UNIVERSITÄT OSNABRÜCK

Institut für Informatik
AG Wissensbasierte Systeme

Bachelorarbeit

Webinterface zur Steuerung eines Rettungsroboters

David Engelhardt

Oktober 2006

Erstgutachter: Prof. Dr. Joachim Hertzberg
Zweitgutachter: Prof. Dr. Oliver Vornberger

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit dem Webinterface zur Steuerung eines Rettungsroboters. Es werden die Kommunikation zwischen dem Roboter und seinem Basisrechner, technische Aspekte der Hardware und genutzte Programmiertechniken erläutert. Weiterhin wird ein Ausblick auf ähnliche Projekte gegeben. Die Umsetzung der Steuerung, speziell die genutzten Netzwerkstrukturen und Benutzerverwaltungsmechanismen werden detailiert dargestellt. Reale mobile Roboter sind für Internetanwendungen zu wartungsintensiv, daher wird die Simulationssoftware USARSim vorgestellt und ihre Anwendung erläutert.

Abstract

This thesis deals with the web interface for controlling a rescue robot. The communication between the robot and its basic server, technical hardware aspects and applied programming techniques are described. Furthermore an overview over related projects is given. The realisation of controlling the robot, with special focus on the network structures and the mechanisms for request management are explained in detail. Real mobile robots have to be maintained and are therefore unsuitable for a web application. The robot simulation software USARSim is envisaged and its use is explained.

Danksagung

Ich danke Professor Hertzberg für die Möglichkeit, meine Bachelorarbeit in der Arbeitsgruppe *Wissensbasierte Systeme* anfertigen zu können, sowie für interessante Einblicke in die Welt der Robotik. Den Doktoranden Kai Lingemann, Andreas Nüchter und Stefan Stiene danke ich für ihre Anregungen und Tipps zur Arbeit. Meiner Freundin Kathrin danke ich sehr herzlich für das Korrekturlesen. Allen, die mich während der Anfertigungszeit missen mussten, allen voran meinen Eltern, danke ich für ihre Nachsicht.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Roboter: eine technische Vorrichtung, die dazu dient, dem Menschen die Arbeit abzunehmen

Encyclopaedia Galactica

1.1 Ziel der Arbeit

Seit vielen Jahrzehnten sind Roboter Mittelpunkt umfangreicher Forschungsprojekte. Während anfangs vornehmlich versucht wurde, Maschinen zu automatisieren und ihre Präzision zu steigern, konzentrieren sich die Bemühungen zunehmend auf die Entwicklung künstlicher Intelligenz. Ein Roboter soll in der Lage sein, anhand von Sensordaten und internem Wissen selbstständig komplexe Entscheidungen zu treffen. Ein Ziel der Anstrengungen ist, Menschenleben zu schützen, indem an Stelle von Feuerwehrleuten Löschroboter ausrücken oder an Stelle von Sprengmeistern hochpräzise Maschinen eine Bombe entschärfen. Spezialroboter können in lebensfeindlichem Gebiet arbeiten, in der Tiefsee, nach ABC-Unfällen und in Kriegsgebieten. Die Vielzahl an verfügbaren Sensoren erweitern die Einsatzgebiete der Roboter laufend, und die Möglichkeit, parallel verschiedene Prozesse auszuführen, lässt sie immer effizienter werden. Viele der Plattformen werden allerdings zunächst noch ferngesteuert, da die Roboter noch nicht leistungsfähig genug sind, um komplexe Situationen komplett autonom zu meistern. Die Sensoren geben dann dem Operator zusätzliche Informationen zur Umgebung und ermöglichen so eine bessere Einordung der Situation. An der Universität Osnabrück beschäftigt man sich unter anderem mit Rettungsrobotern. Hauptforschungsgebiet sind hier semantische 3D-Karten und autonomes Verhalten. Dazu werden Roboter der KURT2-Serie eingesetzt.

Der KURT2 Roboter ist ursprünglich zur Untersuchung von Kanalrohren gedacht[1]. Die Plattform eignet sich daher auch als Träger für aufwändigere Sensorik, wodurch er als Forschungsobjekt und Rettungsroboter interessant wird. Zu diesem Zweck wurde er mit Webcams, einem Laserscanner und weiteren Sensoren ausgerüstet. Zur Steuerung des Roboters und zur Kommunikation mit einem Basisrechner trägt der Roboter ein widerstandsfähiges Notebook. Im Betrieb

[1] daher KURT: Kanal- Untersuchungs-Roboter-Testplattform

nimmt der Roboter fortwährend Bilder und Laserdaten auf und sendet diese an den Basisrechner, der sie in einer grafischen Oberfläche anzeigt. Über diese Oberfläche oder einen angeschlossenen Joystick lässt sich der Roboter steuern, dazu werden die Befehle per WLan an den Roboter gesendet und dort umgesetzt.

Diese Architektur funktioniert soweit nur im Verbund von zwei Computern und erfordert umfangreiche Software. Ziel der Arbeit ist, die aktuelle Netzwerkstruktur in das Internet zu erweitern. Benutzer sollen den Roboter oder eine entsprechende Simulation von zu Hause aus steuern und die Bilder der Kameras und die Laserdaten betrachten können. Dafür wurden eine Oberfläche zur Ausführung in Internetbrowsern, Klassen für die Befehls- und Datenübertragung, sowie eine Benutzersteuerung geschrieben.

Im weiteren Verlauf der Einleitung werden andere Projekte zur Robotersteuerung über das Internet vorgestellt. Anhand der mobilen Plattformen Rhino und Minerva aus dem Projekt TourBot werden Konzepte zur Interaktion und zur Autonomie aufgezeigt. Außerdem wird eine Übersicht über den TeleGarden und weitere Umsetzungen gegeben.

1.2 Aufbau der Arbeit

Diese Arbeit gliedert sich in folgende 4 Kapitel:

Kapitel 1 ist die vorliegende Einleitung

Kapitel 2 enthält die programmiertechnischen Grundlagen, auf die sich das Projekt stützt: Java-Applets und -Netzwerkverbindungen, das Java Native Interface und die Virtual Reality Modelling Language, sowie ein kurze Vorstellung der KURT-Rettungsroboter Plattform. Es werden die Prozesse auf dem Roboter und dem Basisrechner erläutert, sowie Besonderheiten der grafischen Benutzeroberfläche. Die Bemühungen der Arbeitsgruppe werden in die Forschungs- und Wettbewerbswelt Robocup eingeordnet und es wird die Simulationssoftware USARSim vorgestellt.

Kapitel 3 stellt das Webinterface vor, die Prozesse auf Server- und Clientseite werden erläutert, sowie die Techniken und Abläufe der Benutzersteuerung und der Anbindung an die Simulation.

Kapitel 4 gibt einen Ausblick auf Erweiterungsmöglichkeiten zum aktuellen Projekt.

1.3 Rhino und TourBot

Das Institut der Informatik III der Universität Bonn beschäftigt sich unter anderem mit autonomen Robotern in menschengefüllten Umgebungen. Der Roboter Rhino und seine Nachfolger sind speziell entwickelt worden, um unbekannte Umgebungen zu erkunden, mit Menschen zu interagieren oder über das Internet teleoperiert zu werden.

Rhino

Der Roboter Rhino, Abbildung 1.1, wurde passend zur internationalen Ausstellung zur künstlichen Intelligenz entwickelt. Nach einer sechsmonatigen Programmierphase wurde er 1994 vorgestellt. Er verfügt über 24 Sonarabstandssensoren und zwei Farbvideokameras. Gesteuert wird der Roboter über zwei integrierte i486 Prozessoren und über externe Computer. Hindernis-erkennung und -vermeidung, sowie Kartierung sind die wichtigsten Fähigkeiten des Roboters und werden durch integrierte Lerntechniken stetig verbessert. Zur Kollisionsvermeidung werden die '2-Sekunden-Regel' und geschwindigkeitsabhängige Mindestabstandswerte genutzt. Erstere lässt beim Roboter nur Geschwindigkeiten zu, die innerhalb der nächsten zwei Sekunden zu keinem Zusammenstoß führen, um sicherzustellen, dass die Auswertung der Umgebung der Positionsänderung folgen kann.

Zur Katierung und Positionierung werden die Daten der Kameras, der Sonarsensoren und der Odometrie abgeglichen. Aus den Daten entsteht dann eine Karte. Der Roboter vergleicht neue Sensordaten mit den bereits aufgenommenen Daten der Karte. Anhand von auffälligen Merkmalen der vorliegenden Sensordaten und Merkmalen aus der Karte, bestimmt der Roboter seine Position; diese Technik heisst *Markov-Lokalisierung*. Anhand dieser Karte ist der Roboter auch in der Lage seine Position zu korrigieren, steht er schräg zu einer Wand, erkennt er dieses und kann, falls er eigentlich parallel zur Wand stehen sollte, seine Position entsprechend ändern und die Sensordaten besser interpretieren. Bei einer Fahrzeit von 30 Minuten, mit bis zu 90 Zentimetern pro Sekunde ($\approx$ 3.3 km/h), in unbekanntem Terrain, kann mittels dieses Abgleiches der Fehler in der Odometrie von bis zu 30 Metern auf 30 Zentimeter limitiert werden. Der Fehler in der Odometrie ergibt sich durch Schlupf und verzögertes Ansprechen bei kurzen Impulsen. Die Kartierung stützt sich neben den Sensordaten auch auf Beispieldaten für Türen, Flure und Räume, die ständig gegen die Sensordaten abgeglichen werden. Die Gebiete der Karte können so genauer klassifiziert werden, etwa als Raum mit 2 Türen. Probleme bereiten aber kleine Wandabschnitte oder Räume, die häufig nicht korrekt erkannt werden. Die Informationen der Karte werden ständig gegen die aktuellen Sensordaten geprüft und verifiziert oder korrigiert, so dass die Qualität der Karte mit der Betriebsdauer zunehmend besser wird.

Der Roboter verfügt über Fahr- und Routenplaner, die diese Karte nutzen. Es können dem Roboter beliebige Zielpunkte vorgegeben werden, die er ansteuern soll. Die Bereiche der Karte werden dazu mit Streckenkosten bewertet, sehr hohe, wo Hindernisse erkannt wurden, niedrige, wo der Weg frei ist. Aus den Kosten kann dann der günstigste Weg zum Ziel errechnet werden. Wird ein Zielpunkt in noch unbekanntem Gebiet angegeben, wir der Roboter versuchen, direkt dorthin zu fahren. Dabei gibt die Wegeplanung allerdings nur die grobe Bewegungsrichtung vor, während die Routinen zur Kollisionsvermeidung diese umsetzen und gegebenenfalls abändern. Da der Roboter so auf unbekannte Hindernisse stossen kann, werden die neuen Informationen ständig in die Wegeplanung einbezogen und, falls nötig, wird der Pfad korrigiert.

Die Daten der beiden Kameras des Rhino Roboters werden in vier Stufen ausgewertet. Zunächst werden die Bilder herunterskaliert, um die Funkübertragung nicht zu überfordern. Auf den per Funk verbundenen Computern werden die Bilder nach homogenen Flächen abgesucht, um die Dimensionen von Objekten besser erfassen zu können und um bekannte Komponenten wie Decken oder den Boden schneller zu erkennen. Die Auswertung der segmentierten Bilder bildet die dritte

Stufe der Bildbearbeitung. Als Letztes werden wichtige Objekte anhand von Referenzmustern erkannt. Dabei werden Wahrscheinlichkeiten berechnet und geprüft, ob das erkannte Objekt mit dem Referenzmuster ausreichend gut übereinstimmt. Sind die Wahrscheinlichkeitswerte gut genug, wird das neu erkannte Muster wiederum in die Referenzdatenbank aufgenommen.

Das Projekt Rhino wurde inzwischen beendet, die Nachfolgergenerationen sind aber immernoch im Einsatz und werden zum Beispiel an der Universität Freiburg und der Carnegie Mellon University, Pittsburg PA, weiterentwickelt.

TourBot, Roboter als Museumsführer

Die hervorragenden autonomen Eigenschaften der Roboter und die Möglichkeit, über die eingebauten Rechner lokal mit den Robotern zu interagieren, wurden um Funktionen zur Steuerung per Internet erweitert. Ein Ziel war der Einsatz der Roboter in Museen oder auf Messeveranstaltungen. Dazu wurden die neueren Generationen dieser Roboter mit Multimediaperipherie, TouchScreen, Mikrofonen und Lautsprechern ausgestattet. Sie sollen jetzt nicht die Umgebung erkunden, sondern bekommen detailierte Informationen zu ihrer Umgebung, sowie zu den Exponaten und der Veranstaltung an sich. Benutzer können dann am Roboter lokal weitere Informationen erhalten oder, wenn sie sich über das Internet verbunden haben, den Roboter nutzen, um sich selbst in der Umgebung umzusehen. Sie können Fahrbefehle zu bestimmten Ausstellungsstücken geben oder eine vorgegebene Tour aufrufen, bei der sie der Roboter führt und informiert. Um den Benutzern auch bei langsamerer Internetverbindung eine gute Dynamik bieten zu können, wird in der Weboberfläche, Abbildung 1.1, neben den Kamerabildern auch eine virtuelle Ansicht der Umgebung angezeigt. Sie muss nur initial heruntergeladen werden und simuliert dann die Vorgänge in der Umgebung. Personen werden darin als Avatare angezeigt, was erheblich weniger Informationsfluss erfordert als ganze Bilder.

Um die Roboter sympatischer wirken zu lassen, haben sie Augen und einen Mund bekommen, die sogar je nach 'Gemütslage' verändert werden können, um eine Mimik zu erzeugen. Der Roboter Albert legt in Abbildung 1.1 eine freundliche Miene auf. Haben die Roboter einen Fahrauftrag durch einen Benutzer bekommen, verändert sich ihre Stimmung in 'froh' und sie machen sich auf den Weg. Ist dieser durch Besucher versperrt, wird ihre Laune zunehmend schlechter, bis ein verärgertes Gesicht den Umstehenden klarmachen soll, dass der Roboter vorbei will. Die Robotersoftware enthält auch Komponenten zur Spracherkennung, es können einfache Anfragen an die Roboter gestellt werden, die diese aus ihrer Datenbank zu beantworten versuchen. Die Ausgabe erfolgt dabei über ein Sprachmodul.

Museen stellen Roboter vor eine besondere Herausforderung: Ihre Aufteilung und die Besucher erschweren die Navigation und Positionsbestimmung. Um sich selbst zu lokalisieren benötigen Roboter Hindernisse, um sich zu diesen in Relation zu setzen. Sind diese weit weg oder befinden sich Besucher zwischen dem Roboter und dem Hindernis, schlägt die Positionsbestimmung häufig fehl. Um diese Probleme zu umgehen, wurden die TourBots mit einem zusätzlichen Sensor und bestimmten Richtlinien für die Wegfindung ausgestattet. Eine Videokamera zeigt senkrecht nach oben, so kann die Struktur der Decke zur Lokalisierung herangezogen werden. 'Costal Navigation' soll verhindern, dass der Roboter auf freien Flächen die Orientierung verliert. Diese

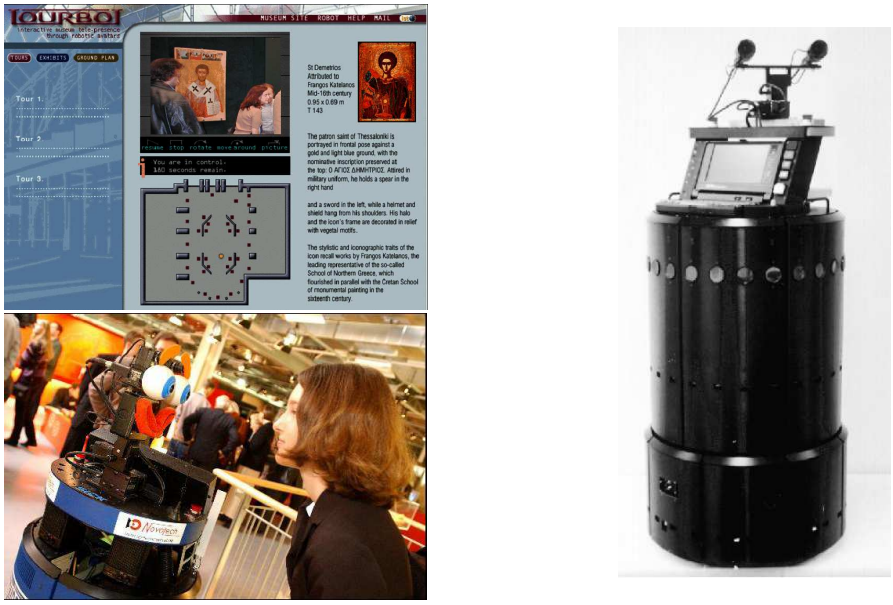

Abbildung 1.1: Oben links: Das Webinterface der TourbotRoboter mit den angebotenen Touren, dem aktuellen Kamerabild, einer Übersichtskarte und näheren Informationen zum ausgewählten Objekt. Unten links: Der Roboter Albert im Forum des Heinz Nixdorf Museums. Rechts: Der Rhino Roboter der Universität Bonn. [1]

Vorgehensweise stammt aus der Seefahrt und bezeichnet das Fahren nahe dem Ufer, wie es kleinere Schiffe machen, damit sie nicht vom Kurs abkommen. Die Roboter meiden also freie Flächen und fahren eher am Rand entlang, in der Nähe von Hindernissen, an denen sie sich orientieren können.

Die Konfigurationssoftware für die Roboter ist inzwischen soweit entwickelt worden, dass die Roboter innerhalb von wenigen Tagen komplett auf eine Veranstaltung eingerichtet werden können. Bei den Vorgängergenerationen, Rhino und Minerva, dauerte es noch 180 bzw. 30 Tage. Da meistens digitale Daten zu den Veranstaltungen vorliegen, sind Programme geschrieben worden, um diese leichter in die Datenbanken der Roboter einspielen zu können. Die Roboter Minerva und Albert sind beides Weiterentwicklungen des Rhino-Roboters. Die unterschiedlichen Namen ergeben sich durch die Abwanderung der Arbeitsgruppen an die Universität Freiburg (Albert) und die Carnegie Mellon University, Pittsburg PA, (Minerva). Mehr Informationen zu den Robotern der Universität Bonn finden sich bei [1].

1.4 Der Telegarden

Der Telegarden ist ein Kunstprojekt der University of Southern California. Die Initiatoren wollten einen Kontrast schaffen zwischen der immer schnelllebigeren Medienwelt und der gleichsam ruhigen Natur. Dazu gründeten sie eine Community, die mittels eines über das Internet gesteu-

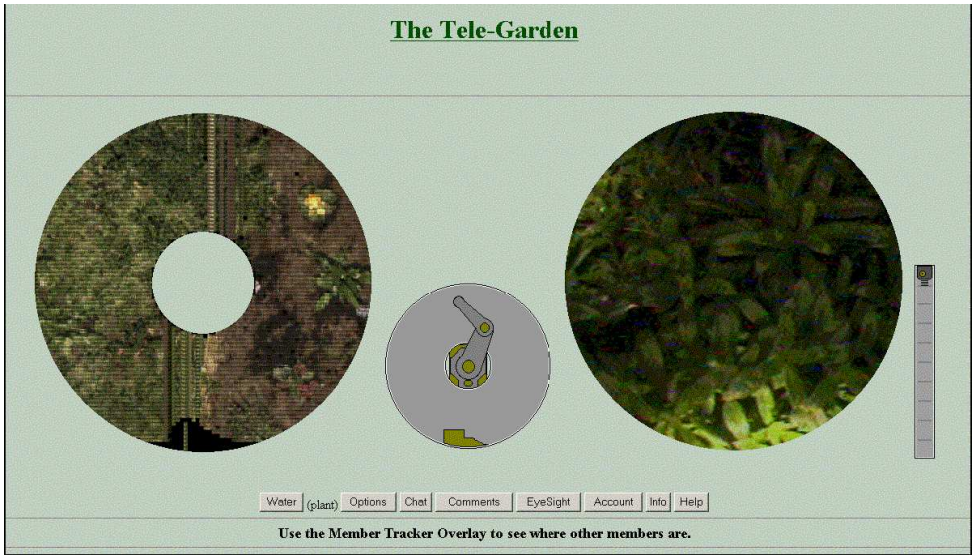

Abbildung 1.2: Grafische Benutzeroberfläche des Telegarden

erten Industrieroboters, Abbildung 1.3, eine etwa 4 m^2 große Pflanzfläche kultivierte.

