis1.advise(myApplet, null);
89        callbackSetAxis2.advise(myApplet, null);
90        callbackSetAxis3.advise(myApplet, null);
91        callbackSetAxis4.advise(myApplet, null);
92        callbackSetAxis5.advise(myApplet, null);
93        setAngelAxis1     = callbackSetAxis1.getValue();
94        setAngelAxis2     = callbackSetAxis2.getValue();
95        setAngelAxis3     = callbackSetAxis3.getValue();
96        setAngelAxis4     = callbackSetAxis4.getValue();
97        setAngelAxis5     = callbackSetAxis1.getValue();
98        actualAngelAxis1 = callbackSetAxis1.getValue();
99        actualAngelAxis2 = callbackSetAxis2.getValue();
100       actualAngelAxis3 = callbackSetAxis3.getValue();
101       actualAngelAxis4 = callbackSetAxis4.getValue();
102       actualAngelAxis5 = callbackSetAxis5.getValue();
103       helpAxis1         = callbackSetAxis1.getValue();
104       helpAxis2         = callbackSetAxis2.getValue();
105       helpAxis3         = callbackSetAxis3.getValue();
106       helpAxis4         = callbackSetAxis4.getValue();
107       helpAxis5         = callbackSetAxis5.getValue();
108       rotArm1In    = (EventInSFRotation)ActualAxis1Node.getEventIn("rotation");
109       rotArm2In    = (EventInSFRotation)ActualAxis2Node.getEventIn("rotation");
110       rotArm3In    = (EventInSFRotation)ActualAxis3Node.getEventIn("rotation");
111       rotArm4In    = (EventInSFRotation)ActualAxis4Node.getEventIn("rotation");
112       rotArm5In    = (EventInSFRotation)ActualAxis5Node.getEventIn("rotation");
113       rotArm1aIn   = (EventInSFRotation)SetAxis1Node.getEventIn("rotation");
114       rotArm2aIn   = (EventInSFRotation)SetAxis2Node.getEventIn("rotation");
115       rotArm3aIn   = (EventInSFRotation)SetAxis3Node.getEventIn("rotation");
116       rotArm4aIn   = (EventInSFRotation)SetAxis4Node.getEventIn("rotation");
117       rotArm5aIn   = (EventInSFRotation)SetAxis5Node.getEventIn("rotation");
118     }
119   }
120    /**
121    *set the actual VRML-Robot to a new Position
122    *
123    *@param newPos Float-Array of the new Position
124    */
125   public void setVRMLactual(float[] newPos){
126     actualAngelAxis1[0]=0;
127     actualAngelAxis2[0]=0;
128     actualAngelAxis3[0]=0;
129     actualAngelAxis4[0]=0;
130     actualAngelAxis5[0]=0;
```

```
131       actualAngelAxis1[1]=1;
132       actualAngelAxis2[1]=1;
133       actualAngelAxis3[1]=1;
134       actualAngelAxis4[1]=1;
135       actualAngelAxis5[1]=1;
136       actualAngelAxis1[2]=0;
137       actualAngelAxis2[2]=0;
138       actualAngelAxis3[2]=0;
139       actualAngelAxis4[2]=0;
140       actualAngelAxis5[2]=0;
141       actualAngelAxis1[3]=-newPos[0];
142       actualAngelAxis2[3]=-newPos[1];
143       actualAngelAxis3[3]=-(newPos[2]-newPos[1]);
144       actualAngelAxis4[3]=-newPos[3]+newPos[2];
145       actualAngelAxis5[3]=newPos[4];
146       actualAngel=newPos;
147       rotArm1In.setValue(actualAngelAxis1);
148       rotArm2In.setValue(actualAngelAxis2);
149       rotArm3In.setValue(actualAngelAxis3);
150       rotArm4In.setValue(actualAngelAxis4);
151       rotArm5In.setValue(actualAngelAxis5);
152     }
153     /**
154      *set the set VRML-Robot to a new Position
155      *
156      *@param newPos Float-Array of the new Position
157      */
158     public void setVRMLset(float[] newPos){
159       actualAngelAxis1[0]=0;
160       actualAngelAxis2[0]=0;
161       actualAngelAxis3[0]=0;
162       actualAngelAxis4[0]=0;
163       actualAngelAxis5[0]=0;
164       actualAngelAxis1[1]=1;
165       actualAngelAxis2[1]=1;
166       actualAngelAxis3[1]=1;
167       actualAngelAxis4[1]=1;
168       actualAngelAxis5[1]=1;
169       actualAngelAxis1[2]=0;
170       actualAngelAxis2[2]=0;
171       actualAngelAxis3[2]=0;
172       actualAngelAxis4[2]=0;
173       actualAngelAxis5[2]=0;
174       actualAngelAxis1[3]=-newPos[0];
175       actualAngelAxis2[3]=-newPos[1];
176       actualAngelAxis3[3]=-(newPos[2]-newPos[1]);
177       actualAngelAxis4[3]=-newPos[3]+newPos[2];
178       actualAngelAxis5[3]=newPos[4];
179       actualAngel=newPos;
180       rotArm1aIn.setValue(actualAngelAxis1);
181       rotArm2aIn.setValue(actualAngelAxis2);
182       rotArm3aIn.setValue(actualAngelAxis3);
183       rotArm4aIn.setValue(actualAngelAxis4);
184       rotArm5aIn.setValue(actualAngelAxis5);
185     }
186     /**
187      *get the actual position Array
188      *
189      */
190     public float[] getVRML(){
191       return actualAngel;
192     }
193    /**
194      *update the actual position Array
195      *
196      */
```

