UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

AGAS
Arbeitsgruppe Aktives Sehen

Erweiterte Integration von USARSim in die Robbie-Architektur

Studienarbeit
im Studiengang Computervisualistik

vorgelegt von

Dennis Holzhäuser

Betreuer:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Erstgutachter:	Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik
Zweitgutachter:	Dipl.-Inf. Johannes Pellenz, Institut für Computervisualistik, Fachbereich Informatik

Koblenz, im Juni 2007

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien der Arbeitsgruppe für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja ☐ nein ☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ☐ nein ☐

Koblenz, den ..... Unterschrift

Inhaltsverzeichnis

1 Einleitung **9**

2 Beschreibung der Simulation **13**

2.1 Überblick 13

2.1.1 Alternativen zu USARSim 13

2.1.2 Client-Server Architektur 14

2.2 USARSim 15

2.2.1 Simulation der Umgebung 15

2.2.2 Simulation der Roboter 16

2.2.3 Simulation der Sensoren 17

2.2.4 Konfiguration des Roboters 18

2.3 Schnittstelle zur Software 20

2.3.1 Systemkomponenten 21

2.3.2 Funktionsweise der Schnittstelle 23

3 Erweiterung der simulierten Robotersensorik **25**

3.1 Einbinden des 3D Scanners 25

3.1.1 Unterschiede zur Hardware 26

3.1.2 Konfiguration und Steuerung des 3D Scanners 27
3.2 Implementation des 3D Scanners 27
3.2.1 Erweiterung des USARSimMessageConverter 28
3.2.2 Weiterleiten der Daten 28
3.3 Einbinden der Kameras 29
3.3.1 Simulation und Akquirierung der Kamerabilder 29
3.3.2 Umwandlung der Kamerabilder 30
3.3.3 Konfiguration und Steuerung der Kameras 31
3.4 Implementation der Kameras 32
3.4.1 Implementation des Kameraservers 32
3.4.2 Implementation der Kameraschnittstelle 33
3.5 Einbinden des Wärmesensors 36
3.5.1 Simulation von Wärmestrahlung 36
3.5.2 Einbinden des VictimRFID-Sensors 37
3.6 Implementation des Wärmesensors 38
3.6.1 Erweiterung des USARSimMessageConverter 38
3.6.2 Berechnung des Thermalbildes 39
3.6.3 Visualisierung der Opferwärme 40
3.7 Simulation realitätsnaher Daten 41
3.7.1 Erzeugung des Sensorrauschens 42

4 Experimente und Evaluation 45
4.1 Versuchsaufbau 45
4.1.1 Anforderungen 46
4.2 Versuchsdurchführung 47

4.2.1 Test der Erweiterungen 47
4.2.2 Test der Autonomie 48
4.3 Ergebnisse 48
4.3.1 3D Karten 48
4.3.2 Tiefenschätzung 50
4.3.3 Opfererkennung 52
4.3.4 Autonome Exploration und Opfersuche 53

5 Zusammenfassung **55**

A Unreal Glossar **59**

Kapitel 1

Einleitung

Ziel des *Robbie* Projektes, der Arbeitsgruppe Aktives Sehen, war die Entwicklung eines Rettungsroboters zur Teilnahme in der *Robocup-Rescue-League*. Hier gilt es, zur Unterstützung von Hilfskräften in Katastrophengebieten, menschliche Opfer aufzuspüren und deren Position in eine Karte einzutragen. Im Rahmen des Projektpraktikums *Robbie 7* wurde zum ersten Mal der Robotersimulator *USARSim*[1] in der Version 2.0.6 zur Erreichung dieses Ziels verwendet. USARSim ist ein Tool zur Simulation von Rettungsrobotern im Hinblick auf Human-Robot-Interaktion (HRI) und Koordinierung von mehreren Robotern [**?**]. Hier sind verschiedene Anwendungsgebiete denkbar: das Auffinden von Personen oder die Koordinierung von großen Rettungsteams nach einer Katastrophe wie zum Beispiel einem Erdbeben (siehe Bild 1.2).

Robbie, der reale Roboter steht nicht immer zur Verfügung oder kann defekt sein. So können Tests der Software nur bedingt oder mit hohem zeitlichen Aufwand durchgeführt werden. Ein Test mit dem Roboter birgt zusätzlich noch das Risiko, die Hardware zu beschädigen, zum Beispiel in schwierigem Gelände. Wesentliche Vorteile der Simulation sind somit die Schonung der Hardware-Ressourcen und die bessere Verfügbarkeit. Zudem lassen sich in einer Simulation, beliebige Szenarien ohne großen Aufwand und Kosten erstellen. In der Simulation wird das physikalische Verhalten von verschiedenen Robotertypen und Sensoren in einer virtuellen Umgebung simuliert. Hierbei ist die Operatorstation

[1] Urban Search And Rescue Simulation (siehe Anhang A)

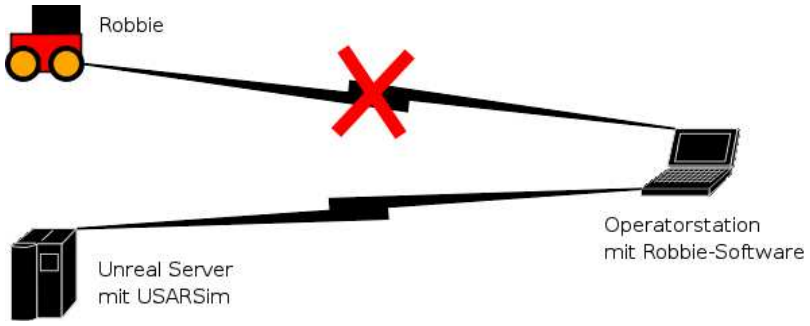

Bild 1.1: Verbindung während der Simulation

anstatt mit dem realen Roboter, mit dem Simulationsserver und USARSim verbunden, wie Bild 1.1 verdeutlicht.

Bisher wurden in der Simulation nur die Sonarsensoren, der 2D Laserscanner sowie die Odometriedaten des Roboters verwendet. Im Rahmen mehrerer Studienarbeiten und des achten Projektpraktikums wurde der Roboter erweitert. Hinsichtlich der Erweiterung von Robbie, sollen die neuen Sensoren ebenfalls simuliert werden können. Alle Erweiterungen dieser Arbeit beziehen sich auf USARSim 2.0.6 und das dazu gehörige Manual [**?**] in der Version 2.1. Die Ziele der vorliegenden Arbeit sind somit die Umstellung des 2D Laserscanners zur Erstellung von 3D Scans, die Implemtation des Wärmesensors zur Erzeugung von Wärmebildern sowie die Simulation von Kamerabildern. Zusätzlich soll eine Evaluation der simulierten Daten, im Vergleich zu den Daten des realen Roboters zeigen, wie realitätsnah die Simulation ist. Um die simulierten Daten zu evaluieren, ist es sinnvoll, zunächst bestimmte Szenarien aus einer virtuellen Umgebung in der Realität nachzustellen, um dann die generierten Daten der einzelnen Sensoren vergleichen zu können.

USARSim nutzt die 3D Gameengine des Spieles *Unreal Tournament 2004* (UT2004), mit dessen fortschrittlicher Graphik und physikalischer Modellierung, zur Simulation und Darstellung der Roboter und Umgebungen [HLN03]. Unreal Tournament bietet somit auch die Möglichkeit, Kamerabilder aus der Sicht eines Roboters oder eines freien Beobachters in einer dreidimensionalen Umgebung zu erzeugen und anzuzeigen. Diese Bilder sind mit einer entsprechenden Schnittstelle an die GUI weiterzuleiten. Bild 1.2 zeigt einen Screens-

Bild 1.2: Simuliertes Katastrophenszenario (Screenshot UT2004)

hot aus Unreal Tournament 2004 in einer, der von USARSim bereitgestellten Maps[2], in welcher ein Erdbebengebiet als Katastrophenszenario nachgestellt ist. Die Simulation eines Wärmesensors wiederum, stellt ein anderes Problem dar. Da USARSim bislang keine Wärmestrahlung unterstützt, ist die Implementation des Wärmesensors nur über die Positionen der Opfer zu erreichen. Mit Hilfe eines RFID Sensors, der die Positionen von Opfern detektieren kann, wird über einen Umweg ein wahrscheinliches Wärmebild generiert. Ferner gilt es noch, den von USARSim bereitgestellten 3D Sensor zu implementieren und die aquirierten Messdaten in ein einheitliches Format zu übertragen.

Der Aufbau der vorliegenden Arbeit ist wie folgt gegliedert:

Kapitel 2 gibt einen Überblick und stellt die einzelnen Simulationskomponenten vor. Dabei werden die Möglichkeiten mit USARSim, sowie das bestehende Setup des Robbie-Systems in der Simulation und die Funktionsweise der Schnittstelle zur Robbie-Software

[2] genauer DM-compWorldDay1 (siehe Anhang A)

beschrieben. In Kapitel 3 werden die Lösungen der aufgeführten Aufgaben im Einzelnen beschrieben. Dazu wird das installierte System, im Hinblick auf die zu erreichenden Ziele erläutert und die nötigen Konfigurationen und Implementationsschritte erklärt. Kapitel 4 vergleicht schließlich die Simulation und die Realität mittels einer Evaluation. Dabei werden die unterschiedlichen Methoden, Probleme und Ergebnisse dieser Evaluation behandelt. In Kapitel 5 werden die Ergebnisse zusammengefasst und die gesammelten Erfahrungen beschrieben. Schließlich wird ein Ausblick auf mögliche Weiterentwicklungen gegeben.

Kapitel 2

Beschreibung der Simulation

In diesem Kapitel erfolgt eine genauere Beschreibung der Systemkomponenten, welche zur Durchführung einer Simulation benötigt werden. Zuerst erfolgt ein kurzer Überblick über die Architektur und alternative Simulationsprogramme. Dann wird USARSim vorgestellt, wobei generelle Sensor- und Robotertypen aufgeführt werden und mit diesem Wissen, die Konfiguration des Roboters erklärt wird. Im Anschluß daran, folgt die Beschreibung der Schnittstelle zwischen USARSim und Robbie-Software.

2.1 Überblick

Dieser Abschnitt gibt einen Überblick der Simulation mittels Unreal Tournament 2004 und USARSim, stellt kurz die Vorteile gegenüber alternativen Simulationstools vor und verdeutlicht die Client-Server Architektur, welche das Grundgerüst und die wichtigsten Bausteine für die Simulation enthält.

2.1.1 Alternativen zu USARSim

Neben USARSim gibt es andere Tools zur Simulation von Robotern, deren Sensoren und interaktiven Umgebungen. Insbesondere für die Simulation von Kameras werden 3D fä-

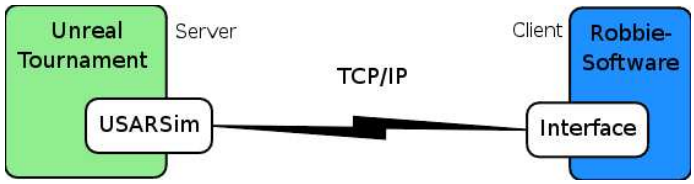

Bild 2.1: Client-Server Architektur der Simulation

hige Simulationsumgebungen benötigt. Das von Microsoft vorgestellte System *Robotic Studio* beinhaltet unter anderem eine Simulationsumgebung, die aber nur auf Plattformen welche das Framewok *.NET 2.0* unterstützen, läuft [Mer07]. Ein plattformübergreifender Simulator *Webots*^{TM} [Mic04], der die ODE Graphikengine (*Open Dynamics Engine*) benutzt, hat den Nachteil, ein kommerzielles Produkt zu sein. Weiter zu erwähnen ist *Delta3D* [mis07a], ein open-source Simulator, dessen Konzept ähnlich dem von USARSim ist. Hauptsächlich werden mit Delta3D militärische Anwendungen entwickelt und getestet [CLW^{+}07].

Mit einer langjährig weiter entwickelten Graphikengine, einem effizientem Physiksystem und einer, in zahlosen Onlinespielen bewährten, Client-Server Architektur, sowie einem plattformunabhängigen Simulator mit vorgefertigten Robotermodellen, besitzt letztlich die Kombination aus Unreal Tournament 2004 und USARSim, beste Vorraussetzungen für die Simulation von Robbie.

2.1.2 Client-Server Architektur

Wie in Unreal Tournament 2004, wo mehrere Spieler als Clients online gegeneinander, auf einem Server antreten können, bildet die Client-Server Architektur die Grundlage der Simulation. Die Rollenverteilung in der Simulation ist dargestellt in Bild 2.1. Während einer Simulation arbeitet Unreal Tournament als Server, im Folgenden *UT-Server* genannt, der Informationen aus der Simulation, über USARSim bereitstellt. Die Robbie-Software dient dem entsprechend als Client, weiter als *Robbie-Client* bezeichnet. Mit Hilfe der in Abschnitt 2.3 beschriebenen Schnittstelle, kann der Robbie-Client die Daten empfangen und Steuerkommandos absetzen. Zwischen Server und Client besteht eine *TCP/IP Verbindung*, über welche die Nachrichten verschickt werden.

2.2 USARSim

Der Kern von USARSim ist die Simulation der interaktiven Umgebung, der Roboter und deren Sensoren und Effecter [**?**], welche nachfolgend erläutert werden. Seit den Erfolgen in Computerspielen werden Gameengines zunehmend in der Wissenschaft eingesetzt [LJ02]. Wie in Kapitel 1 erwähnt, baut USARSim auf der Graphikengine *Unrealengine* der Firma *Epic Games* auf, welche führend im Genre der Ego-Shooter ist und von der Spieleindustrie oftmals genutzt wird [WLC+05]. Sie ist eine der neuesten und am meist verbreiteten Gameengines, mit dem Vorteil streng objektorientiert zu sein und neueste Graphiktechnik zu benutzen. Außerdem ermöglicht sie ein schnelles, hochqualitatives Szenenrendering, und eine effinziente Architektur um mehrere Roboter in einer Umgebung zu steuern. Ferner bietet die Engine eine eigene objekorientierte Programmiersprache *UnrealScript* um die Interaktion mit der Simulation auf eigene Bedürfnisse anzupassen.

2.2.1 Simulation der Umgebung

Es existieren, vom *National Institute of Standards and Technology* (NIST), bereitgestellte Umgebungen, welche zum Testen autonomer mobiler Roboter dienen [JMW+03]. Zur Simulation dieser sogenannten Arenen, gibt es entsprechende Maps, welche mit einem eigens mitgelieferten Leveleditor *UnrealEditor* (siehe Anhang A) erstellt werden können. Mit Hilfe eines Screenshots der gelben Arena (siehe Bild 2.2) lassen sich die wichtigsten Punkte für die Simulation der Umgebungen erläutern.

- Das *geometrische Modell* der Umgebung beinhaltet unbewegbare, statische Objekte wie Wände, Rampen, Treppen und den Boden.

- Weiterhin gibt es *Hindernisse*, wie Stühle, Rohre, und Steine, welche bei Kontakt mit dem Roboter bewegt werden, die aber auch das Verhalten des Roboters beeinflussen können.

- Das *Lichtmodell* (Positionen und Eigenschaften der Lichtquellen) dient zur Simulation der Lichtverhältnisse in der virtuellen Umgebung.

Bild 2.2: Gelbe Arena (Screenshot UT2004)

- Zusätzliche *Spezialeffekte* sind nötig um besondere Objekte, wie Spiegel und Glas, oder Elemente wie Feuer und Rauch (vergleiche mit Bild 1.2) zu simulieren.

- Die *Simulation der Opfer* geschieht durch spezielle Gamebots (siehe Anhang A), welche in der Lage sind, Geräusche von sich zu geben und sich zu bewegen.