Im Juni 1995 nahm die Anlage in Berkley, Kalifornien, den Dienst auf. Bereits im ersten Jahr wuchs die Community auf 9000 Mitglieder, die die Pflanzen bewässerten, beobachteten und neu anpflanzten. Der Industriedreharmroboter wurde dafür mit Kameras und spezieller Peripherie ausgestattet und erlaubte so den Benutzern über das Internet mit den Pflanzen zu interagieren. Nach 7 Jahren in Kalifornien wurde die Anlage im September 2002 nach Österreich umgesiedelt, wo sie im Ars Electronica Museum weitere 2 Jahre in Betrieb war. 2004 ging der Telegarden bisher endgültig vom Netz. Neben der technischen Umsetzung ging es den Initiatoren auch um eher philosophische Ansätze. Es sollte gezeigt werden, dass auch der schnellste Rechner und die beste Internetverbindung das Wachstum der Saatlinge nicht beschleunigen können. Außerdem sollte ein kooperativer Geist entstehen, der die postnomadische Überzeugung bekräftigen soll, dass das Überleben die begünstigt, die zusammenarbeiten. Dieser philosophische Ansatz ist auch auf die Benutzer übergesprungen: Es wurden Urlaubsvertretungen arrangiert, damit die Pflanze nicht vertrocknet, und sogar hohe Geldbeträge geboten, um die Pflanzen zu erwerben.

Der Telegarden arbeitet parallel für mehrere Benutzer. In der grafischen Oberfläche, Abbildung 1.2, können sich die Benutzer anzeigen lassen, welche Bereiche andere Benutzer gerade betrachten. Der Roboter arbeitet alle Anfragen nacheinander und benutzerparallel ab, wodurch die Benutzer keinen Videostream, sondern eher eine Diashow betrachten. Die linke kreisförmige Anzeige zeigt eine Draufsicht auf die Pflanzfläche, hier kann der Roboter positioniert werden, und es werden auf Wunsch die Betrachtungspunkte anderer Benutzer angezeigt. Die rechte Fläche zeigt das für diesen Benutzer aktuelle Kamerabild an. Über die Schaltflächen am unteren Ende der Anzeige können Pflanzen gegossen werden, es können Kommentare und Chatnachrichten versandt werden, und auch der Wechsel in eine aktuelle Draufsicht ist möglich. Ist ein Benutzer lange genug dabei, steht ihm auch der Button 'plant' zur Verfügung, und er kann einen neuen

Abbildung 1.3: Der Telegarden Roboter über der runden Pflanzfläche

Samen in die Pflanzfläche einbringen.

1.5 Weitere Projekte

In diesem Abschnitt werden noch ein paar jüngere Projekte vorstellt, da diese, in Teilen, dieselben Techniken benutzen, die auch in dieser Arbeit verwendet wurden. Leider ist Robotersteuerung, speziell von mobilen Plattformen, eine sehr zeitkritische Aufgabe. Die Netzwerkverbindungen im Internet werden zwar immer breitbandiger und schneller, erreichen aber nur weiche Echtzeit.

1.5.1 Steuerung eines Fünf-Achsen-Knickarmroboters

An der Universität Paderborn wude die Internetsteuerung eines Fünf-Achsen-Knickarmroboters programmiert. Die Benutzer sollten die einzelnen Achsen des Roboters bewegen und Punkte im Arbeitsraum anfahren. Zur Steuerung wurden zwei Oberflächen geschrieben, ein Java Applet mit Slidern zur konkreten Eingabe von einzelnen Werten für jede Achse und eine dreidimensionale VRML-Oberfläche. Diese ist eine detailreiche Umsetzung des CAD-Modells des Roboters in VRML. Die Oberfläche, Abbildung 1.4, stellt den Roboter zweimal dar, in der aktuellen Ausgangspose und als Modell für die neue Situation. Das zweite Modell ist dazu mit Sensorknoten, s.u., bestückt, die die neuen Soll-Koordinaten aus der 3D Welt auslesen und über das Applet an den Roboter weitersenden. VRML ist eigentlich eine reine Beschreibungssprache für virtuelle Welten. Sie kann zwar interne Veränderungen des Modells berechnen und animieren, stellt aber keine Verbindungen nach außen her. Um dennoch die Daten für die Roboterpositionen aus dem Modell heraus oder in es hinein senden zu können, wurde das *External Authoring Interface*

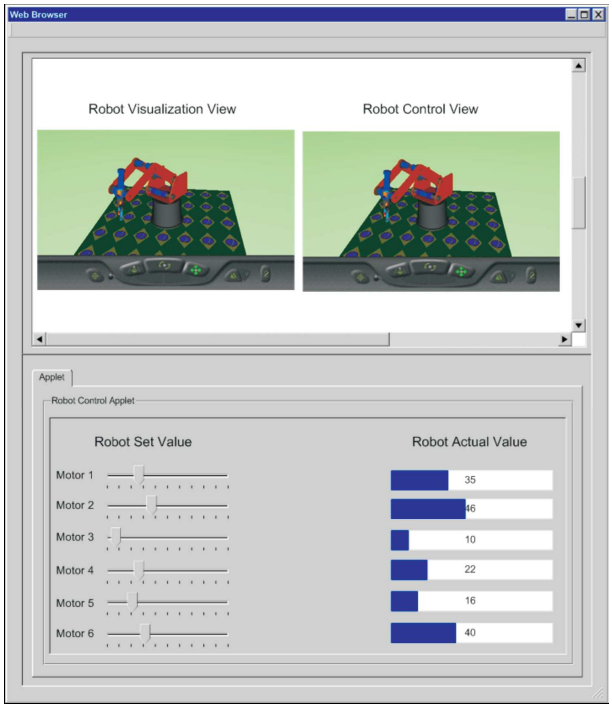

Abbildung 1.4: Internetoberfläche zur Steuerung eines Fünf-Achsen-Knickarmroboters. [15]

(EAI) als Schnittstelle herangezogen. Da viele Implementationen des *EAI* in Java geschrieben werden, ließ es sich auch einfach an das Applet anbinden.

Als Internetanwendung muss das System auch eine Benutzersteuerung implementieren, da meistens mehrere User Zugriff auf die anbietende Webseite haben. Das Applet selbst kann in beliebig vielen Instanzen ausgeführt werden, aber der Roboter ist als einmalige Ressource exklusiv zu behandeln. Da der Roboter ortsfest aufgestellt ist, kann die Benutzerverwaltung allerdings weniger streng ausgelegt sein. Bei einer mobilen Plattform ist es sinnvoll, den Benutzern eine gewisse Mindestzeit anzubieten, damit sie die Umgebung erkunden können. Bei einem stationären Roboter, der nur verschiedene Posen einnimmt, reicht es, dem aktuellen Benutzer eine kurze Zeit exklusiven Zugriffs zu gewähren, um Abfolgen von Posen durchzuführen. Verstreicht diese Bedenkzeit, werden die Soll-Werte aller Benutzer ausgewertet. Bei dieser Art der Benutzersteuerung muss der Server häufig entscheiden, ob die eingehenden Daten vom aktuellen Benutzer oder einem anderen Anwärter kommen, eine Sitzungsverfolgung soll hier bei der Auswertung helfen. Das System birgt einen potenziellen Überschuss an Datenverkehr, da jeder Benutzer, ob aktiv oder nicht, fortwährend Daten senden kann, die der Server verarbeiten muss. Die Steuerungssoftware für den Roboter ist in C++, die Internetanwendungen, das Applet und der AppletServer, in Java geschrieben. Die Schnittstelle bildet auch in diesem Projekt das Java Native Interface; dieses wird im Abschnitt 'Das Java Native Interface' beschrieben.

Zu den Anforderungen an Benutzeroberfläche und Steuerung kommen bei diesem Projekt noch Sicherheitsaspekte. Der Roboter steht in einer Laborumgebung, seine Bewegungsfreiheit wird da-

Abbildung 1.5: Roboka, der Roboter im Kaufhof. [12]

her durch zusätzliche Objekte und auch seine eigenen Komponenten eingeschränkt. Um Schäden zu verhindern, werden alle eingehenden Soll-Werte von einer Kollisionskontrolle geprüft und verworfen oder, wenn sie gültige, sichere Positionen beschreiben, weitergegeben. Weitere Informationen zu diesem Projekt findet man unter [15].

1.5.2 Roboka, der Roboter im Kaufhof

An der Universität Rostock wurde Ende 2002 eine mobile Plattform erstellt, mit der Internetbenutzer die Möglichkeit haben sollten, nach Ladenschluss die Sportabteilung des örtlichen Kaufhofs zu erkunden. Dieser Roboter, Roboka, Abbildung 1.5, ist ebenfalls vom Typ *B2*, wie auch Rhino und Minerva, allerdings ist er mit weniger Sensorik ausgestattet. Er trägt 16 Abstandssensoren zur Kollisionsvermeidung, sowie eine Webcam zur Aufnahme von Umgebungsbildern. Der Roboter wird über ein WLan-Netz gesteuert, welches über Funknetzwerkbrücken mit dem nahe gelegenen Universitätsnetzwerk verbunden ist. Um Benutzern mit einer schlechteren Internetverbindung die Möglichkeit zu geben, den Roboter zu steuern, werden die Kamerabilder auf etwa ein Viertel der Ursprungsgröße heruntergerechnet. Die Universität Rostock nutzte zur Einsatzzeit des Roboters nur ein Funknetzwerk des IEEE 802.11b-Standards, daher wurden Programme eingesetzt, die wenig Bandbreite benötigten. Zur Steuerung des Roboters wurden Remote Method Invocation, RMI, Techniken verwendet. Diese Technik erstellt auf dem RMI-Server die benötigten Java-Objekte, die dann per Referenz auf dem Benutzerrechner genutzt werden können. Die Software dort kennt dann nur eine Signatur des Objektes und kann dessen

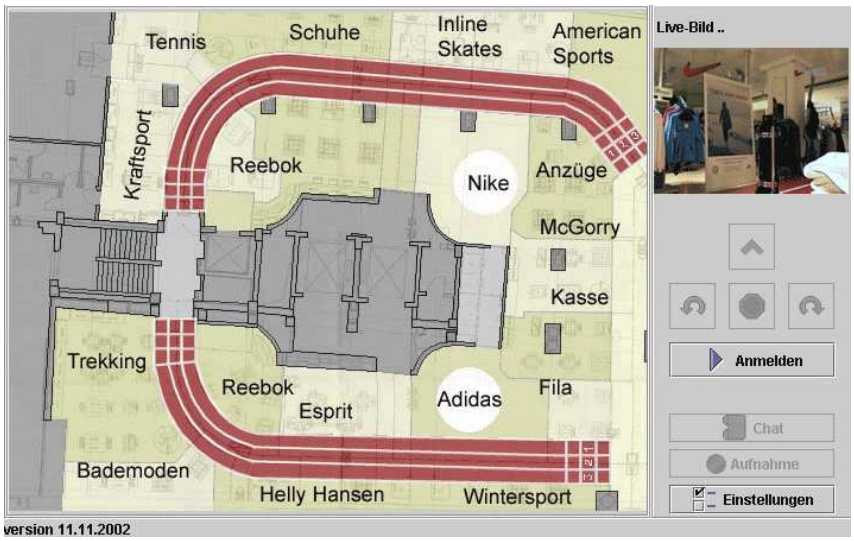

Abbildung 1.6: Grafische Oberfläche zur Steuerung des Roboka. [12]

Methoden fernaufrufen. Diese Technik lehnt sich an die Interfaces in Java an. Für jeden Benutzer wird ein eigener Thread angelegt, alle Threads greifen auf den selben Satz von Objekten zu. Kritische Bereiche müssen daher synchronisiert werden. Da der eingebaute Rechner des Roboters nicht besonders leistungsstark war, wurden eine spezielle Implementation, das TinyRMI, und die Kertasarie VM zum Einsatz auf ressourcenarmen Systemen genutzt. Die Benutzerseite wurde wieder über ein Applet, Abbildung 1.6, realisiert.

Die grafische Oberfläche zeigt auf der linken Seite eine Zeichnung der Sportabteilung. Rot markierte Flächen sind dem Roboter dabei zugänglich, graue sind Nebenräume und anderweitig eingefärbte Bereiche kann der Roboter meist nicht befahren. Ein gelber Punkt auf der Karte soll die Position des Roboters anzeigen. Die rechte Seite beinhaltet das Bild der Webcam auf dem Roboter und die Buttons zur Kontrolle. Ein Benutzer muss sich zunächst anmelden, um in die Warteschlange zur Robotersteuerung aufgenommen zu werden. Ihm wird mitgeteilt, wie lange es maximal bis zur Zuweisung der Kontrolle dauern wird. Da die Benutzungsdauer auf 10 Minuten festgelegt ist, wird vor Zuweisung der Kontrollen getestet, ob wirklich ein Benutzer anwesend ist, um den Roboter zu steuern. Dazu wird ein Dialog geöffnet, der auffordert, binnen 10 Sekunden einen Button zu drücken. Geschieht dieses nicht, wird der Benutzer aus der Warteschlange gestrichen und die Benutzersteuerung versucht, dem nächsten Wartenden die Kontrolle zuzuteilen. Die grafische Oberfläche hat zwei Zusatzfunktionen: Man kann durch Klicken auf das Kamerabild einen Dialog öffnen, der größere Bilder anzeigt und zum Speichern zur Verfügung stellt und, sofern der Benutzer das Java Media API installiert hat, können die Bilder als QuickTime-Filmsequenz aufgezeichnet werden. Ein simpler Textchat zwischen den Benutzern kann ebenfalls per Buttonklick gestartet werden. Das Projekt ist in [12] dokumentiert.

Kapitel 2

Grundlagen

2.1 Der KURT-Rettungsroboter

Die Roboter der KURT-Serie sind mobile Plattformen die meistens in der Forschung und Lehre eingesetzt werden. In ihrer Grundausstattung sind sie nur mit vier Infrarotabstandssensoren ausgestattet, auf Kundenwunsch können aber weitere Sensoren in das Chassis eingebaut werden. Zur Auswahl stehen hier Ultraschallsensoren, Neigungssensorik und Gyroskope zur Messung von Beschleunigung in zwei Richtungen, so können Vibrationen und Winkel gemessen werden. Ein optional erhältliches EvaluationBoard ermöglicht eine einfache Auswertung der Messdaten am Computer. Zur Orientierung, speziell in Gebieten mit wenigen Bezugspunkten (große Hallen, lange Kanalrohre) kann ein Kompassmodul eingebaut werden. Einige der Sensoren benötigen eine zusätzliche Mikrocontrollerkarte zur Ansteuerung. Angetrieben wird der Roboter über zwei 90W Elektromotoren, einen für jede Seite, mit entsprechender Antriebsmechanik. Diese erhalten ihren Strom aus Nickelmetallhydrid-Akkumulatoren mit 24V Betriebsspannung und einer Leistung von 4500 mAh[1].

Es gibt zwei Varianten des KURT2 Roboters: Die erste ist für den Indoorbetrieb gedacht und hat kleinere Räder, die zweite, Abbildung 2.2, ist mit großen, luftgefüllten Profilreifen ausgestattet und hat somit eine höhere Bodenfreiheit. Abbildung 2.1 zeigt schematisch die Aufteilung im Chassis des Roboters. Um die Plattform für Rettungsmissionen oder Erkundungs- und Kartierungsfahrten nutzen zu können, musste die Sensorik aufgestockt und um Equipment zum Datentransfer erweitert werden. Ein installierter Laserscanner soll die Ebene vor dem Roboter, den Kegel vor dem Roboter oder die gesamte Umgebung vermessen, je nach Aufhängung. Dazu nimmt er 181 Abstandswerte in einem Öffnungswinkel von 180 Grad mit einer Schrittweite von einem Grad auf. Je nach Aufhängung wird dieser Vorgang für verschiedene Scannerstellungen wiederholt. Mit der Aufhängung der Roboter an der Universität Osnabrück stehen vertikal die Auflösungen von 361 und 721 Schritten zur Verfügung. Die Abstandswerte geben jeweils die Entfernung an, in der der Laserstrahl reflektiert wird, wo sich also das nächste, undurchsichtige Hindernis befindet.

[1] Dieses sind die Werte der Roboter an der Universität Osnabrück, als Variante können auch NiCd Akkus mit 21.6V und 5000mAh eingesetzt werden.

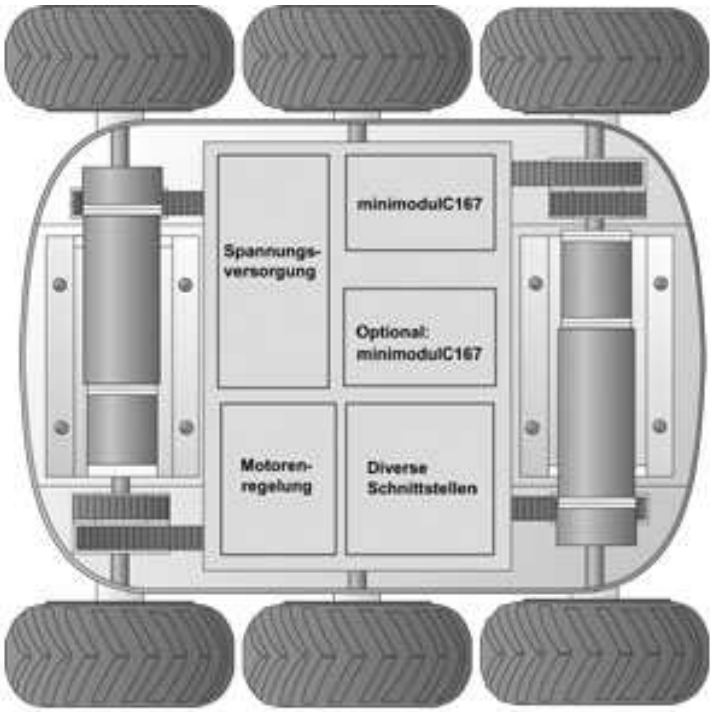

Abbildung 2.1: Schematische Ansicht des KURT2 Chassis. [16]

Abbildung 2.2: Der KURT-Rettungsroboter, Outdoor-Version.

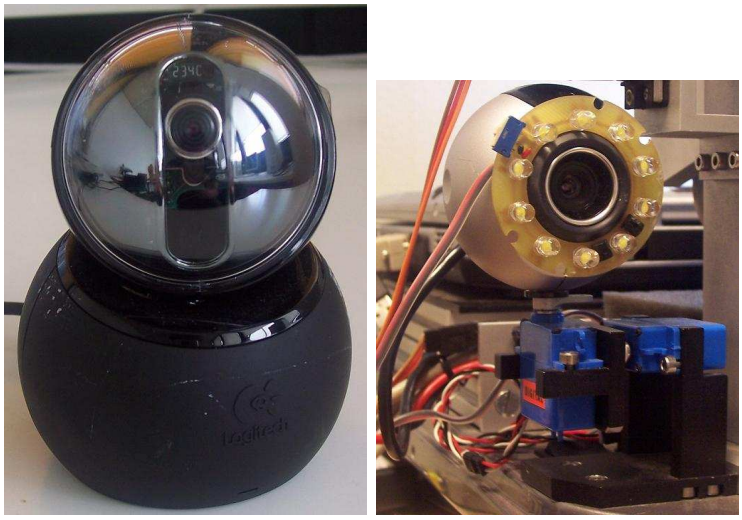

Abbildung 2.3: Links: Logitech Webcam mit integrierten Servos. Rechts: Logitech Webcam mit externen Digital-Servos und LED Beleuchtungskranz

Zur Bildübertragung dienen verschiedene Kameratypen: Webcams zur normalen Bildübertragung und eine Infrarotkamera zur Erfassung von Wärmebildern. Die Webcams, Abbildung 2.3, sind durch eingebaute Motoren oder durch eine externe Servoaufhängung drehbar gelagert. Bei der Auswahl der Kameras wurde auf empfindliche CMOS-Chips geachtet, damit auch im Dunkeln oder bei schlechter Beleuchtung auswertbare Bilder aufgenommen werden.