```
197   public void Fcallback(){
198    //Abrufen der Positions des Soll-Roboters
199    actualAngelAxis1 = callbackSetAxis1.getValue();
200    actualAngelAxis2 = callbackSetAxis2.getValue();
201    actualAngelAxis3 = callbackSetAxis3.getValue();
202    actualAngelAxis4 = callbackSetAxis4.getValue();
203    actualAngelAxis5 = callbackSetAxis5.getValue();
204
205    actualAngel[0] = ((float)-Math.round(actualAngelAxis1[3]*1000))/1000;
206    actualAngel[1] = ((float)Math.round(-actualAngelAxis2[3]*1000))/1000;
207    actualAngel[2] = ((float)Math.round(-(actualAngelAxis3[3]+
208                        actualAngelAxis2[3])*1000))/1000;
209    actualAngel[3] = ((float)-Math.round((actualAngelAxis4[3]+
210                        actualAngelAxis3[3]+actualAngelAxis2[3])*1000))/1000;
211    actualAngel[4] = ((float)Math.round(actualAngelAxis5[3]*1000))/1000;
212   }
213 }
```

E.3.3 robo.jsp

```
1  <html>
2      <head>
3      <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
4      <title>Steuerung technischer Systeme via Internet</title>
5      <jsp:include page="/style.jsp"/>
6  </head>
7
8  <body>
9  <jsp:include page="/Header.jsp"/>
10 <br>
11 <table>
12     <tr>
13     <jsp:include page="/Contents.jsp"/>
14         <td align="left" valign="top" width="100%">
15             <!-- Body -->
16               <p><center>Der Roboter auf brasilianischem Grund stellt den Soll-
                       Wert da, der auf deutschem Grund die Ist-Position</center></p>
17                 <p><embed src="./5dof.wrl" width="1200" height="500"></p>
18               <center><APPLET align="baseline" code="roboApplet.class" archive="
                       ../../classes/" height="225" width="900" mayscript></APPLET></
                       center>
19
20               <!-- Body End -->
21           </td>
22     </tr>
23 </table>
24 <jsp:include page="/Footer.jsp"/>
25 </body>
26 </html>
```

E.3.4 HttpMessage.java

Diese Klasse ist Teil des Packages com.oreilly.servlet

