Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Mirco Gerling

PDA-gestützte Robotersteuerung mit funkbasierter Serveranbindung

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Mirco Gerling

PDA-gestützte Robotersteuerung mit funkbasierter Serveranbindung

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Technische Informatik
am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachter : Prof. Dr. Gunter Klemke

Abgegeben am 16. Dezember 2003

Mirco Gerling

Thema der Diplomarbeit

PDA-gestützte Robotersteuerung mit funkbasierter Serveranbindung

Stichworte

Agenten, PDA, Bluetooth, TCP/IP, mobile Roboter, Subsumption

Kurzzusammenfassung

In dieser Arbeit wird eine PDA-gestützte Robotersteuerung realisiert. Das System besteht aus einer 3-Schicht-Architektur. Die unterste Schicht besteht aus einem Controller-Board. Das Controller-Board stellt die Schnittstelle zu den Motoren und Sensoren dar. Die zweite Schicht besteht aus einem PDA, der über ein serielles Kabel mit dem Controller-Board verbunden ist. Mit Hilfe dieser Verbindung kann der PDA Befehle an das Controller-Board übertragen und Sensorwerte zu empfangen. Die dritte Schicht besteht aus einem Server auf dem ein Server-Programm läuft. Es ermöglicht einem Benutzer die Eingabe von Informationen für den Roboter. Der Server und der PDA können über eine Funkverbindung, die auf Bluetooth basiert, eine TCP/IP-Verbindung aufbauen. Über diese Verbindung kann sich der Roboter die Informationen vom Server abholen. Die Funktion des Systems wird anhand eines Szenarios überprüft.

Mirco Gerling

Title of the paper

PDA-based robot control with wireless server connection

Keywords

Agents, PDA, Bluetooth, TCP/IP, mobile robots, subsumption

Abstract

In this thesis a PDA supported robotic control is realized. The system consists of a 3-layer-architecture. The lowest layer consists of a controller board. The controller board, represents the interface to the engines and sensors. The second layer consists of a PDA, which is connected with the controller board by a serial cable. With the help of this connection the PDA is able to transfer instructions to the controller board and receive sensor values. The third layer consists of a server on which a server program runs. It makes it possible for a user to input information for the robot. The server and the PDA can create a TCP/IP connection over a wireless connection, which is based on Bluetooth. Over this connection the robot can fetch itself the information from the server. The function of the system is examined on the basis of a scenario.

Danksagung

Bei meinem Betreuer Kai von Luck und bei allen, die mich in irgendeiner Form unterstützt haben, möchte ich mich herzlich bedanken.

DANKE, DANKE, DANKE!!!

Inhaltsverzeichnis

Kapitel 1

Einleitung

Roboter gewinnen in unserem Alltag immer mehr an Bedeutung, ob für den Privatgebrauch oder in der Industrie. Es gibt Roboter die Rasen mähen, Staub saugen, Schwimmbäder reinigen oder Waren in einem Lager transportieren. So gibt es staubsaugendende Roboter wie den Roomba (Roomba, 2003) und den Trilobite (Electrolux, 2003). Als Beispiele für Roboter, die Rasen mähen, gibt es den Auto Mower (Husqvarna, 2003) und den RoboMow (RoboMow, 2003). Der RoboMow sorgt selbstständig dafür, dass ein Rasen bis ca. 1500 Quadratmeter immer kurz gehalten wird.

Von der Firma Honda (Honda, 2003) gibt es bereits seit Jahren Bemühungen im Bereich der humanoiden[1] Roboter. Hierbei handelt es sich um Roboter in menschenähnlicher Gestalt. Sie haben Beine zum Gehen und Hände zum Greifen. Das besondere an diesen Robotern ist, dass sie ähnlich wie ein Mensch gehen können. Das Schwierige daran ist der Gleichgewichtssinn. Der Roboter muss das Gleichgewicht halten können, vor allem beim Treppensteigen.

[1] humanoid = menschenähnlich

Abbildung 1.1: Roomba

Abbildung 1.2: Trilobite mit Ladestation

Abbildung 1.3: Husqvarna Rasenmäher Auto Mower

Abbildung 1.4: RoboMow von Friendly Robotics

Abbildung 1.5: Humanoider Roboter Honda ASIMO

Die Roboter unterscheiden sich in einem entscheidenden Punkt der Softwarearchitektur. Es gibt verhaltensbasierte Roboter und modellbasierte Roboter. In dem Buch Behavior-Based Robotics (Arkin, 1998) ist eine Vielzahl von Architekturen beschrieben. Es gibt z.B. Verhaltensweisen wie bei Tieren (Animal Behavior) bzw. Insekten. Verhaltensbasierte Systeme werden auch als reaktive Systeme bezeichnet. Sie reagieren direkt auf äußere Einflüsse. Modellbasierte Systeme besitzen eine innere Karte von ihrer Umgebung und handeln aufgrund dieser Karte. Außerdem gibt es auch Architekturen, bei denen die beiden Verhaltensweisen gemischt werden. Diese werden als hybride Architekturen bezeichnet.

Diese Arbeit baut auf meiner Studienarbeit über Agentenarchitekturen (Gerling, 2002) auf. In der Studienarbeit geht es um die Ausstattung eines, auf einem MIT-6.270-Board basierenden Roboters mit einem seriellen Bluetooth-Modul. Dieses Modul ermöglicht eine Kommunikation mit dem Roboter. Zusätzlich werden in der Studienarbeit die Unterschiede zwischen modellbasierten und verhaltensbasierten Agenten gezeigt. Die Abbildung 1.6 zeigt den Roboter aus der Studienarbeit in seinem Testumfeld. Auf dem Bild ist eine grüne Platine zu erkennen. Hierbei handelt es sich um ein serielles Bluetooth-Modul, über welches der Roboter kommuniziert.

Gegenstand dieser Arbeit ist das Design und die Realisierung eines hybriden Roboters in einer 3-Schicht-Architektur. Als Szenario soll die Metapher einer Tankstelle verwendet werden, an der z.B. die Kreditkartendaten per Bluetooth zwischen Fahrzeug und der Kasse ausgetauscht werden. Außerdem soll es möglich sein z.B. Stauinformationen an der Tankstelle auszutauschen.

Diese Arbeit ist in mehrere Teile gegliedert. Das Kapitel 2 behandelt die allgemeinen theoretischen Grundlagen von Agenten. Im Kapitel 3 wird ein mögliches Szenario, zur Überprüfung des zu entwickelnden Systems, vorgestellt. Mit den Hard-

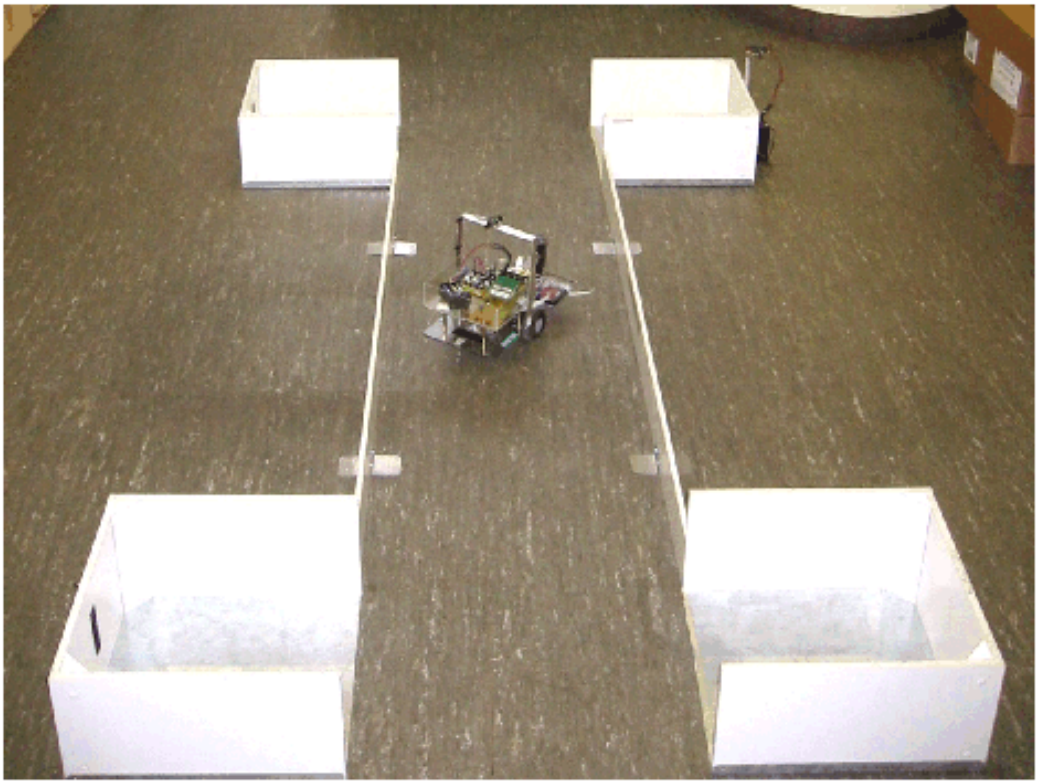

Abbildung 1.6: Abbildung aus der vorangegangenen Studienarbeit

und Softwarevoraussetzungen, die für diese Arbeit benötigt werden, beschäftigt sich das Kapitel 4. Das Kapitel 5 befasst sich mit dem Design und der Realisierung des Systems. Abschließend werden in Kapitel 6 die erreichten Ziele und weitere Entwicklungsmöglichkeiten zusammengefasst.