2.2.2 Simulation der Roboter

Ursprüglich wurde USARSim mit Fokus auf Roboter mit Rädern entwickelt. Mittlerweile unterstützt USARSim verschiedenste Formen wie Unterwasserfahrzeuge, Roboter auf Beinen und Humanoide[1]. Darunter ist der *P2AT* zu finden, der im Robbieprojekt verwendete Typ, von *ActivMedia Robotics* (siehe Abschnitt 2.2.4). Bild 2.3 zeigt Robbie mit aktueller Sensorik, und sein virtuelles Gegenstück im Vergleich. Obwohl mit gleicher Sensorik

[1] In [**?**], Kapitel 10 werden alle unterstützten Robotertypen mit der dazugehörigen Konfiguration und Sensorausstattung beschrieben.

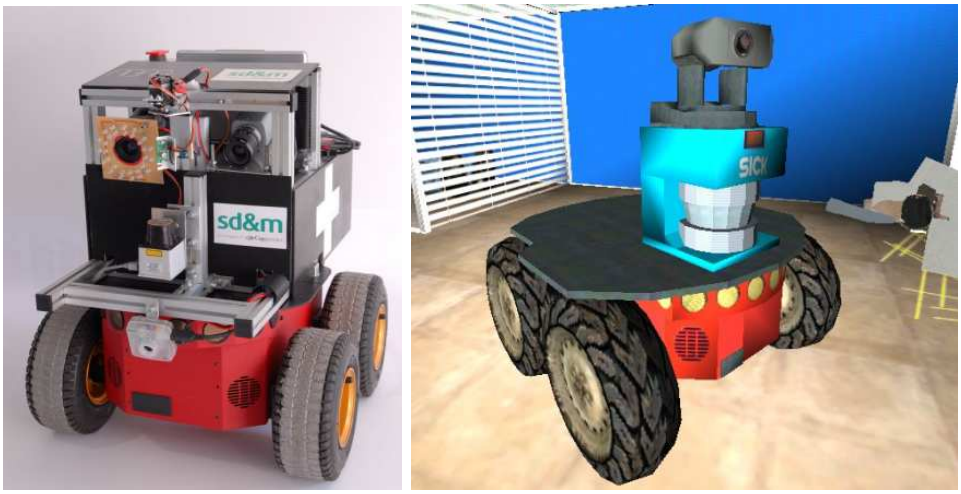

Bild 2.3: Realer Robbie (links) und Pendant (rechts)

ausgestattet, weichen die Bilder voneinander ab. So zeigt das 3D-Modell trotz aktiviertem Stereokamerasystem, nur eine Kamera. Auch die Darstellung des simulierten Laserscanners ist unterschiedlich, hier ist es ein Modell der Firma *SICK*. Das 3D-Modell in der Simulation besteht nur aus einem geometrischen Modell des Chassis und weiteren vorgefertigten Teilen, wie Räder und Sensoren, welche mit dem Chassis verbunden sind.

Entscheident für eine gute Simulation sind laut [CLW+07] ein akkurates Modell des Roboters, eine exakte Kinematik, akkurat modellierte Sensoren und die Interaktion mit der Umgebung. Die Berechnung des physikalischen Verhaltens, inklusive Kollisionserkennung, erledigt die *Karma rigid-body physics engine*[Mat] welche in die Unrealengine eingebettet ist. Sie ermöglicht die Simulation einer komplexen physikalischen Struktur des Roboters und seiner Umgebung.

2.2.3 Simulation der Sensoren

Sensoren ermöglichen dem Roboter das Sammeln von Informationen und bilden so eine Schnittstelle zwischen virtueller Umgebung und dem Robotermodell. Es gibt drei verschiedene Arten von Sensoren in USARSim [WLC+05]. Propriozeptive Sensoren, welche

den Status des Roboters erkennen können, Odometrie- und Inertialsensoren, zur Bestimmung der aktuellen Position und Orientierung, sowie Sensoren zur Wahrnehmung der Umgebung.

Die verschiedenen Sensortypen in USARSim sind als Klassen in UnrealScript implementiert und hierarchisch gegliedert, wie Bild 2.4 zeigt. Dabei sind alle Sensoren von einer Oberklasse abgeleitet.

Um eine gelungene Simulation zu bekommen, ist es notwendig, die verwendeten Sensoren an reale Verhältnisse anzupassen, da es nicht für jede Hardware eine entsprechende Sensorklasse gibt. Jeder Sensor lässt sich, durch Setzen der entsprechenden Attribute, speziell konfigurieren. So können beispielsweise der Öffnungswinkel und die Auflösung des 2D Rangescanners, an die Einstellungen eines realen Sensors angepasst werden. Auf diese Weise lassen sich aber auch leistungsstärkere Sensoren simulieren, indem man die maximale Reichweite vergrößert oder das Scanintervall verkürzt[2]. Die Sensoren werden durch entsprechende Einträge in der Konfigurationsdatei (siehe Abschnitt 2.2.4) eingestellt, sowie mit gewünschter Position und Blickrichtung auf dem Roboter angebracht.

2.2.4 Konfiguration des Roboters

Wie im vorhergehenden Abschnitt beschrieben, ist eine adäquate Konfiguration der Sensoren für eine realitätsnahe Simulation unerlässlich. Die Einstellungen und das Anbringen der Sensoren lassen sich mit einem Texteditor in der Initialisierungsdatei `USARBot.ini` vornehmen. Diese wird beim Starten des Servers (siehe Abschnitt 2.3.1) geladen und bestimmt das Setup während einer Simulation. Ist der Server erst gestartet, lassen sich die Konfigurationen nicht mehr ändern. Ausnahmen bilden Sensorparameter die durch spezielle Kommandos angesteuert werden können. In der benannten Datei erfolgt auch die Konfiguration des Roboters. So lassen sich Einstellungen, wie Motorgeschwindigkeit, Gewicht, oder Akkulaufzeit, und die Sensorausstattung des Roboters, mittels eines Texteditors bearbeiten.

Die reale verwendete Robotik-Plattform ist der Pioneer 3-AT von ActivMedia Robotics

[2] Änderungen an Auflösung oder Scanintervall, können zu Lasten der Performance gehen!

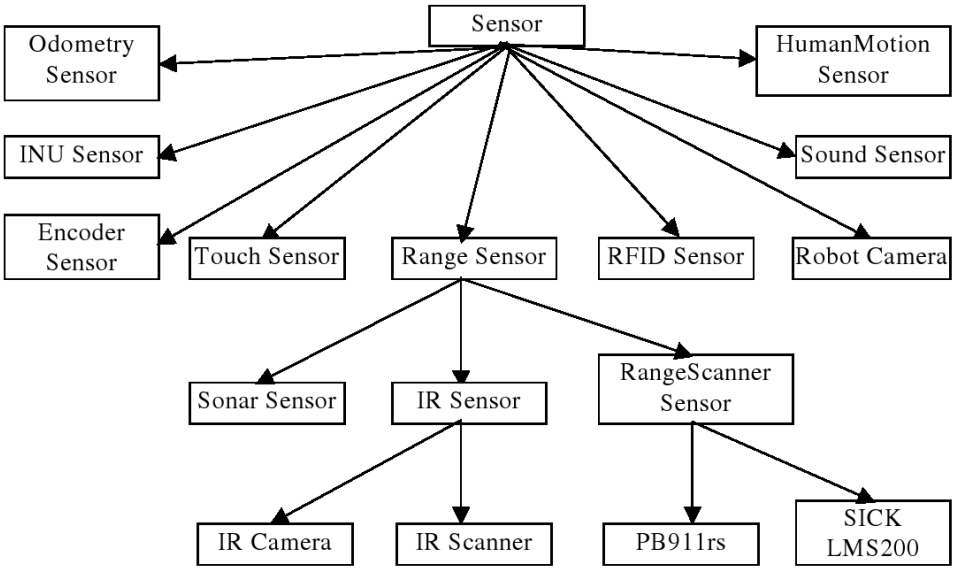

Bild 2.4: Sensorhierarchie in USARSim (aus [**?**])

mit 16 Sonarsensoren. Weitere Informationen können auf der Webseite der ActivMedia[3] gefunden werden. Hier kann auch die Spezifikation der Plattform eingesehen werden[4]. Der Roboter ist ausgestattet mit drei Firewire Farbkameras, wobei sich in der vorliegenden Arbeit nur auf das Stereokamerasystem, bestehend aus den Kameras Sony DFW X700 und Sony DFW X710 mit einer Auflösung von je 1024×768 Pixeln, konzentriert werden soll. Zusätzlich ist der Roboter mit dem Laserscanner *Hokuyo URG-04LX* ausgerüstet, der, von Servomotoren gesteuert, sowohl 2D als auch 3D Aufnahmen liefern kann [PDMP06].

Nachfolgend wird die Konfiguration des simulierten Roboters, ohne die, in dieser Arbeit vorzunehmenden Erweiterungen, beschrieben. Das in der Simulation verwendete Modell enstpricht dem Standartmodell des P2AT mit Stereokamerasystem[5].

Roboter: Der verwendete Robotertyp ist `stereoP2AT` mit einer Größe von 50×49×26 cm, einem Gewicht von 14 kg mit einer maximalen Zuladung von 40 kg. Der Motor läuft

[3]siehe `http://www.activrobots.com/ROBOTS/p2at.html`

[4]siehe `http://www.activrobots.com/ROBOTS/specs.html`

[5]Die Bilder der Stereokameras können bis dahin nur als *Spectator* (siehe Anhang A) im Spiel selbst, angesehen werden

mit einer Maximalgeschwindigkeit von 1.3 rad/sec. Zusätzlich ist der Roboter mit folgenden Sensoren ausgestattet.

Lasersensoren: Der verwendete Lasersensor `RangeScanner` hat eine Auflösung von 682 Punkten bei einem Öffnungswinkel von 240 Grad, was dem realen Laserscanner mit einer Auflösung von 0,352 Grad entspricht.

Sonarsensoren: Insgesamt 16 Sonarsensoren sind am Roboter angebracht, wovon, in speziellen Winkeln, 8 nach Vorn und 8 nach Hinten ausgerichtet sind. Die maximale Reichweite der Sonarsensoren beträgt 5 Meter.

Kameras: Über dem Laserscanner ist ein Stereokamerasystem, montiert auf einem Gestell zum Drehen und Kippen (diese Funktionen werden nicht genutzt), angebracht. Der Öffnungswinkel der Kameras kann auf Kommando angepasst werden.

Sonstige: Weiterhin verfügt der Roboter noch über interne Sensoren welche den aktuellen Status messen. Dazu gehört der Batteriestatus, die momentane Geschwindigkeit sowie Orientierung und die aktuelle Position, welche über Odometriesensoren gemessen wird.

Um den realen Roboter zu konfigurieren gibt es auch in der Robbie-Software Konfigurationsdateien, welche beim Starten eingelesen werden. Für die Benutzung in einer Simulation, gibt es zusätzliche Einstellungsmöglichkeiten, wie IP-Adresse und Port des Servers, Startpunkt des Roboters in der Map und mathematische Konstanten zur Umrechnung von *UnrealUnits* (siehe Anhang A). Diese Einstellungen werden in der Datei `simulation.dat` vorgenommen.

2.3 Schnittstelle zur Software

Um die Daten aus der Simulation nutzen zu können bedarf es einer Schnittstelle zwischen USARSim und der Robbie-Software. USARSim bietet dazu die Socket-API und *MOAST*[6], ein voll funktionsfähiges Framework zur Kontrolle von Roboterplattformen [CLW+06].

[6] Mobility Architecture Simulation and Tools

Die eigens für die Robbie-Software programmierte Schnittstelle, kommuniziert mit USARSim über die Socket-API. Im Folgenden wird die Schnittstelle genauer beschrieben, um die Vorraussetzungen und Bedingungen für Kapitel 3 aufzuzeigen. Dafür ist es sinnvoll zunächst einen Blick auf das Zusammenspiel zwischen USARSim und Robbie-Software zu werfen.

2.3.1 Systemkomponenten

Um die Verbindung und die Funktionsweise von USARSim und Robbie-Software zu verstehen ist es notwendig einen genaueren Blick auf die Systemkomponenten innerhalb der Simulation zu werfen, als das in Abschnitt 2.1 der Fall war. Bild 2.5 zeigt die einzelnen Systemkomponenten genauer und verdeutlicht den Datenfluss zwischen ihnen.

Auf der Serverseite (UT-Server), benutzt die Unrealengine die Maps und andere 3D Modelle zur Berechnung von Daten. So werden beispielsweise Sensordaten durch messen der Distanz, zwischen der Sensorposition und dem ersten Objekt auf der Sichtlinie, generiert. USARSim kann diese Messdaten abfragen und erstellt einzelne Nachrichten daraus, je nachdem welche Sensoren in der Konfiguration des Roboters aufgeführt sind. Für jedes Objekt in USARSim gibt es spezielle Nachrichten, welche es dem Benutzer erlauben, Steuerbefehle zu schicken, bestimmte Konfigurationen[7] zu ändern, den Status abzufragen oder Sensordaten zu empfangen [CLW+06]. Diese Nachrichten werden von USARSim in Form von Strings aus ASCII-Zeichen mit spezieller Syntax zur Verfügung gestellt. Sie werden im Folgenden als *USARSim-Messages* bezeichnet.

Auf der Clientseite (Robbie-Software) bilden das Device *USARSimCommunicator,* der Worker *USARSimMessageConverter* und das Modul *USARSimModule* die Schnittstelle zur Robbie-Software, welche die Nachrichtenübertragung sicherstellt und die Daten eine von Robbie verständliche Form umwandelt. Das weiterleiten der Daten geschieht hier mittels Funktionen. Eine genauere Beschreibung der Funktionsweise erfolgt im nächsten Abschnitt.

[7] Es lassen sich nur dafür vorgesehene Konfigurationen, wie z. B. der FOV der Kameras, ändern

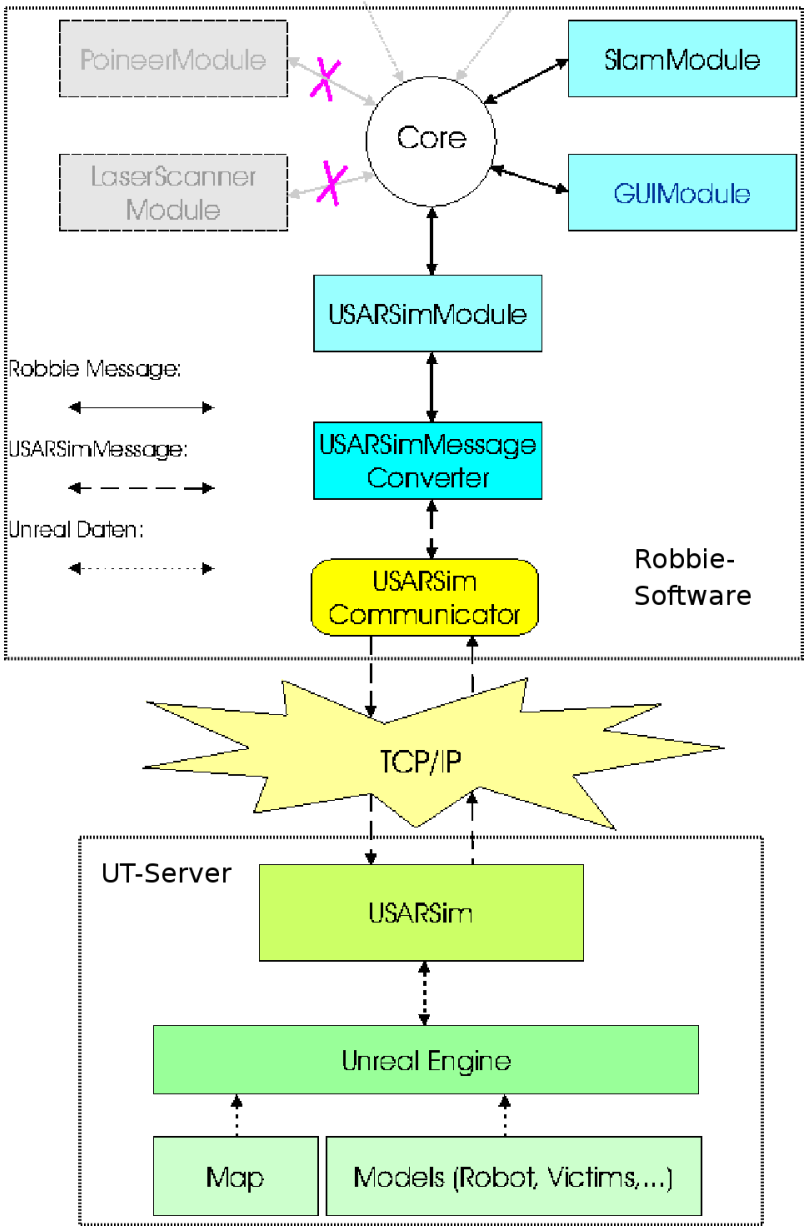

Bild 2.5: Systemkomponenten und Datenfluss während einer Simulation (aus [BH06])

Die Software von Robbie ist modular aufgebaut. Das bedeutet, dass diejenigen Module, die während des Betriebs mit dem realen Roboter die Anbindung und Steuerung der Hardware übernehmen, zum Beispiel *PioneerModule* und *LaserScannerModule* (siehe Bild 2.5), im Simulationsmodus deaktiviert sind. Module, welche die Daten nur verarbeiten, können aktiv beleiben, wie das *GUIModule* oder das *SLAMModule*.