```
1  import java.awt.*;
2  import java.applet.*;
3  import vrml.external.*;
4  import vrml.external.field.*;
5  import java.io.*;
6  import java.net.*;
7  import java.util.*;
8
9   class HttpMessage {
10
11    URL servlet = null;
12    Hashtable headers = null;
13
14    /**
15     * Constructs a new HttpMessage that can be used to communicate with the
16     * servlet at the specified URL.
17     *
18     * @param servlet the server resource (typically a servlet) with which
19     * to communicate
20     */
21    public HttpMessage(URL servlet) {
22      this.servlet = servlet;
23    }
24
25    /**
26     * Performs a GET request to the servlet, with no query string.
27     *
28     * @return an InputStream to read the response
29     * @exception IOException if an I/O error occurs
30     */
31    public InputStream sendGetMessage() throws IOException {
32      return sendGetMessage(null);
33    }
34
35    /**
36     * Performs a GET request to the servlet, building
37     * a query string from the supplied properties list.
38     *
39     * @param args the properties list from which to build a query string
40     * @return an InputStream to read the response
41     * @exception IOException if an I/O error occurs
42     */
43    public InputStream sendGetMessage(Properties args) throws IOException {
44      String argString = "";  // default
45
46      if (args != null) {
47        argString = "?" + toEncodedString(args);
48      }
49      URL url = new URL(servlet.toExternalForm() + argString);
50
51      // Turn off caching
52      URLConnection con = url.openConnection();
53      con.setUseCaches(false);
54
55      // Send headers
56      sendHeaders(con);
57
58      return con.getInputStream();
59    }
60
61    /**
62     * Performs a POST request to the servlet, with no query string.
```

```
63     *
64     * @return an InputStream to read the response
65     * @exception IOException if an I/O error occurs
66     */
67    public InputStream sendPostMessage() throws IOException {
68      return sendPostMessage(null);
69    }
70
71    /**
72     * Performs a POST request to the servlet, building
73     * post data from the supplied properties list.
74     *
75     * @param args the properties list from which to build the post data
76     * @return an InputStream to read the response
77     * @exception IOException if an I/O error occurs
78     */
79    public InputStream sendPostMessage(Properties args) throws IOException {
80      String argString = "";  // default
81      if (args != null) {
82        argString = toEncodedString(args);  // notice no "?"
83      }
84
85      URLConnection con = servlet.openConnection();
86
87      // Prepare for both input and output
88      con.setDoInput(true);
89      con.setDoOutput(true);
90
91      // Turn off caching
92      con.setUseCaches(false);
93
94      // Work around a Netscape bug
95      con.setRequestProperty("Content-Type",
96                             "application/x-www-form-urlencoded");
97
98      // Send headers
99      sendHeaders(con);
100
101     // Write the arguments as post data
102     DataOutputStream out = new DataOutputStream(con.getOutputStream());
103     out.writeBytes(argString);
104     out.flush();
105     out.close();
106
107     return con.getInputStream();
108   }
109
110   /**
111    * Performs a POST request to the servlet, uploading a serialized object.
112    * <p>
113    * The servlet can receive the object in its <tt>doPost()</tt> method
114    * like this:
115    * <pre>
116    *      ObjectInputStream objin =
117    *        new ObjectInputStream(req.getInputStream());
118    *      Object obj = objin.readObject();
119    * </pre>
120    * The type of the uploaded object can be determined through introspection.
121    *
122    * @param obj the serializable object to upload
123    * @return an InputStream to read the response
124    * @exception IOException if an I/O error occurs
125    */
126   public InputStream sendPostMessage(Serializable obj) throws IOException {
127     URLConnection con = servlet.openConnection();
128
```

```
129     // Prepare for both input and output
130     con.setDoInput(true);
131     con.setDoOutput(true);
132
133     // Turn off caching
134     con.setUseCaches(false);
135
136     // Set the content type to be application/x-java-serialized-object
137     con.setRequestProperty("Content-Type",
138                            "application/x-java-serialized-object");
139
140     // Send headers
141     sendHeaders(con);
142
143     // Write the serialized object as post data
144     ObjectOutputStream out = new ObjectOutputStream(con.getOutputStream());
145     out.writeObject(obj);
146     out.flush();
147     out.close();
148
149     return con.getInputStream();
150   }
151
152   /**
153    * Sets a request header with the given name and value.  The header
154    * persists across multiple requests.  The caller is responsible for
155    * ensuring there are no illegal characters in the name and value.
156    *
157    * @param name the header name
158    * @param value the header value
159    */
160   public void setHeader(String name, String value) {
161     if (headers == null) {
162       headers = new Hashtable();
163     }
164     headers.put(name, value);
165   }
166
167   // Send the contents of the headers hashtable to the server
168   private void sendHeaders(URLConnection con) {
169     if (headers != null) {
170       Enumeration enum = headers.keys();
171       while (enum.hasMoreElements()) {
172         String name = (String) enum.nextElement();
173         String value = (String) headers.get(name);
174         con.setRequestProperty(name, value);
175       }
176     }
177   }
178
179   /**
180    * Sets a request cookie with the given name and value.  The cookie
181    * persists across multiple requests.  The caller is responsible for
182    * ensuring there are no illegal characters in the name and value.
183    *
184    * @param name the header name
185    * @param value the header value
186    */
187   public void setCookie(String name, String value) {
188     if (headers == null) {
189       headers = new Hashtable();
190     }
191     String existingCookies = (String) headers.get("Cookie");
192     if (existingCookies == null) {
193       setHeader("Cookie", name + "=" + value);
194     }
```