Eingetragene Warenzeichen

Kapitel 2

Agenten

2.1 Charakteristika

Agenten sind in der Regel Programme, denen man einen Auftrag gibt, den sie daraufhin selbständig abarbeiten. Genauere Informationen gibt es im Handbuch der KI (Goerz u. a., 2000). Es gibt reine Softwareagenten und Agenten in Form von Robotern. Jan Philip Porschke (Porschke, 2002) beschreibt in seiner Diplomarbeit genauer die Merkmale von Agenten, wie sie in dem Handbuch der KI dargestellt werden. In dem Handbuch der KI werden Agenten durch folgende Merkmale, der Taxonomie der Agenten, charakterisiert:

- Andauernde Verfügbarkeit/Aktivität: Agenten sind über einen längeren Zeitraum hinweg ansprechbar und nehmen "Aufträge" von Nutzern oder anderen Agenten entgegen. Sie können eigenständig aktiv werden.
- Interaktion mit einer Umwelt: Agenten nehmen Informationen aus ihrer Umwelt auf. Sie agieren in ihrer Umwelt, um sie (auftragsgemäß) zu beeinflussen.
- Situiertheit: Auch hier geht es um die Einbettung von Agenten in einer Umwelt. Es wird der Aspekt betont, dass ein komplexes Agentenverhalten auch als Resultat von direkten Reaktionen auf Umwelteinflüsse erzeugbar ist ("emergentes Verhalten").
- Eigenständigkeit (Autonomie) im Handeln: Agenten unterliegen keiner (unmittelbaren) Steuerung und Kontrolle durch den Nutzer. Sie handeln eigenständig, aber "im Sinne ihrer Auftraggeber". Die Auswahl/Planung ihrer Handlungen kann (aber muss nicht) nach sehr komplexen Methoden erfolgen.

- Reaktivität: Im engeren Sinne wird darunter das unmittelbare Reagieren auf Umweltereignisse ("Stimulus-Responsus") verstanden. Im allgemeinen betrifft es die Interaktion mit der Umwelt insgesamt.
- Zielgerichtetheit: Agenten verfolgen Ziele bzw. Aufträge, die angepasstes Handeln über lange Zeiträume hinweg erfordern können.
- Pro-Aktivität: Der Begriff ist verwandt mit Zielgerichtetheit (und wird teilweise gleichwertig benutzt). Eine spezielle Betonung liegt dabei auf der "Eigeninitiative" des Agenten.
- Deliberatives Verhalten: "Deliberation" bezeichnet die explizit modellierte Auswahl von Zielen bzw. Absichten. Auch dieser Begriff wird als Gegensatz zu reaktivem Verhalten gebraucht.
- Intelligenz: Der Begriff kann im Sinne analoger Erklärungen des Begriffs "Künstliche Intelligenz" so interpretiert werden: Agenten vollführen Handlungen, die beim Menschen als intelligent gelten würden.
- Rationalität: Agenten treffen sinnvoll erscheinende Entscheidungen in angemessener Zeit auch unter dem Aspekt beschränkt verfügbarer eigener Ressourcen.
- Lernfähigkeit: Agenten können ihr Verhalten an die Umwelt anpassen, indem sie zum Beispiel Fähigkeiten oder Entscheidungsprozesse geeignet variieren.
- Mobilität: Agenten können eigenständig auf andere Plattformen (Rechner) migrieren. Dabei wird das Agenten-Programm mit seinen aktuellen Daten übertragen und setzt seine Arbeit dort fort mit der Möglichkeit des Zugriffs auf lokale Ressourcen.
- Kooperation: Agenten arbeiten zusammen mit Menschen und anderen Agenten.
- Wohlwollen (benevolenz): Agenten führen Aufträge nach Möglichkeit im Sinne der Auftraggebers aus, wobei auch andere Agenten Auftraggeber sein können.
- Soziales Verhalten: Agenten halten sich bei ihrer Arbeit mit anderen Agenten (und Menschen) an vorgegebene Regeln und Normen. Sie bilden Gruppen oder Koalitionen. Vorbild sind soziale Strukturen der Menschen.

- Emotionales Verhalten: Emotionen sollen einerseits das Verhalten steuern (sie treten damit neben andere Begriffe wie Ziele, Absichten usw.), und sie sollen die Interaktion mit dem Menschen verbessern.
- Glaubwürdigkeit: Die Erscheinungsform (z.B. Gestik animierter Agenten) soll sich in glaubhafter Übereinstimmung mit ihren Aktionen befinden. Agenten sollen als Individuen mit eigenen Zielen, Bedürfnissen und Emotionen erscheinen.

Ein Agentensystem sollte mindestens eins oder besser mehrere dieser Kriterien erfüllen, um als Agent bezeichnet werden zu können.

2.2 Deliberative Architektur (Modellbasierte)

Es gibt zwei unterschiedliche Gruppen von modellbasierten Agenten. Die einen besitzen ein vorgefertigtes Modell ihrer Umgebung und die anderen erkunden ihre Umgebung und fertigen sich ihr Modell selbst an. Ein Modell kann zum Beispiel aus einer Karte der Umgebung, aus Stationen bzw. Häusern, Türen und Botschaften in Form von Briefen bestehen. Beide Systeme versuchen ihr Wissen über die Umgebung logisch umzusetzen. Das größte Problem der modellbasierten Agenten ist die Bestimmung der eigenen Position innerhalb der Umgebung. Nur wenn der Roboter weiß, wo er sich befindet, kann er auch im Bereich einer Karte navigieren. Das Prinzip basiert auf der Planung des Verhaltens mit Hilfe der Karte.

Die modellbasierte Architektur weist einige Schwächen auf. So muss das Modell der Umgebung vollständig definiert sein, damit der Roboter auch auf alle Ereignisse reagieren kann. Es ist praktisch unmöglich das gesamte Modell zu erfassen. So können in einer dynamischen Umgebung unerwartet Hindernisse auftreten, die im Modell nicht erfasst sind. Ein weiteres Problem ist die Erkennbarkeit des Modells. Da man in der Realität keine hundertprozentige Sensor-Qualität antrifft, kann es passieren, dass sich der Roboter im Modell nicht mehr zurecht findet, da er seine Position nicht zuverlässig erkennt. Ein weiteres Problem ist der Handlungserfolg. Wenn der Roboter eine Aktion ausführen möchte, gibt es zwei Möglichkeiten. Entweder gelingt die Aktion oder sie schlägt fehl. Für beide Fälle müssen entsprechende Behandlungen vorgesehen sein. Jedoch lassen sich auch hier nicht alle Eventualitäten vorhersehen.

2.3 Verhaltensbasierte Architektur

Verhaltensbasierte Agenten benötigen kein Wissen über ihre Umwelt. Diese Agenten haben keinen Willen z.B. ein Tor zu schießen. Sie verhalten sich rein reaktiv. Wenn ein bestimmtes Ereignis eintritt, reagieren sie nach einem vorher für dieses Ereignis definierten Muster. Dies tun sie unabhängig von ihrem Standort. Dieses Verhalten kann durchaus in eine Sackgasse führen. Aus diesem Grund sollten möglichst alle Ereignisse dem Agenten schon vorher bekannt sein, damit er darauf reagieren kann. So würde man eine möglichst geringe Fehleranfälligkeit erreichen. Verhaltensbasierte Roboter sind nicht in der Lage planvoll zu handeln. Rolf Pfeifer (Pfeifer und Scheier, 1999) vertritt die Ansicht, dass ein verhaltensbasierter Roboter immer in irgendeiner Weise reagiert. Es ist nur nicht sicher, ob er sich angemessen verhält. So kann man ein Verhalten für den Notfall vorsehen, jedoch ist nicht sicher, ob der Roboter zuverlässig den Notfall erkennt.

Ein Beispiel für verhaltensbasierte Agenten, in Form eines Roboters, ist das Braitenberg Vehikel (Braitenberg, 1993; Arkin, 1998). Braitenberg hat einige verhaltensbasierte Vehikel entwickelt. Hier werden analoge Sensoren direkt mit den Motoren einer Plattform gekoppelt. Dadurch reagiert das System direkt auf äußere Einflüsse. Durch unterschiedliche Beschaltungen lassen sich diverse Verhaltensweisen realisieren. So ist es möglich durch lichtempfindliche Sensoren das Vehikel lichtabhängig reagieren zu lassen. Je nach Beschaltung nähert sich das Vehikel einer Lichtquelle oder entfernt sich von ihr. Diese Vehikel sind rein reaktive Systeme, d.h. sie reagieren nur auf die Einflüsse ihrer Umgebung. Es gibt Varianten mit nur einem Sensor und einem Motor (siehe Abbildung 2.1 links) und es gibt Varianten mit zwei Motoren und zwei Sensoren (siehe Abbildung 2.1 mitte und rechts). Auf diese Weise wurden im Media Lab des Massachusetts Institute of Technology (MIT) verschiedene Vehikel entwickelt, die auf dem Prinzip von Braitenberg basieren. Es wurden z.B. ein Schatten-Sucher, ein Schatten-Kantenfinder, ein Vehikel welches Angst vor Schatten hat und ein Licht suchendes Vehikel entwickelt.

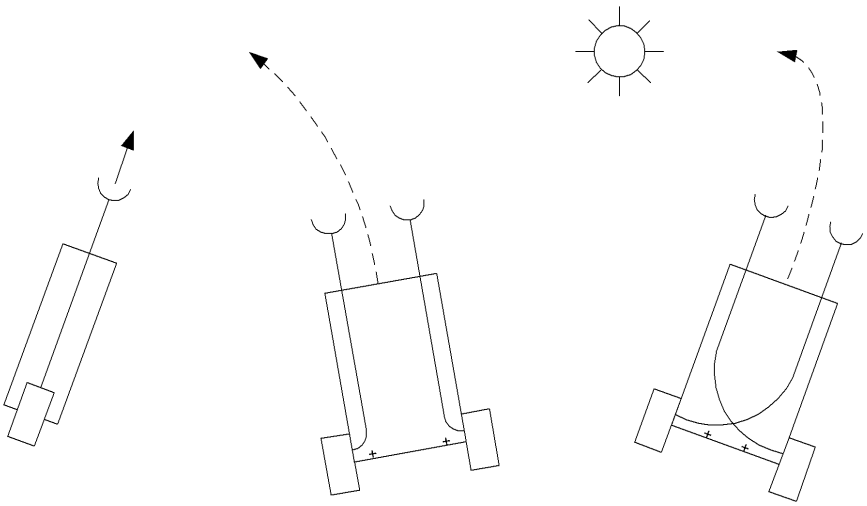

Abbildung 2.1: Braitenberg Vehikel

Subsumption-Architektur

Die Subsumption-Architektur wurde von Rodney Brooks Mitte der achtziger Jahre am MIT entwickelt. Brooks stellte das sense-plan-act Modell in Frage. Nach diesem Modell wurden einige der ersten autonomen Roboter entwickelt. Mitte der achtziger Jahre befürwortete er die Verwendung eines geschichteten Kontrollsystems (Brooks, 1985). Mit der Subsumption-Architektur muss ein komplexes Verhalten nicht unbedingt das Ergebnis eines komplexen Kontrollsystems sein. Diese Architektur macht es möglich, verschiedene Verhaltensweisen zu kapseln. Für Brooks war es nicht einfach, die Menschen zum Umdenken zu bewegen, dass dies der neue Weg sein soll, intelligente Roboter bzw. intelligente Systeme auf diese Weise zu konstruieren.