Für die Kommunikation unter den einzelnen Modulen der Robbie-Software werden spezielle Message-Klassen genutzt, welche mit den Daten aus der Simulation initialisiert werden. Diese werden im Weiteren als *Robbie-Messages* bezeichnet. Sie werden über Queues an den Kern und die Module weitergeleitet.

2.3.2 Funktionsweise der Schnittstelle

Wie bereits beschrieben besteht die Schnittstelle aus drei Klassen. Bild 2.6 zeigt den Datenfluss innerhalb der Klassen und verdeutlicht die Funktionsweise der Schnittstelle.

Das Device USARSimCommunicator übernimmt ausschließlich die Datenübertragung mit Hilfe eines Sockets, der über WLAN mit der Socket-Api von USARSim verbunden ist. Ein- und ausgehende Daten liegen als USARSim-Messages vor. Durch die Funktionen `sendUSARSimMessage()` und `receiveUSARSimMessage()` werden die Daten gesendet und empfangen. Das Device kann nur vom Worker angesteuert werden.

Der Worker USARSimMessageConverter ist für die Bearbeitung und Umwandlung der Nachrichten zuständig. USARSim sendet seine Nachrichten aneinandergereiht in einem Charakterstream. Aufgrund der Datenübermittlung per TCP/IP an den Robbie-Client, wird dieser in einzelne Teilstrings zerlegt, was dazu führen kann, dass unvollständige Daten ankommen. Hat das Modul neue Daten angefordert, empfängt der Worker so lange Teilstrings vom Device bis mindestens eine komplette USARSim-Message vorliegt. Dies wird von der Funktion `updateUSARSimData()` sichergestellt. Dann parst der Worker die einzelnen USARSim-Messages und speichert die enthaltenen Daten, wobei für jeden Nachrichtentyp eine eigene `convert...Data()` Konvertierungsfunktion existiert. Die so gewonnenen Rohdaten werden danach in Robbie-Messages umgewandelt und weitergeleitet.

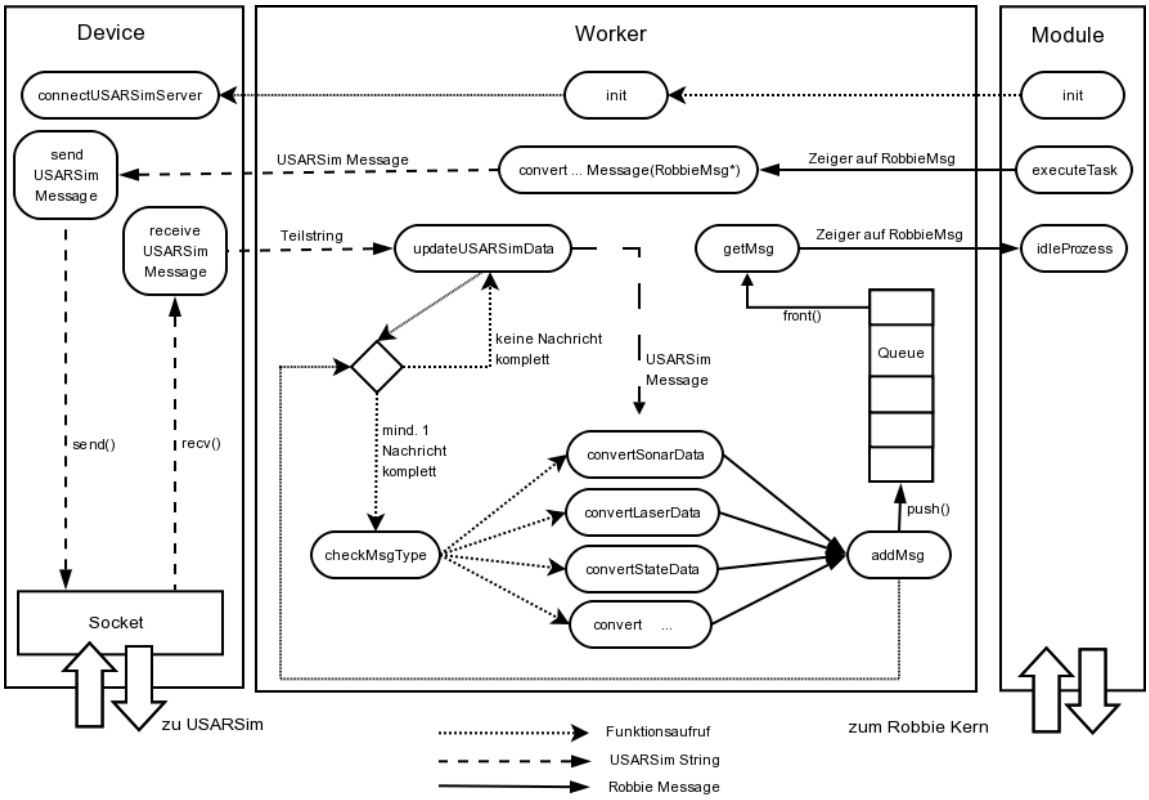

Bild 2.6: Datenfluss innerhalb der Schnittstelle

Steuerbefehle, die von der Robbie-Software kommen, werden umgekehrt bearbeitet. So werden die Robbie-Messages vom Worker in USARSim-Messages konvertiert. Hier gibt es für jede Robbie-Message eine eigene `convert...Message()` Funktion, welche einen, von USARSim verständlichen String, über das Device versendet.

Das Modul *USARSimModule* steuert die Datenübertragung per Robbie-Messages, indem es Steuerbefehle an den Worker weitergibt und immer neue Sensordaten aus der Simulation anfordert, um die Robbie-Software mit aktuellen Informationen zu versorgen. Dabei regelt das Modul die Verbindung zum Robbie-Kern durch Empfangen und Weiterleiten der Robbie-Messages. Es hat ansonsten nur Zugriff auf den Worker.

Kapitel 3

Erweiterung der simulierten Robotersensorik

Um die Robbie-Software weiterhin in der Simulation testen zu können, ist die Anpassung des virtuellen Roboters an die reale Hardware unumgänglich. In diesem Kapitel werden die einzelnen Erweiterungen vorgestellt, wobei zuerst auf den Lösungsansatz, im Sinne der Konfiguration und Benutzung unter USARSim, sowie möglichen Schwierigkeiten, und dann auf die Implementation in der Robbie-Software eingegangen wird.

3.1 Einbinden des 3D Scanners

Seit USARSim Version 2.0.6 existiert ein eigenständiger 3D Laserscanner, der den einfachen 2D Lasersensor im Rahmen der Programmierung um eine Kippbewegung erweitert. Dieser ist durch die Klasse `USARBot.RangeScanner3D` definiert. Wie jeder andere Sensor, ist er einfach durch einen Eintrag in der Konfigurationsdatei einzubinden und auf dem Roboter zu montieren. Anders als beim simulierten Roboter der Universität Osnabrück *Kurt3D* [AHL+06], dessen 3D Scanner auf der Geräteebene angesprochen wird, ist bei der Simulation von Robbie das Laserscannermodul deaktiviert (vergleiche Abschnitt 2.3) und das USARSimModul übernimmt entsprechend die Aufgaben.

3.1.1 Unterschiede zur Hardware

Zu beachten sind jedoch die unterschiedliche Funktionsweisen des realen und des simulierten Sensors. Der reale Sensor wird von einem Servomotor 180 Grad um die Sichtlinie rotiert, wodurch die Scanebene ebenfalls rotiert wird. Die Scanebene des simulierten Laserscanners hingegen, wird entsprechend einem vertikalen Öffnungswinkel gekippt. Dadurch entstehen verschiedene Interpretationen der gemessenen Raumpunkte, die in ihrem jeweiligen sphärischen Kordinatensystem nicht kompatibel sind. Aufgrund dieser Unterschiede ist es notwendig, die ermittelten Daten in ein einheitliches, karthesisches Koordinatensystem relativ zur Sensorposition zu überführen.

Auch bei der Performanz werden Unterschiede deutlich. Während der reale Sensor mit Zeiten von 10 Sekunden bei 68.300 Messpunkten [PDMP06] aufwarten kann, liegt der simulierte Sensor deutlich darunter. Die Zeit für einen 3D Scan in der Simulation ist abhängig von der Anzahl der akquirierten Messpunkte. Tabelle 3.1.1 zeigt verschiedene Messergebnisse bei einer Auflösung von je einem Grad.

Die erreichten Messungen sind abhängig von der Performanz des verwendeten Rechners und des Betriebsystems, auf dem der UT-Server läuft und können somit einige Sekunden, von den hier erreichten Zeiten abweichen. Für diese Messung wurde ein DELL INSPIRON 6400 mit 2.0 GHz verwendet, auf dem beide, UT-Server und Robbie-Client ausgeführt wurden.

Horizontaler FOV	Vertikaler FOV	Messpunkte	Sekunden
$120°$	$120°$	14.641	$14,473$
$180°$	$120°$	21.901	$33,57$
$240°$	$120°$	29.161	$60,452$
$240°$	$180°$	43.621	$142,783$

Tabelle 3.1: Messzeiten des simulierten 3D Laserscanners

3.1.2 Konfiguration und Steuerung des 3D Scanners

Da der Lasersensor nur von Zeit zu Zeit 3D Aufnahmen der Umgebung erstellen soll, und 21.901 Punkte ausreichend für eine Weiterverarbeitung für 6D-SLAM im *ICPModule* [**?**] sind, wurde als Standart eine Auflösung von 180×120 Messpunkten eingestellt. Damit ist der 3D-Laserscanner wie folgt konfiguriert:

```
[USARBot.RangeScanner3D]
MaxRange=4.000000
ScanInterval=0.0
VerticalFOV=2.0944
HorizontalFOV=3.1416
VerticalResolution=0.017453
HorizontalResolution=0.017453
```

Neben vertikalem, beziehungsweise horizontalem Öffnungswinkel, wird auch die Auflösung, berechnet als Anzahl der Scans pro FOV, in Radianten angegeben. Zusätzlich vermerkt `MaxRange` die maximale Reichweite des Scanners mit 4 Metern. `ScanInterval` gibt an in welchen Zeitabständen der Sensor eine Messung vornimmt. Auf Null geschaltet, befindet sich der Sensor im manuellen Modus und muss durch das nachstehende Kommando angesteuert werden.

```
SET {Type 3DRangeScanner} {Name Scanner2} {Opcode SCAN}
```

War die Messung erfolgreich, wird von USARSim eine Nachricht zur Bestätigung gesendet. Erst danach sendet USARSim die Sensorantwort, in Form der nacheinander gemessenen Entfernungswerten.

3.2 Implementation des 3D Scanners

Um den 3D Scanner zu implementieren werden nur die Klassen der Schnittstelle benötigt. Einzig der Worker USARSimMessageConverter bedarf einiger Änderungen, um mit den neuen USARSim-Nachrichten umgehen zu können.

3.2.1 Erweiterung des USARSimMessageConverter

Die tatsächliche Implementierung des Sensors in die Robbie-Software, ist mit nur zwei zusätzlichen Funktionen im Worker USARSimMessageConverter erledigt, welche im Folgenden erläutert werden. Ferner muss die Funktion `updateUSARSimData()` die neue USARSim-Message auslesen können.

- `void convertGetLaserscan3DMessage();`

  Nachdem das USARSimModul, die Robbie-Message `GetLaserscan3DM` vom Kern empfangen hat, ruft es diese Funktion des Workers auf. Der Worker übermittelt dann das oben benannte Kommando über das Device zu USARSim. Das Messen der Entfernungen geschieht USARSim intern.

- `void convert3DLaserMessage(string* sim3DLaserMessage);`

  Hat der Worker eine neue Sensornachricht *sim3DLaserMessage* des 3D Laserscanners vom Device empfangen, ruft er diese Funktion auf, wobei er die Nachricht als Zeiger auf einen String übergibt. Die Funktion parst den String und liest die Entfernungswerte heraus. Diese Rohdaten werden mit Hilfe eines weiteren Workers, dem *CoordinateConverter* in karthesische Koordinaten umgewandelt, wobei der Nullpunkt der Position des 3D Laserscanners entspricht.

3.2.2 Weiterleiten der Daten

Die so ermittelten Daten werden dem Modul in der bereits vorhandenen Robbie-Message `Laserscan3DDataM` zur Verfügung gestellt und können so zum Softwarekern weitergeleitet werden. Schließlich werden die 3D Punkte im Reiter *3DScans* der Robbie-GUI angezeigt. Auch andere Module, wie das ICPModule, bekommen die Robbie-Message und können mit dem 3D Scan arbeiten, ohne zu wissen ob die Daten realen oder virtuellen Ursprungs sind. Bild 3.1 zeigt einen Screenshot der Robbie-GUI mit simulierten 3D Laserdaten bei einer Auflösung von 180×120 Grad.

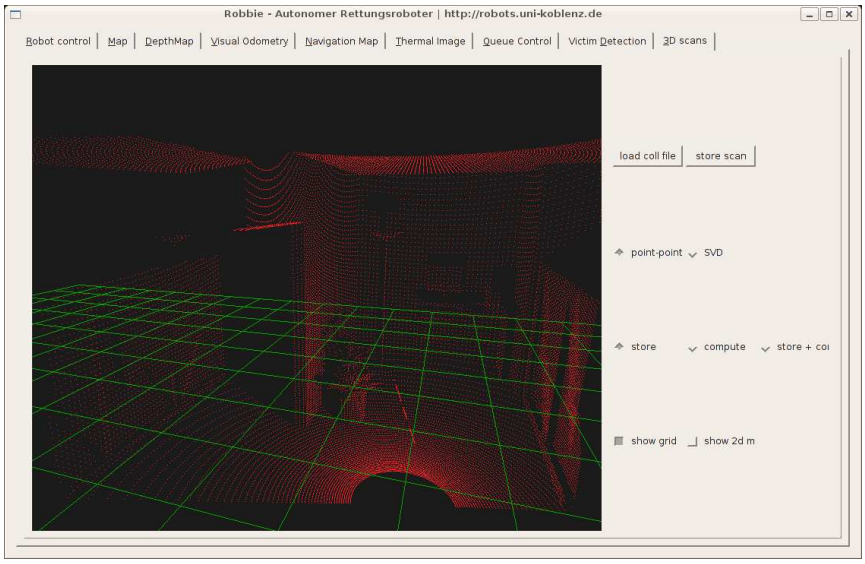

Bild 3.1: Anzeige der simulierten 3D Daten in der Robbie-GUI (Screenshot)

3.3 Einbinden der Kameras

Ein Herzstück von Robbie ist das Stereokamerasystem, mit dem er Opfer anhand der Hautfarbe erkennen, und visuelle Odometrie betreiben kann. Ferner bieten die Kameras, dem Operator eine optische Unterstützung bei der manuellen Steuerung. Aus diesen Gründen sind Kamerabilder auch in der Simulation wünschenswert.