Um bei schlechtem Licht die Umgebung aufzuhellen ist einer der Rettungsroboter der Universität Osnabrück mit spezieller Lichttechnik ausgestattet. Eine Lichtleiste mit extrahellen LEDs vorne, sowie zwei Kränze um die Kameraobjektive ermöglichen Fahrten in absoluter Finsternis. Abbildung 2.4 zeigt den Roboter mit voller Fahrbeleuchtung. Um später auch verschüttete Opfer finden zu können oder bei Wettkämpfen die Puppen mit Gasflasche aufzuspüren, kann der Roboter mit einem Kohlenstoffdioxidsensor ausgestattet werden. Dieser hat allerdings klare Einschränkungen, da es eine gewisse Ansprechzeit braucht, bis dieser veränderte Werte zurückgibt. Umgebungsgeräusche können dem Operator Informationen über Opfer und Gefahren geben. Sie werden entweder über ein eigenständiges Funkmikrofon übertragen, oder mit dem Laptop und einem angeschlossenen (Richt-) Mikrofon aufgenommen und per WLan versandt. Die erste Methode ist erheblich zuverlässiger und qualitativ besser, da die Kontrollschleifen des Roboters nicht belastet werden und das Funknetz für andere Daten frei bleibt, die zweite ist kostengünstiger und später auch für das autonome Verhalten des Roboters nutzbar.

Der Rettungsroboter ist eine modifizierte Version des KURT2 mit speziellen Motoren. Sie haben mehr Drehmoment, damit der Roboter stärkere Steigungen überwinden kann. Diese Variante fährt maximal einen Meter pro Sekunde. Es gibt auch eine HighSpeed-Variante, hauptsächlich für Fahrten auf ebenen Flächen. Da diese keine größeren Steigungen bewältigen muss, sind die Motoren auf eine höhere Endgeschwindigkeit ausgelegt. Sie können den Roboter auf maximal 5.0 m/s beschleunigen, allerdings sind nur 4.0 m/s algorithmisch kontrollierbar, das heißt bis zu

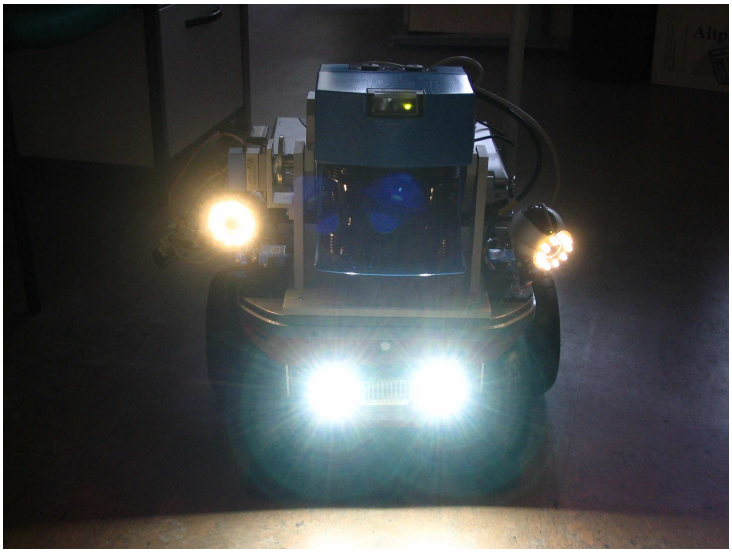

Abbildung 2.4: Kurt3D Rettungsroboter mit voller Beleuchtung.

dieser Geschwindigkeit werden Hindernisse erkannt und umfahren.

2.2 RoboCup

Robotik, die Entwicklung von künstlicher Intelligenz und Konzepte zum Informationsaustausch zwischen Mensch und Maschine sind seit vielen Jahren Gegenstand der Forschung. Da die Forschungsgruppen ihre Anstrengungen zu unterschiedlichen Aufgaben und Themengebieten vorantreiben, ist der Informationsfluss zwischen den Gruppen oft eingeschränkt und 'das Rad wird häufig neu erfunden'. Um eine Plattform für den Wissensaustausch zu schaffen, Fortschritte besser beobachten zu können und einen Vergleich verschiedener Projekte herstellen zu können, wurde die RoboCup-Initiative ins Leben gerufen. Seit etwa 1993 treten bei den Wettkämpfen dieser Organisation internationale Teams mit ihren Robotern zum Fussballspielen an. Die Wahl des Themengebietes fiel auf Fussball, da hier alle in Zukunft interessanten Techniken der Informatik zusammentreffen. Bilderkennung zur Einstufung des Umfeldes, Entscheidungstreffung anhand von Sensordaten und Informationen von aussen, Multiagentenverwaltung zum Teamspiel, etc.

Im Jahr 2000 nahm die RoboCup-Vereinigung ein weiteres Feld in ihre Bemühungen auf, den RoboCupRescue. In dieser Initiative treten Rettungsroboter gegeneinander an und es werden Konferenzen zur Robotik in Gefahrensituationen abgehalten. Ziel ist, Roboter zu entwickeln, die ferngesteuert oder autonom in der Lage sind, Unfall- oder Katastrophengebiete zu erkunden, zu kartieren und so den Rettungsmannschaften eine schnelle Übersicht über die Einsatzlage zu geben. Einfache mechanische Funktionen zur Schadensbegrenzung sollen ebenfalls umgesetzt werden. Neben den physikalischen Robotern gibt es auch eine Simulationsliga. Verschiedene Teams lassen die Agenten ihrer Roboter gegeneinander antreten. So können die künstlichen Intelligenzen verglichen werden. Simulationen haben den Vorteil, dass Arbeitsgruppen auf teure

und fehleranfällige Hardware verzichten können, und dennoch ihre Algorithmen und Techniken in realistischen Umgebungen testen können. Im folgenden Kapitel *Simulation mit Unreal Tournament* wird die Software *USARSim* vorgestellt.

Die Arbeitsgruppe *Wissensbasierte Systeme* der Universität Osnabrück tritt bei den Wettkämpfen des RoboCup Rescue als Teil des Teams *Deutschland 1* an, zusammen mit der Universität Bonn und unterstützt von der Universität Hannover. Bei den realen Robotern gibt es zwei Wettbewerbe: In der ersten Kategorie werden die Roboter teleoperiert. Ein oder mehrere Operatoren steuern den Roboter aus einem vom Spielfeld separierten Raum, nur anhand der Sensordaten. Ziel ist es, die Opfer im Einsatzgebiet zu finden und ihre Position zu markieren. Dazu werden Karten der Umgebung erstellt und entsprechende Zeichen eingefügt. Eine Kollision mit Opfern oder Hindernissen führt dabei zu Punktabzug. In dieser Kategorie sind die KURT-Roboter inzwischen klar im Nachteil, da ihre Bau- und Antriebsform kein Treppensteigen oder das Befahren von höheren *StepFields* erlaubt. In dieser Kategorie schneiden häufig die Roboter von Mechatronik-Instituten gut ab, die von bis zu sechs Ketten angetrieben werden. Der zweite Wettbewerb fordert autonomes Verhalten der Roboter. Sie werden in eine weniger komplizierte Arena geschickt und sollen dort selbstständig erkunden und Opfer erkennen. In dieser Disziplin schneiden die Informatik-Institute meistens besser ab, da hier Algorithmen gefordert sind, die Hardware allerdings nur eine untergeordnete Rolle spielt. Die Abbildungen in 2.5 zeigen aktuelle Rettungsroboter.

Der RoboCup fördert und fordert zunehmend auch den Nachwuchs der Robotik. Sowohl bei den Fussball- als auch bei den Rettungsrobotern gibt es Wettkämpfe für Schüler und Jugendliche. Hier wird häufig auf die Robotikkomponenten der *Mindstormer* des Spielwarenherstellers *Lego* zurückgegriffen. Einfache Sensoren und Motoren werden über einen Mikrokontroller gesteuert, der sich mit der beiliegenden Software einfach am Computer programmieren lässt. Den gesamten Umfang der RoboCup-Initiative erfährt man auf [7] und [8].

2.3 Simulation mit Unreal Tournament

Das Webinterface ist zur Steuerung eines Rettungsroboters gedacht, soll aber als Webapplikation auch möglichst wartungsarm laufen. Ein realer Roboter muss ständig kontrolliert werden, Batterien müssen geladen werden und es muss ein geeignetes Areal zur Verfügung stehen. Zudem ist nicht auszuschließen, dass weniger freundlich gesinnte Benutzer ihre Fahrzeit bei einem realen Roboter nutzen würden, um zu testen, wie oft man diesen vor die Wand fahren lassen kann, bis er Schaden nimmt. Eine Lösung für diese Probleme wäre eine Simulation des Roboters.

USARSim ist eine Simulationssoftware für Roboter. Der Ansatz ist hier, Menschenleben zu schützen, indem Roboter entwickelt werden, die in Gefahren- und Kriegsgebieten eingesetzt werden. Wirkliche Katastrophen sind aber zum Glück nur sehr selten, erfordern dann aber die gesamte Aufmerksamkeit der Rettungskräfte und bieten kaum Möglichkeit, Roboter einzusetzen und zu testen, wenn ihre Arbeit gefahrlos und schneller von menschlichen Rettungskräften erledigt werden kann. Um dennoch auf den Gebieten von Rettungstechnik, Multiroboterkontrolle und dem Zusammenwirken von Robotern und Hilfskräften forschen und testen zu können, wurde

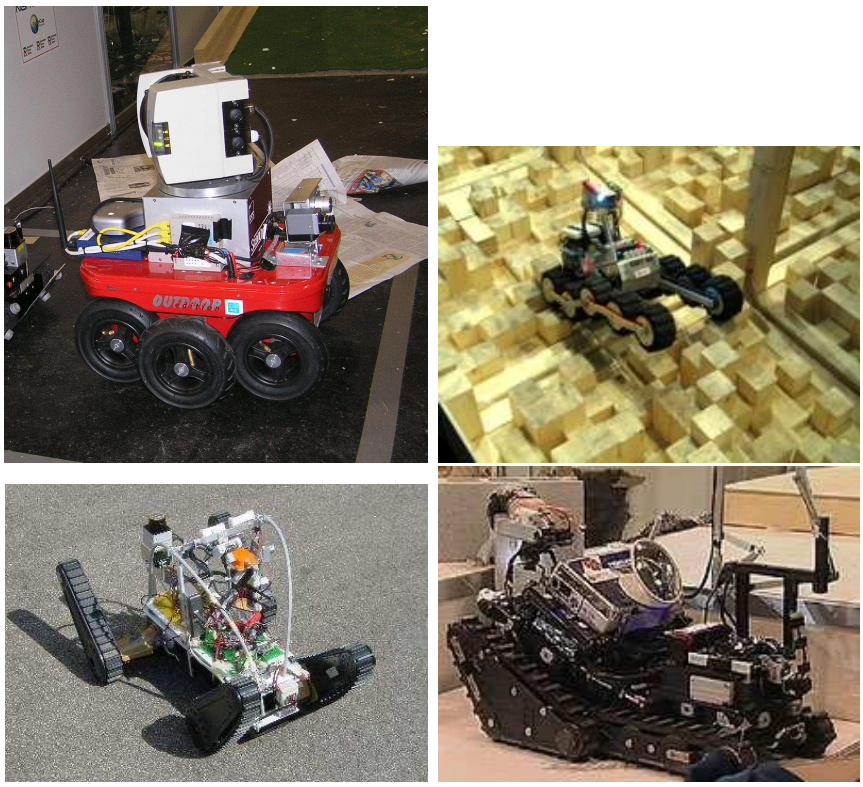

Abbildung 2.5: Oben links: Der Kurt3D des Team Deutschland1, Universität Bonn und Universität Osnabrück, mit einem vertikal drehbar gelagerten Laserscanner der Universität Hannover, der Abstandswerte der gesamten Umgebung liefert. Oben rechts: Einer der Rettungsroboter des Teams Independent aus Thailand, das Team gewann den Gesamtpreis beim RoboCup Rescue 2006. Unten links: Der Lurker-Rettungsroboter der Universität Freiburg, Gewinner in der Autonomiewertung. Unten rechts: Der Roboter des Teams Shinobi aus Japan wurde für die beste Fortbewegungsfähigkeit ausgezeichnet.

eine Simulation für 'städtisches Suchen und Retten' geschrieben.[2] Um den Fokus auf die Entwicklung von Sensoren, Aktuatoren und Schnittstellen zwischen Testsoftware und Simulation richten zu können, wurde als Plattform ein verbreitetes, leistungsfähiges und erweiterbares Computerspiel ausgewählt. *Unreal Tournament* bietet als kommerzielle Software eine solide Grundlage und kann mit Hilfe von Editoren und Zusatzsoftware, die zum Teil durch den Hersteller *Epic* selbst angeboten werden, erweitert und angepasst werden. Die exzellente Grafik und die eingesetzte *Karma Physics Engine* ermöglichen detailierte Simulationswelten und realistisches Objektverhalten. Die physikalischen Berechnungen umfassen derzeit solide Objekte, können also Räder, Chassis, Gelenke und Federn modellieren. Da das Spiel für die große Menge der Computerspieler entwickelt wurde, sind einzelne Lizenzen mit etwa 10 Euro (Unreal Tournament 2004, Preis bei Markteinführung etwa 50 Euro) sehr günstig.

Bei der Erstellung der Simulationssoftware, USARSim, orientierten sich die Programmierer an

[2]Daher USARSim: Urban Search And Rescue Simulation

den Richtlinien des *National Institute of Standards*, *NIST*. Dort gibt es Vorgaben zu den Testumgebungen für reale USAR-Roboter, die auch für die Simulation übernommen wurden.

Die Karten der NIST-konformen Arenen, Modelle für kommerzielle und experimentale Roboter sowie Modelle für Gegenstände und Personen sind im Umfang der Simulationssoftware enthalten. So können Arbeitsgruppen schnell mit den Forschungen beginnen, ohne vorher einen eigenen Roboter erschaffen zu müssen. Sie sind dann gefordert, künstliche Intelligenzen und Fahrtensteuerungsalgorithmen bereitzustellen, um den Roboter in der Simulation zu steuern. Unreal Tournament benutzt ein eigenes, weniger standardisiertes Kommunikationsprotokoll. Die Verbindung mit anderen Applikationen ist daher nicht problemlos aufzubauen. Als Schnittstelle dient hier die Software *Gamebots*, die einen Netzwerkdienst zwischen der Unreal Tournament Engine und externen Programmen erstellt. Über diese Verbindung können dann Befehle und Statusdaten ausgetauscht werden.

Neben den mitgelieferten Robotermodellen können eigene Roboter entworfen und zusammengesetzt werden. Dazu werden die einzelnen Komponenten initialisiert und mit virtuellen Verbindungspunkten aneinandergefügt. Diese Verbindungspunkte bilden auch den Bezug für Modifikationen wie Kipp- und Drehbewegungen. Der gemeinsame Bezugspunkt der Einzelteile ist meistens das Chassis des Roboters. Es kann als Wurzel des Komponentenbaumes verstanden werden. Sensoren wie Kameras, Abstandsmesser und Batterietester können dann an das Modell angebracht werden. Ihre Werte werden über das oben beschriebene *Gamebot*-Interface abgeschickt. In den neueren Versionen kam auch ein RFID-Tag-Reader hinzu, der ein schnelleres Erkennen von Personen, Opfern und Gegenständen ermöglichen soll.

Die Simulation benötigt viel Rechenleistung. Der Egoshooter Unreal Tournament 2004 läuft normalerweise ab etwa 1 GHz, jedoch benötigen die Roboteroperationen wie Laserscans und Odometrie viel Rechenleistung. An der Universität wird daher ein Verbund von 4 leistungsstarken Computern für die Simulation eingesetzt. Auf dem ersten läuft der Unreal-/ USARSim-Server. Es werden die Umgebung berechnet und die Status der verschiedenen Objekte. Der Server berechnet keine 3D Grafiken, sondern Positionen und Sensordaten. Der zweite Rechner startet den Unreal-/USARSim-Client. Dieser berechnet aus den Daten des Servers die 3D-Bilder und stellt sie in einem Fenster dar. Als zweiter Prozess läuft auf dem Rechner der `Cam_Server`: Er nimmt fortwährend Screenshots dieses Fensters auf, teilt diese in der Mitte und versendet sie als Bilder der linken und rechten Kamera. Die Sensordaten vom Server und die Bilddaten des `Cam_Servers` laufen beim dritten Rechner zusammen. Er startet die Prozesse, die sonst auf dem Notebook des Roboters laufen. Anstelle von USB und CanBus werden die Daten jetzt allerdings über eine *TCP/IP* Netzwerkverbindung empfangen. Es wird die normale Robotersoftware benutzt, lediglich beim Kompilieren werden Flags gesetzt, die den Empfang der Sensordaten auf die Netzwerkschnittstelle verlegen. Es gibt keine eigene Robotersoftware für die Simulation. Der letzte Rechner ist der Kurt_Server, der die grafische Benutzeroberfläche und die Empfangsdienste für die Sensordaten zur Verfügung stellt. Es fällt auf, dass die Benutzeroberfläche unabhängig von anderen Programmen der Simulation läuft: Es macht also keinen Unterschied, ob die Daten von einem realen oder einem simulierten Roboter stammen.

Lässt man alle diese Dienste auf einem Rechner laufen, wird die Simulation zur Diashow. Ein 3D-Laserscan, der sonst etwa eine Minute dauert, nimmt dann drei in Anspruch. Durch die

hohe Prozessorauslastung reagiert die Simulation nur schwerfällig auf Fahrbefehle, wodurch der Benutzer schnell übersteuert. Eine Steuerung über das Internet ist dann nur mit niedrigen Robotergeschwindigkeiten möglich. Weitere Informationen zu USARSim und der Umsetzung an der Universität Osnabrück findet man in [17] und [14].

2.4 Das Java Native Interface JNI

Es gibt viele verschiedene Anforderungen an Programmiersprachen: Sie sollen zum Beispiel guten Hardwarezugriff ermöglichen, unkompliziert grafische Benutzeroberflächen erzeugen oder eine breite Auswahl an Stringmodifikationsmethoden bereitstellen. Für quasi jedes dieser Anwendungsgebiete gibt es auch eine Programmiersprache, die darauf ausgelegt ist. Will man jetzt aber mehrere dieser Gebiete in einem Projekt verbinden, muss man oft den Kompromiss eingehen, dass man zum Beispiel eine gute grafische Oberfläche hat, die Textmodifikation aber umständlicher ist. Im Projektrahmen der Rettungsroboter sind die Anforderungen an die benutze Programmiersprache eine gute Hardwareverwaltung sowie eine gute grafische Oberfläche. Da Treiber häufig für C oder C++ geschrieben sind, ist das die Wahl für die Hardwareseite. Grafische Oberflächen lassen sich leicht in Java programmieren und bieten dann auch die Möglichkeit, als Applets umgesetzt zu werden.

Es gilt nun, Programmiertechniken zu finden, die eine Kommunikation zwischen den Programmiersprachen ermöglichen. Als Schnittstelle tritt hier das Java Native Interface, JNI, ein. Es umfasst Techniken und Methoden, um zwischen der Java Virtual Machine und darin laufenden Programmen und Funktionen in anderen Programmiersprachen wie C, C++ und Assembler zu vermitteln. Die Konzepte hinter dem JNI stehen allerdings in Konflikt mit der Plattformunabhängigkeit von Java. Sollen native Methoden aufgerufen werden, müssen sie in dynamischen Bibliotheken zur Verfügung stehen. Unter Linux sind das die *shared libraries*, während unter Microsoft Windows *Dynamic Link Libraries*, *DLL's*, eingesetzt werden. Beide Dateitypen arbeiten zwar nach dem gleichen Prinzip, müssen aber plattformabhängig kompiliert werden. Es gibt zwar auch Bibliotheken, die unter verschiedenen Plattformen laufen, diese stellen aber eine Ausnahme dar. Ebenfalls problematisch ist ein einheitliches Interface, da es mehrere Entwickler für die Java Virtual Machine gibt und jeder einen leicht anderen Satz von Funktionen und internen Abläufen zugänglich macht.