```
195       else {
196         setHeader("Cookie", existingCookies + ";␣" + name + "=" + value);
197       }
198     }
199
200     /**
201      * Sets the authorization information for the request (using BASIC
202      * authentication via the HTTP Authorization header). The authorization
203      * persists across multiple requests.
204      *
205      * @param name the user name
206      * @param name the user password
207      */
208     public void setAuthorization(String name, String password) {
209       String authorization = Base64Encoder.encode(name + ":" + password);
210       setHeader("Authorization", "Basic␣" + authorization);
211     }
212
213     /*
214      * Converts a properties list to a URL-encoded query string
215      */
216     private String toEncodedString(Properties args) {
217       StringBuffer buf = new StringBuffer();
218       Enumeration names = args.propertyNames();
219       while (names.hasMoreElements()) {
220         String name = (String) names.nextElement();
221         String value = args.getProperty(name);
222         buf.append(URLEncoder.encode(name) + "=" + URLEncoder.encode(value));
223         if (names.hasMoreElements()) buf.append("&");
224       }
225       return buf.toString();
226     }
227 }
```

E.3.5 Base64Encoder.java

Diese Klasse ist Teil des Packages com.oreilly.servlet

```
1  import java.awt.*;
2  import java.applet.*;
3  import vrml.external.*;
4  import vrml.external.field.*;
5  import java.io.*;
6  import java.net.*;
7  import java.util.*;
8
9  class Base64Encoder extends FilterOutputStream {
10
11   private static final char[] chars = {
12     'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
13     'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
14     'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd',
15     'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
16     'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
17     'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7',
18     '8', '9', '+', '/'
19   };
20
21   private int charCount;
22   private int carryOver;
23
24   /**
25    * Constructs a new Base64 encoder that writes output to the given
26    * OutputStream.
27    *
28    * @param out the output stream
29    */
30   public Base64Encoder(OutputStream out) {
31     super(out);
32   }
33
34   /**
35    * Writes the given byte to the output stream in an encoded form.
36    *
37    * @exception IOException if an I/O error occurs
38    */
39   public void write(int b) throws IOException {
40     // Take 24-bits from three octets, translate into four encoded chars
41     // Break lines at 76 chars
42     // If necessary, pad with 0 bits on the right at the end
43     // Use = signs as padding at the end to ensure encodedLength % 4 == 0
44
45     // Remove the sign bit,
46     // thanks to Christian Schweingruber <chrigu@lorraine.ch>
47     if (b < 0) {
48       b += 256;
49     }
50
51     // First byte use first six bits, save last two bits
52     if (charCount % 3 == 0) {
53       int lookup = b >> 2;
54       carryOver = b & 3;        // last two bits
55       out.write(chars[lookup]);
56     }
57     // Second byte use previous two bits and first four new bits,
58     // save last four bits
59     else if (charCount % 3 == 1) {
60       int lookup = ((carryOver << 4) + (b >> 4)) & 63;
61       carryOver = b & 15;       // last four bits
62       out.write(chars[lookup]);
```