3.3.1 Simulation und Akquirierung der Kamerabilder

Um Bilder aus der Simulation zu bekommen muss die virtuelle Umgebung mit allen Objekten gerendert werden. Dies erledigt wieder die Graphikengine von UT2004. Indem man das Spiel als Client, im Folgenden als *UT-Client*[1] bezeichnet, startet, kann man sich als Spectator mit einer frei beweglichen Kamera in die laufende Simulationsumgebung *spawnen* (siehe Anhang A). Diese Kamera lässt sich auf die Position der Roboterkamera setzen, um so die Umgebung aus der Sicht des Roboters wahrzunehmen.

[1] Im Gegensatz zu Robbie-Client und UT-Server

Dies ist zunächst die einfachste und schnellste Möglichkeit um Kamerabilder aus der Simulation zu visualisieren. Für eine weitere Verarbeitung der Bilder, müssen sie aber aufgenommen und zur Verfügung gestellt werden können. Nach [mis07b] gibt es bisher zwei Alternativen um die Bilder des UT-Clients zu akquirieren.

- Mit Hilfe des *Imageservers* und der Bibliothek *Hook.dll* von USARSim, lassen sich die Bilder direkt vom DirectX Graphikspeicher abgreifen und speichern [**?**]. Dies hat den Vorteil einer hohen Framerate und das, selbst bei verdeckten Fenstern, die Bilder noch aufgenommen werden können. Unter Linux lässt sich der Imageserver allerdings nicht einsetzen.

- Eine einfache Möglichkeit unter Linux ist das Aufnehmen der Bilder über Screenshots. Der Kameraserver der Universität Osnabrück verwendet dazu das Konsolenprogramm *xwd*, welches Screenshots von einzelnen Fenstern aufnimmt [**?**]. Nachteilig ist hier die schlechtere Performance und das Auftreten von Überlappenden Fenstern. Da die Robbie-Software aber nur unter Linux läuft, wird für diese Arbeit der Osnabrücker Kameraserver in leicht geänderter Form benutzt (siehe Abschnitt 3.4.1).

3.3.2 Umwandlung der Kamerabilder

Ein vom Kameraserver gelieferter Screenshot, enthält sowohl das rechte als auch das linke Kamerabild[2]. Wie in Bild 3.2 verdeutlicht wird, müssen die Bilder deshalb umgewandelt werden. Zunächst muss der Screenshot (links) in rechtes und linkes Bild getrennt werden. Aufgrund der Auflösung des UT-Clients sind die gerenderten Stereobilder entsprechend horizontal gestaucht (Mitte) und müssen auf die doppelte Breite skaliert werden (rechts). Die resultierenden Bilder besitzen wieder die Auflösung des UT-Client Fensters. Für eine bessere Darstellung in der Robbie-GUI können die gestauchten Bilder auch kleiner, zum Beispiel auf halbe Höhe, skaliert werden.

[2]Dies ist nicht der Fall wenn der Roboter P2AT statt stereoP2AT simuliert wird!

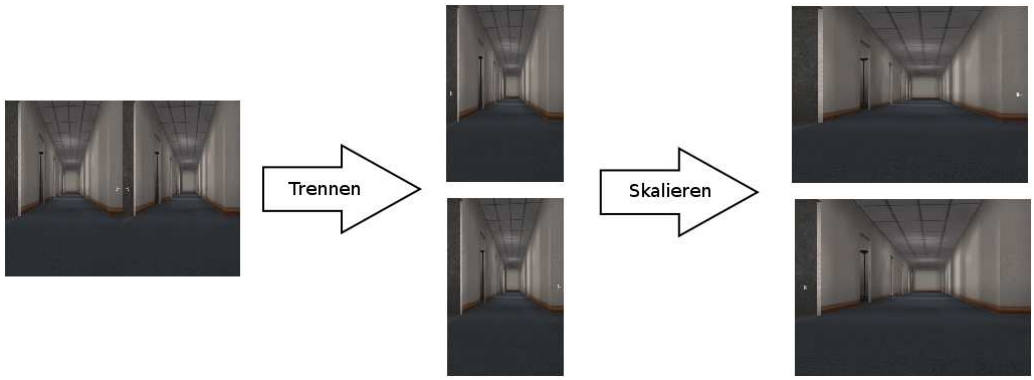

Bild 3.2: Umwandlung der Stereobilder aus der Simulation

3.3.3 Konfiguration und Steuerung der Kameras

Ausser der Position und der Blickrichtung, welche schon durch die Konfiguration des Roboters gegeben sind, ist die Konfiguration der Kamera auf die Angabe von minimalem und maximalem Zoomfaktor beschränkt. Der Zoomfaktor entspricht dabei dem Öffnungswinkel der Kamera in Radianten. Mit dem nachstehenden Befehl, lässt sich dieser während einer Simulation, durch Angabe des Radianten nach `Params`, innerhalb der definierten Grenzen anpassen.

```
SET {Type Camera} {Name Camera} {Opcode Zoom} {Params 1.5708}
```

Wie in Abschnitt 3.3.1 beschrieben, muss der Fokus der Kamera vom Spectator auf den Roboter gesetzt werden. Dies geschieht mit dem folgenden Befehl.

```
SET {Type Camera} {Robot Robbie7} {Name Camera} {Client ip}
```

Dabei ist `Robbie7` der Name des Roboters[3] in der Simulation und `ip` die Angabe der IP-Adresse des UT-Clients. Ferner sind IP-Adresse und Port in der Datei `simulation.dat` angegeben. Nach einer Simulation muss der Fokus wieder auf den Spectator übertragen werden. Dies geschieht mit dem selben Befehl ohne Angabe des Roboternamens.

```
SET {Type Camera} {Name Camera} {Client 141.26.69.9}
```

[3]Diese Angabe ist wichtig da auch mehrere Roboter in einer Umgebung simuliert werden können

3.4 Implementation der Kameras

Das Aufnehmen von Screenshots der simulierten Szenerie und die Verwendung der Bilder als Video-Feedback ist ein Ansatz, der realen Kameras nicht unähnlich ist [WLC+05]. Zuständig für die Steuerung der Kameras ist eine im Folgenden aufgeführte Funktion des USARSimMessageConverters, welche anhand einer übergebenen ID eine der, in Abschnitt 3.3.3 beschriebenen, Kommandos ausführt.

```
void setUSARSimCameraMessage(unsigned int id);
```

Nachdem die Kameras in der Simulation aktiviert sind, können die Bilder mit Hilfe des Kameraservers aufgenommen und verschickt werden. Um die gerenderten Stereobilder in verarbeitbarer Form an die Robbie-Software und insbesondere an die GUI anzubinden, bedarf es einer *Kameraschnittstelle*. Die Implementation des Kameraservers und der Kameraschnittstelle werden nachfolgend beschrieben.

3.4.1 Implementation des Kameraservers

Im Grunde dient der Kameraserver nur zum Aufnehmen und Verschicken der Kamerabilder. Nachdem ein Client akzeptiert wurde, wird dazu in einer Schleife das nachstehende Programmfragment ausgeführt, welches einen Prozess startet, der einen Screenshot erzeugt, ihn als Filestream zurückliefert und in `cam` zwischenspeichert.

```
cam = popen("xwd -name \"Unreal Tournament 2004\" -nobdrs -silent
            | xwdtopnm 2>/dev/null | pnmtojpeg","r");
```

Der gestartete Prozess `popen()` benutzt das Programm *xwd* um vom Fensterinhalt des UT-Clients einen Screenshot aufzunehmen. Per Umleitung an das Programm *xwdtopnm*, wird die Ausgabe ins PNM Format umgewandelt, wobei die Konsolenausgaben ins Nichts geführt werden. Weiter wird der Screenshot mit dem Programm *pnmtojpeg* ins JPEG Format umgewandelt und komprimiert. Die Variable `cam` enthält nun den fertig aufgenommenen und für die Weiterleitung komprimierten Screenshot.

Nachdem der Screenshot in ein Array aus ASCII-Zeichen eingelesen wurde, wird dieses Array über den Serversocket an den Client gesendet. Um die einzelenen, aufeinander folg-

nenden Bilder zu trennen wurde der Kameraserver so modifiziert, dass er nach einem Bild, ein Trennwort aus den Zeichen `CMP` für *complete* sendet und die Anzahl der gesendeten Bytes ausgibt. So kann der Empfänger sichergehen, dass ein komplettes Bild übertragen und vollständig empfangen wurde.

Mit der modifizierten Version des Kameraservers wird letztlich eine Bildübertragungsrate von einem Bild pro Sekunde, plus minus 100 ms erreicht.

3.4.2 Implementation der Kameraschnittstelle

Anhand Bild 3.3 lässt sich der Weg der Stereobilder vom UT-Client bis zur GUI verfolgen, wobei die Funktion der einzelnen Klassen der Kameraschnittstelle deutlich wird. Zuoberst ist der UT-Server aufgeführt, auf dem nun auch der UT-Client läuft. Wie bereits beschrieben nimmt der Kameraserver parallel dazu, die Bilder mittels Screenshots auf und sendet sie über eine TCP/IP Verbindung als Bytestream zur Robbiesoftware.

Ähnlich wie die Schnittstelle zur Anbindung von USARSim besteht auch die Kameraschnittstelle aus einem Device, einem Worker und einem Modul.

- Das Device *USARSimCameraClient* dient zur Verbindungsaufnahme und zur Kommunikation mit dem Kameraserver.
- Der Worker *USARSimImageConverter* wandelt die Stereobilder um (siehe Bild 3.2).
- Das Modul *USARSimImageReceiverModule* steuert die Kameraschnittstelle und leitet die Bilder weiter.

Nachdem das Device USARSimCameraClient ein vollständiges Stereobild des Typs JPEG vom Kameraserver empfangen hat, gibt es dieses in Form eines Arrays aus ASCII-Zeichen an das Modul weiter. Um die Komprimierung des JPEG Bildes rückgängig zu machen, bedient sich das Modul der statischen Funktion `decompress()`, der aussenstehenden Klasse `ImageCompressor`. Das entkomprimierte Bild wird in Form eines Zeigers auf ein PUMA `ColorImage` zurück liefert und als `cameraImage` gespeichert.

Zusammen mit zwei weiteren Zeigern des Typs `ColorImage` für die Ergebnisbilder, `leftImage` und `rightImage`, wird `cameraImage` an die Funtion `convert()` des

Workers übergeben. Der Worker trennt und skaliert das Stereobild, wobei `leftImage` und `rightImage` auf die resultierenden Bilder zeigen. Diese wiederum, werden, wie die Bilder der realen Kameras, mit der Robbie-Message `ImageM` über den Kern an die GUI weitergeleitet.

Verwendet man in der Simulation die Roboterklasse P2AT mit monokularer Kamera, entfällt die Umwandlung durch den Worker. In dem Fall wird das `cameraImage` direkt an die GUI weitergeleitet.

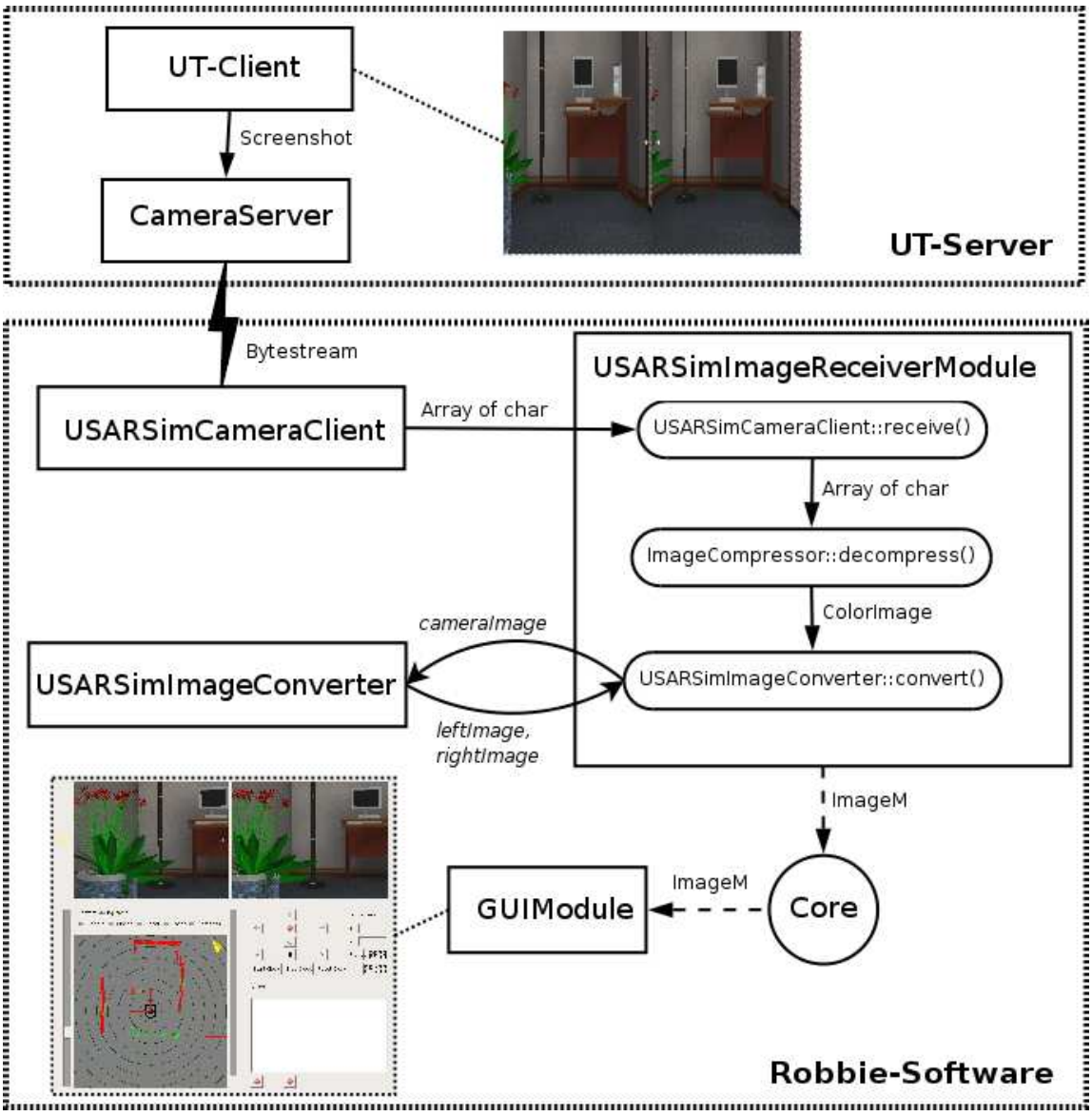

Bild 3.3: Datenfluss der simulierten Kamerabilder

3.5 Einbinden des Wärmesensors

Eine wesentliche Aufgabe von Robbie ist das Auffinden von verletzten Menschen in einem Katastrophengebiet. In erster Linie wird dafür der Wärmesensor benutzt, welcher die Körperwärme der Opfer detektiert. Mit Hilfe der Opfererkennung wird das Verhalten des Roboters, besonders beim autonomen Fahren, gesteuert um beispielsweise ein gefundenes Opfer anfahren zu können. Für die Simulation der Opfererkennung und des autnomen Fahrens ist somit ein simulierter Wärmesensor erforderlich.

3.5.1 Simulation von Wärmestrahlung

Bislang ist in USARSim noch keine Wärmestrahlung implementiert. Eine neue Methode zur Simulation von Gas [Wan06], insbesondere der Verteilung der Gaskonzentrationen und deren Detektion mit Gassensoren, zum Beipiel einem CO2-Sensor, soll in neueren Version von USARSim implementiert werden. Laut [mis07b] kann dieser Ansatz auch zur Simulation von Strahlung und Wärme verwendet werden.