2.4.1 Deklaration von Methoden

In Java gibt es das Konzept von Interfaces. Diese Dateien, die mit dem Schlüsselwort `interface` gekennzeichnet sind, enthalten anstelle von kompletten Methoden nur deren Signatur, bestehend aus Rückgabetyp, Name und Parameterliste. Häufig werden auch noch Sichtbarkeitsattribute angegeben, allerdings sind alle Methoden in einem Interface implizit `abstract` und `public`. Implementiert eine Java-Klasse nun dieses Interface, so muss sie entweder alle geforderten Methoden implementieren oder durch das Schlüsselwort `abstract` anzeigen, dass die Methoden

durch eine von ihr abgeleitete Klasse definiert werden. Eine abstrakte Klasse kann nicht instantiiert werden. Diese Konzepte werden innerhalb von Java angewandt. Das JNI ermöglicht es nun, dass eine Java-Klasse eine Methodensignatur angeben kann, ohne dass diese durch eine Java-Klasse implementiert wird. Die entsprechende Methode wird dann mit dem Schlüsselwort `native` versehen, das anzeigt, dass diese Methode in einer anderen Programmiersprache definiert wird. Die Klasse muss nicht als abstrakte Klasse oder Interface gekennzeichnet werden, auch wenn sie eine Methode ohne Methodenrumpf enthält.

Die native Methode muss nun geschrieben und kompiliert werden. Dazu muss die Signatur in der implementierenden Klasse entsprechend aufgeführt sein. Der neue Methodenname ergibt sich aus dem Präfix Java_, dem Paketnamen gefolgt von einem Unterstrich, dem Klassennamen gefolgt von einem Unterstrich und dem Methodennamen in der deklarierenden Klasse. Ist die Klasse keinem Paket zugeordnet, entfallen Paketname und Unterstrich. Bei C oder C++ steht dieser Name sowohl in der Header- als auch in der Programmdatei, dabei ist zu beachten, dass die Signatur um zwei Parameter erweitert wird. Der erste Parameter, vom Typ `JNIEnv`, ist ein Pointer, der der nativen Programmiersprache den Zugriff auf die Parameter und Objekte in der `Java Virtual Machine` ermöglicht, aus der der Aufruf stammt. Der zweite ist eine Selbstreferenz auf die aufrufende Javaklasse und vom Typ `jobject`. Als Typ des Aufrufes werden dem Methodennamen die Attribute `JNIEXPORT` 'Rückgabetyp' *JNICALL* vorangestellt. Letztlich muss noch die Headerdatei 'jni.h' eingebunden werden, die die Methoden für das Interface bereitstellt. Da das JNI in das Java SDK eingebettet ist, gibt es auch Tools, um direkt die entsprechenden Signaturen zu erzeugen. Durch Aufruf von `javah` wird die entsprechende Headerdatei erstellt, die die erforderlichen Methodenköpfe für die Programmdatei enthält.

Die Implementation der Methode kann nun nicht einfach zu einer normalen Binärdatei kompiliert werden, sondern wird als eine dynamische Bibliothek für die entsprechende Systemplattform erzeugt. Diese wird dann von der Javaklasse mit der Anweisung 'static System.loadLibrary ('*Bibliothekenname*') geladen und es können die implementierten Methoden genutzt werden. Soll ein Programm native Methoden zur Verfügung stellen, aber nicht als dynamische Bibliothek kompiliert werden, können Methoden benutzt werden, um die Bindung zwischen dem nativen Programm und der entsprechenden Javaklasse manuell herzustellen.

Werden mit dem Aufruf von nativen Methoden Daten übergeben, ob als Rückgabewert oder als Parameter, muss entsprechender Speicherplatz reserviert oder freigegeben werden. Die Datenübertragung erfolgt hier nach dem 'Call-by-reference' Mechanismus. Hier muss darauf geachtet werden, dass Java und zum Beispiel C, C++ teilweise verschiedene Speicherformen für prinzipiell gleiche Datentypen verwenden. Bei trivialen Datentypen wie `integer` oder `float` sorgt das Java Native Interface automatisch für die richtige Speicherform, bei Objekten, wie Strings, muss der Benutzer bestimmte Methoden des JNI aufrufen, um etwa aus dem String in Java ein Feld von Charaktern in C, C++ zu erhalten. Daher werden meistens direkt nach Aufruf einer nativen Methode sämtliche übergebenen Parameter kopiert und konvertiert. Danach können die Übergabevariablen freigegeben werden, um ihren Speicherplatz für andere Daten nutzen zu können. Tabelle 2.1 zeigt die Datentypen in Java und der nativen Sprache auf. Auch Felder können über das JNI übergeben werden, ein direkter Zugriff auf die Elemente ist allerdings nicht möglich. Jedes Array wird als Typ `jarray` geführt und es ist dem Programmierer überlassen, den richtigen Datentyp zu wählen. Da Java-Felder Längeninformationen tragen, kann mit

Typ in Java	Typ in nativer Sprache	Größe in Bits
boolean	jboolean	8, unsigned
byte	jbyte	8
char	jchar	16, unsigned
short	jshort	16
int	jint	32
long	jlong	64
float	jfloat	32
double	jdouble	64
void	void	-

Tabelle 2.1: Typenzuordnungen im JNI

Funktion	Array Typ
GetBooleanArrayElements	boolean
GetByteArrayElements	byte
GetCharArrayElements	char
GetShortArrayElements	short
GetIntArrayElements	int
GetLongArrayElements	long
GetFloatArrayElements	float
GetDoubleArrayElements	double

Funktion	Array Typ
ReleaseBooleanArrayElements	boolean
ReleaseByteArrayElements	byte
ReleaseCharArrayElements	char
ReleaseShortArrayElements	short
ReleaseIntArrayElements	int
ReleaseLongArrayElements	long
ReleaseFloatArrayElements	float
ReleaseDoubleArrayElements	double

Tabelle 2.2: Arrayzugriffs- und Freigabemethoden im JNI

dem Aufruf `GetArrayLength` die Anzahl der Elemente bestimmt werden. Nun wird über einen Kopierbefehl das übergebene Array in ein lokales Array übernommen, das entsprechend viele Einträge hat. Auf das neue Array kann über den Index zugegriffen werden. Der Kopierbefehl legt dabei fest, welcher Datentyp erwartet wird und als welcher Datentyp die Informationen des Feldes dargestellt werden. Die Tabellen in 2.2 beinhalten die entsprechenden Methodennamen. Mit diesen Befehlen wird das gesamte Array kopiert. Bei großen Feldern oder wenn nur wenige Einträge aus dem Feld ausgelesen werden sollen, ist es sinnvoller, nur bestimmte Regionen aus dem Feld zu kopieren. Für solche Operationen stehen im JNI wieder verschiedene datentypspezifische Methoden zur Verfügung.

2.4.2 Aufruf von Javamethoden aus nativen Sprachen

Das Java Native Interface arbeitet in zwei Richtungen, es lässt, wie oben beschrieben, die Implementation von Methoden für Java in anderen Programmiersprachen zu, ermöglicht es aber auch, von außen Javamethoden aufzurufen. Dieses macht besonders Sinn, wenn eine grafische Ober-

Signatur	Datentyp in Java
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
Lkompletter Klassenname;	kompletter Klassenname
[type	type[]

Tabelle 2.3: Signaturen beim Identifizieren von Java Methoden über das JNI

fläche zu steuern ist, wie zum Beispiel in der aktuellen Kurt-Server-Software. Der Aufruf erfolgt dabei in zwei Schritten: Zunächst muss die aufzurufende Methode identifiziert werden und kann dann über die erhaltene `jmethodID` aufgerufen werden. Zum Identifizieren werden mehrere Anfragen über den `JNIEnv`-Pointer gestellt. Die erste sucht nach der entsprechenden Javaklasse, die bei einem Aufruf über das JNI einen Pointer auf sich selbst übergibt, und speichert ihre Kennung als `jclass` ab. Die zweite nutzt diese Kennung, den Namen der Methode und deren Signatur, um eine eindeutige ID für die Methode zu erhalten. Diese ID ist nur gültig, solange die Instanz der Javaklasse nicht beendet wurde. Es können sowohl Instanz- als auch Klassenmethoden über das JNI aufgerufen werden, wozu die Funktionen `GetMethodID` und `GetStaticMethodID` dienen. Diesen Funktionen werden dabei vier Argumente übergeben: Der `JNIEnv`-Pointer, die Referenz auf die Klasse, der Name der Funktion und ihre Über- und Rückgabewerte in der Form (Übergabewerte)Rückgabewerte. Hier muss wieder auf die entsprechenden Kürzel geachtet werden. Tabelle 2.3 gibt zu den Datentypen in Java die entsprechenden Signaturkürzel in der nativen Sprache an. Soll also zum Beispiel ein Objekt wie eine URL zurückgegeben werden, ist die Signatur entsprechend `Ljava/net/URL`.

Ist eines der Argumente ein Array, wird es mit einer öffnenden eckigen Klammer gekennzeichnet. Da es recht aufwändig ist, die Methoden-ID aufzunehmen, ist es sinnvoll, diese für spätere Zugriffe zu speichern. Dabei ist aber wieder darauf zu achten, dass sich die Javaklasse nicht ändern darf. Wie schon bei den Arrayzugriffen muss auch beim Aufruf von Methoden eine datentypspezifische Funktion genutzt werden. Das JNI stellt für alle trivialen Datentypen solche zur Verfügung, dabei bezieht sich das Argument auf den Rückgabetyp. Zwischen dem Aufruf von Klassen- und Instanzmethoden wird mit den Funktionen `Call'Rückgabetyp'Method` und `CallStatic'Rückgabetyp'Method` unterschieden. Gibt die Methode nichts zurück, muss als Rückgabetyp `void` eingetragen werden. Es gibt allerdings nicht für alle Javaklassen einzelne Methoden, da ja auch eigene Klassen definiert werden können. Es werden alle Methodenaufrufe, bei denen nicht triviale Datentypen oder Arrays zurückgeben werden, durch die Funktionen `CallObjectMethod` oder `CallStaticObjectMethod` abgesetzt. Das JNI bietet Methoden zur

Laufzeiterkennung von Klassen und Objekten.

2.4.3 Aufruf von Klassen- und Instanzvariablen

Die Funktionalität des Java Native Interface beschränkt sich nicht nur auf den Aufruf von Methoden. Über den `JNIEnv`-Pointer ist es auch möglich, auf Klassen- und Instanzvariablen von Java zuzugreifen. Der Aufruf erfolgt ähnlich dem von Methoden: Zunächst muss die beherbergende Klasse identifiziert werden, um über diese die Kennung der Variablen zu erlangen. Die Funktionen zum Registrieren des Datenfeldes sind dabei ebenfalls auf die verschiedenen Datentypen abgestimmt.

2.4.4 Threads und das Java Native Interface

Die Methoden des JNI erlauben es fast jedem Javaprogramm, native Methode zu deklarieren und auszuführen. Da Java aber in verschiedenen nebenläufigen Prozessen laufen kann, ist nicht auszuschließen, dass native Methoden gleichzeitig auf wichtige Systemressourcen zugreifen. Innerhalb von Java löst man dieses Problem mit synchronisierten Programmabschnitten. Diese werden durch einen Monitor überwacht, einem Objekt, dass nur einem Prozess den Zugriff auf den Programmabschnitt gewährt. Das JNI stellt auch für den Einsatz in C und C++ Funktionen zur Verfügung, um dieses Prinzip zu nutzen. Dazu werden Funktionen aufgerufen, die entweder den Monitor belegen wollen, `MonitorEnter`, oder den synchronisierten Abschnitt wieder freigeben, `MonitorExit`. Diesen Funktionen wird dabei ein `jobject` übergeben, eine Referenz auf das Monitorobjekt des betreffenden Blockes. Diese Techniken sind in anderen Programmiersprachen auch unter dem Begriff `Mutex`, `Mutual Exclusion`, bekannt.

2.4.5 Exceptions im JNI

Die nativen Methoden, die mit dem JNI definiert werden können, laufen außerhalb von Java und sind somit nicht Teil des `Exception`-Prinzipes. Das erschwert die Fehlersuche häufig, da es zu Abstürzen der `Java Virtual Machine` kommen kann oder Fehler in C oder C++ auftreten, die nicht näher erläutert werden oder keine aussagekräftigen Fehlermeldungen produzieren. Um dennoch eine Fehlerbehandlung zu ermöglichen, beinhaltet das JNI Methoden, um Fehler zu erkennen und um `Exceptions` zu werfen. Die Fehlerbehandlung in der nativen Sprache erfolgt in mehreren Schritten:

- Ausführen von kritischem Code.
- Durch Aufruf der Methode `ExceptionOccured` wird getestet, ob es ausstehende Fehlermeldungen gibt.
- Per `if`-Abfrage kann ein Programmteil zur Fehlerbehebung gestartet werden.
- Die Fehlermeldung wird mit `ExceptionDescribe` genauer betrachtet und mit `ExceptionClear` aufgehoben.

- Ist der Fehler damit endgültig behoben, kann das Programm normal weiterlaufen, wenn nicht, können `Exceptions` generiert und an das Javaprogramm propagiert werden.

Es sollte bei kritischem Code häufiger geprüft werden, ob Fehlermeldungen vorliegen, da die Ergebnisse von Operationen bei unverarbeiteten `Exceptions` unsicher sind.

2.4.6 Die Java Virtual Machine und das JNI

Die oben angeführten Vorgehensweisen setzen immer voraus, dass bereits eine `Java Virtual Machine` existiert, zum Beispiel durch den Aufruf von `java 'Javaklasse'`. Häufig wird aber die Softwarearchitektur von der nativen Seite, also durch C oder C++ aufgebaut. Es muss nun also eine `Java Virtual Machine` erschaffen werden, um Javamethoden ausführen zu können. Das JNI stellt entsprechende Methoden für C und C++ bereit. Die Initialisierung einer `Virtual Machine` erfolgt hierbei in zwei Schritten. Zunächst werden durch den Aufruf der Methode `JNI_GetDefaultJavaVMInitArgs` die Standardeinstellungen für eine `Virtual Machine` auf der aktuellen Systemplattform ermittelt. Mit diesen kann dann `JNI_Create JavaVM` aufgerufen werden, wodurch die `Java VM` gestartet wird. Sie wird dabei wie eine Variable behandelt und hat eine entsprechende Referenz. Beim Aufruf wird auch der `JNIenv`-Pointer erzeugt, dessen Einsatz oben beschrieben wird. Beide Pointer stehen nur dem Thread zur Verfügung, der die `Java VM` gestartet hat, sie können nicht weitergegeben werden. Werden dennoch Aufrufe von anderen nativen Threads über diese Pointer abgesetzt, treten Speicherzugriffsfehler auf oder die `Java VM` stürzt ab. Die Methode `DestroyJavaVM` versucht, die `Java Virtual Machine` zu beenden. Schlägt diese Operation fehl, wird eine Fehlermeldung zurückgegeben und die `JVM` läuft weiter, bis die gesamte Prozessstruktur der Aufrufe beendet wird. Über das JNI können auch native Threads an die `Java Virtual Machine` gebunden werden oder aus dieser Bindung wieder befreit werden. Dazu dienen die Methoden `AttachCurrentThread` und `DetachCurrentThread`. Mehr Informationen zum JNI bekommt man bei [11], [10] und [6].

2.5 Die aktuelle Kurt-Netzwerkarchitektur

Die aktuelle Kurt-Netzwerkarchitektur besteht aus dem Basis- oder Serverrechner und dem Laptop des Roboters als Client. Auf beiden Rechnern laufen Prozesse, die kontinuierlich Daten austauschen, diese verarbeiten oder zur Verfügung stellen.

2.5.1 Prozesse auf dem KURT-Rettungsroboter

Der Rettungsroboter ist mit einem Notebook bestückt, auf dem das Betriebssystem Linux läuft. Die Peripheriegeräte werden über den Universal Serial oder den Can Bus angesteuert. Die Netzwerkverbindungen werden über den internen WLan Adpater aufgebaut. In den Startroutinen der Robotersoftware werden individuelle Einstellungen aus den Konfigurationsdateien eingelesen. Diese bestimmen den Satz an aktiven Sensoren und Aktuatoren, die Einstellungen für selbige und somit das Verhalten des Roboters. Sind die Einstellungen ausgelesen, werden die

Treiber und Schnittstellen geladen und vorbereitet. Es werden die Netzwerkverbindungen aufgebaut, über die die Daten der einzelnen Sensoren über separierte Ports gesendet werden. Sind alle Vorbereitungen abgeschlossen, springt die Robotersoftware in eine Endlosschleife, die nur durch Fehler oder auf Benutzerwunsch verlassen wird.

Die Hauptschleife des Roboters soll möglichst schnell durchlaufen, um die Verzögerung zwischen Änderung der äußeren Situation und dem Eingang der Sensordaten gering zu halten. Dazu darf die Schleife zu keinem Zeitpunkt stehen oder warten müssen. Alle Treiber und Schnittstellen laufen daher im `nonblocking` Modus, die Schleife fragt bei jedem Treiber an, ob Daten zum Abruf bereitstehen, anstelle den Treiber anzuweisen, eine gewisse Menge Daten zu übergeben. So muss niemals gewartet werden, bis der Treiber genug Daten gesammelt hat. Sind Daten verfügbar, werden diese ausgelesen und per WLan an den Serverrechner geschickt oder lokal verarbeitet. Dazu steht dem Roboter allerdings nur begrenzte Rechenzeit zur Verfügung. Die Hauptschleife der Robotersoftware läuft folgende Punkte, falls aktiv, nacheinander ab:

- Verarbeiten der GPS Daten
- Verarbeiten der Odometriedaten
- Auswerten der HAYAI-Lokalisierung
- Auswerten der Routenplanung und Fahranweisung
- Ermitteln der Positionsdaten
- Auslesen der Roll-, Kipp- und Funknetzwerkwerte
- Senden der Positionsdaten
- Tonübertragung
- Einlesen und Verarbeiten von Joystickeingaben oder Kommandos vom Server
- Verarbeiten der Laserscannerdaten
- Auswertungen zum autonomen Fahren
- Auslesen und Senden der RGB-Kamerabilder
- Auslesen und Senden der IR-Kamerabilder
- Aufnahme eines 3D Scans

2.5.2 Prozesse auf dem Server

Auch das Verhalten des Servers wird über eine Konfigurationsdatei eingestellt. Diese hat den Vorteil, dass sie mit einfachen Texteditoren bearbeitet werden kann und Änderungen der Einstellungen kein erneutes Kompilieren der Software erfordern. Bei der Initialisierung der Serversoftware werden die Parameter aus der Konfigurationsdatei geladen und beim Aufruf von

Schnittstellen und Treibern umgesetzt. Außerdem werden eine Reihe Javaklassen registriert und gestartet, die die grafische Benutzeroberfläche erzeugen und steuern. Nachdem die Vorbereitungen abgeschlossen sind, springt die Serversoftware in einen Wartezustand. Sobald sich ein KURT-Rettungsroboter mit dem Server verbindet, startet die Serverschleife, die ebenfalls nur durch Fehler oder Benutzereingaben endet. In dieser Schleife werden die verschiedenen Prozesse angewiesen zu testen, ob die Verbindungen zum Roboter noch stehen, und dann diese auf neue Daten zu prüfen. Folgende Prozesse werden, falls aktiv, nacheinander angesprochen:

- Entgegennahme und Verarbeitung von Joystickeingaben[3]
- Empfangen der Laserdaten
- Empfangen der RGB-Kamerabilder
- Empfangen der IR-Kamerabilder
- Empfangen der 3D Laserdaten
- Empfangen der Positionsdaten
- Empfangen der Tonübertragung
- Empfangen der Kommandos vom Commando_Server

Parallel laufen Prozesse, die die Daten verarbeiten, auswerten und speichern. Zur Verarbeitung zählt hierbei auch die Visualisierung in geeigneten grafischen Oberflächen. Dazu werden die Daten gespeichert und entweder per Dateiladebefehl oder direkt nach dem Speichern als Speicherreferenz oder Datenfeld an die grafische Oberfläche übergeben. Da die Steuerungssoftware in C++ geschrieben ist, während die grafische Benutzeroberfläche in Java erstellt wurde, werden hier die speziellen Techniken des JNI benutzt, um die Javamethoden, nachdem sie zuvor registriert wurden, aus C++ heraus aufzurufen.