In der Subsumption-Architektur arbeiten verschiedene Schichten konkurrierend und asynchron an unterschiedlichen Aufgaben. Die Koordination erfolgt über Prioritäten. Der Roboter führt zurzeit nur ein Verhalten einer bestimmten Schicht aus. Dadurch ist es möglich, z.B. der Kollisionsvermeidung die höchste Priorität zu geben und anderen Aufgaben, die nicht ganz so wichtig sind eine niedrigere Priorität zu geben. Die Schicht mit der höchsten Priorität befindet sich ganz oben. So ist sie in der Lage, die anderen Schichten zu unterdrücken. Die Schicht mit der niedrigsten Priorität befindet sich demnach ganz unten. Alle Schichten haben unabhängig davon, ob sie gerade die Kontrolle über den Roboter haben, Zugriff auf einen Pool von Sensordaten.

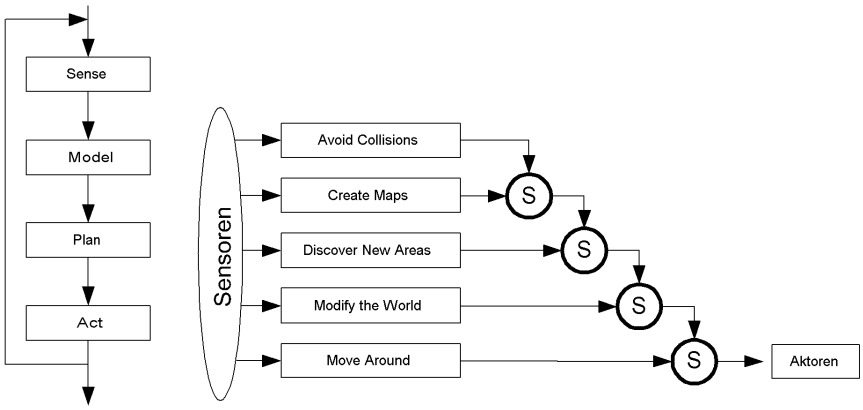

Abbildung 2.2: Sense-plan-act Modell und Subsumption (reaktives) Modell

2.4 Hybride Architektur

Die hybride Architektur besteht aus einer Mischung von modellbasierter und verhaltensbasierter Architektur. Hier werden modellbasierte und verhaltensbasierte Implementationen konkurrierend parallel zueinander betrieben. Über Prioritäten und die Anknüpfung von Verhalten an externe Signale wird nun das Verhalten ausgewählt, das den Roboter kontrollieren soll.

In dem Buch Behavior-Based Robotics (Arkin, 1998) werden Argumente, die für eine hybride Architektur sprechen, diskutiert. Rein reaktive Architekturen können in komplexen und in dynamischen Umgebungen eine robuste und leistungsfähige Lösung sein. In einigen Fällen können die festen Annahmen, die rein reaktive Systeme machen, zu einem Nachteil werden. Zu diesen Annahmen gehört:

1. Es fehlt zeitweise an der Übereinstimmung und an Beständigkeit der Umgebung mit dem Weltmodell
2. Es gibt keine oder nur wenige Informationen über das Weltmodell der Umgebung.
3. Es ist schwierig den Roboter relativ zum Weltmodell zu lokalisieren.
4. Die unmittelbare Sensorik ist ausreichend für die momentane Aufgabe.

In einigen Umgebungen sind diese Annahmen nicht immer korrekt. Rein reaktive Roboter-Systeme sind nicht für alle Roboter-Anwendungen geeignet. In Situationen, in denen die Welt genau modelliert werden kann, also jegliche Unsicherheiten begrenzt sind und eine Garantie besteht, dass sich während der Ausführung die Welt nicht verändert, werden deliberative Methoden häufig bevorzugt, da ein Plan meist effektiver ausgeführt werden kann.

In der realen Welt existieren biologische Agenten, wie z.B. Insekten. In der realen Welt gibt es keine Bedingungen, in denen man rein deliberative Architekturen einsetzen kann. Das liegt an der Vielfalt der realen Welt. Um unvorhersehbare Situationen zu beherrschen, ist ein reaktives Verhalten notwendig. Dementsprechend ist für den Einsatz eines Roboters in der realen Welt eine reaktive Kontrolle des Roboters erforderlich. Eine hybride Architektur aus deliberativer und verhaltensbasierter Architektur macht es möglich, das volle Potential aus einem Roboter-System herauszuholen.

Die Mischung von unterschiedlichen Informationen in einer Roboter-Architektur kann die Navigation flexibler und allgemeiner gestalten. Deliberative Systeme ermöglichen es, das Wissen über das Weltmodell für planerische Zwecke zu verwenden, bevor das System eine Aktion ausführt.

Ein vorheriges Wissen über das Weltmodell kann, wenn vorhanden und beständig, zur Konfiguration und Rekonfiguration eines Systems verwendet werden. Dynamisch erstellte Weltmodelle können vor unvorhergesehenen Gefahren (Hindernissen) schützen.

Hybride Architekturen erlauben eine Rekonfiguration von reaktiven Kontroll-Systemen basierend auf verfügbarem Wissen über das Weltmodell. Die dynamische Rekonfiguration des Kontroll-Systems ist eine wichtige Ergänzung zu den allgemeinen Fähigkeiten eines Roboters.

2.5 Multiagenten Systeme

Es gibt Systeme in denen nicht nur ein Agent tätig ist, sondern auch mehrere. Der Einsatz von Multiagenten hat einige positive Aspekte, wie zum Beispiel eine höhere Effektivität und Leistung durch die Aufgabenverteilung auf mehrere Agenten, die gemeinsame Nutzung von Informationen, die gleichzeitige Bewältigung von Aufgaben an mehreren Orten, die Fehlertoleranz durch Redundanz und reduzierte Komplexität und die Fähigkeit mehrerer Roboter Aufgaben zu erledigen, die ein einzelner nicht bewältigen kann.

Jedoch gibt es auch eine Vielzahl von negativen Aspekten, die ein Multiagenten-System mit sich bringt. So kann es zu Kollisionen oder Blockaden unter den Robotern kommen. Je mehr Roboter zum Einsatz kommen, desto weniger Raum steht ihnen zum Navigieren zur Verfügung. Bei mehreren Agenten ist auch eine Kommunikation nötig. Die Kommunikation hat ihren Preis, es wird zusätzliche Hardware, Rechenleistung und Energie benötigt. Die Zuverlässigkeit der Kommunikation kann durch gegenseitige Störungen und verrauschte Kanäle beeinträchtigt werden. Die Koordination der einzelnen Agenten erfordert die Kenntnis über die Tätigkeit der jeweiligen Agenten. Wenn dies wegen fehlender Kenntnis oder schlechter Kommunikation unklar ist, kann dies dazu führen, dass die Roboter gegeneinander und nicht miteinander arbeiten. Es kann sein, dass ein System mit zwei Robotern insgesamt auf Dauer mehr kostet. Wenn ein Team mit einfacheren, weniger komplexen Robotern erstellt werden kann, als wenn man einen Roboter einsetzt, dann muss das nicht zutreffen.

2.6 Zukunft der Agenten

In der Forschung über Roboter gibt es noch einige Herausforderungen zu meistern. So sollten die Roboter der großen Menge der Menschen zugänglich gemacht werden. Dies ist aber nur möglich, wenn entsprechend einfache Benutzer- und Programmier-Schnittstellen entwickelt werden. In der Sensorik gibt es auch noch viele Möglichkeiten durch die Verbesserung von Sensoren. Die Kenntnisse über die Zusammenhänge zwischen deliberativen und reaktiven Systemen sollten noch tiefer sein. Im Bereich der lernenden Roboter gibt es noch einige Kapazitäten.

Kapitel 3

Ein Szenario

Die Aufgabe

Das in dieser Arbeit zu entwickelnde System soll anhand eines Szenarios überprüft werden. Daher wird an dieser ein Szenario beschrieben, welches in vereinfachter Form die Möglichkeiten dieses Systems zeigt.

Der Roboter soll in der Lage sein, selbstständig durch einen Parcours zu fahren. In dem Parcours gibt es zwei Stationen, die Tankstellen darstellen sollen. Es gibt zwei verschiedene Wege, um von einer Tankstelle zu der anderen zu kommen. Unterwegs befinden sich weitere Plätze, die angefahren werden sollen. Bei den Tankstellen können Transportaufträge für den Roboter anfallen. Es soll möglich sein, die Transportaufträge und Stauinformationen an jeweils einer der beiden Tankstellen zu hinterlegen. Diese Informationen kann sich der Roboter dann abrufen. Erreicht der Roboter eine der Tankstellen, so soll es möglich sein, als erstes die Kreditkartendaten zu übertragen. Erst nachdem die Kreditkartendaten überprüft wurden kann der Roboter weitere Informationen erfragen. Wenn die Ressourcen des Roboters knapp werden, soll er zu einer der Tankstellen fahren und seine Ressourcen erneuern. In dieser Zeit werden die anstehenden Transportaufträge zurückgestellt. Erst wenn die Ressourcen erneuert wurden, werden eventuell anstehende Transportaufträge abgearbeitet. (siehe Abbildung 3.1)

Der Roboter verfügt über eine Funkverbindung, um mit den Tankstellen zu kommunizieren. Kommt er an eine der Tankstellen, nimmt er per Funk Kontakt mit ihr auf. Wenn jetzt z.B. der Tankvorgang abgeschlossen ist, werden Informationen ausgetauscht. Ist der Vorgang abgeschlossen, wird die Verbindung zwischen dem Roboter und der Tankstelle getrennt. Der Roboter soll nun eine der beiden Strecken

befahren. Welche der Strecken benutzt wird, richtet sich nach den Stauinformationen und nach den Transportaufträgen. Kommt der Roboter an die andere Tankstelle, nimmt er ebenfalls mit ihr Kontakt auf. Nach dem Aufenthalt an der zweiten Tankstelle fährt der Roboter wieder auf dem vom Server empfohlenen Weg in die Richtung der ersten Tankstelle. So wiederholt sich dann der Vorgang. Der Roboter kennt die Welt in der er sich bewegt. Dies ist eine Voraussetzung dafür, dass er sich in dem Parcours zurecht findet und grob einschätzen kann, wo er sich gerade befindet. Eine weitere Beschreibung des Szenarios, speziell in Bezug auf die Stauinformationen, ist in Kapitel 5.8 zu finden.