Primär ist für Robbie nur die Wärmestrahlung von Opfern interessant. Als Alternative zur Wärmestrahlung werden deshalb in der vorliegenden Arbeit, die Positionen der Opfer zur Vortäuschung einer Wärmequelle verwendet. Andere Wärmequellen, wie Feuer oder Heizelemente, werden so allerdings nicht erkannt. Mit Hilfe eines *RFID-Sensors* werden die Positionen der Opfer in der virtuellen Umgebung bestimmt. RFID[4] bezeichnet die kabellose Technik der Identifikation und den Informationsaustausch über Funkwellen. Mit RFID-Technology lassen sich viele Aufgaben der Robotik, wie Exploration, Lokalisation oder Mapping, lösen [KPN06].

In der Simulation sowie in den realen Arenen, ist jedes Opfer mit einem *RFID-Tag*, einem kleinen Chip der die Identifikationsnummer des Opfers aussendet, ausgestattet. Die USARSim Sensorklasse *VictimRFID-Sensor* liefert Position, ID und medizinischen Status aus der Simulation. Im Folgenden wird beschrieben wie der VictimRFID-Sensor eingebunden und die Position eines Opfers genutzt wird um daraus ein Wärmebild zu generieren.

[4] Radio Frequency Identification

3.5.2 Einbinden des VictimRFID-Sensors

Der VictimRFID-Sensor wird durch Einbinden der Sensorklasse `RFIDVictSensor` aktiviert. Der Sensor sollte vorne über den Kameras angebracht sein [**?**]. Er wird durch folgende Einstellungen konfiguriert.

```
[USARBot.RFIDVictSensor]
VictMinRange=1.5
MaxRange=2.5
VictStatusRange=1.3
RFIDFOV=2.0944
```

Dabei erkennt der Sensor die Position eines Opfers ab 2,5 m, dem Wert von `MaxRange`. Das entspricht in Etwa der Reichweite des realen Wärmesensors. Ab 1,5 m, dem Wert von `VictMinRange`, erkennt der Sensor die ID des Opfers. Zusätzlich kann der Sensor den medizinischen Status eines Opfers ab 1,3 m erfassen. Bei einem Standartscan erfasst der reale Wärmesensor einen Bereich von 120 Grad, was der Wert `RFIDFOV` beim simulierten Sensor festlegt. Die simulierte Umgebungstemperatur ist frei wählbar und als `USARSIM_ENV_TEMP` in der Datei `simulation.dat` anzugeben.

Je näher der Roboter einem Opfer kommt, desto mehr Informationen werden empfangen. Eine vollständige Nachricht des VictimRFID-Sensors hat folgende Form.

```
SEN {Type VictRFID} {Name VictRFID} {ID Bobby_X_250}
    {Status Facial abrasions} {Location 1.23,2.34,3.45}
```

Daraus ist zu entnehmen, dass sich an der Position (1,23/2,34/3,45) ein Opfer Namens Bobby, mit Schürfwunden im Gesicht, befindet. Für die Simulation eines Wärmesensors ist jedoch nur die Position von Bedeutung. Angegeben wird die Position als 3D Punkt im Sensorkoordinatensystem, ein linkshändiges System mit der Sensorposition als Ursprung. Aus der bekannten Position lässt sich, wie in Abschnitt 3.6.2 beschrieben wird, ein Thermalbild generieren.

3.6 Implementation des Wärmesensors

Ziel der Implementation ist es, aus der Position eines Opfers, ein Thermalbild zu generieren und dieses an die Robbie-Software weiter zu leiten. Im Gegensatz zum realen Thermalsensor, der mit einer Robbie-Message zum Scannen aufgerufen wird, sendet der simulierte Sensor aktiv seine Daten. Die Lösung dieses Problems und die Erstellung eines Thermalbildes obliegt einem eigens dafür implementierten Worker, dem *USARSimThermalGenerator*.

3.6.1 Erweiterung des USARSimMessageConverter

Zunächst muss jedoch der Worker USARSimMessageConverter im Stande sein, die Sensornachrichten und Steuerbefehle zu bearbeiten. Ähnlich wie beim 3D Laserscanner werden dazu zwei neue Funktionen hinzugefügt (siehe Abschnitt 3.2.1).

- `void convertGetThermalImageMessage();`

  Nachdem das USARSimModul, die Robbie-Message `GetThermalImageM` vom Kern empfangen hat, ruft es diese Funktion des USARSimMessageConverters auf. Dieser erstellt ein Thermalbild indem er die Funktion `generateThermalImage()` des Workers USARSimThermalGenerator aufruft. Das generierte Bild wird dann als RobbieMessage `ThermalImageM`, über das USARSimModul an die Robbie-Software weitergeleitet.

- `void convertVictRFIDMessage(string* simVictRFIDMessage);`

  Hat der USARSimMessageConverter eine neue USARSim-Message *simVictRFIDMessage* des VictimRFID-Sensors vom Device empfangen, ruft er diese Funktion auf, wobei er die empfangene Nachricht als Zeiger auf einen String übergibt. Die Funktion parst den String und liest die Position, die ID und den Status eines Opfers heraus. Über die Funktion `setVictim(x,y,z,id,state)` bekommt der USARSimThermalGenerator alle Daten vermittelt um ein Thermalbild zu erstellen.

3.6.2 Berechnung des Thermalbildes

Alle mit der Funktion `setVictim()` empfangenen Opferdaten, insbesondere die Position, werden im USARSimThermalGenerator zur weiteren Verarbeitung gespeichert. Damit andere Module mit dem Thermalbild etwas anfangen können, muss ein detektiertes Opfer durch einen *Hotspot*[5] im Bild kenntlich gemacht werden. Dazu muss aus der 3D Position des Opfers ein 2D Punkt im Wärmebild ermittelt werden.

Die Berechnung des Hotspots geschieht im USARSimThermalGenerator in zwei Schritten:

- Zunächst wird die Opferposition, vorliegend in karthesischen Koordinaten p_x, p_y, p_z in sphärische Koordinaten α, β, r umgewandelt. Wie beim realen Sensor, wo die gemessenen Temperaturen anhand des horizontalen und vertikalen Winkels, im Thermalbild eingetragen werden, wird auch der Hotspot ins Bild eingefügt. Dabei bezeichnet α den horizontalen, und β den vertikalen Winkel. Verantwortlich für die Größe des Hotspots im Bild ist die Entfernung r zum Opfer.

- Aus den sphärischen Koordinaten werden dann die Pixelkoordinaten x, y und der Durchmesser d des Hotspots im Bild berechnet. Dabei ist x abhängig vom Öffnungswinkel γ des Wärmesensors, vom horizontalen Winkel α, von der Winkelposition α_1 bei der ein Scan endet und von der Breite des Bildes I_w gemessen in Pixel.

  $$x = \frac{(\alpha + \alpha_1)}{\gamma I_w} \tag{3.1}$$

  Hingegen ist y abhängig von dem aktuellen vertikalem Winkel β, den minmal anzunehmenden vertikalem Winkel β_{min} und der Höhe des Bildes I_h, gemessen in Pixel.

  $$y = \frac{\beta}{\beta_{min} I_h} \tag{3.2}$$

  Die Größe des Hotspots, gemessen in Pixel, ist umgekehrt proportional zu der Entfernung des Opfers. Sie ist begrenzt durch d_{min} und d_{max}. Die maximale Entfernung

[5] Ein Punkt in einem Thermalbild, der sich durch erhöte Temperatur deutlich von seiner Umgebung abgrenzt

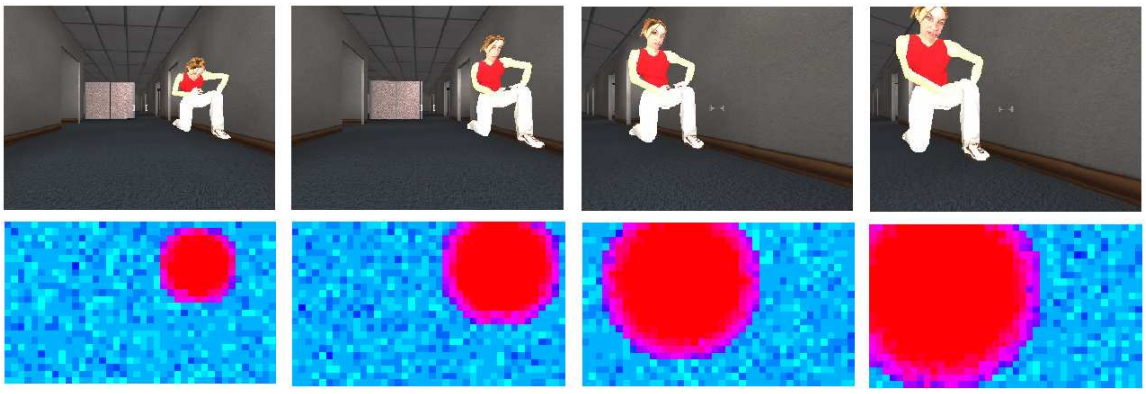

Bild 3.4: Bildserie aus Kamerabildern (oben) mit zuhgehörigen Wärmebildern (unten)

zu einem Opfer r_{max} ist durch den Wert `MaxRange` in der Konfiguration des Sensors gegeben.

$$d = d_{max} - \frac{r}{r_{max}(d_{max} - d_{min})} \tag{3.3}$$

Bild 3.4 zeigt eine Bildserie aus der Simulation, wobei die Kamerabilder mit den generierten Wärmebildern verglichen werden. Besonders gute Übereinstimmung finden dabei die horizontale Lage und die Größe des Hotspots, gegenüber der Position des Opfers im Kamerabild.

3.6.3 Visualisierung der Opferwärme

Für die weitere Verarbeitung, insbesondere die Segmentierung der Hotspots im simulierten Thermalbild ist eine reale Darstellung wichtig. Bild 3.5 zeigt ein reales Wärmebild (links) im Vergleich zu einem simulierten Wärmebild (rechts) nach der Vorverarbeitung, bestehend aus Einfärben und Glätten.

Ebenso wie das Wärmebild des realen Sensors, ist auch das simulierte Bild ein Grauwertbild, wobei ein Pixelwert der gemessenen Temperatur in Grad Celsius entspricht. Nachdem Größe und Position des Hotspots berechnet wurden, wird dieser beim Aufruf der Funktion `generateThermalImage()` in das Thermalbild eingezeichnet. Dazu zeichnet der Worker einen Kreis, der von der hellen Mitte ausgehend, nach Aussen dunkler wird. Der

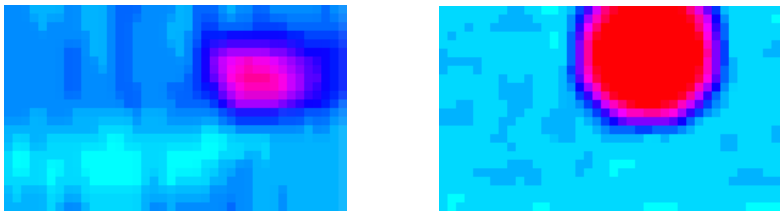

Bild 3.5: Vergleich zwischen realem (links) und simuliertem Wärmebild (rechts)

Hintergrund des Bildes entspricht dabei der Umgebungstemperatur. Sie wird aufgrund der fehlenden Wärme in der Simulation als 22 Grad Celsius angenommen.

Ein Wärmebild aus realer Umgebung weist viele thermische Störungen auf. Um die erstellten Wärmebilder realistischer zu gestalten, wird deshalb ein gaussverteiltes Rauschen über das Bild gelegt. Dazu werden die einzelnen Pixelwerte vor dem Zeichnen um einen Abweichungswert, welcher von der Funktion `randomGauss(v)` ermittelt wird, erhöht oder tiefer gesetzt. Die Abweichung `v` kann maximal 3 Grad Celsius zur gemessenen Temperatur betragen.

3.7 Simulation realitätsnaher Daten

Alle Messdaten die von den simulierten Sensoren zurückgeliefert werden, sind im Rahmen der Simulationsgenauigkeit perfekt. Das liegt daran, dass alle messbaren Objekte in der Simulation, wie Wände, Roboter, Opfer, RFID-Tags, usw. letztlich auf binären Daten beruhen. Messungen in der realen Welt hingegen, sind alles andere als perfekt. Hier gehen Daten verloren, durch die Unebenheiten von Oberflächen werden Laserstrahlen abgelenkt oder verfälscht, und etliche andere Störungen beeinflussen das Ergebnis.

Um realitätsnahe, verrauschte Daten zu erzeugen, bietet USARSim die Möglichkeit, die Messdaten mit einem Rauschen zu belegen. Bevor die gemessenen Daten abgeschickt werden, werden sie USARSim-intern mit einem wählbaren Störfaktor verrechnet. Damit hat der Benutzer die Möglichkeit seine Algorithmen mit perfekten Daten zu versorgen oder die Robustheit seiner Software gegen Sensorrauschen zu testen [CLW+07]. Das Definieren des Störfaktors und die Verrechnung mit den Daten wird im Folgenden erläutert.

3.7.1 Erzeugung des Sensorrauschens

Jeder USARSim Sensor zur Bestimmung von Entfernungen kann durch Angabe des Störfaktors `Noise` mit einem Rauschen belegt werden, wie die nachstehende Kofiguration eines Sonarsensors mit 0,5% Rauschen zeigt.

```
[USARBot.SonarSensor]
Weight=0.4
MaxRange=5.0
BeamAngle=0.349
Noise=0.005
```

Der Störfaktor ist bei der Konfiguration (siehe Abschnitt 2.2.4) eines Sensors als Prozentwert zu definieren. Somit ist das resultierende Rauschen Abhängig von der jeweils gemessenen Entfernung.

Um verrauschte Daten D zu bekommen, wird der Störfaktor f intern von USARSim, in einen perfekten Datensatz D_0, gemäß Gleichung 3.4 [**?**] eingerechnet. Die Funktion $\mathrm{r}(f)$ liefert dabei einen gaussverteilten Zufallswert im Bereich $[-f, ..., f]$.

$$D = D_0 + \mathrm{r}(f)D_0 \tag{3.4}$$

Ein perfekter, simulierter 2D Laserscan mit 682 Einzelmessungen ist in Bild 3.6 zu sehen. Dabei ist die Nummer der Einzelmessung auf der X-Achse, und die entprechend gemessene Entfernung auf der Y-Achse eingetragen. Derselbe Laserscan mit einem, dem realen Sensor entsprechenden, Störfaktor von 1% ist in Bild 3.7 zu sehen.

Deutlich wird der Unterschied, dargestellt in Bild 3.8, wenn man nur die Abweichungen (blau) von den Originaldaten (rot) aus Bild 3.6 betrachtet. Hier sind ebenfalls die Einzelmessungen auf der X-Achse und die gemessene Abweichung in Metern auf der Y-Achse angegeben. Die Abweichung verhält sich proportional zur gemessenen Entfernung.

Die Verteilung des Rauschens wird in Bild 3.9 deutlich. Hier ist horizontal die Stärke des Rauschens in Prozent, und vertikal die Häufigkeit mit welcher der Rauschfaktor auftritt, angegeben. Hierfür wurden 3500 Rauschwerte gemessenen und bis auf die dritte Nachkommastelle gerundet.