Die grafische Benutzeroberfläche, Abbildung 2.6, stellt die Kamerabilder, die Laserscandaten, CO_2-Messwerte, Roll- und Kippdaten sowie den Funknetzwerkstatus dar. Um den Roboter zu steuern sind Schaltflächen und Schieberegler vorhanden. Diese simulieren Joystickeingaben, die an den Roboter weitergegeben werden. Die meisten Kommandos an den Roboter bestehen aus zwei Ketten mit je vier Zeichen. Die ersten vier Zeichen geben die Funktion an, zum Beispiel 'vorwärts fahren' oder 'links drehen', die zweiten legen den Status der Funktion fest, also ob die Funktion aktiv ist oder nicht. Somit muss zu jedem Fahrbefehl auch der entsprechende Stoppbefehl gesendet werden, da der Roboter sonst nicht wieder anhalten würde. Diese Eigenschaft ist in der benutzten Implementation der Actionlistener berücksichtigt. Des Weiteren gibt es Befehle, die einen Wert übergeben sollen, hier die aktuelle Höchstgeschwindigkeit. Diese wird über einen Schieberegler eingestellt, der kontinuierlich die aktuelle Höchstgeschwindigkeit weiterleitet. Dazu wird zunächst als Funktion die Höchstgeschwindigkeit ausgewählt, dann wird aber kein Wert wie aktiv oder nicht aktiv gesetzt, sondern der neue Wert für die Geschwindigkeit. Die Befehle

[3] In der grafischen Oberfläche werden alle Kommandos als Joystickeingaben simuliert

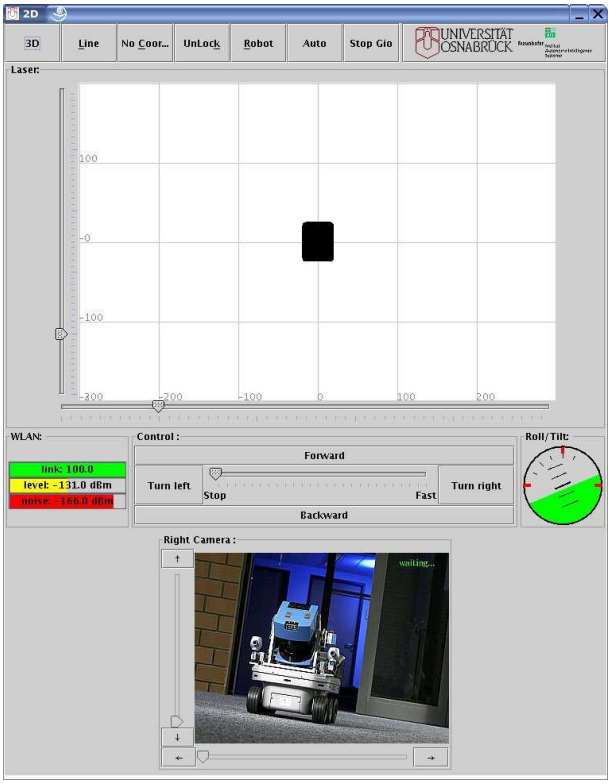

Abbildung 2.6: Die ursprüngliche Benutzeroberfläche mit Anzeige für Laserdaten, eine Kamera, Roll- und Kippwerte, sowie WLandaten. Zur Steuerung dienen die Schaltflächen.

für die Kameras sind hier ein Sonderfall. Sie bestehen aus drei Einzelbefehlen. Der erste gibt an, welche Kamera bewegt werden soll, der zweite, in welche Richtung sie bewegt werden soll, und der letzte ist das entsprechende Stoppsignal. Um die Befehle aus der Java-Oberfläche in das C++ Steuerprogramm zu übertragen wird ebenfalls das Java-Native-Interface benutzt. Es werden Methoden aus dem C++ Steuerprogramm in Java registriert und von dort aufgerufen.

Einige KURT-Rettungsroboter sind in der Lage, Abstandswerte in drei Dimensionen aufzunehmen, diese werden dann als eine dreidimensionale Punktwolke in einer eigenen Applikation dargestellt. Der Benutzer hat dort die Möglichkeit, diese Wolke zu drehen oder zu verschieben, um den besten Eindruck der Umgebung zu gewinnen. Werden mehrere 3D Scans erstellt, werden diese zu einem großen Objekt zusammengefügt, so dass eine räumliche Karte des Fahrgebietes entsteht.

Während des Betriebes werden je nach Einstellung die gemessenen Daten gespeichert. So ist eine spätere Rekonstruktion und Auswertung möglich.

Abbildung 2.7: Panoramabild, aufgenommen mit dem Rettungsroboter.

2.5.3 Besondere Features der grafischen Oberfläche

Die grafische Benutzeroberfläche stellt die vom Roboter gemessenen und übertragenen Daten in geeigneter Form dar, ist aber auch mit einigen Zusatzfeatures ausgestattet. Die installierten Webcams sind drehbar gelagert, sie können durch Servos gesteuert werden und decken damit einen größeren Sichtkegel ab. Diese Eigenschaft wird genutzt, um Panoramabilder zu erstellen. Es werden nacheinander bis zu fünf verschiedene Kamerapositionen mit einer Webcam angefahren. Die dabei aufgenommenen Bilder werden dann anhand von markanten Bildbereichen aneinander gefügt, wodurch ein Bild mit etwa vierfacher Breite entsteht, das dem Benutzer eine Art Rundblick gewährt. Das Panoramabild wird etwas kleiner als die Gesamtbreite der Einzelbilder, da diese überlappen müssen, um aneinandergefügt werden zu können. Abbildung 2.7 zeigt ein Panoramabild zusammengesetzt aus fünf Einzelbildern.

Für den Einsatz in Katastrophenumgebungen oder für den Einsatz in der Nähe von Menschen verfügt die Software über zwei Bilderkennungsalgorithmen. Die Bilder werden auf Punkte geprüft, die Hautfarbe darstellen, um Opfer leichter ausmachen zu können. Solche Bildbereiche werden wie im linken Teil von Abbildung 2.8 gekennzeichnet. Außerdem werden die Bewegungen des Roboters mit den Änderungen der Bilder abgeglichen. Ändern sich die Bilder, obwohl der Roboter steht, markiert die *MotionDetection* die betroffenen Bildbereiche, um den Benutzer auf die Bewegung aufmerksam zu machen. Die rechte Seite von Abbildung 2.8 zeigt die Visualisierung von Bewegung.

Java arbeitet mit einem Cache; wird eine Datei über ihren Dateinamen geladen, so landet sie im Zwischenspeicher. Erfolgt nun ein Dateiladebefehl mit demselben Dateinamen, wird die Datei nicht erneut von ihrer ursprünglichen Position geladen, sondern aus dem Zwischenspeicher bezogen. Daher ist es nicht möglich, die Kamerabilder an festen Stellen und mit gleichen Namen abzulegen und regelmäßig neu zu laden, sondern die Bilder bekommen unterschiedliche Dateinamen, bestehend aus der Art des Bildes, also links, rechts oder infrarot, und einer fortlaufenden Nummer. Wird ein Bild vom Roboter empfangen, wird ihm ein neuer Dateiname zugewiesen, es wird unter diesem Namen gespeichert und der Dateiname wird an die Javaoberfläche weitergegeben, damit diese dann das Bild von der Festplatte laden kann. Gleiches Prinzip gilt in der Appletversion dann auch für die Laserdaten und VRML-Welten.

Abbildung 2.8: Links: Diese Hand ist hautfarben und wird somit als Opfer erkannt; Rechts: Dieser Student bewegt sich und wird daher sofort markiert.

2.6 Grundlegendes zu Applets und Netzwerkverbindungen

2.6.1 Das Applet Prinzip

Applets entstammen der Programmiersprache Java und sind grundlegende Strukturen um, zumeist grafische, Javaprogramme in Internetbrowsern auszuführen. Sie werden in HTML-Seiten eingebettet und können dort fast wie Bilder behandelt werden. Ihnen wird eine Größe zugewiesen und ein Ersatztext, falls sie nicht angezeigt werden können. Zusätzlich können sie aber auch bereits bei ihrem Aufruf in der HTML-Seite mit Parametern versorgt werden. Dieser Mechanismus kommt häufig zum Einsatz, wenn die HTML-Seite dynamisch erzeugt wird, und es daher nicht möglich ist, die Parameter bereits bei der Programmierung des Applets zu berücksichtigen. Diese Technik stellt quasi eine einseitige Schnittstelle zwischen Sprachen zur Erzeugung von dynamischen Webseiten, zum Beispiel dem CGI umgesetzt in Perl oder anderen Programmiersprachen, und Java dar. Um Applets besonders einfach benutzen zu können, verfügen sie über grundlegende Netzwerkfunktionen. Die philosophische Frage 'woher komme ich?' beantwortet das Applet simpel durch Aufruf der Methode `getCodeBase` und bekommt als Antwort die URL des Servers, von dem es heruntergeladen wurde. Somit kann es dynamisch den Server identifizieren, von dem es geladen wurde, wodurch Referenzen auf Bilder oder andere Dateien stark vereinfacht werden. Das Applet muss dann bei der Kompilierung nur wissen, wo sich die Daten relativ zum Applet befinden, nicht aber, von welchem Server es bereitgestellt werden wird.

Um Multimediainhalte besser verarbeiten zu können, ist ein Applet in der Lage, Bilder und Tondateien von dem Rechner zu beziehen, von dem es selbst geladen wurde, und diese dann darzustellen oder abzuspielen. Nicht alle Funktionalitäten von Java sind einem Applet zugänglich. Sie unterliegen dem 'Sandbox-Prinzip', das Internetinhalten den Zugriff auf lokale Dateien und Systemfunktionen verbietet. Ein normales Applet

- darf keine Bibliotheken laden oder native Methoden definieren

- darf üblicherweise keine Dateien des ausführenden Rechners lesen oder schreiben
- darf nur Netzwerkverbindungen zu dem Rechner aufbauen, von dem es geladen wurde
- darf keine Programme auf dem ausführenden Rechner starten
- darf nur ausgewählte Systemeigenschaften auslesen
- erzeugt Fenster, die anders aussehen als normale Java Fenster

Die meisten dieser Sicherheitseinstellungen können umgangen werden, wenn das Applet signiert wird.

Zu den Funktionen zur Datenbeschaffung vom bereitstellenden Server kommen noch Methoden zur Interaktion mit dem Browser. Das Applet kann testen, ob weitere Applets gestartet wurden, und gegebenenfalls mit diesen kommunizieren, den Browser anweisen, andere URLs im aktuellen oder separaten Fenstern anzuzeigen und die Statusleiste des Browsers für Ausgaben nutzen. Die Standardausgabefunktionen, zum Beispiel `System.out.println`, schreiben bei einem Applet in die *Java Console*. Hier werden auch Fehlermeldungen angezeigt. Zur schnelleren Übertragung und zur leichteren Organisation werden die Applets meistens nicht als Klassendateien zur Verfügung gestellt, sondern zusammen mit allen benutzten Klassen zu einem komprimierten '.jar' Archiv zusammengefasst. Nur '.jar' Archive lassen sich signieren, einzelne Klassen nicht.

Applets werden von dem abrufenden Internetbrowser meistens ebenfalls in einen Zwischenspeicher geladen. Wird eine Änderung an dem Applet vorgenommen und hochgeladen, reicht es beim Benutzer häufig nicht aus, die Seite mit dem Applet neu zu laden. Er muss sämtliche Instanzen des Browsers schließen und die Webseite mit dem Applet dann erneut aufsuchen, damit die veränderte Appletversion in den Zwischenspeicher geladen wird.

2.6.2 Lebenszyklus eines Applets

Jedes Applet hat vier wichtige Methoden, die entweder überschrieben oder von der Klasse Applet übernommen werden.

- init
- start
- stop
- destroy

Anstelle eines Konstruktors wird die Methode `init` aufgerufen, sobald der Browser das Applet lädt. In dieser Methode werden alle nötigen Variablen initialisiert, Netzwerkverbindungen aufgebaut und grafische Komponenten erstellt und arrangiert. Danach und jedes Mal, wenn das Applet wieder in den Fokus des Browsers kommt (also wenn die Seite, die das Applet enthält, wieder die aktive Seite wird oder wenn das Applet wieder in den aktiven Bildbereich gescrollt

wird), ruft der Browser die Methode `start` auf. Sie vermittelt dem Applet, dass es aktiv werden soll und Rechenzeit verbrauchen darf. Damit ein Applet nicht fortwährend rechnet oder Ausgaben produziert, während es nicht aktiv ist, wird die Methode `stop` aufgerufen, sobald das Applet den Fokus des Browsers verlässt. `start` und `stop` eignen sich also sehr gut, um nebenläufige Prozesse zu stoppen oder zu starten. Die ursprünglichen Methoden `Thread.suspend` und `Thread.resume` sind allerdings veraltet und von ihrer Benutzung wird abgeraten, da sie häufig Auslöser für Verklemmungen (Deadlocks) waren. Stattdessen sollen Statusvariablen in den Threads gesetzt werden, die dem Threads anzeigen, dass er im Moment nicht arbeiten soll. Diese werden in regelmäßigen Abständen überprüft, während der Prozess zwischen den Überprüfungen ruht (`Thread.sleep`). `destroy` ist die letzte Methode, die aufgerufen wird. Sie dient dazu, verwendete Ressourcen freizugeben, wenn das Browserfenster mit dem Applet geschlossen wird. An dieser Stelle werden hauptsächlich Netzverbindungen getrennt und erzeugte nebenläufige Prozesse beendet.

2.6.3 Nebenläufigkeit in Applets

Als grafische Oberfläche läuft ein Applet automatisch in verschiedenen Prozessen, da jeder Listener einen solchen darstellt. Es kann allerdings nützlich sein, weitere Prozesse manuell anzulegen, um zum Beispiel Netzverbindungen zu verwalten. Dazu wird der entsprechende Thread in der `init` Methode des Applets initialisiert und gestartet. Es ist meistens sinnvoll, dem Thread eine niedrige Priorität zuzuordnen, damit er die grafische Oberfläche nicht ausbremst.

2.6.4 Netzwerkverbindungen unter Java

Netzwerkverbindungen werden unter Java mit Sockets realisiert. Dabei muss eine eindeutige Server-Client-Beziehung bestehen. Der Server öffnet einen `ServerSocket`, der auf dem übergebenen Port horcht, ob ein Client von außen eine Verbindung zu erstellen versucht. Sobald diese erste Kontaktaufnahme erfolgt ist, wird mit der Serversocketmethode `accept` ein neuer Socket auf Serverseite erstellt, der Ein- und Ausgabeströme bereitstellt. Die Methode `accept` wartet dabei so lange, bis eine neue Anfrage erfolgt. Mit einem ServerSocket sind zwar Leseoperationen auf der Netzwerkverbindung möglich, es kann jedoch nicht in den Strom hineingeschrieben werden. War der Aufruf `accept` erfolgreich, besteht eine Netzwerkverbindung zwischen dem Socket auf Client- und dem Socket auf Serverseite. Beide können in die Netzwerkverbindung schreiben und aus ihr lesen. Soll der Datenverkehr über die Netzwerkverbindung verschlüsselt werden, kann statt des normalen Socket ein SSL Socket verwendet werden.

2.6.5 Netzwerkverbindungen mit Applets

Applets haben die Möglichkeit eine Netzwerkverbindung zu dem Rechner herzustellen, von dem sie geladen wurden. Dazu nutzen sie das oben beschriebene Socketverfahren, es muss allerdings auch noch entschieden werden, welche Art von Daten übertragen werden soll. Es können einfache Zeichenketten, Strings, übertragen werden oder ganze Objekte. Diese müssen dafür das Interface `Serializable` implementieren. Da die Applets in eine Webseite eingebunden werden, können

sich auch Verbindungen über das *Hyper Text Transfer Protocoll* aufbauen. Diese Möglichkeit bietet allerdings einen wichtigen Nachteil, *HTTP* Kommunikation geht dabei immer vom Client aus, das Applet kann also Informationen an den Server senden, der dann einmalig die Gelegenheit hat, zu antworten. Da das vorliegende Projekt aber dringend vollständig bidirektional sein muss, fällt diese Möglichkeit heraus. Genau genommen sind Strings in Java, wie fast alle anderen Variablentypen auch, Objekte, dennoch fällt ihnen speziell bei der Netzwerkthematik eine gesonderte Rolle zu, da viele Methoden speziell auf sie ausgelegt sind. Sie können fast ohne Überschuss verschickt werden und sind leicht zu dekodieren. Viele Dateioperationen können mit Strings als Parametern aufgerufen werden, daher werden im aktuellen Projekt alle Informationen als Strings übertragen.

2.7 VRML und 3D Laserscans

Die Fähigkeit des Roboters, Abstandswerte in drei Dimensionen aufzunehmen, ermöglicht einen guten Überblick über die Umgebung zu gewinnen. In der Kurt-Server-Software ist eine Applikation zur Anzeige der Scans enthalten, allerdings ist sie nicht in Java geschrieben und kann so nicht als Applet genutzt werden. Die Sicherheitseinstellungen von Applets verbieten das Aufrufen anderer Prozesse auf dem ausführenden Rechner, und da das Applet ja mit möglichst wenig Installationsaufwand laufen soll, fiel die Wahl der Darstellung von 3D Laserscandaten auf die VRML.

2.7.1 VRML - Virtual Reality Modelling Language

Die Virtual Reality Modelling Language ist eine beschreibende Sprache für 3D Welten, wie HTML eine beschreibende Sprache für Webseiten ist. Sie kann von einigen Browsern direkt umgesetzt werden, andere benötigen verbreitete Plug-Ins wie den CosmoPlayer oder das Cortona Plug-In. VRML beruht auf dem Prinzip von Knoten, jeder Baustein und jede Form ist ein Knoten. Es gibt dabei Haupt- und Unterknoten; erstere beschreiben das Objekt generell, zweitere definieren konkrete Eigenschaften. Werden die Unterknoten nicht explizit gesetzt, verwendet der Interpreter Standardwerte für diese. Es wird zwischen drei wichtigen Arten von Knoten unterschieden:

- Gruppenknoten
- Blattknoten
- Untergeordnete Knoten

Gruppenknoten:

Gruppenknoten tragen am stärksten zur Verschachtelung in VRML-Dateien bei. Sie können wiederum Gruppen-, Blatt- oder untergeordnete Knoten enthalten. Die wichtigsten Gruppenknotenarten sind `Group`, sie wird genutzt, um mehrere Kindknoten zusammenzufassen; diese können

dann besser verwaltet werden und erhalten einen gemeinsamen Schwerpunkt, der für Rotationen und andere Bewegungen interessant ist; `Transform`, eine Erweiterung von `Group`, die neben einer Menge an Kindknoten auch direkt Werte für Rotation, Verschiebung und Skalierung bereitstellt; und zuletzt `Inline`. Diese Art von Gruppenknoten lädt eine weitere Datei in die VRML-Welt, wodurch sich Wartbarkeit und Übersichtlichkeit stark verbessern, da die Szenerie in einzelne Module aufgeteilt werden kann.

Blattknoten:

Blattknoten treten in den Dateien als eigenständige Knoten oder als Kindknoten in Gruppenknoten auf. Sie stellen die einzelnen Objekte in der VRML Landschaft dar. Die Art des Blattknoten bestimmt seine generellen Eigenschaften, die angegebenen untergeordneten Knoten definieren weitere. Es sollen die wichtigsten aufgezählt und erläutert werden.

Shape: ist die Knotenart, die alle sichtbaren Objekte beschreibt. Sie beinhaltet als Parameter zwei untergeordnete Knoten, `appearence` und `geometry`, die das Aussehen und die Form des darzustellenden Objektes definieren. Eine VRML-Welt ohne `shape` Knoten gibt es quasi nicht.

DirectionalLight: ist einer der lichterzeugenden Knotentypen. Diese werden immer als Blattknoten in die VRML-Welt eingesetzt. Das `DirectionalLight` ist parallel einfallendes Licht, das wie Sonnenlicht verstanden werden kann, es hat überall in der Szenerie die gleiche Richtung. Die Grundeigenschaften des Knotens sind seine Intensität, die Richtung in die das Licht strahlt, die Lichtfarbe und ob die Lichtquelle gerade ein- oder ausgeschaltet ist.