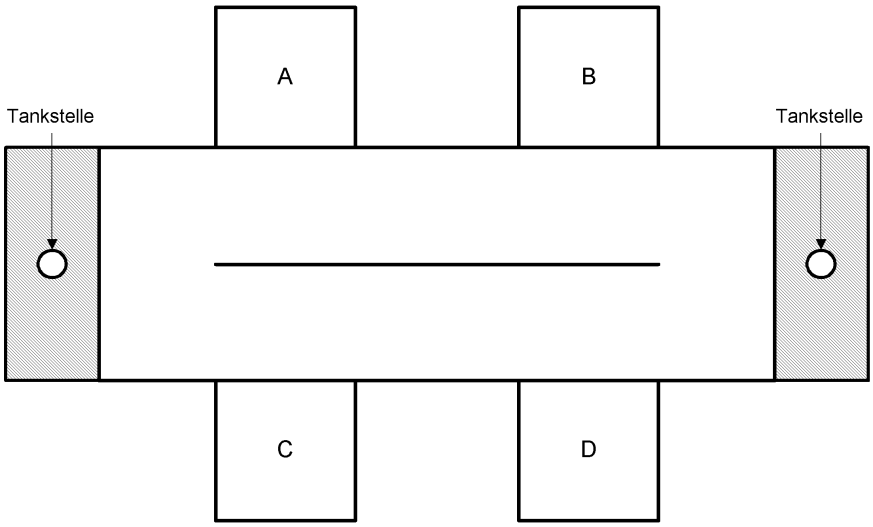

Abbildung 3.1: Plan/Modell des Szenarios

Verhaltensbasierter Anteil

Bumping

Der verhaltensbasierte Anteil soll darin bestehen, dass auf dem Controller-Board des Roboters die Behandlung von Kollisionen mit der Umgebung durchgeführt wird. Dies bedeutet, dass diese Kollisionsbehandlung höchste Priorität hat. Die übergeordneten Funktionalitäten haben bei einer Kollision keine Auswirkungen. Das heißt, Steuersignale die an das Controller-Board übermittelt werden, werden ignoriert. Sie werden erst wieder angenommen, wenn die Kollision behoben wurde. Dieses Verhalten dient um an einer Wand entlang zu fahren.

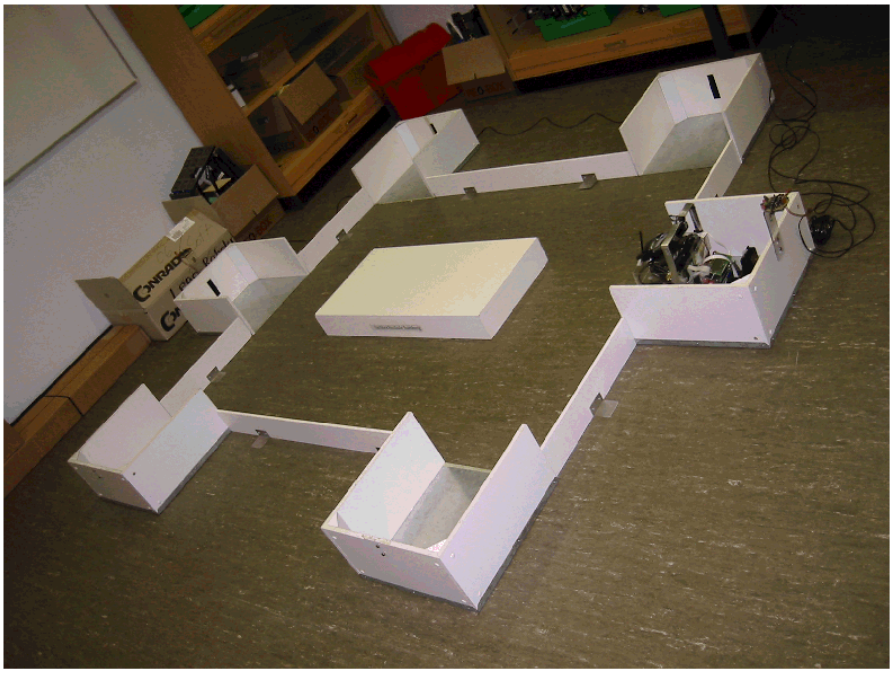

Abbildung 3.2: Photo des Szenarios

Hunger

Ein weiterer Bestandteil des verhaltensbasierten Anteils soll die Hungerfunktion sein. Werden die Ressourcen des Roboters knapp, steuert er eine der Tankstellen an. Eine Maßeinheit für den Hunger könnte hier z.B. die zurückgelegte Strecke sein. Der Hunger könnte sich in zwei Stufen bemerkbar machen. Erst tritt ein leichter Hunger auf. Trifft der Roboter nun auf eine Tankstelle, verbleibt er dort und erneuert seine Ressourcen. Wird der Hunger größer, bekommt der Roboter starken Hunger. In diesem Fall sucht er umgehend eine der beiden Tankstellen auf. Eventuell anstehende Transportaufträge werden vorerst zurückgestellt.

Modellbasierter Anteil

Planning

Beim modellbasierten Anteil geht es darum, dass der Roboter eine innere Karte von seiner Umgebung hat bzw. eine grobe Abschätzung machen kann, wo er sich gerade

befindet. Der Roboter hat Informationen, dass es zwei Tankstellen-Objekte gibt und welche Daten er mit ihnen austauschen muss. Außerdem merkt er sich, wo er sich gerade in dem Parcours befindet und wo er hinfahren muss.

Planerische Ebene

Auf dem Server bzw. den Servern sollen Informationen über den weiteren Verlauf der Fahrt des Roboters bereitgestellt werden.

Außerdem soll die Auftragsverwaltung auf den Servern abgewickelt werden. Es soll möglich sein, unterwegs noch weitere Stationen anzusteuern und dort etwas abzuliefern bzw. abzuholen. Dies soll abhängig von der Auftragslage und von den Stauinformationen geschehen.

Kapitel 4

Hard-/Softwarevoraussetzungen

4.1 Die Laborumgebung

Bei dem Labor der HAW Hamburg handelt es sich um ein speziell für das Experimentieren mit Robotern eingerichtetes Labor. Es verfügt über eine umfangreiche Ausstattung, die zum Bau von autonomen Robotern benötigt wird.

In dem Labor gibt es verschiedene Bereiche, an denen gearbeitet wird. Es gibt Roboter, die mit dem MIT-6.270-Board ausgestattet sind bzw. ausgestattet werden können (Martin, 2002; MIT, 2003). Seit wenigen Semestern gibt es auch Projekte in denen mit den MINDSTORM Baukästen von LEGO gearbeitet wird (LEGO, 2002).

Zudem gibt es noch die Pioneer Roboter. Diese sind fertig gekaufte Roboter von der Firma ActivMedia Robotics (Activmedia, 2002). Die Pioneer Roboter verfügen über eine gute Ausstattung. Informationen zu dem Labor gibt es unter: (von Luck, 2002).

Seit kurzem werden auch Projekte mit mobilen Computern bzw. mit PDA's durchgeführt. In dieser Arbeit wird ein PDA in Kombination mit einem Roboter verwendet.

4.2 Hardware-Umgebung für die Arbeit

4.2.1 Der Roboter

Der Roboter basiert auf einer Plattform, die bereits in meiner Studienarbeit verwendet wurde. Bisher war der Roboter mit einem seriellen Bluetooth-Modul ausge-

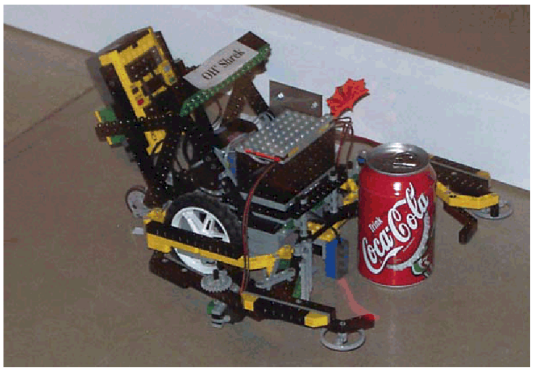

Abbildung 4.1: LEGO MINDSTORM Roboter als Dosensammler

Abbildung 4.2: Pioneer Roboter von ActivMedia

stattet. Da immer mehr mit modellbasierten Architekturen gearbeitet wird, soll der Roboter mehr Ressourcen erhalten. Für modellbasierte Systeme ist mehr Rechenleistung nötig. Daher wird das Bluetooth-Modul durch einen PDA ersetzt. Auf diese Weise verfügt der Roboter über deutlich mehr Möglichkeiten. Der PDA ist mit einer schnurlosen Kommunikation ausgestattet und daher von besonderer Bedeutung. Für Server-gestützte Anwendungen ist die schnurlose Kommunikation von großem Interesse. Um die Anforderungen des Szenarios erfüllen zu können, wird die folgende Ausstattung für den Roboter verwendet.

MIT-6.270-Board

Die Zentrale Hardware des Roboters ist ein MIT-6.270-Board (MIT, 2003). Dies ist eine Einheit, die speziell für den Einsatz mit Robotern entwickelt wurde. Die CPU

ist ein Motorola 68HC11. An dieses Board können diverse Aktuatoren und Sensoren angeschlossen werden. Es können sowohl digitale als auch analoge Sensoren verwendet werden. Als Sensoren kommen z.B. Lichtschranken, Photowiderstände und Taster in Frage.

Das MIT-6.270-Board gehört zur Familie des Handyboard und des LEGO MINDSTORM RCX (LEGO, 2002). Das Handyboard ist der Nachfolger des MIT-6.270-Board.