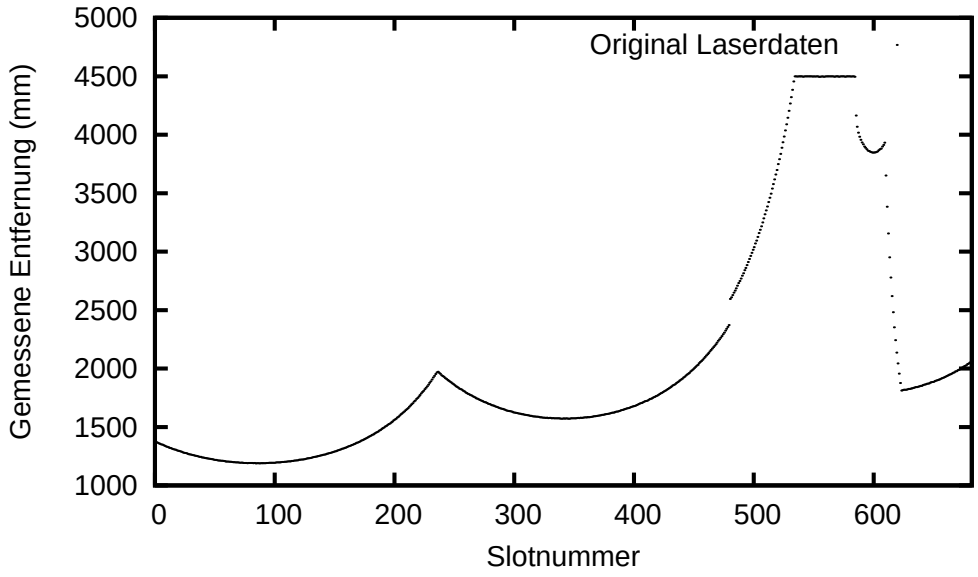

Bild 3.6: Simulierter 2D Laserscan

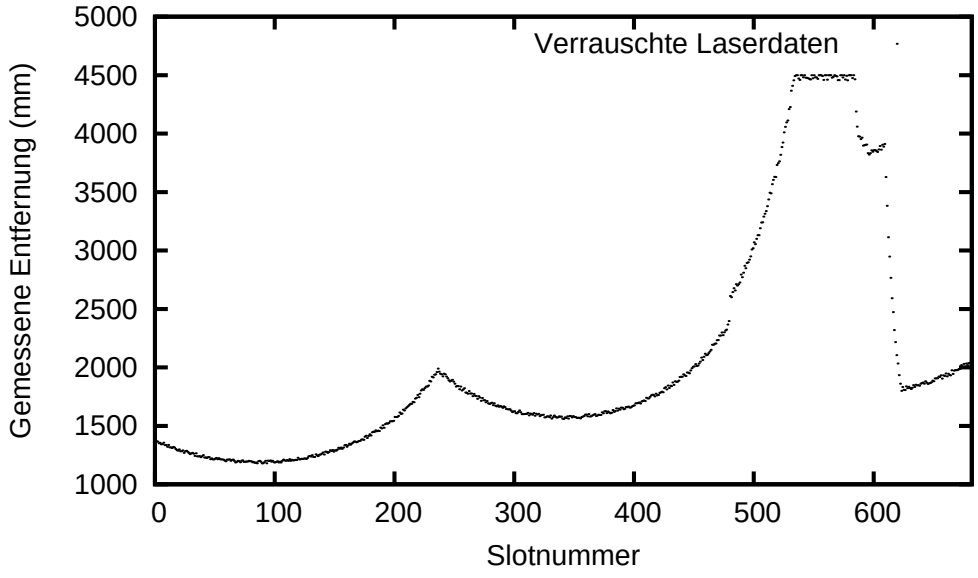

Bild 3.7: Simulierter 2D Laserscan mit Sensorrauschen

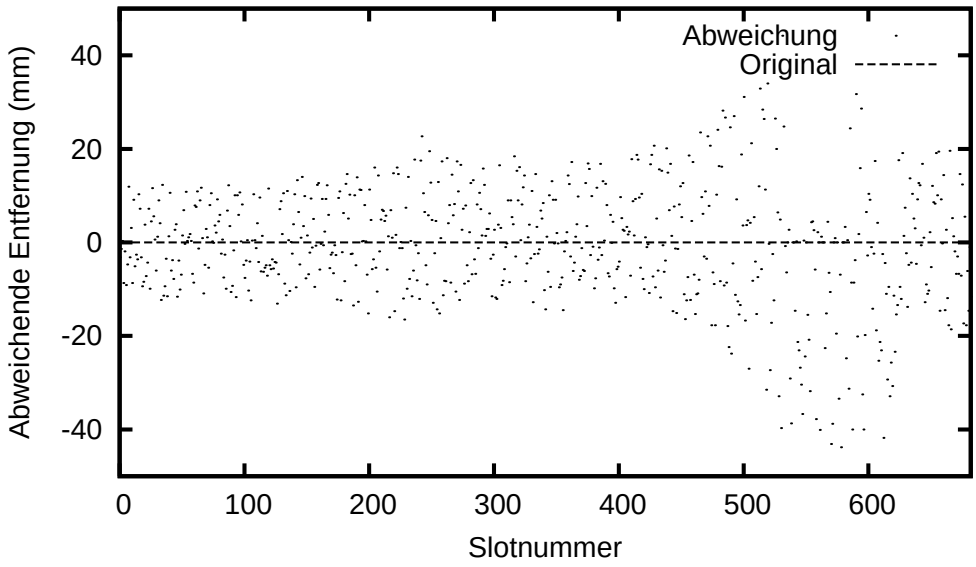

Bild 3.8: Abweichungen durch Sensorrauschen

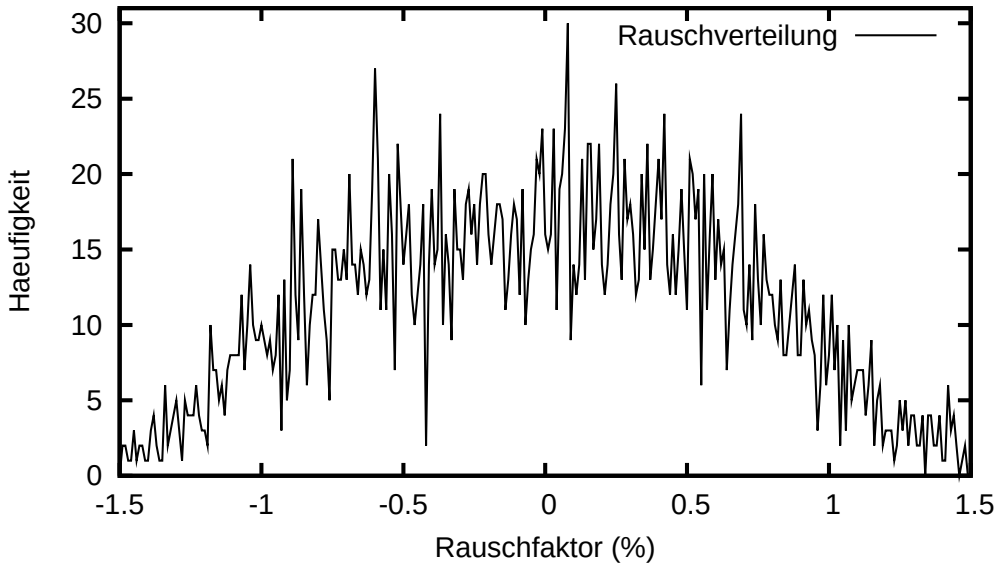

Bild 3.9: Abweichungen durch Sensorrauschen

Kapitel 4

Experimente und Evaluation

Nach [CLW+06] ist die Nützlichkeit einer Simulation, als Werkzeug zum Entwickeln und Testen von Software, stark abhängig vom Grad der Validierung. Eine Simulation besteht nur aus emulierten Daten, ist also nur ein Abbild der Realität. Inwiefern eine Simuation der Realität entspricht, ist nur durch direktes Vergleichen feststellbar. In diesem Kapitel erfolgt der Vergleich zwischen simuliertem und realem Roboter.

4.1 Versuchsaufbau

In [WLC+05] sind die Ergebnisse einer Evaluierung des Hokuyo Laserscanners vorgestellt. Hierfür wurden 2D Datensätze in einer realen Arena und im simulierten Pendant aufgenommen und verglichen. Für eine Evaluierung der visuellen Perzeption in [CLW+06] sollten Probanten in realer und simulierter Umgebung einen Gelben Kegel anfahren. Auch hier wurde eine Arena mit entsprechender Map verwendet. Die von USARSim bereitgestellten Maps entsprechen exakt den realen, von NIST entworfenen Arenas [JMW+03], was einen genauen Vergleich der Daten ermöglicht.

Maßgebend für eine Vergleichbarkeit der Ergebnisse ist die Vergleichbarkeit der Umgebung. Neben der Wahl der Testumgebung ist also zu beachten, das eine vergleichbare Map existiert.

Bild 4.1: Vergleich zwischen AGAS-Flur (links) und simuliertem Korridor (rechts)

Zu den Hauptaufgaben von Robbie gehört das autonome Explorieren der Umgebung und das Auffinden von Opfern in dieser. Als Testumgebung kommen, neben den engen und verwinkelten Arenen, besonders solche mit langen Korridoren und vielen Räumen in Frage, wie sie in öffentlichen Gebäuden oder Bürokomplexen zu finden sind.

Für die Evaluation der Simulation von Robbie eignet sich der Hauptkorridor der Arbeitsgruppe Aktives Sehen (*AGAS*) im Universitätsgebäude. Zur Zeit existiert aber noch keine Map des AGAS-Flures. Vergleichbar bezüglich Einteilung und Größe, eignet sich jedoch der Flur des Bürogebäudes aus Bild 1.2 als simuliertes Gegenstück. Bild 4.1 zeigt den AGAS-Flur und den simulierten Korridor im Vergleich.

4.1.1 Anforderungen

Die Robbie-Software kann gleichermaßen mit realen sowie simulierten Daten umgehen. Alle Module, welche für die Weiterverarbeitung von Sensordaten zuständig sind, arbeiten mit Sensordaten, unabhängig ob sie real oder simuliert sind. Für die Umstellung von Simulation auf reale Anwendung, ist somit keine Änderung des Codes notwendig. Allein die Form der simulierten Sensordaten muss den realen Daten entsprechen.

Zunächst werden die Erweiterungen, welche den Hauptteil der vorliegenden Arbeit ausmachen, getestet, um festzustellen ob die simulierten Daten den Anforderungen genügen.

Danach folgt ein Vergleich des Verhaltens bei komplexen, koordinierten Aufgaben wie autonomes Fahren und gleichzeitige Opfersuche.

4.2 Versuchsdurchführung

Nachdem eine geeignete Testumgebung gefunden wurde, gibt dieser Abschnitt einen Überblick über die einzelnen Test und wie sie durchzuführen sind.

4.2.1 Test der Erweiterungen

Durch Einzeltests der Erweiterungen soll festgestellt werden, ob andere Module mit den gelieferten Sensordaten arbeiten können. Dazu werden bestimmte Module mit den simulierten Daten versorgt und das Ergebnis mit realen Ergebnissen verglichen. Im Folgenden sind die durchzuführenden Tests genauer beschrieben.

Test des 3D Scanners: Mit Hilfe des ICP-Modules sollen mehrere 3D Scans aus dem virtuellen Bürogebäude, zu einem einzigen 3D Datensatz zusammengefügt und mit Datensätzen aus dem realen Flur verglichen werden.

Test der Stereokameras: In [CSN+06] sind verschiedene Ergebnisse zu Tests der visuellen Perzeption zu finden, insbesondere der *Feature extraction* mittels verschiedener Algorithmen. Für Tests von Stereobildern dient ein anderer Ansatz. Ursprünglich ist Robbie in der Lage mit den Stereokameras eine optische Tiefenkarte zu erstellen. Es soll festgestellt werden inwieweit dies mit simulierten Bildern möglich ist

Test des Wärmesensors: Zunächst soll mit Hilfe des *VictimDetectionModules* Opferwärme in den Thermalbildern segmentiert werden. Die Überprüfung der weiteren Verarbeitung erfolgt beim Test der autonomen Opfersuche.

Tests realitätsnaher Daten: Das Sensorrauschen wird bei allen Tests aktiviert sein.

4.2.2 Test der Autonomie

Als Test der gesamten Robbie-Software eignet sich das autonome Fahren und die autonome Opfererkennung am besten, da hier fast alle Sensoren gleichzeitig involviert sind. So lässt sich das Zusammenspiel von Lasersensoren, Sonarsensoren, Odometriesensoren, Kameras und Wärmesensor testen.

Der Roboter wird simuliert und real am Anfang der, in Abschnitt 4.1 vorgestellten, Korridore positioniert. Danach wird die autonome Exploration inklusive Opfersuche aktiviert. Die dabei erstellten Karten und eingetragene Positionen der gefundenen Opfer werden ausgewertet.

4.3 Ergebnisse

In diesem Abschnitt erfolgt die Vorstellung der Testergebnisse und die Auswertung der Daten. Dabei wird insbesondere auf die Unterschiede, zwischen Ergebnissen aus Simulation und realer Anwendung, und deren Entstehung eingegangen. Zunächst wird auf die Einzeltests eingegangen, dann folgt der Autonomie Test.

4.3.1 3D Karten

Sowohl in der Simulation als auch im realen Betrieb, lassen sich mit dem ICP-Module mehrere 3D-Scans zu einem Gesamtscan zusammenfügen. Aufgrund der besseren Odometriedaten in der Simulation, geschieht das Zusammenfügen hier schneller als mit realen Daten. Selbst ein aufgerechneter Fehlerwert von 3% auf Position und Orientierung führt nur zu kleinen Abweichungen. Tests und Ergebnisse zur Fehleranfälligkeit sind in [**?**] zu finden.

Bild 4.2 zeigt zusammengefügte, mit dem realen, rotierenden Lasersensor aufgenommene, 3D Scans des AGAS Flures. Links im Bild war die Rotationsachse nach vorne gestellt, im rechten Bild zeigte sie nach oben. Letzteres zeigt durch die schwarzen Kreise am Boden deutlich die Positionen, von denen die einzelnen 3D Scans aufgenommen wurden.

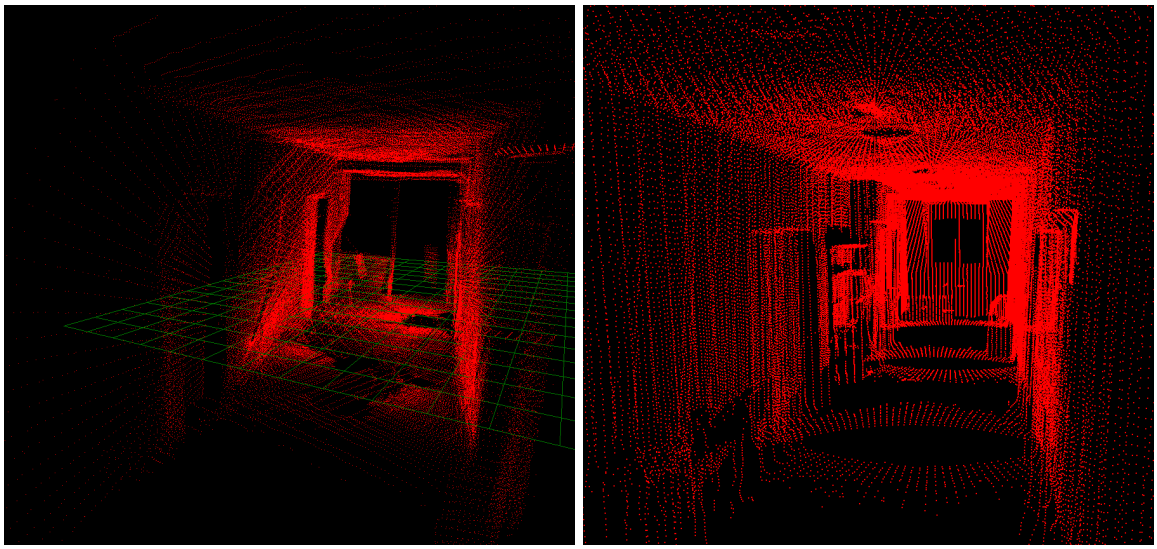

Bild 4.2: 3D Scans des AGAS-Flures aus realen Einzelscans (aus [**?**])

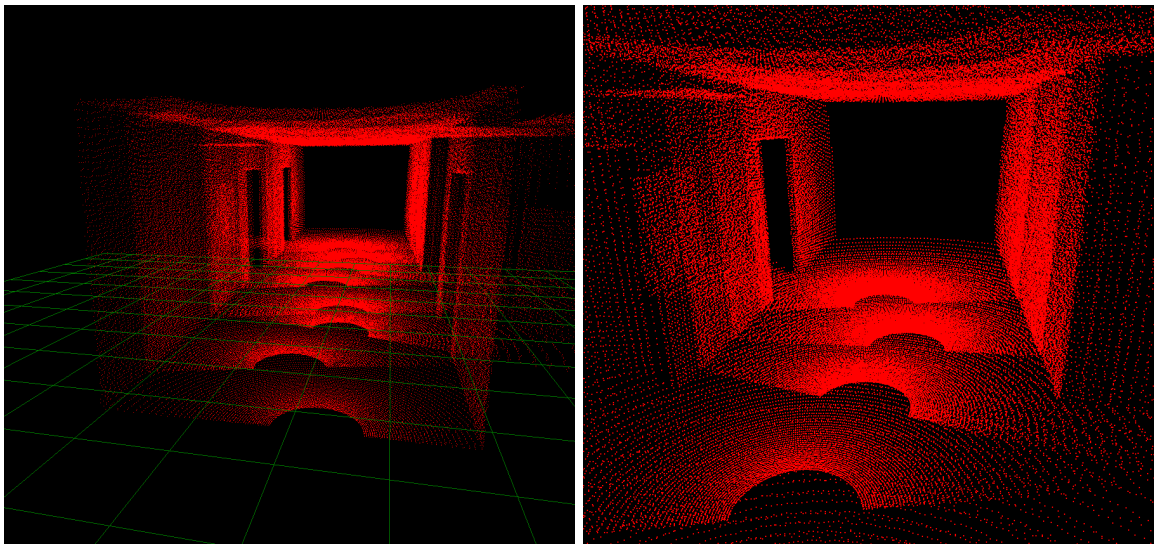

Bild 4.3: 3D Scans des simulierten Korridors aus der Map DM-compWorldDay1

Bei den 3D Scans aus der Simulation, in Bild4.3, wird deutlich, dass zwischen den einzeln aufgenommenen Scans wesentlich kleinere Abstände sind. Da in der Simulation eine niedrigere Auflösung des Laserscanners eingestellt ist (siehe Abschnitt 3.1), müssen die Abstände kleiner sein um noch genügend korrespondierende Punkte zu finden.