PointLight: erzeugt eine punktförmige Lichtquelle, die Licht in alle Richtungen abstrahlt. Neben den Eigenschaften des `DirectionalLight` muss diese Lichtquelle plaziert werden, dazu wird eine `location` angegeben, außerdem wird ihr ein Abschwächungsfaktor `attenuation` zugeordnet, der angibt, in welcher Entfernung das Licht welche Restintensität haben soll, sowie ein Radius, in dem andere Knoten bestrahlt werden sollen. Dazu werden Faktoren gesetzt, die die Abschwächung linear und quadratisch mit dem Abstand berechnen.

SpotLight: erzeugt einen Lichtkegel. Dieser muss platziert, mit Hilfe eines Vektors ausgerichtet und in seinem Öffnungswinkel definiert werden.

BackGround: ermöglicht den Himmel, den Horizont und den Boden der VRML-Welt zu verändern. Es können Farbverläufe erzeugt werden oder URLs auf Grafikdateien angegeben werden, die als Panorama für den jeweiligen Ausschnitt dienen sollen.

ViewPoint: Zur besseren Orientierung in der virtuellen Welt können Aussichtspunkte angelegt werden. Der Standardaussichtspunkt liegt nahe dem Ursprung mit Blickrichtung in Y-Richtung. Die meisten Plug-Ins unterstützen den direkten Aufruf von vorhandenen Aussichtspunkten. Ein Aussichtspunkt definiert sich über seinen Namen, seine Position, der Blickrichtung als Vektor und einer Drehung um diesen Vektor.

Untergeordnete Knoten:

Bei den untergeordneten Knoten unterscheidet man meistens zwischen drei Sorten. Die erste umfasst die geometrischen Formen, die zweite zusätzliche Attribute und die dritte die untergeordneten Knoten, die das Aussehen von Objekten weiter verändern. Untergeordnete Knoten können nicht als eigene Knoten in einer Datei existieren, da sie sich auf ein Objekt beziehen müssen.

Geometrische untergeordnete Knoten:

Box: stellt einen Quader dar. Wird keine besondere Position angegeben, befindet sich der Mittelpunkt des Quaders im Ursprung. Als Parameter können Breite, Höhe und Tiefe übergeben werden.

Cone: ist ein Kegel. Hier sind der Radius des Grundkreises und die Höhe des Kegels anzugeben. Zusätzlich kann bestimmt werden, ob Mantelfläche und /oder Bodenplatte sichtbar sein sollen.

Cylinder: erzeugt einen Zylinder. Radius und Höhe sind einstellbar, sowie Sichtbarkeit von Ober-, Unter- und Seitenfläche.

Sphere: stellt eine Kugel mit Zentrum im Ursprung dar. Es kann lediglich der Radius eingestellt werden.

Text: erzeugt einen sichtbaren Text. Dieser wird durch eine Zeichenfolge mit dem Inhalt, der Schriftart als Styleknoten sowie Ausdehnungsgrenzen angegeben. Sind die Ausdehnungsgrenzen gesetzt, führt dies zu einer Skalierung des Textes.

ElevationGrid: heißt übersetzt Höhengitter. Es stellt eine Matrix aus Grundflächen dar, denen verschiedene Höhenwerte zugeordnet werden. Als Parameter werden die Ausdehnung der Grundflächen in X- und Y- Richtung gesetzt, die Anzahlen der Grundflächen in den beiden Richtungen, eine Schrittweite in Z- Richtung sowie ein zweidimensionales Feld mit Höhenwerten über den einzelnen Grundflächen. Es werden dabei die Verbindungsflächen zwischen den einzelnen Höhenpunkten berechnet, so dass ein kantiger Berg, nicht aber ein Würfelhaufen erzeugt wird.

PointSet: ist eine Menge von Punkten. Ihnen werden gemeinsame Eigenschaften zugeteilt. Das PointSet wird meistens zum Anzeigen von Punktwolken oder zur Erstellung von Linien oder Flächen genutzt.

IndexedLineSet: erzeugt eine Menge von Linien, die durch das Verbinden von Punkten entstehen. Die wichtigen Parameter sind hier eine Tabelle von Punkten und eine Liste von Linien, die durch eine Abfolge von Punkten und einen Endmarker repräsentiert wird.

IndexFaceSet: erzeugt beliebige Formen, indem geschlossene Linienzüge als Umrandungen für die einzelnen Seiten angegeben werden. Bestandteil sind wieder eine Menge von Punkten und eine Liste von Linienzügen, nur müssen diese nun mit dem selben Punkt beginnen und enden.

Hilfsknoten:

Color: gibt einen Farbwert an. Er besteht aus drei Gleitkommazahlen zwischen 0 und 1, es wird damit das Mischungsverhältnis der Grundfarben angegeben.

Coordinate: stellt einen Raumpunkt dar, wie er in den verschiedenen Sets benutzt wird. Er besteht aus drei Gleitkommazahlen für X-, Y- und Z-Koordinate.

Normal: wird genutzt, um explizit einen Normalenvektor anzugeben. Diese werden hauptsächlich für Lichtberechnungen genutzt.

TextureCoordinate: legt fest, wie eine Textur auf der Oberfläche eines Objekts verschoben wird. Die Koordinate bezieht sich dabei auf das Koordinatensystem des Objektes, nicht auf das der allgemeinen VRML-Welt.

Untergeordnete Knoten zum Aussehen von Objekten:

Appearance: ist eine Zusammenfassung von Unterknoten. Sie nimmt Material-, Textur- und Texturmodifikationsknoten auf.

ImageTexture: lädt über eine URL eine Bilddatei, die dann auf die Oberfläche eines Objektes aufgebracht werden kann. Es wird bestimmt, ob sie in s- und/ oder in t-Richtung wiederholt werden soll. (s und t sind dabei die Achsen im Koordinatensystem des Objektes)

MovieTexture: ist wie eine ImageTexture, lädt aber Filmsequenzen. Diese werden wie Bilder angeordnet, bekommen aber zusätzlich eine Wiedergabegeschwindigkeit und eine Wiederholanweisung.

PixelTexture: wird benutzt, um simple Punktmuster direkt in eine VRML-Datei zu schreiben, anstatt sie aus einer URL zu laden. Zu dem Grundmuster gibt es Parameter für die Skalierung, Drehung und Verschiebung der Pixel.

Material: beschreibt die Oberflächenbeschaffenheit der geometrischen Form. Hier werden die Reflektionseigenschaften, das Eigenleuchten und die Transparenz festgelegt.

2.7.2 Interaktion und Animation in VRML

Eine VRML-Welt beschreibt häufig eine statische, passive 3D-Landschaft. Sie kann aber auch durch spezielle Konstrukte dynamisch und interaktiv werden. Bei Animationen spielen meistens drei Objekte zusammen: Das geometrische Objekt, das animiert, zum Beispiel verschoben, werden soll, ein Taktgeber, der anhand von Zyklendauern den Fortschritt der Animation berechnet und ein Interpolator, der diesen Fortschritt in Koordinaten, Größen oder Drehungen umrechnet. Über `Routen` werden diese Objekte miteinander verbunden, damit die entsprechenden Werte übergeben und ausgewertet werden können. Interaktivität erreicht man durch den Einsatz von Sensoren.

VisibilitySensor: Dieser Sensor überwacht einen quaderförmigen Bereich. Kann der Benutzer diesen Bereich teilweise oder ganz sehen, schaltet der Sensor auf `true` und kann zum Beispiel Animationen starten oder Filme abspielen. Ist der Bereich komplett vor dem Benutzer verborgen, das heißt, es werden andere Objekte vor dem Bereich dargestellt oder der Benutzer schaut in eine andere Richtung, stellt der Sensor auf `false` und kann Animationen und Filme stoppen, um keine unnötige Rechenleistung zu verbrauchen. (Vergleiche Applet-Methoden `start` und `stop`.)

TimeSensor: Der TimeSensor stellt eine Stoppuhr dar. Sie kann auf einmalige Zeiten reagieren oder Schleifen berechnen. Der TimeSensor nutzt die Systemuhr als Referenz. Im Schleifenmodus berechnet er je nach Durchlauflänge und bereits verstrichener Zeit einen Wert zwischen 0.0 und 1.0 als aktuellen Fortschritt.

TouchSensor: Der TouchSensor reagiert auf den Mauszeiger. Er registriert als Ereignisse, wenn sich der Mauszeiger über dem zugeordneten Objekt befindet und wenn die Maustaste dabei noch gedrückt wird. Dabei ist die Entfernung vom Standpunkt des Betrachters zum Objekt nicht relevant. (Vergleiche Java-Listener-Methoden `mouseEntered` und `mouseClicked`.)

ProximitySensor: Der `ProximitySensor` überwacht ebenfalls einen quaderförmigen Bereich. Betritt der Benutzer den Bereich, verlegt also seinen Standpunkt in ihn hinein, werden angeschlossene Events ausgelöst. Hierbei können Größe und Mittelpunkt des Bereiches festgelegt werden. Positions- und Orientierungsänderungen innerhalb des überwachten Bereiches werden ebenfalls übertragen. Mit diesem Sensor können aufwändige Animationen gestartet oder gestoppt werden, wenn der Benutzer diese nicht sinnvoll wahrnehmen könnte, weil diese z.B. zu

weit weg sind, oder andere Sensoren aktiviert werden, zu denen der Betrachter nur einen maximalen Abstand haben soll. Der `VisibilitySensor` zum Beispiel unterscheidet dagegen nur, ob der Betrachter ein Objekt sehen kann, nicht aber, wie weit es weg ist.

Sensoren zur Objektbewegung: Es gibt drei Sensoren, die Objekten Freiheitsgrade zuweisen, so dass sie vom Benutzer bewegt oder gedreht werden können. Der `PlaneSensor` ermöglicht dabei eine Bewegung in der XY-Ebene, der `CylinderSensor` eine Rotation um die Y-Achse und der `SphereSensor` ein freies Drehen um den Ursprung. Mit diesen Sensoren lassen sich zum Beispiel Türen simulieren.

Weitere Anwendungsgebiete von VRML und die Beschreibung anderer Techniken finden sich bei [2], [4] und [3].

Kapitel 3

Das Webinterface und die Simulation

3.1 Das Webinterface

Die aktuelle Netzwerkarchitektur sieht eine Steuerung des Roboters durch den Basisrechner vor. Die Funktionalität soll nun um eine Netzwerkverbindung erweitert werden und soll so Internetbenutzern die Möglichkeit bieten, den Rettungsroboter von ihrem Browser aus fernzusteuern. Dazu eignet sich am besten ein Java Applet, da es alle dazu notwendigen Funktionen bereitstellt, ohne (ungewöhnliche) Zusatzsoftware auf jedem neueren Rechner lauffähig ist und viele Komponenten der bereits geschriebenen Software in nur leicht modifizierter Form übernehmen kann. Die Architektur wird dazu um einen Serverdienst, den `AppletServer`, erweitert. Das Bild 3.1 zeigt die Teilbereiche des Projektes auf. Da die Simulation nicht für lange Betriebszeiten ausgelegt ist, und Probleme in der aufwändigen Kurt-Server-Software das gesamte System zum Stillstand bringen würden, läuft nur der `AppletServer` als ständiger Prozess. Die anderen Programme werden bei Bedarf gestartet und über interne Netzwerkverbindungen angesteuert. Um die Instanzen der Applets zu verwalten und nur einem die exklusive Kontrolle über den Roboter zu gewähren, startet der `AppletServer` eine Instanz des `UserUpdater`. Dieser kontrolliert regelmäßig, ob es Anfragen durch Applets gibt oder ob ein Benutzerwechsel vorgenommen werden muss. Des Weiteren steuert er zusammen mit dem `UTManager` das Starten und Beenden der Simulationsprogramme. Der Datenfluss zwischen den einzelnen Programmteilen ist textbasiert, alle Befehle an den Roboter und Informationen für die Applets werden als `Strings` verschickt. Durch vorangestellte Buchstaben kann das Applet die Art der Information erkennen und sie an die entsprechende Methode weiterleiten. Der Datenfluss zwischen den Programmteilen ist in der Abbildung 3.2 dargestellt.

3.1.1 AppletServer

Der `AppletServer` ist der Hauptprozess auf Serverseite. Er verwaltet die Netzwerkverbindungen und stellt aktuelle Dateinamen zur Verfügung. Die Warteschlange der Benutzer, die auf Kontrolle über den KURT hoffen, wird hier angelegt und gepflegt. Der `AppletServer` startet die Prozesse `ServerWindow` und `UserUpdater`, die ebenfalls Teil der Benutzerkontrolle sind, sowie

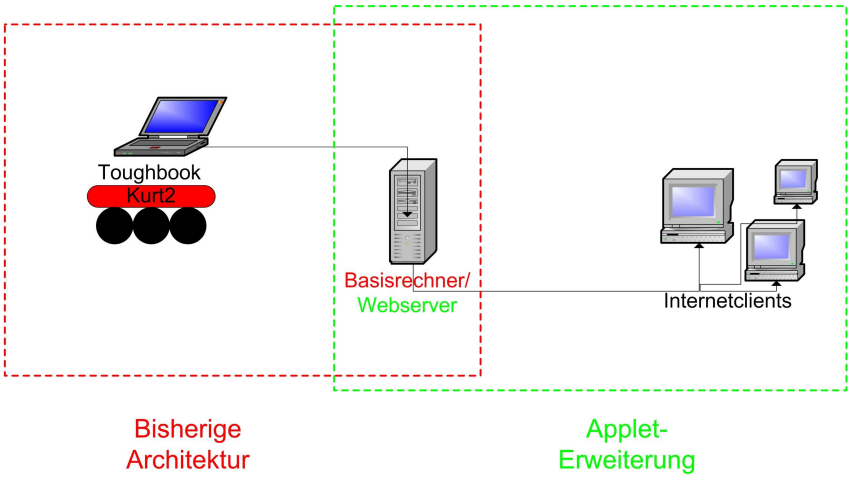

Abbildung 3.1: Alte und neue Netzwerkstruktur.

`UTManager` und `KSConnector`, die zum Initialisieren und Steuern der Simulation benötigt werden. Der `AppletServer` wird als einziger Prozess auf dem Server manuell gestartet, alle anderen sind von ihm abhängig.

In der Hauptschleife wird kontinuierlich ein `ServerSocket` aufgebaut und gehorcht, ob sich ein `TwoDApplet` von außen verbinden möchte. Sobald eine Anfrage kommt, stellt der `ServerSocket` die Verbindung her und liest einen `String` aus der Netzwerkverbindung ein. Dieser `String` identifiziert die anfragende Instanz des `TwoDApplets` und ist Teil der Benutzerkontrolle. Jetzt wird kontrolliert, ob weniger als 10 Benutzer auf Zuweisung der Kontrolle über den KURT warten. In diesem Fall wird der gerade angenommene `String` an eine `LinkedList` angehängt, die die Warteschlange darstellt. Um später diesen `String` wieder aus der Liste entfernen zu können, wird das neue letzte Element der Liste in einer temporären Variablen gespeichert. Eine Referenz auf dieses Objekt wird später dem `AppletClientThread`, der für dieses Applet erzeugt wird, übergeben. Dem Benutzer wird mitgeteilt, dass er für die Steuerung registriert wurde, und es wird aus der Anzahl der wartenden Benutzer berechnet, wie lange es maximal dauern wird. Warten bereits mehr als 10 Benutzer, wird dem hinzukommenden User mitgeteilt, dass er keine Kontrolle über den Roboter erlangen wird. Er kann aber dennoch die Kamerabilder und Laserscans, sowie virtuelle Umgebungsmodelle betrachten.

Nachdem diese Vorarbeiten durchlaufen wurden, wird der `Socket` an einen `AppletClientThread` übergeben, der die weitere Kommunikation mit dem `TwoDApplet` übernimmt. Neben den Netzwerkmethoden stellt der `AppletServer` auch `get`- und `set`-Methoden bereit, um die gespeicherten Variablen zu ändern oder auszulesen. Als Grundlage für das `ServerWindow` wird die Anzahl der laufenden `AppletClientThreads` gespeichert. Sie kann mit den Methoden `threadsup` und `threadsdown` verändert werden. Der Parameter bei `threadsdown` ist das Objekt in der `LinkedList`, das dem entsprechenden `TwoDApplet`/`AppletClientThread`-Paar zur Identifizierung zugeordnet wurde. Die Methode `threadsdown` löscht das Objekt, falls vorhanden, aus der `LinkedList`. So wird verhindert, dass ein nicht mehr existierendes `TwoDApplet` Kontrolle über

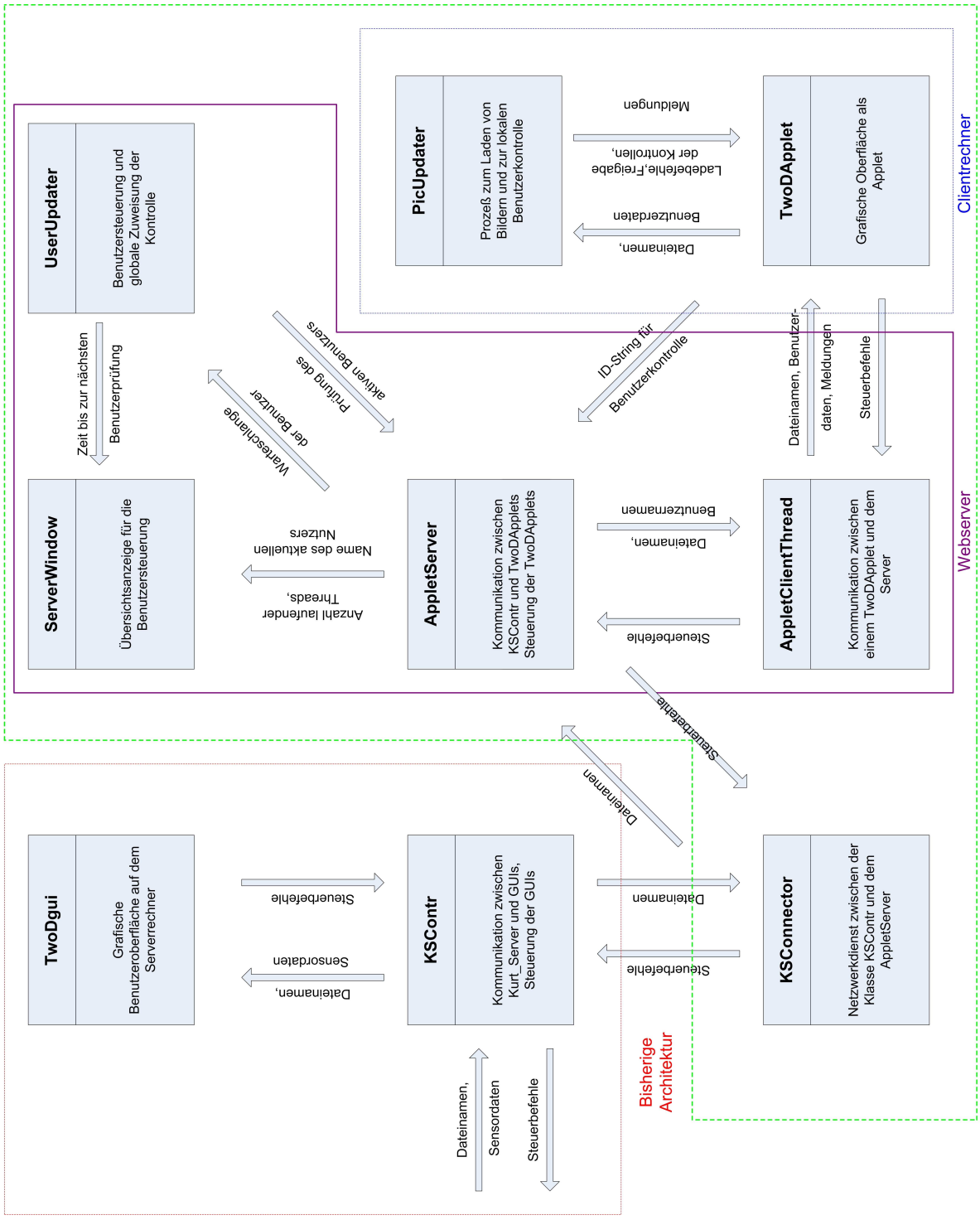

Abbildung 3.2: Klassentafel:

den Roboter bekommt, und die anderen Benutzer warten müssen, ohne dass sich der Roboter bewegt.