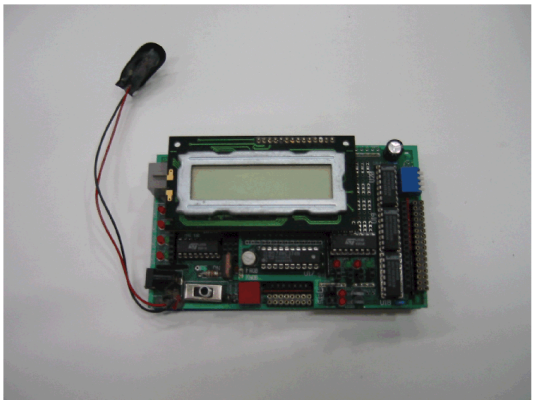

Abbildung 4.3: MIT-6.270-Board

Compaq iPAQ H3970

Der PDA Compaq iPAQ H3970 verfügt über ein eigenes integriertes Bluetooth-Modul. Als Prozessor dient ein Intel XScale mit einer Taktfrequenz von 400 MHz und einem RAM Speicher von 64 MB und einem Flash-ROM Speicher von 32 MB. Das Gerät verfügt über eine serielle Schnittstelle und über eine Infrarot-Schnittstelle. Über die Bluetooth-Verbindung lassen sich z.B. serielle Verbindungen und TCP/IP-Verbindungen erstellen.

4.2.2 Die Plattform

Die Plattform basiert auf einer Differentialsteuerung. Dies bedeutet, dass es ein Pendelrad hinten gibt. Auf der linken und der rechten Seite gibt es jeweils ein angetriebenes Rad. Mit diesen Antriebsrädern kann der Roboter angetrieben und gelenkt werden. Dies ist ein großer Vorteil. Die Plattform ist dadurch sehr wendig. Sie kann um den Mittelpunkt der Antriebsachse gedreht werden.

Abbildung 4.4: Compaq iPAQ H3970

Aktuatoren

Angetrieben werden die Räder über umgebaute Servomotoren, die aus dem Bereich des Modellbaus stammen. Üblicherweise werden sie in Fernsteuermodellen benutzt. In nicht umgebauter Form lassen sich die Servomotoren um einen festen Winkelbetrag drehen. Bestandteile sind ein DC-Motor, ein Getriebe und ein Positionssensor, mit dessen Hilfe über eine Schaltung der Drehwinkel kontrolliert werden kann. Die Positionierung geschieht durch Pulsweitenmodulation (PWM) und die Kontrollinformation durch den Positionssensor. Ein Servomotor ist durch das eingebaute Getriebe relativ kräftig. Ein weiterer Vorteil ist, dass sie sich leicht montieren und ansteuern lassen.

Sensoren

Die Plattform des Roboters ist mit diversen Sensoren ausgestattet.

Bumper 1

Mit Dipp-Schaltern kann man recht einfach Kontakte zum Umfeld registrieren. Für links und rechts wird jeweils ein Schalter verwendet, um Kollisionen mit den Wän-

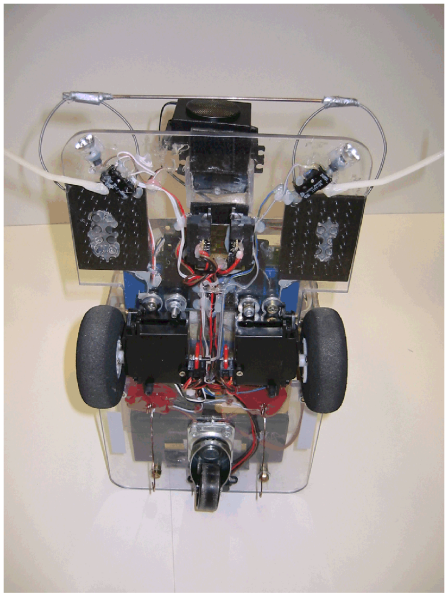

Abbildung 4.5: Roboter in der Ansicht von unten

den zu registrieren. Diese Schalter werden mit kurzen Stücken von Kabelbindern verlängert, um möglichst rechtzeitig eine Kollision zu erkennen.

Bumper 2

Zur Erkennung von Frontal-Kollisionen wird ein weiterer Sensor montiert. Es handelt sich um zwei Drahtschlingen aus Gittarensaiten, die locker auf LEGO Plättchen verklebt sind. Die beiden Drahtschlingen sind mit einem festen Draht verbunden um die Aufprallfläche zu vergrößern. Werden die Drahtschlingen zu sehr durch Druck belastet, so berühren sie einen Drahtkontakt, der dann die Kollision signalisiert.

Shaft Encoder

Zur Messung von Wegstrecken und Rotationen dienen die Shaft-Encoder. Sie bestehen aus kleinen Lochscheiben, die auf derselben Achse wie die Antriebsräder montiert sind. Es gibt in der Regel eine rote LED, die Licht in Richtung der Lochscheibe aussendet. Auf der anderen Seite der Lochscheibe befindet sich ein Phototransistor. Mit diesem werden die Impulse aufgenommen, die durch das Verdecken der LED durch die Lochscheibe entstehen. Die Shaft-Encoder werden an die Ports 0

und 1 angeschlossen. Für das Ansteuern und Auslesen ist eine Programmbibliothek in Interactive C vorhanden.

Sie bietet folgende Funktionen:

```
void enable_encoder(int encoder)
void disable_encoder(int encoder)
void reset_encoder(int encoder)
int read_encoder(int encoder)
```

Photowiderstände

Photowiderstände reagieren auf Licht. Sie verändern ihren Widerstand mit den Lichtverhältnissen. Daher sind sie leicht zu verwenden.

Die Plattform ist mit zwei Photowiderständen ausgerüstet, die auf den Boden gerichtet sind. Damit kann man Unterschiede am Boden erkennen. Am Besten sind sie für schwarzweiß-Unterschiede geeignet.

Zwei weitere Photowiderstände sind vorne links und rechts am Roboter montiert. Sie sind mit kleinen Reflektorschirmen versehen, damit sich eine Beleuchtung stärker auswirkt.

Abstandssensoren (SHARP GP2D12)

Zur Abstandsmessung zu Hindernissen werden Sensoren von der Firma Sharp vom Typ GP2D12 benutzt. Sie haben den großen Vorteil, nicht besonders auf fremdes Licht zu reagieren. Sie arbeiten in einem Bereich von ca. 10 - 80 cm recht zuverlässig. Der Stromverbrauch liegt bei ca. 50 mA/h je Sensor. Versorgt werden die Sensoren über einen Verstärker, der über eine eigene 7,2 V Stromversorgung verfügt. Er besitzt vier Anschlüsse für Sensoren. Die Signale werden über Operationsverstärker an die Ausgänge geleitet und an die analogen Eingänge des 6.270-Boards gelegt.

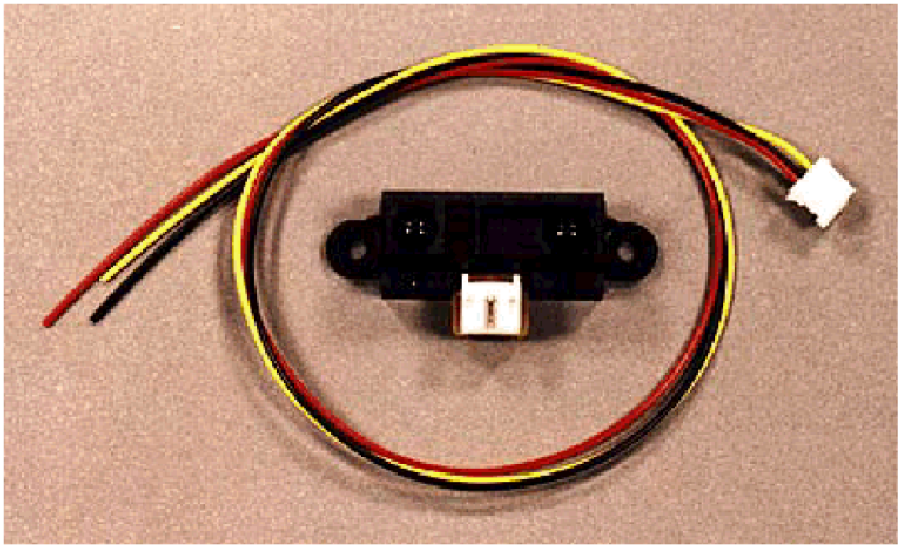

Abbildung 4.6: Abstandssensor (SHARP GP2D12)

Sonar

Außerdem ist es möglich, einen Sonarsensor anzuschließen. In dieser Arbeit wird ein Sensor von der Firma Polaroid verwendet. Es gibt einige Funktionen, die von Oliver Koeckritz und Gunter Klemke (Klemke und Koeckritz, 2002) entwickelt wurden, die den Zugriff auf den Sonarsensor ermöglichen. Der Sensor ermittelt die Entfernung zu einem Hindernis. Auf dieser Plattform ist der Sensor auf einem Servo montiert. Dadurch lässt sich der Sensor in einem Bereich von 180° schwenken.

Der Sonarsensor gehört zu den aktiven Sensoren. Er funktioniert nach demselben Prinzip, nach dem Fledermäuse in der Nacht in ihrer Umgebung navigieren. Ein hochfrequenter Impuls wird ausgesendet und an einer Oberfläche in der Nähe reflektiert. Dieser Impuls kehrt in einer messbaren Zeit zurück. Die Zeitverzögerung kann nun verwendet werden, um die zurückgelegte Strecke zu ermitteln und somit die Distanz zu bestimmen.

Beacon

Ein weiterer Sensor ist ein Infrarotsensor für moduliertes Infrarot-Licht. Er dient zum Finden von so genannten Infrarot-Beacons bzw. Leuchtfeuern. Der Sensor erkennt moduliertes Infrarot-Licht mit einer Trägerfrequenz von 40 kHz. Auf die Trägerfrequenz kann eine Frequenz von 100 Hz oder 125 Hz moduliert werden. Die

Sensoren können an die Ports 4 bis 7 des 6.270-Boards angeschlossen werden. Sie müssen jedoch abgeschirmt werden, damit nicht fremdes Licht das Ergebnis verfälscht.

Bodenkontakte

Zur Erkennung eines Bodenbleches ist die Plattform mit zwei Schleifkontakten ausgestattet. Sie bestehen aus zwei großen, zurecht gebogenen Büroklammern aus Kupfer. Diese Kontakte werden kurzgeschlossen, sobald sich der Roboter auf einem leitfähigen Untergrund befindet.