Ferner fällt die Genauigkeit der simulierten Scans auf. Trotz Sensorrauschens passen die einzelnen Scans gut aneinander, was auf die genauen Odometriedaten aus der Simulation zurückzuführen ist.

4.3.2 Tiefenschätzung

Wie in Abschnitt 4.2 erwähnt, ist Robbie anhand der Stereobilder in der Lage, Tiefenkarten zu erstellen, um zum Beispiel visuelle Odometrie betreiben zu können. Dazu müssen mittels einer Kalibierung die intrinsischen Kameraparameter festgestellt werden, um beispielsweise durch die Linse verursachte Verzerrungen auszugleichen.

Bild 4.4 zeigt ein Beispiel aus dem Projektpraktikum Robbie 7 mit realen Kameras. Links ist die Szene aus dem AGAS Flur und rechts das, aus den entsprechenden Stereobildern erstellte Tiefenbild zu sehen. Dabei sind hellere Punkte im Bild näher gelegen als dunklere.

Aufgrund der geringen Bildrate in der Simulation ist visuelle Odometrie nicht möglich. Auch eine Kalibierung der simulierten Kameras ist, mangels fehlender Kalibriermuster nicht möglich. Anwendungen welche keine hohe Bildrate benötigen sind aber verwendbar. So können beispielsweise Opfer über die Hautfarbe in den Kamerabildern, durch entsprechendes Training mit der Erkennungssoftware, detektiert werden.

Wie das Beispiel aus [Dec06][1], dargestellt in Bild 4.5, zeigt, ist auch die optische Tiefenschätzung durchführbar. Da die simulierten Bilder einer exakten, perspektivischen Projektion ohne Verzerrung entsprechen, ist eine Kalibrierung nicht notwendig. Lediglich der Fokus der Kameras ist durch einen entsprechenden Wert zu schätzen.

[1] Die in [Dec06] verwendeten Kamerabilder sind als Screenshot entstanden. Die Tiefenkarte wurde nicht mit Modulen aus der Robbie-Software erstellt.

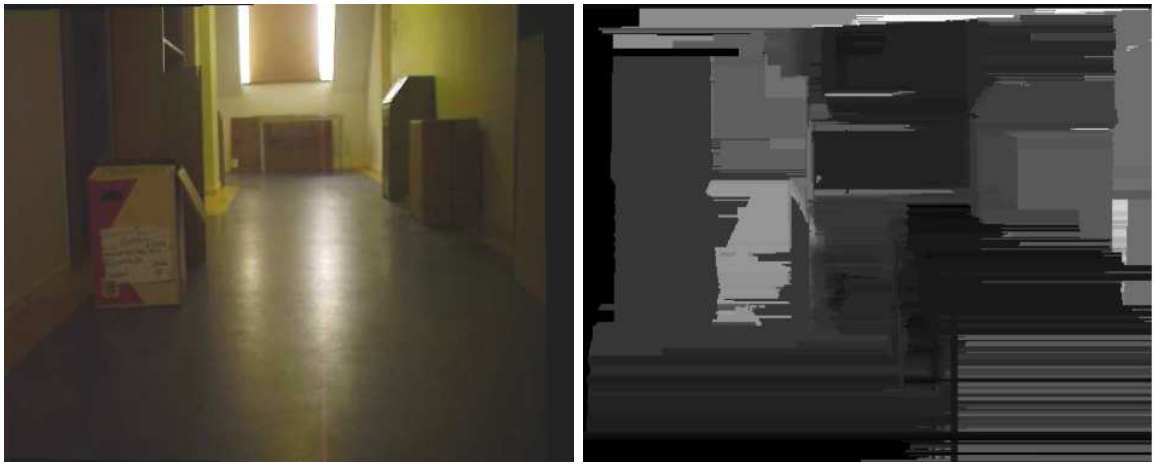

Bild 4.4: Reale Szene im AGAS-Flur und zugehörige Tiefenkarte (Robbie 7)

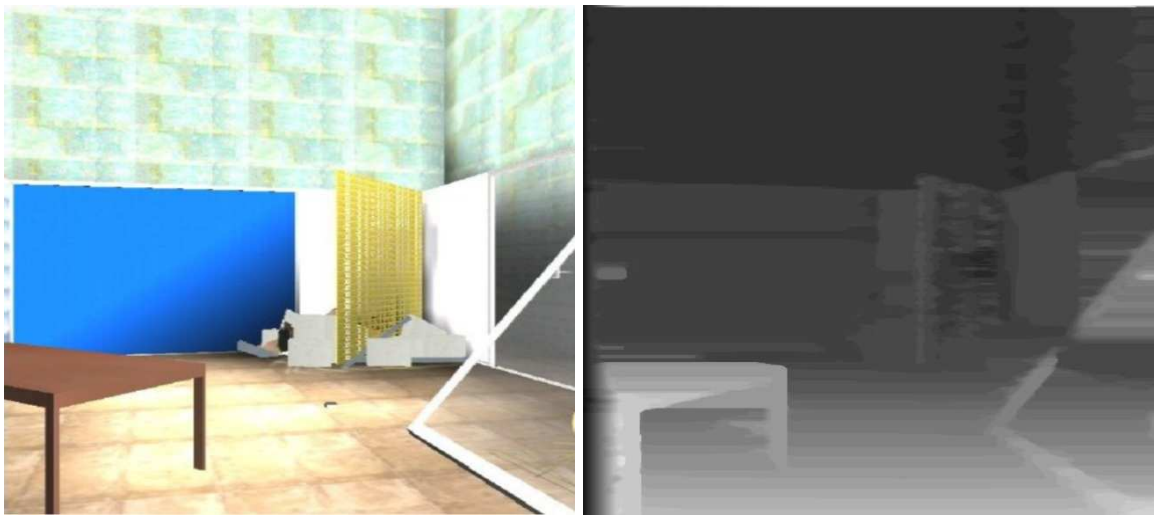

Bild 4.5: Simulierte Szene in der gelben Arena und zugehörige Tiefenkarte (aus [Dec06])

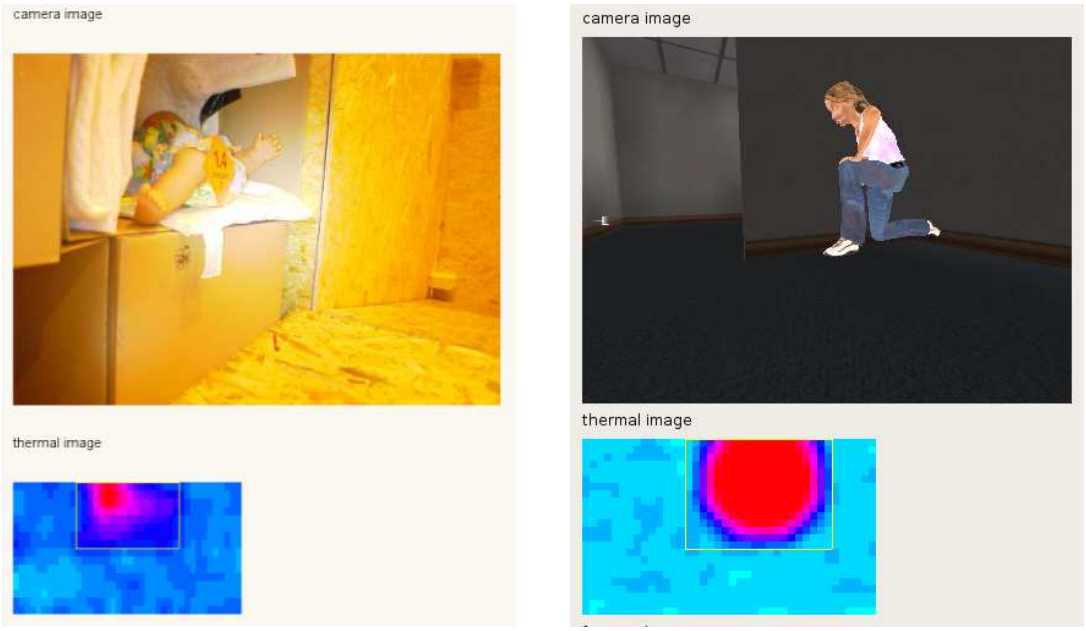

Bild 4.6: Real erkanntes Opfer (links) und erkanntes Opfer in der Simulation (rechts)

4.3.3 Opfererkennung

Die wichtigste Aufgabe des Wärmesensors ist das Erfassen von Opfern. Die Erkennung der Opfer in den Thermalbildern erfolgt in der weiteren Verarbeitung. Dafür müssen die Wärmebilder in der Simulation sowie im realen Betrieb in geeignetem Format vorliegen.

Um aus den, in Abschnitt 3.6.3 vorgestellten Thermalbildern die Position eines Opfers in der Karte zu ermitteln, muss die Opferwärme im Bild zunächst segmentiert werden. Diese Aufgabe übernimmt das VictimDetectionModule. In der GUI wird dabei das Kamerabild des Opfers und der zugehörige Thermalscan angezeigt, wobei eine Boundingbox um die segmentierte Opferwärme gelegt wird.

Jeweils einen Ausschnitt der GUI zeigt Bild 4.6. Links ist ein real gefundenes Opfer, aus der Arena bei den *RoboCup German Open 2007*, rechts ein Opfer aus der Simulation zu sehen. Oben sind die, zu den Thermalscans korrespondierenden, Kamerabilder zu sehen. Unten kennzeichnet ein gelber Rahmen die segmentierte Opferwärme im Thermalbild.

4.3.4 Autonome Exploration und Opfersuche

Abschließend zeigt ein Vergleich des Autonomieverhaltens in der Simulation und im realen Betrieb, wie real die Simulation bezüglich komlexer Aufgaben arbeitet. Für autonome Exploration und Opfersuche arbeiten viele Sensoren und Module der Robbiesoftware zusammen. So werden 2D Laserscans und Odometriedaten für die Kartenerstellung und Wegeplanung benötigt. Kamera- und Wärmebilder dienen der Opfererkennung und Opferverifikation. Da dies die Hauptaufgabe von Robbie ist, muss auch die Simulation entsprechendes leisten können.

Opfersuche: Wurde ein Opfer gefunden, verifiziert und vom Operator akzeptiert, wird seine Position durch einen nummerierten, roten Kreis auf der Karte markiert. Bild 4.7 zeigt jeweils ein autonom gefundenes Opfer in der Simulation (rechts) und real (links). Der grüne Pfeil bezeichnet dabei die Startposition des Roboters. In beiden Fällen besteht eine gute Übereinstimmung zwischen eingetragener und vom Laserscanner erkannter Position.

Exploration: Um Opfer zu finden exploriert der Roboter seine Umgebung und merkt sich dabei den bereits abgesuchten Bereich, wobei er zunächst dem einfachsten Weg folgt. Der so gefahrene Explorationspfad ist in Bild 4.8, in die entstandene Karte rot eingezeichnet. Links im Bild ist der real gefahrene Weg und rechts der simuliert gefahrene Weg zu sehen.Der Kreis markiert dabei die Startposition und das rote Dreieck die aktuelle Position des Roboters in der Karte. Auch hier zeigt der simulierte Roboter das selbe Verhalten wie der Reale. Die Zacken im Pfad entstehen, wenn der Roboter seine Position anpassen muss, um einen übersehenen Bereich nachträglich mit dem Wärmesensor auf Opfer zu testen.

Obwohl die Laser– und Odometriesensoren aus der Simulation wesentlich exaktere Daten liefern, weisen real erstellte Karten aufgrund der besseren Performanz eine höhere Genauigkeit auf. Laser- und Odometriedaten kommen in der Simulation weniger oft und nicht gleichzeitig, was einen höhere zeitliche Unschärfe hervorruft, als sie im realen Betrieb vorkommt.

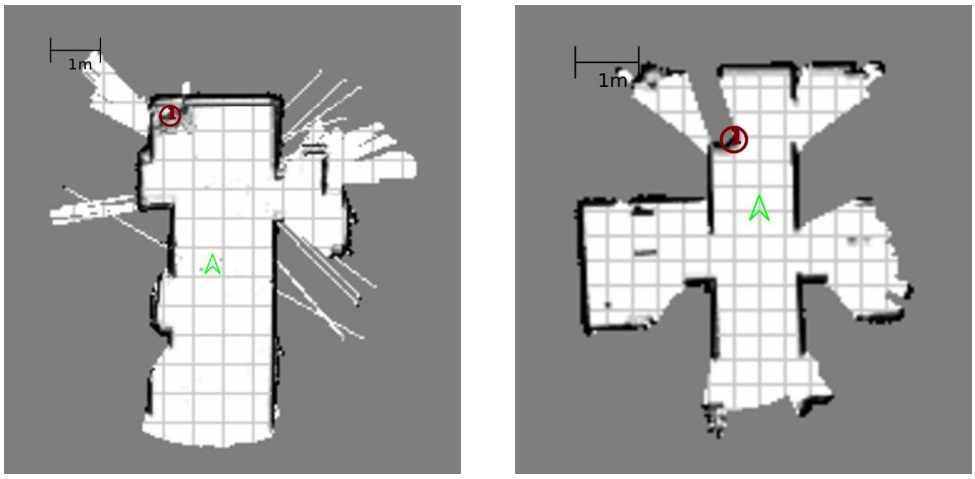

Bild 4.7: Autonom gefundenes Opfer in realem Betrieb (links) und simuliert (rechts)

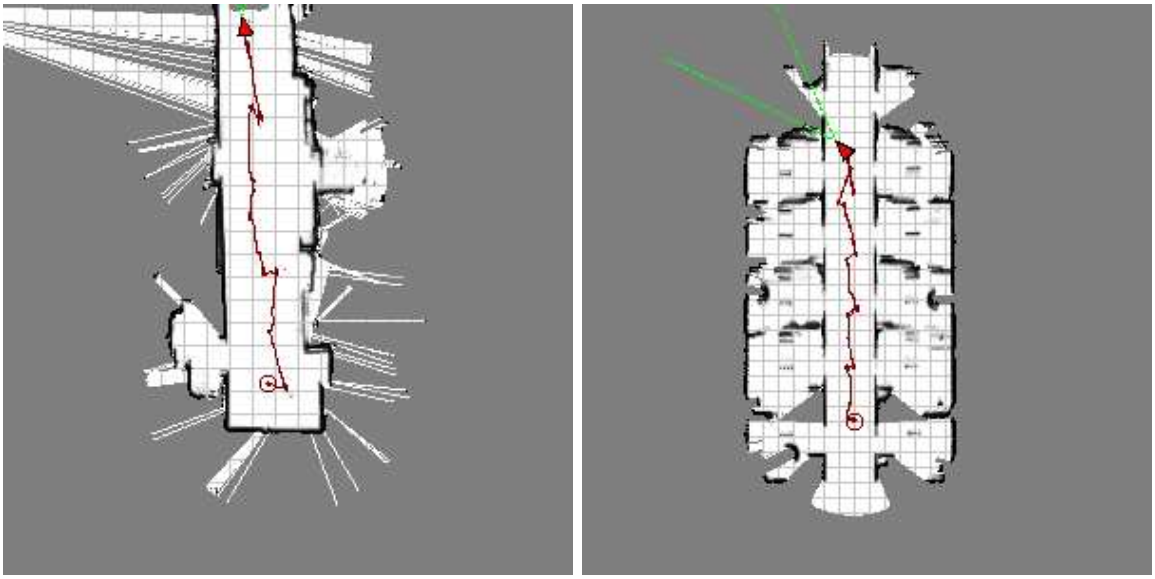

Bild 4.8: Autonomer Explorationspfad in realem Betrieb (links) und simuliert (rechts)

Kapitel 5

Zusammenfassung

Abschließend gibt diese Zusammenfassung nochmal einen Überblick über die Simulation, deren Vorteile und Grenzen. Die beschriebenen Erweiterungen werden resümiert und ein Gesamtbild der aktuell genutzten Simulation gegeben. Weiter werden die Erfahrungen mit der Simulation beschrieben und mögliche Erweiterungen vorgestellt.