3.1.2 AppletClientThread

Der `AppletClientThread` ist ein nebenläufiger Prozess, der die Kommunikation mit einer Instanz des `TwoDApplets` regelt. Er enthält dazu einen eigenen Satz Dateinamen sowie Referenzen auf den `AppletServer` und den Netzwerksocket. Der Konstruktor bekommt als Parameter den Netzwerksocket zum anfragenden `TwoDApplet`, die aufrufende Instanz des `AppletServer`, das identifizierende Objekt für die Benutzerkontrolle, den Status der Simulation, sowie den `String`, den er an den Benutzer schicken soll. Dieser sollte nicht vorher versendet werden, da sonst die Netzwerkverbindung gestört wird. Sobald die übergebenen Parameter kopiert wurden, werden die Ein- und Ausgabeströme erstellt und an einen `PrintWriter` und einen `BufferedReader` weitergegeben.

Der `AppletClientThread` initialisiert dann seinen Satz von Dateinamen durch Aufrufen der entsprechenden `get`-Methoden des AppletServers. Diese werden dann mit einem führenden Buchstaben zur Identifizierung an das `TwoDApplet` weitergeschickt. Die verschiedenen Buchstaben regeln den korrekten Umgang mit den Daten:

- r steht vor dem Dateinamen für ein Bild der rechten Kamera
- l steht vor dem Dateinamen für ein Bild der linken Kamera
- s steht vor dem Dateinamen für einen 2D Laserscan
- u steht vor einem `String`, der den aktuellen Benutzer ausweisst
- v steht vor dem Dateinamen für einen 3D Laserscan im VRML-Format

Sobald ein `AppletClientThread` initialisiert und gestartet wurde, ruft er die Methode `AppletServer.threadsup` auf. Jetzt betritt er eine Schleife, in der er ständig testet, ob neue Anweisungen vom `TwoDApplet` eingetroffen sind (was nur der Fall sein kann, wenn das entsprechende Applet gerade die Kontrolle über den Roboter hat) oder ob sich die Dateinamen der aktuellen Daten geändert haben. Bei Änderung der Dateinamen werden die neuen mit dem identifizierenden Buchstaben versehen und an das Applet gesendet. Ist ein Befehl vom Benutzer angekommen, wird dieser an den `AppletServer` weitergeleitet, der ihn in über eine Netzwerkverbindung an die Klasse `KSContr` sendet, die ihn an den Roboter schickt.

Diese Schleife wird beendet, wenn entweder die `destroy` Methode des `TwoDApplet` den `String` 'EoL' sendet oder wenn die Netzwerkverbindung unterbrochen wird. Dann wird versucht, das identifizierende Objekt aus der Warteschlange zu entfernen, und anschließend wird die Zahl der Threads wieder um eins gesenkt. Häufig dauert es lange, bis ein `Socket` registriert, dass er keine Verbindung mehr hat. Daher startet jeder `AppletClientThread` eine Instanz von `Closer`.

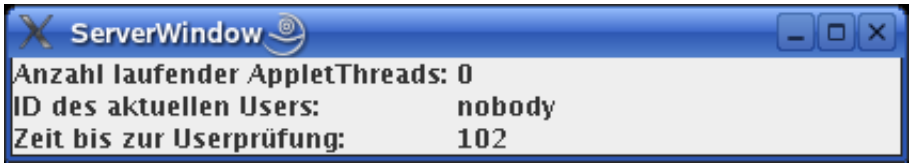

Abbildung 3.3: Das ServerWindow zur Anzeige der aktuellen Benutzerlage.

3.1.3 Closer

Der `Closer` ist ein Prozess, der testet, ob eine Netzwerkverbindung zwischen dem aufrufenden `AppletClientThread` und dem zugeordneten `TwoDApplet` besteht. Letzteres sendet ständig einen 'Ping-String', der vom `AppletClientThread` empfangen wird. Bei jedem Eingang dieses 'Lebenszeichens' wird ein Zähler im `Closer` zurückgesetzt. Geschieht dieses nicht innerhalb von 10 Sekunden, gilt die Verbindung zum Applet als verloren. Das 'EoL'-Flag des `AppletClient-Thread` wird gesetzt und dieser beendet sich.

3.1.4 ServerWindow

Das `ServerWindow`, Abbildung 3.3, ist eine kleine Statusanzeige für den `AppletServer`. Sie besteht aus einigen `JLabels`, die die Anzahl der laufenden `AppletClientThreads`, den ID-String des aktuell Kontrollierenden sowie die Zeit bis zur nächsten Überprüfung des Benutzers anzeigen. Beim Aufruf wird dem `ServerWindow` eine Referenz auf den `AppletServer` übergeben, da das `ServerWindow` von dort die Anzahl der Threads bezieht. Das Fenster wird einmal erstellt und danach nur noch durch Aufruf von `threadsup` und `threadsdown`, sowie durch den `UserUpdater` geändert. Das `ServerWindow` wird beim Beenden des `AppletServers` geschlossen.

3.1.5 UserUpdater

Der `UserUpdater` stellt den Hauptteil der Benutzerkontrolle dar. Er prüft kontinuierlich, ob es einen aktiven Benutzer gibt und reagiert auf weitere Benutzeranfragen. Beim Aufruf werden dem `UserUpdater` Referenzen auf den `AppletServer` und das `ServerWindow` übergeben. Nachdem der Thread gestartet wurde, durchläuft er zwei ineinander verschachtelte Schleifen. Die äußere ist eine 'while'-Schleife, die den Thread aufrechterhält, bis ein Stoppflag (EoL, End of Life) gesetzt wird. Die innere Schleife wird 200 mal durchlaufen. Gibt es keinen aktiven Benutzer und ist ein Benutzer in der Warteschlange, so wird dieser direkt zum aktiven Benutzer erklärt. So wird verhindert, dass ein anfragender Benutzer die vorgesehene Zeit bis zur Benutzerprüfung abwarten muss, obwohl derzeit niemand den Roboter bedient. Bei jedem Durchlauf der inneren Schleife ruht der `UserUpdater` für drei Sekunden, somit ergibt sich eine Benutzungsdauer von 10 Minuten.

Wenn die innere Schleife verlassen wird, wird getestet, ob es mehr als eine Anfrage gibt. Wenn ja, wechselt die Kontrolle vom aktiven Benutzer zum nächsten Wartenden. Gibt es keine weiteren Anfragen, gibt es keinen Grund, dem aktuellen Benutzer die Kontrolle zu entziehen. Er darf

weiterhin den Roboter steuern. Der `UserUpdater` stellt darüber hinaus die Methode `setEoL` zur Verfügung. Durch den Aufruf wird das Flag zum Beenden des Threads gesetzt. Diese Methode dient dem Aufräumen.

3.1.6 UTManager

Der `UTManager` verwaltet die Simulation auf dem Serverrechner. Wenn ein Benutzer das `TwoD-Applet` aufruft, wird es in der Warteschlange registriert. War die Warteschlange vorher leer, muss die Simulationssoftware, sowie die Kurt-Client und Kurt-Server Dienste erst gestartet werden. Dazu werden über die `Runtime` Shellskripte auf dem Server ausgeführt. Entsprechende Wartezeiten in der Abfolge der Aufrufe gewährleisten, dass es keine Zugriffe auf noch nicht vorhandene Resourcen gibt. Der `UTManager` benutzt ebenfalls einen Zähler, der nach 800 Sekunden die Simulation schließt. Dieser Zähler wird jedesmal zurückgesetzt, wenn der `UserUpdater` einen neuen Benutzer aktiv setzt oder dem aktuellen Benutzer weitere Kontrollzeit einräumt. Um den Benutzer nicht zu verwirren, werden beim Herunterfahren der Simulation die Dateinamen für die Bilder und Scans wieder auf die Anfangswerte zurückgesetzt, so dass die typischen Startbilder angezeigt werden.

3.1.7 KSConnector

Die bisherige Kurt-Server-Architektur benutzt das Java Native Interface. Dabei erzeugt es eine eigene `Virtual Machine`. Zugriffe zwischen den beiden Programmteilen, zum Beispiel über Klassenvariablen und -methoden sind daher nicht ohne Weiteres möglich. Für die Kommunikation wurden daher simple Netzwerkprozese geschrieben, die den Daten- und Befehlsverkehr übernehmen.

Die Klasse `KSConnector` stellt dabei die Serverseite dar. Hier wird ein `ServerSocket` geöffnet, der auf eine Anfrage durch die Klasse `KSContr` wartet. Ist eine Verbindung hergestellt, werden Dateinamen und Fahrbefehle über die Netzwerkverbindung ausgetauscht. Um keinen Rückstau zu erhalten, werden die angesammelten Dateinamen, die von der Klasse `KSContr` abgeschickt wurden, regelmäßig von der Klasse `KSConnector` gelöscht. Da die Übermittelung der Fahrbefehle erheblich weniger Bandbreite benötigt, kann hier auf ein Löschen verzichtet werden. Alle Fahrbefehle werden an den Roboter weitergegeben. Da die Netzwerkverbindung nur intern aufgebaut wird, erzeugt sie praktisch keine Verzögerung.

3.2 Benutzerseitige Prozesse

Der Benutzer lädt beim Aufrufen der Internetseite ein 'jar'-Archiv herunter, in dem sich die kompilierten Dateien der Javaklassen befinden. Diese werden ausgeführt und es entsteht die grafische Oberfläche. In den folgenden Abschnitten sollen dabei die Komponenten der Oberfläche vorgestellt werden, von den Einzelbausteinen bis zur Komposition. Einige der folgenden Klassen haben das Präfix *Applet*, es zeigt an, dass es sich bei der Klasse um eine Modifikation

einer bestehenden Klasse der grafischen Oberfläche handelt. Die Klasse `TwoDApplet` ist aber die einzige, die von `JApplet` erbt.

3.2.1 AppletButtonListener

Viele der beschriebenen Dateien sind Modifikationen der ursprünglichen Klassen. Sie wurden um einige Funktionen und Parameter erweitert, wodurch sie umbenannt werden mussten, um keine Konflikte mit der vorhandenen Architektur zu bekommen. Man hätte die Konstruktoren und Methoden auch überladen können, doch wäre dies unübersichtlicher und speicherplatzaufwändiger gewesen.

Die Klasse `AppletButtonListener` ist eine Erweiterung von `JoyMoveButtonListener` und erbt von `MouseAdapter`. Bei der Konstruktion werden zwei `Strings` und eine Referenz auf einen `PrintWriter` übergeben. Der erste `String` ist der Startbefehl, er wird über den `PrintWriter` an den `AppletServer` gesendet, wenn die entsprechende Schaltfläche gedrückt wird(`mousePressed`). Der zweite `String` ist der zugehörige Stoppbefehl, der gesendet wird, wenn die Maustaste losgelassen wird (`mouseReleased`), oder wenn der Mauszeiger die Schaltfläche verlässt (`mouseExited`). Um zu verhindern, dass zu viele Stoppkommandos gesendet werden, wird beim Drücken der Maus eine Statusvariable gesetzt. Nur wenn diese aktiv ist, wird das Stoppkommando in den oben genannten Fällen gesendet. Danach wird die Statusvariable zurückgesetzt.

3.2.2 AppletAction

Es wird eine zweite Listenerklasse verwendet, die ebenfalls von `MouseAdapter` erbt. Die Klasse `AppletAction` ist eine Anpassung von `JoyCamAction`. Dieser Adapter wird genutzt, um die Kameras des Roboters zu steuern. Hier werden nicht nur Start- und Stoppbefehle gesendet, sondern auch ein vorgestellter Befehl, der die betroffene Kamera identifiziert. Das Verhalten von `AppletButtonListener` und `AppletAction` ist gleich, mit dem einzigen Unterschied, dass vor jedem Befehl die Kennung der Kamera gesendet wird. Eine Instanz von `AppletAction` bekommt ebenfalls eine Referenz auf den `PrintWriter`, um die Befehle an den `AppletServer` senden zu können.

3.2.3 AppletMovePanel

Das `AppletMovePanel` ist eine erweiterte Version des `MovePanels`. Es ist ein `JPanel` mit Schaltflächen und einem Slider, wie in 3.4 abgebildet. Über die Schaltflächen können dem Roboter die Befehle 'fahre vorwärts', 'fahre rückwärts', 'drehe nach links' und 'drehe nach rechts' erteilt werden. Dazu werden die Schaltflächen mit entsprechenden `AppletButtonListenern` bestückt. In der ursprünglichen Version wurden anstelle der direkten Befehlsstrings Referenzen auf `Strings` in anderen Klassen übergeben. Dieses wurde geändert, um die Anzahl der nötigen Dateien zu verkleinern. Zur besseren Wartung und zum schnelleren Ändern könnte man die Befehlsstrings allerdings auch als Konstanten in der Klasse `TwoDApplet` speichern.

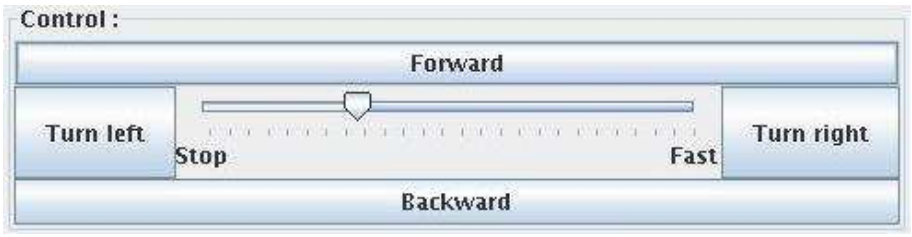

Abbildung 3.4: AppletMovePanel mit Buttons und Slider.

Beim Erstellen einer Instanz von `AppletMovePanel` wird ebenfalls die Referenz auf den `PrintWriter` der Klasse `TwoDApplet` übergeben. Dieser wird zum einen an die angehängten `AppletButtonListener` weitergegeben, zum anderen aber auch direkt genutzt um, die aktuellen Werte des Sliders abzuschicken, mit dem die Geschwindigkeit gesteuert wird. Der Slider reicht von 0 bis 512, angehängt ist ein `VelListener`, eine Implementation von `ChangeListener`, der `ChangeEvents` auffängt, den neuen Wert für die Geschwindigkeit ausliest und wegschickt. Dazu müssen entsprechend führende Nullen eingefügt werden, um das richtige Format für den Befehl zu erreichen.[1]

3.2.4 AppletScanObject

Die Klasse `AppletScanObject` stellt eine Oberfläche zum Anzeigen von 2D-Laserdaten und Ultraschallabstandswerten zur Verfügung. Sie erbt von `JComponent` und ist daher weniger eingeschränkt als ein `JPanel`. Die Messwerte des Laserscanners werden als kleine rote, die der Ultraschallsensoren als kleine blaue Punkte visualisiert. Die Ultraschalldaten werden nicht im Applet angezeigt, da sie nicht von der Simulation, s. u., berechnet werden. Zur besseren Orientierung wird der Roboter schematisch angezeigt, sowie ein skaliertes Gitter, um die Umgebung besser einschätzen zu können. Beide Features können per Button abgeschaltet werden. Ein Klick auf die Anzeige des `AppletScanObjects` arretiert den Roboter am unteren Rand der Anzeige, um möglichst viel der Fläche für Laserdaten zu nutzen, denn der normale KURT scannt nur nach vorne. Ein weiterer Klick, und der Roboter ist wieder in der Mitte.

Die Bildschirme in der Universität werden mit einer Auflösung von 1280 x 1024 Pixeln betrieben und bieten somit sehr viel Platz für die Benutzeroberfläche. Die meisten Benutzer stellen ihre Bildschirme allerdings nur auf 1024 x 768 Bildpunkte ein; zusätzlich wird die Oberfläche in einem Browser angezeigt, der durch Symbolleisten weiteren Platz einnimmt. Daher wurde das `AppletScanObject` ein wenig verkleinert, um es dennoch vollständig anzeigen zu können. Das `AppletScanObject` ist mit einem Slider ausgestattet, der ein Zoomen in das Bild hinein oder aus dem Bild heraus ermöglicht.

Die Anzeige der Scandaten ist außerdem Eingabefläche für Fahrbefehle: Wird auf eine Stelle ein Rechtsklick vollführt, berechnet das `AppletScanObject` aus dem Ort des Klicks, dem aktuellen Zoomfaktor und der Roboterposition die Zielkoordinaten und aus der Bewegung des Mauszeigers

[1] Ein Fehler in den Befehlstabellen führte dazu, dass die ersten Versionen des Applets beim Verschieben des Reglers den Befehl 'fahre rückwärts' sendeten. Leider ohne den Stoppbefehl.

den Zielwinkel und sendet diese Daten zur Auswertung an den Roboter. Dort werden die Daten durch die Giovanni Fahrplanung ausgewertet und in Befehle an die Motoren umgesetzt. Die Fahrplanung kann über einen Button im `TwoDApplet` gestoppt werden. Die Fahrplanung ist aber in der Simulation noch nicht verfügbar.

3.2.5 Pinger

Der `Pinger` ist ein Prozess, der jede Sekunde einmal den String 'ping' über die Netzwerkverbindung des aufrufenden `TwoDApplets` abschickt. Diese Lebenszeichen werden vom `AppletClientThread` empfangen und veranlassen diesen, den Zähler der Klasse `Closer` zurückzusetzen. So wird gewährleistet, dass alle `AppletClientThreads`, deren Gegenseite geschlossen wurde, nach spätestens 10 Sekunden beendet werden. Es ist wichtig, immer die genaue Anzahl an laufenden Threads zu kennen, um die Simulation entsprechend starten und stoppen zu können. Würde sich ein `AppletClientThread` nicht beenden, obwohl seine Gegenseite weggefallen ist, würde er diese Mechanismen blockieren.

3.2.6 TwoDApplet

Das `TwoDApplet` ist die eigentliche grafische Oberfläche zur Ausführung im Webbrowser. Dabei wurde viel des vorhandenen Quellcodes übernommen und neu angeordnet, allerdings wurden auch viele Methoden und Abläufe für die Netzwerkverbindungen hinzugefügt. Um eine funktionelle Anordnung zu erreichen, besteht die Oberfläche aus mehreren verschachtelten `GridLayouts`. Die grobe Aufteilung des Applets besteht aus vier Teilen, die in einer 2x2 Matrix angeordnet sind. Oben links ist das Statuspanel platziert, oben rechts befindet sich das `AppletScanObject`. Die unteren beiden Teile stellen das linke und rechte Kamerabild dar.

Das `TwoDApplet` wird nicht wie normale Klassen über einen Konstruktor erzeugt, sondern wie für Applets üblich über die `init` Methode. Diese lässt das Applet zunächst eine Sekunde ruhen, damit die Architektur nicht durch ständiges Benutzen der 'erneut laden'-Funktion des Browsers beeinträchtigt wird. Danach ermittelt das Applet den Host, der es bereitgestellt hat und speichert dessen Adresse für weitere Anfragen. Es wird eine Statusvariable gesetzt, die angibt, dass diesem Applet bisher noch nicht die Kontrolle über den Roboter zugewiesen wurde. Mit der gerade ermittelten Adresse des Hostes lässt sich nun eine Netzwerkverbindung erstellen, um mit dem AppletServer zu kommunizieren. Über diese Verbindung wird ein String abgeschickt, der das Applet für die Benutzersteuerung identifiziert.

Als letzter Punkt in der `init` Methode folgt ein Aufruf an die Methode `Init`, die die grafische Oberfläche erzeugt. Diese legt zunächst die Grundaufteilung des Applets fest und beginnt dann, die einzelnen Komponenten zu erstellen und anzuordnen. Die Oberfläche wird horizontal und vertikal in vier Teile aufgeteilt, wie in Abbildung 3.5 dargestellt. Das Logo der Universität wird als Button geladen, ein `JLabel` für Statusanzeigen wird erstellt und eine Reihe von Schaltflächen wird initialisiert, mit `ActionListenern` ausgestattet und in einem `GridLayout` angeordnet. Diese drei Komponenten bilden die obere Hälfte des linken oberen Feldes. Die untere Hälfte nimmt ein `AppletMovePanel` ein. Rechts oben wird ein `AppletScanObject` platziert, die beiden unteren

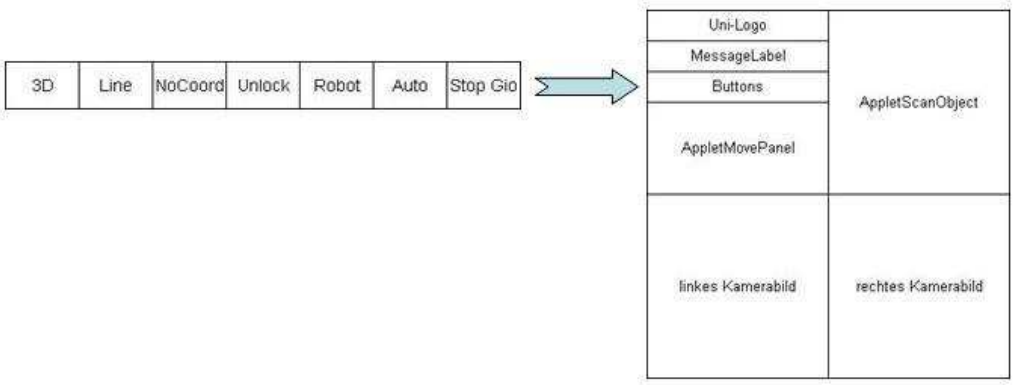

Abbildung 3.5: Aufteilung des TwoDApplets.