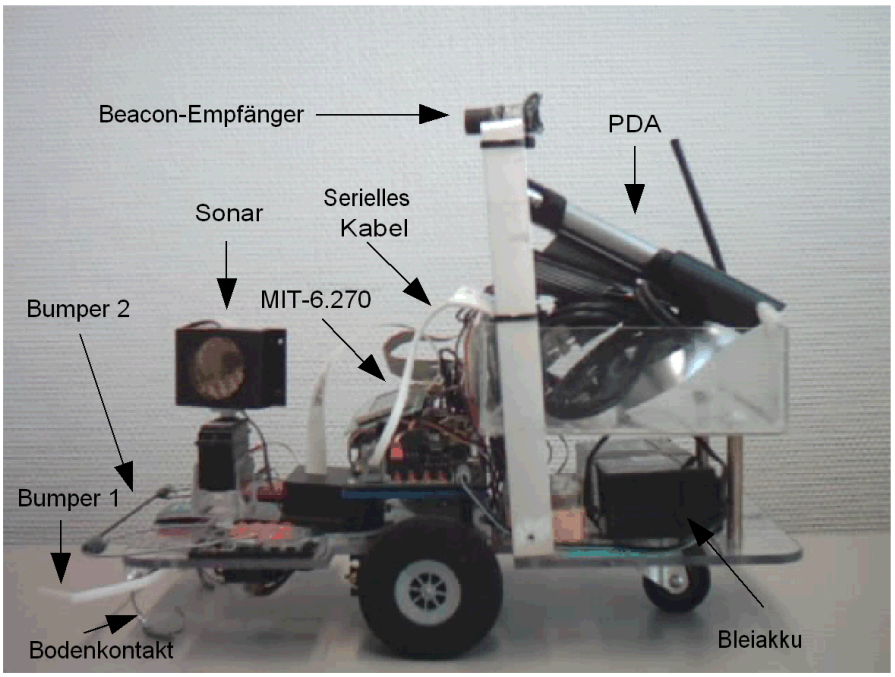

Abbildung 4.7: Seitenansicht des Roboters

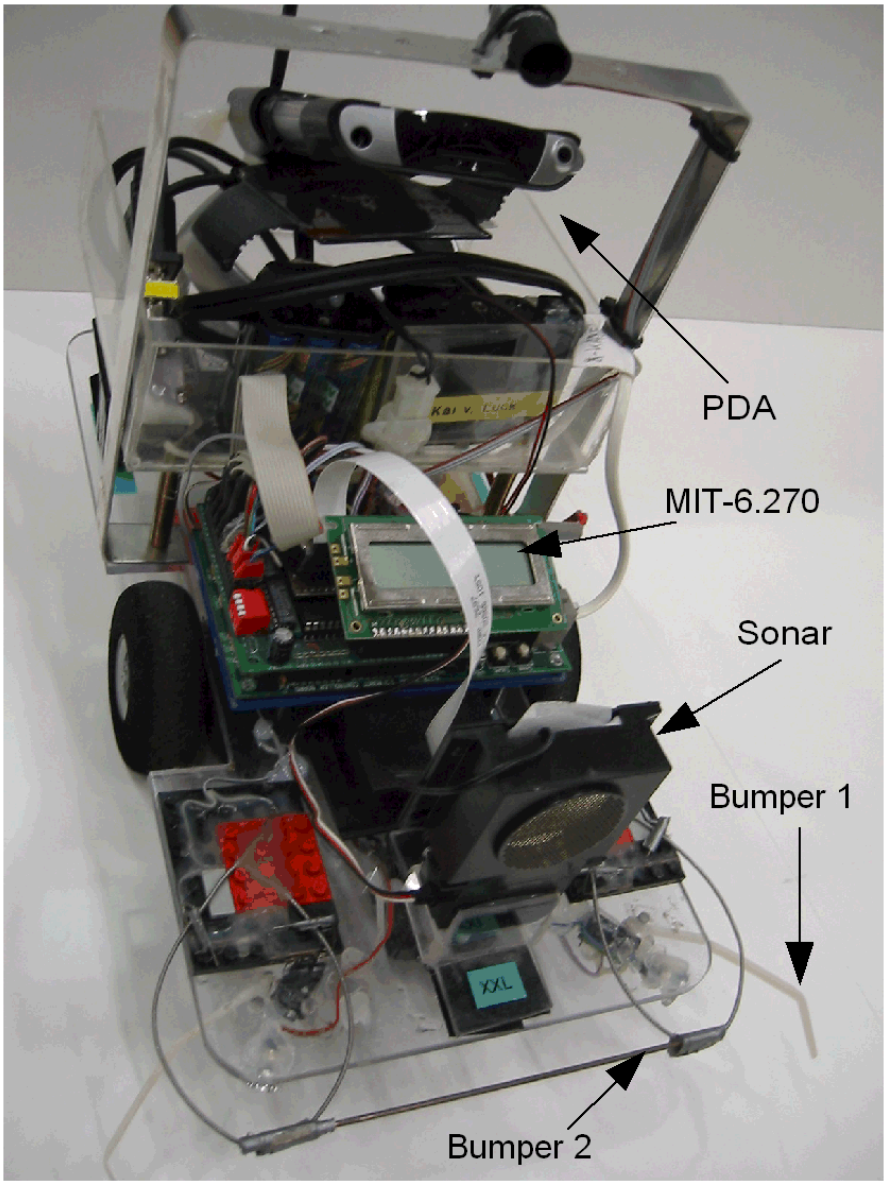

Abbildung 4.8: Frontansicht des Roboters

4.2.3 Hardware für Kommunikation

Die Kommunikation soll über Bluetooth realisiert werden. Der PDA verfügt über ein integriertes Bluetooth-Modul.

Der PC wird mit einem Bluetooth USB-Adapter der Firma Acer ausgestattet. Dieser Adapter hat den großen Vorteil, dass er sein klein ist und wenig Strom verbraucht. Er benötigt keine separate Stromversorgung.

Abbildung 4.9: Acer BT500

4.3 Die Software-Umgebung

Die Programmiersprache Interactive C

Als Entwicklungsumgebung wird das Programm Interactive C Version 3.2 von den Newton Research Labs verwendet. Interactive C ist ein Subset von der Programmiersprache C. Es bietet eine Bibliothek mit diversen Funktionen, die speziell für das 6.270-Board bestimmt sind. Interactive C bietet leider nur einfache Kontrollstrukturen wie for, while, if-else und break. Ein einfaches Multitasking ist auch möglich. Es bietet keine Semaphoren oder ähnliche Mechanismen.

In der Interactive C-Bibliothek gibt es Funktionen für das Einlesen von Sensordaten und die Bereitstellung von Steuerdaten auf die Ausgänge. Es gibt Zeit-, Ton-, Menü- und Diagnose-Funktionen.

Die für die Steuerung des Roboters verwendete Programmiersprache Interactive C bietet diverse Funktionen. Zu den wichtigsten für diese Arbeit zählen z.B. die Funktionen für das Shaftencoding. Hiermit lassen sich die Wegstrecken bestimmen.

```
void enable_encoder(int encoder)
void disable_encoder(int encoder)
void reset_encoder(int encoder)
int read_encoder(int encoder)
```

Zusätzlich sind die Funktionen für die Beaconerkennung von Bedeutung. Die Funktionen sind für den Empfang von Infrarot Signalen bestimmt.

```
void ir_receive_on()
void ir_receive_off()
void set_ir_receive_frequency(int f)
int ir_counts(int p)
```

Für die serielle Kommunikation werden folgende Befehle für den Speicherzugriff benötigt.

Zur Aktivierung wird die Funktion

```
void poke(int loc, int byte)
```

und zum Senden und Auslesen wird die Funktion

```
int peek(int loc)
```

verwendet.

Die Entwicklungsumgebung von Interactive C bietet eine grafische Oberfläche zur Entwicklung und Übertragung von Programmen auf den Roboter. Es gibt ein Interaction-Window, in dem Statusmeldungen während der Übertragung von Code zu sehen sind. Im unteren Bereich des Interaction-Window gibt es einen kleinen Abschnitt, in dem man direkt Funktionen aufrufen kann und damit die Funktionstüchtigkeit des Roboters überprüfen kann.

Die Entwicklungsumgebung versucht beim Start das Board des Roboters anzusprechen und den PCode für den Grundbetrieb des Boards herunterzuladen. Ist kein Board vorhanden, so kann man in den Einstellungen angeben, dass das Board ignoriert werden soll. So kann man auch ohne Board an dem Code arbeiten.

Weitere Funktionalitäten der Entwicklungsumgebung von Interactive C sind einfache Editorfunktionen wie Find, Replace und Goto Line. Eine Hilfe zu den Funktionen wird ebenfalls angeboten. Die hier verwendete Version der Software ist die Version 3.2 (Labs, 2002).

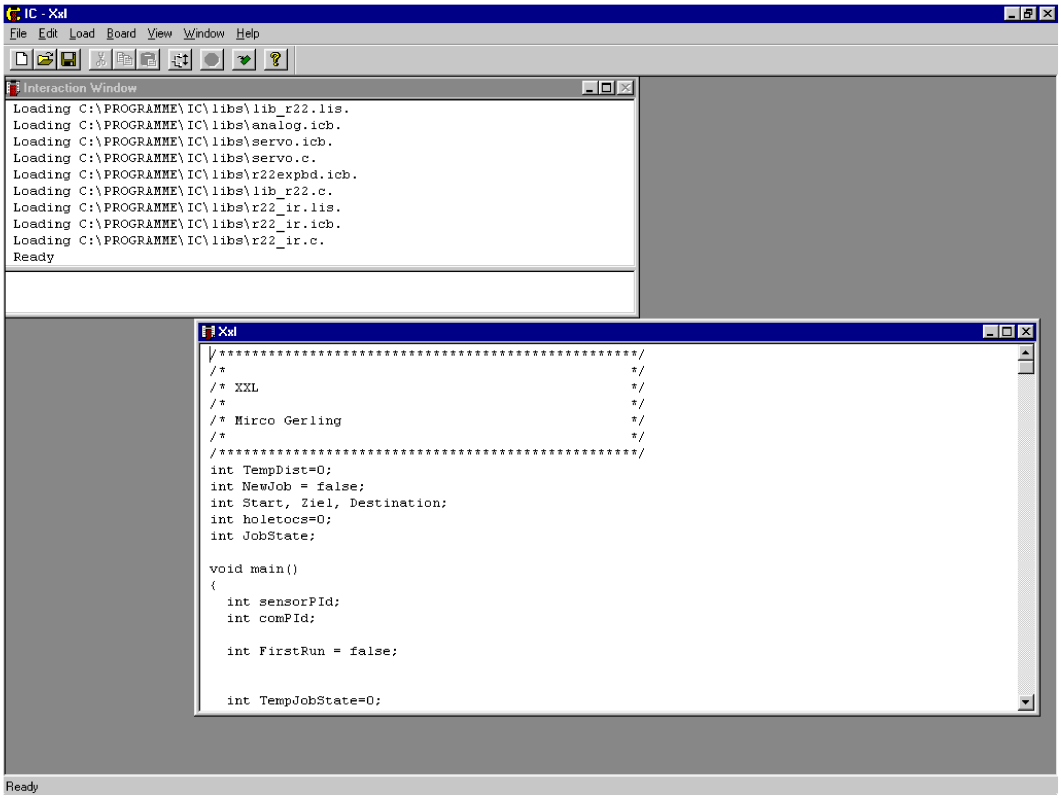

Abbildung 4.10: Entwicklungsumgebung Interactive C

Microsoft Visual C++ 6.0

Zur Entwicklung der Windows-Anwendung für den Server wird Visual C++ 6.0 von Microsoft benutzt. Mit dieser Entwicklungsumgebung lassen sich recht einfach und schnell Visual C++-Anwendungen für Windows erstellen. Für die Entwicklung werden die Microsoft Foundation Classes (MFC) (Microsoft, 2003) benötigt. Die MFC stellen unter anderem besondere Funktionalitäten für die Verwendung von Sockets zur Verfügung. Die Abbildung 4.11 zeigt die grafische Oberfläche der Entwicklungsumgebung.