```
63      }
64      // Third byte use previous four bits and first two new bits,
65      // then use last six new bits
66      else if (charCount % 3 == 2) {
67        int lookup = ((carryOver << 2) + (b >> 6)) & 63;
68        out.write(chars[lookup]);
69        lookup = b & 63;           // last six bits
70        out.write(chars[lookup]);
71        carryOver = 0;
72      }
73      charCount++;
74
75      // Add newline every 76 output chars (that's 57 input chars)
76      if (charCount % 57 == 0) {
77        out.write('\n');
78      }
79    }
80
81    /**
82     * Writes the given byte array to the output stream in an
83     * encoded form.
84     *
85     * @param b the data to be written
86     * @param off the start offset of the data
87     * @param len the length of the data
88     * @exception IOException if an I/O error occurs
89     */
90    public void write(byte[] b, int off, int len) throws IOException {
91      // This could of course be optimized
92      for (int i = 0; i < len; i++) {
93        write(b[off + i]);
94      }
95    }
96
97    /**
98     * Closes the stream, this MUST be called to ensure proper padding is
99     * written to the end of the output stream.
100    *
101    * @exception IOException if an I/O error occurs
102    */
103   public void close() throws IOException {
104     // Handle leftover bytes
105     if (charCount % 3 == 1) {  // one leftover
106       int lookup = (carryOver << 4) & 63;
107       out.write(chars[lookup]);
108       out.write('=');
109       out.write('=');
110     }
111     else if (charCount % 3 == 2) {  // two leftovers
112       int lookup = (carryOver << 2) & 63;
113       out.write(chars[lookup]);
114       out.write('=');
115     }
116     super.close();
117   }
118
119   /**
120    * Returns the encoded form of the given unencoded string.
121    *
122    * @param unencoded the string to encode
123    * @return the encoded form of the unencoded string
124    */
125   public static String encode(String unencoded) {
126     ByteArrayOutputStream out =
127       new ByteArrayOutputStream((int) (unencoded.length() * 1.37));
128     Base64Encoder encodedOut = new Base64Encoder(out);
```

```
129
130     byte[] bytes = null;
131     try {
132       bytes = unencoded.getBytes("8859_1");
133     }
134     catch (UnsupportedEncodingException ignored) { }
135
136     try {
137       encodedOut.write(bytes);
138       encodedOut.close();
139
140       return out.toString("8859_1");
141     }
142     catch (IOException ignored) { return null; }
143   }
144
145   public static void main(String[] args) throws Exception {
146     if (args.length != 1) {
147       System.err.println(
148         "Usage: java com.oreilly.servlet.Base64Encoder fileToEncode");
149     }
150
151     Base64Encoder encoder = null;
152     BufferedInputStream in = null;
153     try {
154       encoder = new Base64Encoder(System.out);
155       in = new BufferedInputStream(new FileInputStream(args[0]));
156
157       byte[] buf = new byte[4 * 1024];  // 4K buffer
158       int bytesRead;
159       while ((bytesRead = in.read(buf)) != -1) {
160         encoder.write(buf, 0, bytesRead);
161       }
162     }
163     finally {
164       if (in != null) in.close();
165       if (encoder != null) encoder.close();
166     }
167   }
168 }
```

E.4 Excel- Makro