Felder stellen die Kamerabilder dar. Dazu werden die Bilder als `ImageIcons` in `JLabel` geladen, um die Slider und Buttons angeordnet sind. Die Buttons sind mit Listenern bestückt, die auf Mausklick Befehle versenden, die Kamera entsprechend horizontal oder vertikal zu drehen. Die Slider sind nur passiv und sollen die aktuelle Ausrichtung der Kamera darstellen, sie reagieren nicht auf Mausklicks und können nicht verschoben werden.

Bei der Initialisierung in der normalen Umgebung, also im Verbund Roboter-Basisrechner, werden alle Komponenten sofort als sichtbar erstellt. In der Appletversion werden die Schaltflächen und das `AppletMovePanel` zunächst auf unsichtbar gesetzt und sind nur dann sichtbar, und somit anwählbar, wenn das aktuelle Applet die Kontrolle über den Roboter hat. Die Benutzeroberfläche mit eingeschalteten Kontrollen ist in Abbildung 3.6 dargestellt. Ist die Steuerzeit des Benutzers vorbei, werden die Kontrollen wieder ausgeblendet und, er wird darüber informiert, dass er nicht erneut die Kontrolle erhält.

3.2.7 PicUpdater

Die Klasse `PicUpdater` steuert den Datenfluß zum `TwoDApplet`. Er ist ein nebenläufiger Prozess, der in einer (fast) Endlosschleife das Applet anweist, den nächsten `String` aus der Netzwerkverbindung zu lesen und an ihn weiterzugeben. Wird ein nichtleerer `String` übergeben, wird anhand des Anfangsbuchstabens entschieden, welche Art von Anweisung hier vorliegt, und der `String` wird entsprechend an die richtige Methode des `TwoDApplets` weitergegeben. Dort wird dieser entweder sofort umgesetzt und zum Beispiel als neuer aktiver User gespeichert, oder er wird Teil eines Ladevorganges für ein Bild, eine Datei mit Scandaten oder eine VRML-Welt. Kann der `PicUpdater` keine sinnvollen Informationen vom Applet erlangen, gibt er eine Fehlermeldung aus und weist das Applet an, eine entsprechende Nachricht anzuzeigen. Danach beendet sich der Thread selbstständig.

Die Schleife läuft sehr schnell durch und würde das System unnötig belasten, daher wird die Priorität des Threads auf das mögliche Minimum gesetzt. Zusätzlich ruht der Thread bei jedem Durchlauf für 50 Millisekunden. Die dadurch entstehende Verzögerung ist klein gegen die

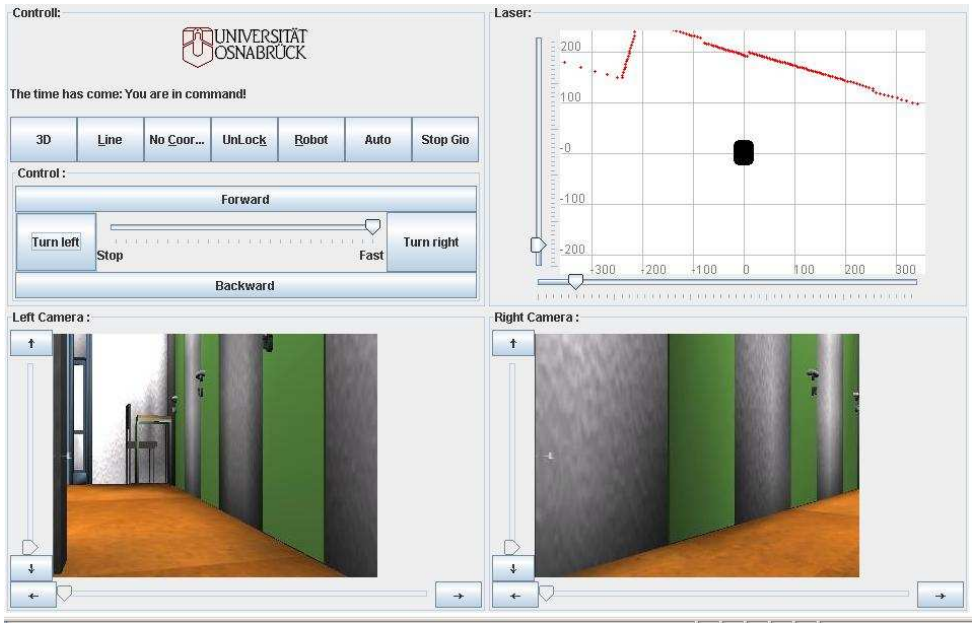

Abbildung 3.6: TwoDApplet, aktiv mit Kontrollen und Daten aus der Simulation.

Verzögerung der Roboter-Basisrechner-Architektur und die Ladezeiten für die Daten.

3.2.8 Synchronisierung

Die Kameras des Roboters können sehr viele Bilder, bis zu 30 pro Sekunde, aufnehmen. Bei den Bildern von zwei Kameras, den Laserscannerdaten und den Ausgaben der Benutzersteuerung kommt eine große Anzahl an Anweisungen beim Benutzer an. Nicht jede Internetverbindung ist gleich schnell und nicht jeder Rechner kann die Daten in der gleichen Geschwindigkeit verarbeiten, in der sie erzeugt werden. Es muss nun verhindert werden, dass beim Internetbenutzer eine Warteschlange an Dateinamen oder Anweisungen entsteht, die auf längst veraltete Kamerabilder oder Laserdaten zeigt.

Zu diesem Zweck werden die Sende- und Empfangsbuffer der Netzwerkverbindung zwischen `AppletServer` und `TwoDApplet` sehr klein gesetzt, um nur maximal die letzten 5 Befehle abzuspeichern. Die Befehle, die darüber hinaus gehen, werden einfach verworfen. Da aber so Anweisungen der Benutzersteuerung gelöscht werden könnten, werden diese nicht mehr nur bei Änderung des aktiven Benutzers gesendet, sondern kontinuierlich abgeschickt. Gleiches gilt für die Dateinamen der VRML-Welten. Eine interne Abfrage verhindert allerdings, dass bei jedem Eintreffen des VRML-Dateinamens ein neues Browserfenster erzeugt wird. Es können nur Befehle auf Client-Seite verworfen werden, Fahrbefehle an den Roboter werden immer empfangen und weitergegeben.

3.3 3D Scans in VRML

Der Kurt3D Rettungsroboter ist mit einem neigbaren Laserscanner ausgestattet. Er nimmt Wertepaare von Winkel des Scanners vertikal, Winkel der Abstrahlung horizontal, sowie Abstand in dieser Richtung auf. Diese Koordinaten in Kugelkoordinaten werden dann umgerechnet in kartesische Koordinaten und an den Server versendet. Dabei gelten folgende Gleichungen:

$$\begin{aligned} x &= r \cdot cos(\theta) \cdot cos(\varphi) \\ y &= r \cdot cos(\theta) \cdot sin(\varphi) \\ z &= r \cdot sin(\theta) + h \end{aligned}$$

Dabei ist h die Höhe des Scanners über dem Boden, θ der vertikale Winkel des Scanners zum Boden, φ der Abstrahlwinkel in der Horizontalen und r der gemessene Abstand zum Objekt in der Scannerstellung.

Die aktuelle Kurt-Server-Software stellt die räumlichen Scannerdaten meistens als Punktwolke dar. Verschiedene Algorithmen können aber hinzugeschaltet werden, um Linien und Flächen aus den Sensordaten zu extrahieren und anzuzeigen. Der einfachste Ansatz, das Anzeigen von Punktwolken, wurde aufgegriffen und in VRML umgesetzt. Dazu wird an der entsprechenden Stelle der Scandatenverarbeitung die neu geschriebene Methode `scanTovrml` aufgerufen. Sie erstellt eine neue VRML-Datei, schreibt den Standardkopf und erstellt dann ein `PointSet` mit allen in der jeweiligen Scandatei enthaltenen Punkten. Zur besseren Übersicht wird der Hintergrund schwarz gesetzt und die einzelnen Punkte emitieren die Standardbeleuchtung in der VRML-Welt grün.

Ist die Datei geschrieben, wird der Dateiname zurückgegeben, an den `AppletServer` weitergeleitet und an die `TwoDApplets` versendet. Wenn die Applets einen VRML Dateinamen empfangen, gekennzeichnet durch ein führendes 'v', weisen sie den Browser an, ein neues Fenster zu laden, in dem die VRML-Welt angezeigt wird. Der Laserscanner berechnet die Daten um sich herum, er liegt also im Ursprung des Koordinatensystems. Da in der VRML-Welt keine Skala sichtbar ist, wird ein roten Quader als Ersatz für den Roboter in das Modell eingebracht. Er wird etwa 30 Zentimeter unter dem Ursprung platziert und ermöglicht so eine gute Orientierung in der Punktwolke.

Abbildung 3.7 zeigt einen Laserscan mit Roboter. Es fällt auf, dass in der 3D-Darstellung Objekte nahe am Roboter angezeigt werden, die nicht im Kamerabild erscheinen. Dieser Zusammenhang wird auch ,,Blick durch den Strohhalm" genannt, da Kamerabilder häufig das Gefühl vermitteln, man befinde sich etwa einen halben Meter weiter in Blickrichtung. Um die Szenerie schneller überblicken zu können, werden acht `ViewPoints` in die Darstellung eingebracht. Sie sind auf einem Würfel um den Ursprung angeordnet. Diese Viewpoints sind meistens hilfreich, um ein wenig Abstand zum Roboter zu gewinnen und so die Umgebung besser betrachten zu können. Allerdings sind die 'fit'-Funktionen der VRML-Plug-Ins, s. u., hier auch sehr hilfreich. Sie fahren so weit aus der Szene heraus, dass sie alle oder möglichst viele Objekte in den Sichtbereich rücken. Auf zweidimensionalem Papier kommt die Tiefenwahrnehmung natürlich schlecht heraus. Erst

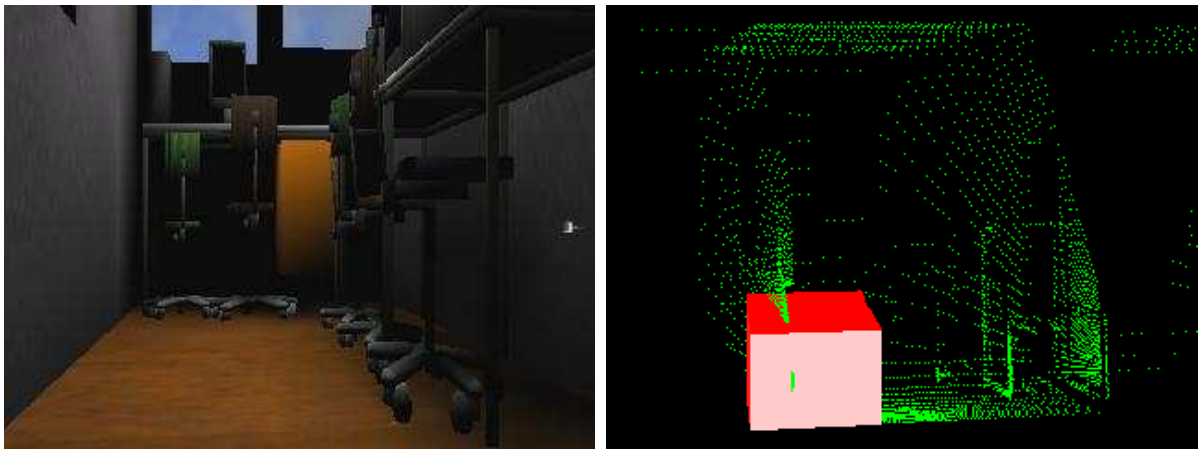

Abbildung 3.7: Links: Kamerabild aus der Simulation. Rechts: VRML-Anzeige der 3D-Laserscandaten

durch Drehen oder Bewegen der Szene werden Abstände klar und es lassen sich Rückschlüsse auf die räumliche Anordnung ziehen.

3.3.1 VRML Plug-Ins

Die meistgenutzen Browser, Internet Explorer und Firefox, bieten leider von Haus aus keine VRML-Unterstützung, diese kann aber über Plug-Ins nachgerüstet werden. Das bekannteste Plug-In ist hier der Cosmo Player von CA. Die Herstellerfirma hat allerdings den Support eingestellt, da mit X3D der Nachfolger von VRML auf dem Markt ist. Die neuesten Standards und Browserversionen werden nicht mehr unterstützt. Ein anderes ebenfalls verbreitetes Plug-In ist das Cortona Plug-In von Parallel Graphics. Diese Erweiterung ist aktueller und unterstützt auch neuere Browserversionen und Betriebssysteme. Bei der Rechenleistung kann bestimmt werden, mit welcher Schnittstelle die Grafikinformationen bearbeitet werden sollen. Es stehen der `OpenGL`, der `DirectX` und der `R98` Renderer zur Auswahl.

Zur Navigation und Betrachtung der VRML-Welten stellen die Plug-Ins verschiedene Bewegungsmöglichkeiten bereit. Der Betrachter kann seinen eigenen Standpunkt und seine Blickrichtung ändern und sich somit um das Objekt herum bewegen, oder er ruht und dreht das Modell. Es werden `align` Funktionen angeboten, die den Blickpunkt des Betrachters an der Umgebung orientieren, ihn zum Beispiel parallel zum Boden ausrichten. Wie bereits angesprochen, können die Wegpunkte angesteuert werden. Dazu gibt es eine Liste mit allen enthaltenen Punkten oder die Sichten können der Reihe nach durchgeschaltet werden. Um die Szene gut zu beleuchten, können zu den vorhandenen Lichtquellen auch Scheinwerfer aktiviert werden, die die Szenerie vom Blickpunkt des Betrachters aus erhellen. Damit die Benutzer des Applets die VRML-Welten in ihrem Browser betrachten können, besteht ein Link auf der Webseite, der sie zur Parallel Graphics Homepage weiterleitet, wo das Cortona Plug-In kostenlos heruntergeladen werden kann.

Kapitel 4

Ausblick

Das Ziel der Arbeit war die vorhandene Netzwerkstruktur in das Internet zu erweitern. Dazu wurde ein lauffähiges Applet zur Steuerung des Rettungsroboters programmiert. Es kann Bilder und Scandaten anzeigen und hat Routinen zur Benutzerkontrolle. Welche Erweiterungsmöglichkeiten und Verbesserungen gibt es aber noch?

Das Applet unterstützt noch nicht alle Funktionen des Roboters. Die meisten wurden deshalb nicht implementiert, weil sie bisher noch nicht von der Simulation in Unreal Tournament unterstützt werden. Sobald aber auch die Werte für die Roll- und Kippsensoren, die Infrarotkamera, der CO2-Sensor und der WLanstatus berechnet werden können, könnte das Applet entsprechend erweitert werden.

Die Simulation erschafft derzeit lediglich einen Roboter, der in einer leblosen Karte herumfahren kann. Es gibt für den Roboter weder zeitliche noch anderweitige Einschränkungen. Um die Simulation realistischer oder interessanter zu gestalten, könnten sich die Piloten des Rettungsroboters auf eine Rettungsmission begeben, bei der sie in vorgegebener Zeit Opfer finden müssen oder einer Gruppe von Rettern einen Weg weisen müssen. Eine Bewertungsfunktion könnte dann jeder Mission eine Punktezahl zuordnen, anhand derer sich die Piloten vergleichen können. Um die Besten zu ehren und somit einen Anreiz zum erneuten Probieren zu geben, könnte eine Bestenliste online gestellt werden. Dieser infantile Ansatz hat selten seine Wirkung verfehlt.

Es ist im Moment vorgesehen, dass sich immer nur ein Rettungsroboter auf der Karte befindet, allerdings wäre es auch möglich, mehrere Roboter zusammen oder konkurrierend arbeiten zu lassen. So könnten Internetbenutzer eine Rettungsmission zusammen koordinieren oder aber gegeneinander antreten, um festzustellen, wer in einer vorgegebenen Zeitspanne die meisten Opfer findet. Eine Liga für kooperierende Roboter und Agenten gibt es zwar, die Kurt-Software-Architektur ist aber noch nicht darauf ausgelegt.

Es gibt mehr als eine Forschungsgruppe, die sich mit Rettungsrobotern beschäftigt, insbesondere gibt es mehr als eine Gruppe, die ihre Roboter simuliert. Es könnte also eine Community geschaffen werden, in der die angeschlossenen Arbeitsgruppen ihre Simulationen zentral vorstellen und anbieten und in der Verbesserungen und Vorschläge diskutiert werden können.

Literaturverzeichnis

[1] W. Burgard, P. Trahanias, D. Hähnel, D. Schulz, H. Baltzakis, and A. Argyros. TelePresence in Populated Exhibitions Through Web-Operated Mobile Robots. *Journal of Autonomous Robots*, 15:299–316, 2003.

[2] Uwe Debacher. VRML-Einführung. *http://www.debacher.de/vrml/vrml.htm*, 2006.

[3] Rolf Däßler. Das Einsteigerseminar VRML. *http://fabdq.fh-potsdam.de/bhv/content.html*, 2000.

[4] Jörg H. Kloss et al. VRML 97 : der internationale Standard für interaktive 3D-Welten im World Wide Web. *Addison-Wesley Longman*, 1998.

[5] Epic Games. Unreal Tournament. *http://www.unrealtournament.com/*, 2006.

[6] Michael Härtfelder. Kaffee mit Vitamin C. *http://www.haertfelder.com/jni.html*, 2000.

[7] RoboCup Initiative. Robocup Official Site. *http://www.robocup.org/02.html*, 2006.

[8] RoboCup Rescue Initiative. RoboCup-Rescue Official web page. *http://www.rescuesystem.org/robocuprescue/*, 2006.

[9] Thailife Magazine. Mission ausgeführt: Thailändisches King Mongkut Team ist stolzer Gewinner beim World Robot Wettbewerb in Bremen . *Thailife Magazine*, page http://www.thailife.de, 2006.

[10] Sun Microsystems. JavaTM 2 Platform Standard Edition 5.0 API Specification . *http://java.sun.com/j2se/1.5.0/docs/api/*, 2006.

[11] Sun Microsystems. The Java Tutorials for Java SE 6. *http://java.sun.com/docs/books/tutorial*, 2006.

[12] Universität Rostock. Roboka, Der Roboter im Kaufhof. *http://www.roboka.uni-rostock.de/pages/info.html*, 2002.

[13] Toni Schornböck. C++ An introduction. *http://tutorial.schornboeck.net/struct.htm*, 2006.

[14] Kai Lingemann Andreas Nüchter Jochen Sprickerhof Stefan Stiene Sven Albrecht, Joachim Hertzberg. Device Level Simulation of Kurt3D Rescue Robots. *http://kos.informatik.uni-osnabrueck.de/download/UOSSim/index.html*, 2006.

[15] Jens Twiefel. Sichere internetbasierte echtzeitfähige Robotersteuerung, Studienarbeit, Universität Bonn. *www.twiefel.de*, 2003.

[16] KTO Kommunikation und Technologietransfer Odenthal. Kommunikation und Technolgietransfer Odenthal. *http://www.kurt2.de/*, 2003.

[17] Jijun Wang. USARSim-manual V2.0.2. *Sourceforge.net; Project UsarSim*, 2006.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Osnabrück, Oktober 2006