Gegenüber dem realen Robbie bietet die Simulation gewisse Vorteile, wie in Kapitel 1 beschrieben. Neben vielen bereitstehenden Maps lassen sich mit dem UnrealEditor beliebige Szenarien erzeugen. Während ein Lauf des realen Roboters viel Aufwand durch Vorbereitung, Aufbau einer Testumgebung und herstellen der WLAN Verbindung bedeutet, geschieht dies in der Simulation alles automatisch. Weiter wurden die bessere Verfügbarkeit und die geringen Kosten erwähnt. Allerdings sind der Simulation auch Grenzen gesetzt. Die Simulation ist nur eine virtuelle, idealisierte, auf digitalen Daten beruhende Welt. Während die reale Welt von Störquellen und von physikalischen Gesetzen beinflusst wird, ist dies alles in der Simulation synthetisch erzeugt. Weiter ist die Simulation begrenzt durch die Performanz des Rechners und der Leistung der Graphikkarte.

Praktisch hat sich die Simulation im Robbie Projektpraktikum durchaus bewährt. Insbesondere wenn der reale Roboter defekt war, wurde die Simulation für die Entwicklung des autonomen Fahrens bei Robbie 7 herangezogen. Später bei Robbie 8 leistete sie gute Dienste bei der Verbesserung der Autonomie in Verbindung mit der Opfererkennung. Auch während dieser Zeit wurde die Anbindung der Simulation stetig weiterentwickelt.

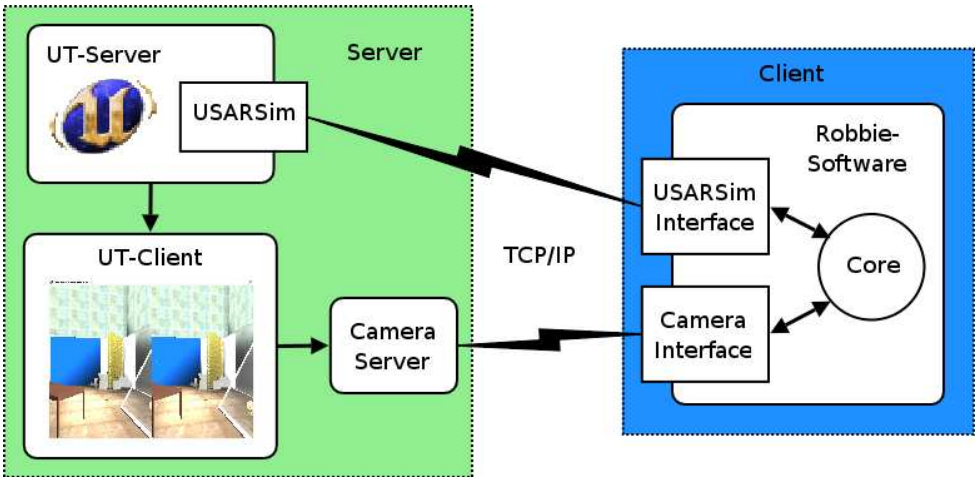

Bild 5.1: Datenaustausch innerhalb der aktuellen Simulationskomponenten

Eine Übersicht über alle Komponenten der Simulation, wie sie aktuell mit den, in Kapitel 3 beschriebenen Erweiterungen vorliegt, liefert Bild 5.1. Die Pfeile im Diagramm geben die Richtung der Datenübertragung zwischen den Komponenten an. Im Vergleich mit Bild 2.1 wird neben der Client-Server Architektur deutlich, dass auf Serverseite zwei neue Komponenten hinzugefügt wurden. Wie das Camera Interface, beschrieben in Abschnitt 3.4.2, dienen diese der Anbindung der Kamerabilder.

Die im Hauptteil dieser Arbeit vorgestellten Erweiterungen, ermöglichen trotz der Unterschiede zur Hardware, eine funktional umfangreichere Simulation von Robbie. Obwohl der simulierte 3D Scanner dem realen 3D Sensor in Punkto Schnelligkeit und Auflösung unterliegt, liefert er, dank exakter Daten aus den Maps, ein genaues Abbild der Umgebung. Ähnlich verhält es sich mit den Stereokameras. Trotz der niedrigen Bildrate können die Kamerabilder für optische Tiefenschätzung und Hautfarbendetektion verwendet werden. Die syntehtisch erzeugeten Rauscheffekte steigern die Realität und damit die Qualität der Simulation. Ebenso wichtig für eine realitätsnahe Simulation ist der Wärmesensor. Auch wenn die, in Abschnitt 3.5 vorgestellte Methode, keine Wärme sondern nur Opfer detektiert, ist sie für die Simulation unerlässlich. Insbesondere bei autonomer Exploration und Opfererkennung spielen die Wärmebilder eine wesentliche Rolle. Letztlich stellt die Performanz der Simulation noch eine Hürde auf dem Weg zu einer idealen Simulation dar.

Insgesamt zeigte sich bei den durchgeführten Tests jedoch ein vergleichbares Verhalten in Simulation und Realität. Wie die praktischen Erfahrungen bestätigen, sind neben den Sensordaten und den ähnlich qualitativen Karten auch das Fehlverhalten, zum Beispiel in der Autonomie, vergleichbar. Besonders zum Testen von Software ist dies eine wichtige Eigenschaft. Die, in Abschnitt 4.3 vorgestellten, Ergebnisse belegen somit den Nutzen der Simulation bei speziellen sowie komplexen Aufgaben.

Für die weitere Zukunft sind noch einige Punkte offen. Es ist geplant, die in Abschnitt 2.3 beschriebene Schnittstelle auf die Änderungen in der aktuellen USARSim Version 3.1 anzupassen. Für einen besseren Vergleich zwischen dem Roboterverhalten in der realen Welt und in der Simulation, beschrieben in Abschnitt 4.1, wäre eine Map als simuliertes, maßstabgetreues Abbild des realen AGAS Flures sinnvoll. Ein 3D Modell von Robbie zur Visualisierung des Roboters, würde die Simulation nicht nur optisch anspruchsvoller gestalten, sondern würde das physikalische Verhalten, zum Beispiel bei Kollisionen, verbessern. Nicht zuletzt wäre eine Implementation von Wärmestrahlung in USARSim wünschenswert.

USARSim wird ständig weiterentwickelt, was besonders an der Frequenz mit der neue Releases erscheinen, deutlich wird. Es gilt nur diese Erneuerungen zu nutzen.

Danksagung

Besonderer Dank gilt meinem Betreuer Johannes Pellenz, der mir mit Rat und Tat hilfreich zur Seite stand und mich bei inhaltlichen sowie technischen Problemen unterstützte. Weiter möchte ich mich bei Stephan Wirth und Ioannis Mihailidis bedanken, die mir insbesondere bei programmiertechnischen Fragen und linuxspezifischen Schwierigkeiten weitergeholfen haben.

Ferner bedanke ich mich bei Peter Schneider, Peter Decker und Christian Delis für Material und Erläuterungen zu ihren Studienarbeiten. Auch sei Stefan Burghardt gedankt, der immer ein offenes Ohr und gute Ratschläge für mich hatte. Für das Korrekturlesen und das Finden eingeschlichener Fehler und verrutschter Kommata, bedanke ich mich bei Janine Gasteier, Andreas Harder und Ramesh Christudhas.

Anhang A

Unreal Glossar

Alle in der vorliegenden Arbeit verwendeten, unreal-spezifischen Begriffe sind in diesem Glossar nachstehend, mit weiterführenden Weblinks, erläutert. Weitere Informationen zu Unreal sind zu finden unter
`http://www.unrealtournament.com`
`http://www.epicgames.com`

Effecter: Als Effecter, werden die Teile eines Roboters bezeichnet, die im Gegensatz zu den passiven Sensoren, aktiv operieren können, wie z. B. Greifarme.

Gamebot: Ein Gamebot ist der virtueller Charakter, ein texturiertes 3D-Modell einer Figur (eines Menschen, eines Roboters,...), welches das Alter-Ego eines Spielers oder KI-gesteuerter Kontrahenten darstellt.

Maps: Als Map bezeichnet man in Computerspielen, die virtuelle Spielumgebung, die der Spieler (in diesem Fall der Roboter) sieht. Der Begriff meint hier die gesamte Zusammenstellung von Objekten (Wände, Hindernisse, Opfer,...), aus denen ein RoboCup-Szenario besteht.

Spawnen: Als spawnen (Englisch: *spawn* = *laichen*) bezeichnet man den Vorgang der Initialisierung eines Spielers oder eines Objektes und das Erscheinen des entsprechenden 3D-Modelles in der Map.

Spectator: Im Spiel Unreal Tournament 2004 besteht die Möglichkeit, die Szenerie als frei bewegbarer Beobachter oder aus der Sicht eines Mitspielers (in diesem Fall der simulierte Roboter) zu betrachten.

UnrealEditor: Ein spezielles Tool zur Erstellung von Maps und Levels für Unreal Tournament und anderen Spielen, welche die *Unrealengine* benutzen. Ferner können auch eigene Robotermeshes für die Simulation erzeugt werden.
`http://www.unrealed.de`

Unrealengine: Die hier verwendete Gameengine besteht aus den Basisfunktionalitäten für Graphik, Physiksystem, Steuerung und Netzwerkcode. Die benutzete Version UnrealEngine 2.0 stammt aus dem Spiel Unreal Tournament 2004 von Epic Games, auf welches USARSim aufbaut.
`http://www.unrealtechnology.com`

UnrealUnits: In der verwendeten USARSim Version werden Entfernungswerte, wie auch in Unreal intern genutzten, eigenen Einheiten, den UnrealUnits (UU) angegeben. Ferner werden Winkel in RotationalUnrealUnits (RUU) angegeben.
`http://wiki.beyondunreal.com/wiki/Unreal_Unit`

USARSim: USARSim steht für „Urban Search And Rescue Simulation“. Maps, Robotermodelle und Interfaces zur Robotersoftware werden von USARSim bereitgestellt.
`http://sourceforge.net/projects/usarsim`

Literaturverzeichnis

[AHL+06] ALBRECHT, Sven ; HERTZBERG, Joachim ; LINGEMANN, Kai ; NÜCHTER, Andreas ; SPRICKERHOF, Jochen ; STIENE, Stefan: Device Level Simulation of Kurt3D Rescue Robots / Universität Osnabrück. 2006. – Forschungsbericht

[BH06] BURGHARDT, Stefan ; HOLZHÄUSER, Dennis: *CR07: Anbinden des Unreal Simulators*. 5 2006. – Change request Robbie7

[CLW+06] CARPIN, Stefano ; LEWIS, Mike ; WANG, Jijun ; BALAKIRSKY, Stephen ; SCRAPPER, Chris: Bridging the gap between simulation and reality in urban serch and rescue. In: *Robocup 2006: Robot Soccer World Cup X*, Springer, 2006

[CLW+07] CARPIN, Stefano ; LEWIS, Mike ; WANG, Jijun ; BALAKIRSKY, Stephen ; SCRAPPER, Chris: USARSim: a robot simulator for research and education. In: *Proceedings of the 2007 IEEE Conference on Robotics and Automatio*, 2007

[CSN+06] CARPIN, S. ; STOYANOV, T. ; NEVATIA, Y. ; LEWIS, M. ; WANG, J.: Quantitative assessments of USARSim accuracy. In: *Proceedings of PerMIS*, 2006

[Dec06] DECKER, Peter: *Erstellung einer Tiefenkarte aus Stereobildern für den RoboCup Rescue Wettbewerb*, Universität Koblenz-Landau, Studienarbeit, 2006

[HLN03] HERTZBER, Joachim ; LINGEMANN, Kai ; NUECHTER, Andreas: USARSim - Game-Engines in der Robotiklehre / Universität Osnabrück. 2003. – Forschungsbericht

[JMW+03] JACOFF, Adam ; MESSINA, Elena ; WEISS, Brian A. ; TADOKORO, Satoshi ; NAKAGAWA, Yuki: Test Arenas and Performance Metrics for Urban Search and Rescue Robots. In: *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robotics and Systems*, 2003

[KPN06] KLEINER, Alexander ; PREDIGER, Johann ; NEBEL, Bernhard ; : RFID Technology-based Exploration and SLAM for Search and Rescue. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2006

[LJ02] LEWIS, Michael ; JACOBSON, Jeffrey: Game engines in scientific research. In: *Communications of the ACM* 45 (2002), 1, Nr. 1, S. 27 – 31

[Mat] MATHENGINE (Hrsg.): *MathEngine Karma User Guide*. MathEngine, `http://udn.epicgames.com/Two/KarmaReference.html`

[Mer07] MERKELE, Bernhard: Digitale Mitspieler, Microsoft stellt Robotics Studio vor. In: *iX Magazin für professionelle Informationstechnik* (2007), 3

[Mic04] MICHEL, Olivier: Webots: Professional Mobile Robot Simulation. In: *International Journal of Advanced Robotic Systems* Bd. 1, 2004, S. 39 – 44

[mis07a] *Delta3D, Open source gaming & simulation engine*. `http://www.Delta3D.org`. Version: 5 2007

[mis07b] *USARSim discussion forums: Help*. `http://sourceforge.net/forum/forum.php?forum_id=486389`. Version: 4 2007

[PDMP06] PELLENZ, Johannes ; DELIS, Christian ; MIHAILIDIS, Ioannis ; PAULUS, Dietrich: Low-Cost 3D-Laserscanner für mobile Systeme im RoboCup Rescue Wettbewerb. In: *9. Anwendungsbezogener Workshop zur Erfassung, Modellierung, Verarbeitung und Auswertung von 3D-Daten* (2006)

[Wan06] WANG, Jijun: *The Gas Simulation*. 10 2006. – Document on simulating gas diffusion and sensors in USARSim

[WLC+05] WANG, Jijun ; LEWIS, Mike ; CARPIN, Stefano ; BIRK, Andreas ; JACOFF, Adam: High fidelity tools for rescue robotics: results and perspectives. In: *Robocup 2006: Robot Soccer World Cup IX*, Springer, 2005