```
Sub variiere() help1    = Range("k6").Value help2    =
Range("k7").Value help3    = Range("k8").Value unten1   =
Range("q5").Value unten2   = Range("q6").Value unten3   =
Range("q7").Value oben1    = Range("r5").Value oben2    =
Range("r6").Value oben3    = Range("r7").Value schritt1 =
Range("s5").Value schritt2 = Range("s6").Value schritt3 =
Range("s7").Value gnr = 46 For winkel1 = unten1 To oben1 Step
schritt1
  Range("k6").Value = winkel1
  For winkel2 = unten2 To oben2 Step schritt2
    Range("k7").Value = winkel2
    For winkel3 = unten3 To oben3 Step schritt3
      Range("k8").Value = winkel3
      Calculate
      If Range("q8").Value Then
          Cells(gnr, 16).Value = winkel1
          Cells(gnr, 17).Value = winkel2
          Cells(gnr, 18).Value = winkel3
          Cells(gnr, 19).Value = "0"
          Cells(gnr, 20).Value = Range("n42").Value
          If Range("n42").Value Then
           Cells(gnr, 19).Value = "1"
          End If
          gnr = gnr + 1
      End If
    Next winkel3
    For winkel3 = oben3 To unten3 Step -schritt3
      Range("k8").Value = winkel3
      Calculate
    Next winkel3
  Next winkel2
  For winkel2 = oben2 To unten2 Step -schritt2
    Range("k7").Value = winkel2
    Calculate
  Next winkel2
Next winkel1 For winkel1 = oben1 To unten1 Step -schritt1
  Range("k6").Value = winkel1
  Calculate
  Next winkel1
Range("k6").Value = help1 Range("k7").Value = help2
Range("k8").Value = help3 End Sub
```

Literaturverzeichnis

[1] Dennis J Bouvier. Getting Started with the Java 3D API, 1999. http://developer.java.sun.com/developer/onlineTraining/java3d/.

[2] Florian Dittmann. Applet Servlet Kommunikation unter Echtzeitaspekten, Juli 2002. http://www.upb.de/cs/ipl/Lehre/Sisi/Ss02/sisi.html.

[3] William Grosso. *Java RMI*, chapter 10 - Serializalization. O'Reilly, October 2001. http://www.oreilly.de/catalog/javarmi/chapter/.

[4] I.N. Bronstein, K.A. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbusch der Mathematik*. Verlag Harri Deutsch, fünfte edition, 2000.

[5] Sun Microsystems Inc. Java 2 platform, standard edition (j2se) product homepage. http://java.sun.com/j2se/.

[6] Sun Microsystems Inc. The Java 3D API. http://java.sun.com/products/java-media/3D/collateral/j3d_api/j3d_wp_SMCC.pdf.

[7] Sun Microsystems Inc. Java Remote Method Invocation Specification. ftp://ftp.java.sun.com/docs/j2se1.4/rmi-spec-1.4.pdf.

[8] Sun Microsystems Inc. Java Servlet Technology. http://java.sun.com/products/servlet/whitepaper.html.

[9] Sun Microsystems Inc. The JavaServer Pages. http://java.sun.com/products/jsp/whitepaper.html.

[10] Jasson Hunter and William Crawford. *Java Servlet Programmierung*. O'Reilly, erste deutsche ausgabe der 2. edition, 2002.

[11] Rafael Cardoso Klein. Rapid Prototyping System - A case-study: Interacting Robots, March 2003.

[12] Sheng Liang. *The Java Native Interface - Programmer's Guide and Specification*. Addison-Wesley, 1999. ftp://ftp.javasoft.com/docs/specs/jni.pdf.

[13] M. C. Zanella, M. Robrecht, A. de Freitas Francisco, A. Horst, T. Lehmann, and R. Gielow. RABBIT A Modular Rapid Prototyping Platform for Distributed Mechatroinc Systems, 2001. http://xmobile.upb.de/publications/_pdf/sbc2001.pdf.

[14] Rikk Carey and Gavin Bell. The Annotated VRML97 Reference. http://www.web3d.org/resources/vrml_ref_manual/Book.html.

[15] S. Ihmor, N. Bastos Jr., R. Cardoso Klein, M. Visarius, and W. Hardt. Rapid Prototyping of Real-Time Communication A Case-Study: Interacting Robots, 2003.

[16] Scott Oaks and Herry Wong. *JINI.* in a nutshell - deutsche Ausgabe. O'Reilly, erste edition, 2001.

[17] Horst Stöcker. *Taschenbuch mathematischer Formeln und moderner Verfahren.* Verlag Harri Deutsch, drite edition, 1995.

[18] Stefan Middendorf and Reiner Sienger. *Java Programmierhandbuch und Referenz für die Java-2-Platform.* dpunkt.verlag, zweite edition, 1999.

[19] Jens Twiefel. Interaktive Animation auf Basis von VRML, Juni 2002. http://www.twiefel.de/SISI/.

[20] Web3D. Extensible 3D (X3D)ISO/IEC FCD 19775:200x. http://www.web3d.org/fs_specifications.htm.

[21] Web3D. VRML97 Functional specification and VRML97 External Authoring Interface (EAI) ISO/IEC 14772-1:1997, ISO/IEC 14772-2:2002. http://www.web3d.org/technicalinfo/specifications/ISO_IEC_14772-All/index.html.