Microsoft Embedded Visual C++ 3.0

Für die Entwicklung der Software für den PDA wird die kostenlose Entwicklungsumgebung Embedded Visual Tools 3.0 benutzt. Diese Entwicklungsumgebung er-

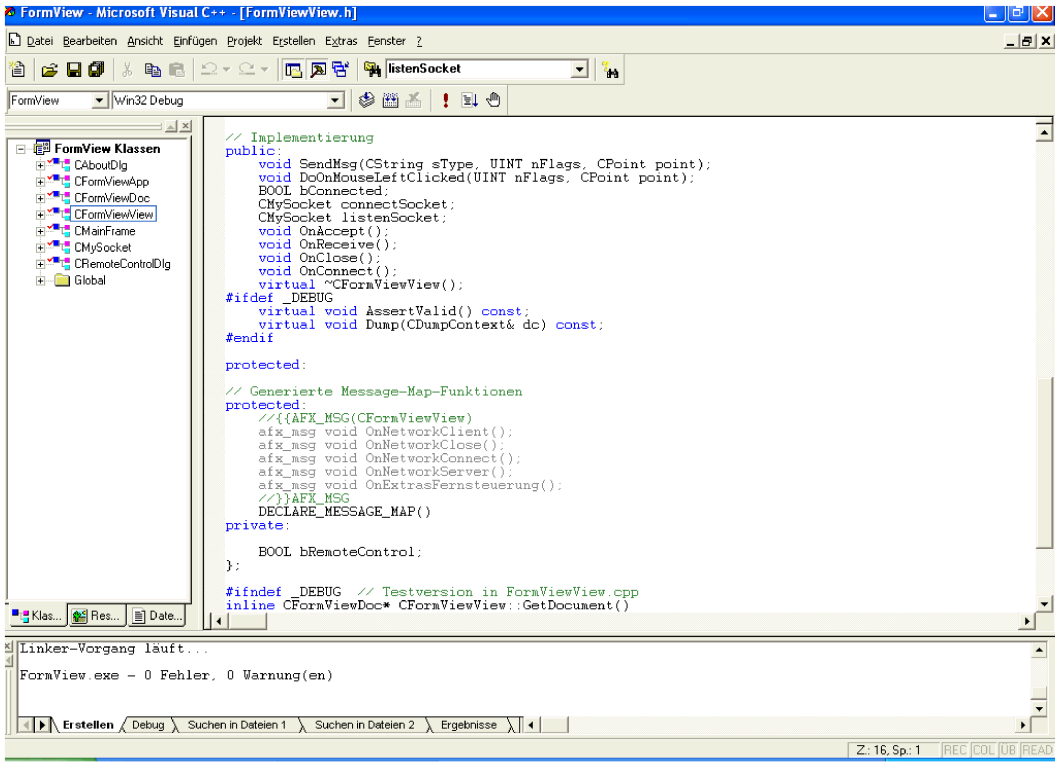

Abbildung 4.11: Entwicklungsumgebung Visual C++ 6.0

möglicht die Entwicklung von Windows CE Anwendungen mit Embedded Visual Basic und Embedded Visual C++. In dieser Arbeit wird Embedded Visual C++ 3.0 verwendet, da es eine hardwarenahe und performante Programmierung bietet. Auch hier werden die MFC für Windows CE verwendet. Außerdem wird die Pocket PC 2002 SDK benötigt. Es gibt bereits eine neuere Version der Entwicklungsumgebung. Leider unterstützt diese nicht die Pocket PC 2002 SDK.

Da die Entwicklungsumgebung speziell für Entwicklung von Software für mobile Geräte gedacht ist, gibt es die Möglichkeit den Code bzw. die ausführbare Datei auf das Gerät herunterzuladen. Die Abbildung 4.12 zeigt die grafische Oberfläche der Entwicklungsumgebung von Embedded Visual C++ 3.0.

High Point BTAccess

Für den dynamischen Auf- und Abbau von Bluetooth-Verbindungen wird für den PDA eine zusätzliche Bibliothek benötigt. Die Bibliothek BTAccess von der Firma

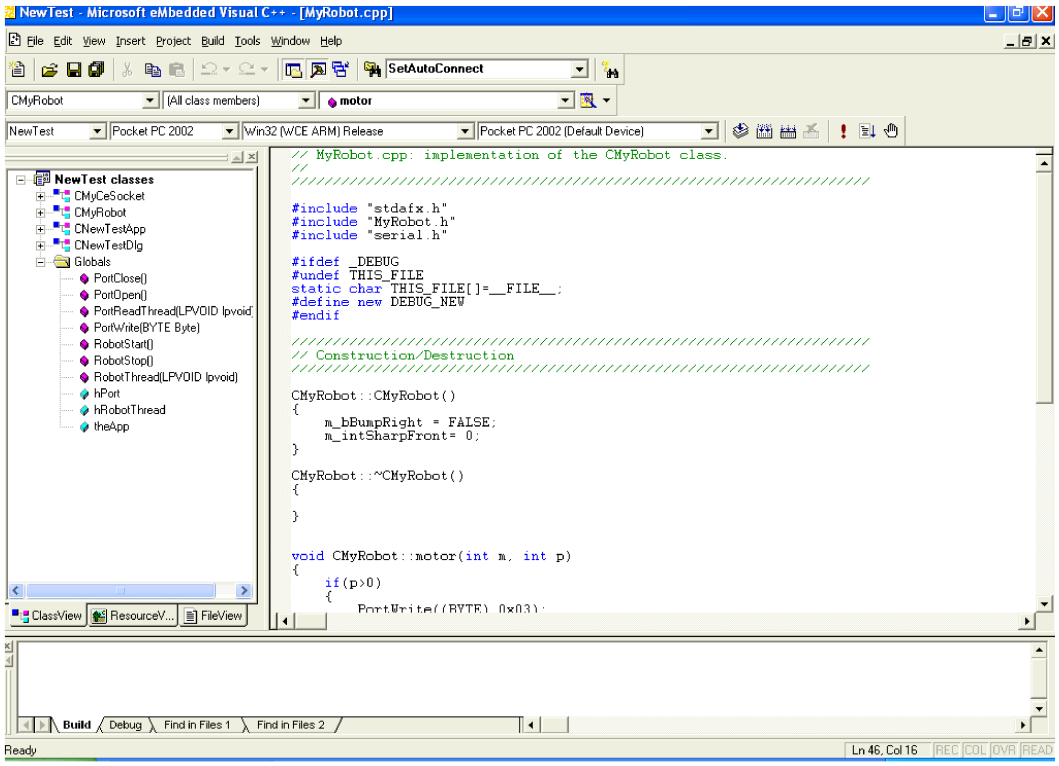

Abbildung 4.12: Entwicklungsumgebung Embedded Visual C++ 3.0

High Point Software (BTAccess, 2003) bietet für den Compaq iPAQ H3970 Funktionalitäten um das Bluetooth-Modul zu bedienen. Es ist möglich das Bluetooth-Modul ein- und auszuschalten und dynamisch nach anderen Bluetooth-Geräten zu suchen. Wurde ein Gerät gefunden, so ist es möglich die angebotenen Dienste des Gerätes zu verwenden. Wird z.B. ein LAN-Access angeboten, so ist es möglich eine Netzwerkverbindung mit dem Gerät aufzubauen und auch später wieder abzubauen. Eine weitere nützliche Funktionalität ist die Anzeige der Sendeleistung des anderen Gerätes.

Microsoft ActiveSync

Microsoft ActiveSync ist eine Software, die es ermöglicht zwischen einem PC und einem PDA oder Mobiltelefon persönliche Daten zu synchronisieren. Dies können Termine, Adressen oder E-Mails sein. ActiveSync benutzt z.B. eine serielle Verbindung um die Daten zu synchronisieren. Dies lässt sich über ein serielles Kabel bzw.

über ein USB-Kabel machen oder aber über das serielle Bluetooth-Profil der Geräte. Über die ActiveSync-Verbindung ist es zusätzlich möglich eine TCP/IP-Verbindung zu erstellen. Diese Verbindung lässt sich verwenden um eine Netzwerkverbindung zwischen den Sockets von Client und Server zu erstellen.

Bluetooth-Software auf dem PC

Auf dem PC mit einem Bluetooth-Modul läuft ein Bluetooth-Stack. Dieser Stack ist für den Auf- und Abbau von Bluetooth-Verbindungen zuständig. Die Software stellt z.B. virtuelle COM-Ports zur Verfügung um eine serielle Verbindung über Bluetooth aufzubauen.

Kapitel 5

Design und Realisierung

5.1 Das Gesamtsystem

Das Gesamtsystem besteht aus drei Komponenten. Ein Controller-Programm, welches auf dem Controller-Board läuft. Ein Planning-Programm, welches auf dem PDA läuft und ein Server-Programm, welches auf einem PC läuft. Die Kommunikation zwischen dem Controller-Board und dem PDA soll über eine serielle Verbindung erfolgen. Die Kommunikation zwischen dem PDA und dem PC soll über eine TCP/IP-Verbindung stattfinden.

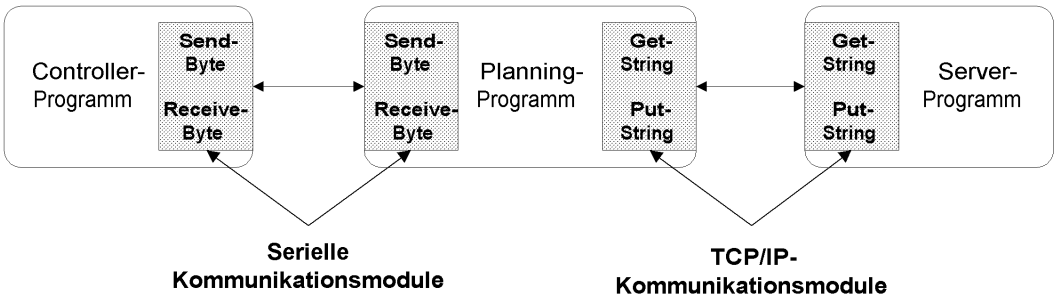

Abbildung 5.1: Die Komponenten des Systems

Für die Verteilung der Aufgaben der einzelnen Komponenten gibt es mehrere Alternativen. An dieser Stelle sollen die verschiedenen Möglichkeiten der Aufgabenverteilung diskutiert werden. Jede Alternative hat seine gewissen Vor- und Nachteile, die hier erläutert werden sollen. Für welche Variante man sich entscheidet, hängt hauptsächlich von der Art der Anwendung ab.

5.1.1 Aufgabenverteilung Controller- <-> Planning-Programm

Die Verteilung zwischen Controller-Programm und Planning-Programm gibt Anlass zur Diskussion. Das hängt damit zusammen, dass sie beide sehr ähnliche Aufgaben erfüllen können und beide nahezu den gleichen Bedingungen ausgesetzt sind. Viele Aufgaben kann das Controller-Programm selbst erledigen, jedoch kann es Sinn machen, gewisse Aufgaben an das Planning-Programm weiter zu geben. So hat das Planning-Programm mehr Informationen zur Verfügung, auf die es zurückgreifen kann. Je mehr Funktionalitäten vom Controller-Programm in das Planning-Programm verlegt werden, desto mehr Datenfluss entsteht auch zwischen den beiden Programmen. Dies kann bei zeitkritischen Abläufen zu einem Problem werden.

Durchreichen der vollen Interactive C-Funktionalität

Eine Möglichkeit ist es, in dem Planning-Programm die volle Funktionalität der Interactive C-Bibliothek abzubilden. So ist es möglich den vollen Funktionsumfang an das Planning-Programm weiter zu reichen. Diese Lösung bringt allerdings ein sehr hohes Datenaufkommen mit sich. Das hängt damit zusammen, dass die Rückgabewerte von Funktionen und sämtliche Sensorwerte, die benötigt werden, über die serielle Verbindung an das Planning-Programm übertragen werden müssen. Außerdem werden sämtliche Funktionsaufrufe über die serielle Verbindung vom Planning-Programm zum Controller-Programm übertragen. Diese Lösung hat den Vorteil, dass der volle Funktionsumfang von Interactive C dem Planning-Programm zur Verfügung steht. Ein Nachteil kann sein, dass die Reaktionsgeschwindigkeit nicht besonders hoch ist. So sollten zum Beispiel Kollisionen mit Hindernissen nicht vom Planning-Programm behandelt werden, sondern von dem Controller-Programm. In diesem Fall braucht das Controller-Programm überhaupt keine Kenntnisse von der Welt zu haben, in der sich der Roboter bewegt. In dem Controller-Programm ist ein rein reaktives Verhalten implementiert. Die innere Karte von der Umgebung ist auf das Planning-Programm verlagert.

Steuerung über Vektordaten

Bei der Steuerung über Vektordaten bietet das Controller-Programm zum Beispiel eine Funktion an, um sich einen Sensorvektor zu holen. Das bedeutet, dass man eine ganze Reihe von Sensorwerten geliefert bekommt. Diese Sensorwerte können dann ausgelesen und ausgewertet werden. Daraufhin trifft das Planning-Programm eine Entscheidung, wie die Motoren anzusteuern sind. Eine weitere Funktion könnte es ermöglichen nun einen Motorvektor vom Planning-Programm zu Controller-Programm zu übertragen. Das Controller-Programm liest diesen Motorvektor aus und steuert dementsprechend die Motoren an. Bei dieser Lösung fallen nicht ganz so viele Daten an, da es nicht so viele Funktionsaufrufe gibt. Das hängt damit zusammen, dass nicht für jeden Sensorwert eine Funktion aufgerufen wird, sondern eine Funktion alle Sensorwerte liefert. Ein weiterer Grund ist, dass nicht für jeden Motor eine einzelne Funktion aufgerufen wird, sondern ebenfalls ein Vektor übertragen wird und daraufhin die Motoren gesteuert werden. Auch in diesem Fall hat das Controller-Programm keine Kenntnis von der Umgebung. Das Verhalten ist rein reaktiv.

Abstraktion der Funktionalitäten

Eine weitere Möglichkeit besteht darin, die Funktionalitäten in einer höheren Abstraktion anzubieten. So könnte das Controller-Programm Funktionen anbieten, wie zum Beispiel `turn90()`, `turn45()` oder `followwalluntil()`. Somit wäre es möglich dem Controller-Programm nur den Befehl zu geben, den Roboter um einen bestimmten Winkel zu drehen oder einer Wand zu folgen, bis beispielsweise eine Abrisskante oder eine Einbuchtung auftritt. Diese Lösung hat den Vorteil, dass der Datenfluss gering ist. Das Controller-Programm braucht nur Rückmeldungen zu senden, ob die gewünschte Aktion ausgeführt wurde oder nicht. Jedoch werden in diesem Fall auch keine Sensorwerte an das Planning-Programm übertragen. So lassen sich diese auch nicht in dem Planning-Programm weiterverarbeiten. Auch in diesem Fall braucht das Controller-Programm keine Kenntnisse von der Umwelt zu haben. Die innere Karte ist auch hier in dem Planning-Programm angesiedelt.

Steuerung durch Ortsangaben oder Landmarken

Eine noch höhere Abstraktionsebene wäre die Lösung, indem das Planning-Programm dem Controller-Programm den Befehl gibt, zu einem bestimmten Ort

zu fahren. Dies könnte zum Beispiel der Befehl sein, zu einer der beiden Tankstellen zu fahren. In diesem Fall müsste das Controller-Programm eine Karte der Welt, in der sich der Roboter bewegt, führen. So wäre es möglich das Controller-Programm nach einer Abschätzung zu fragen, wo sich vermutlich der Roboter befindet. Hier ist der Datenfluss sehr gering, allerdings lasten auch viele Aufgaben auf dem Controller-Programm.

Eine andere Variante der Steuerung als über Ortsangaben wäre die Steuerung über Landmarken. Eine Landmarke für eine Tankstelle könnte ein Infrarot Leuchtfeuer mit einer bestimmten Frequenz sein. So könnte das Planning-Programm dem Controller-Programm einen Befehl geben, wie "Fahre zum nächsten Leuchtfeuer mit der Frequenz 125 Hz". Eine weitere Landmarke kann eine Station sein. Die Station hat höhere Wände als die Trennwände des Parcours und verfügt über einen Metallboden. So könnte das Planning-Programm dem Controller-Programm den Befehl geben: "Fahre, bis du hohe Wände siehst und Metall unter dir hast". Bei dieser Variante würde die innere Karte in dem Planning-Programm abgelegt sein. So würde das Controller-Programm keine Informationen bereithalten, wo sich der Roboter gerade befindet.

Mischung aus voller Funktionalität und Erweiterung

In dieser Arbeit soll die volle Funktionalität von Interactive C durchgereicht werden. Zusätzlich soll es möglich sein, die Funktionalität durch eigene Funktionen zu erweitern. So ist es möglich die volle Kontrolle über sämtliche Abläufe zu haben und zusätzlich auf einer höheren Abstraktionsebene Programme zu entwickeln.

Die Erweiterung über die Funktionalitäten von Interactive C hinaus bietet die Möglichkeit des situativen bzw. dynamischen Abstrahierens. Im normalen Betrieb können auf einer hohen Abstraktionsebene die Erweiterungen benutzt werden. Tritt eine Ausnahmesituation ein, ist es auch möglich, bis auf die Sensorwerte "hinunter" zu gehen um genau die Situation beurteilen zu können. Ist die Ausnahmesituation beseitigt, so kann wieder auf eine höhere Abstraktionsebene übergegangen werden.

5.1.2 Aufgabenverteilung Planning- <-> Server-Programm

Bei der Aufgabenverteilung zwischen Planning-Programm und Server-Programm gibt es ebenfalls Alternativen.

Steuerung durch Planning-Programm

Eine Möglichkeit ist, dass die Steuerung von dem Planning-Programm übernommen wird. Das Planning-Programm kann dem Server-Programm Funktionen anbieten, die es dem Server ermöglichen, Aufträge in eine priorisierte Warteschlange zu stellen, den Status dieser Aufträge abzufragen und die momentane Position des Roboters zu erfragen, sofern er sich in der Reichweite eines Servers befindet. Der Vorteil bei dieser Lösung ist, dass keine permanente Funkverbindung notwendig ist.

Steuerung durch Server-Programm

Es ist auch denkbar, dass der Roboter vom Server aus gesteuert wird. Dieses Vorgehen ist jedoch für zeitkritische Vorgänge nicht besonders geeignet. Für planerische Vorgänge kann dies durchaus sinnvoll sein. Wenn zum Beispiel mehrere Roboter im Einsatz sind und über einen Kameraserver Informationen über die aktuellen Positionen der Roboter vorhanden sind. So könnte das Server-Programm die strategische Steuerung der Roboter übernehmen. Ein Nachteil bei dieser Variante ist, dass die ganze Zeit eine Funkverbindung bestehen muss.

5.2 Das mögliche dynamische Zusammenspiel der Komponenten

In diesem Abschnitt sollen zwei Möglichkeiten für das Zusammenspiel der Komponenten gezeigt werden. Dies soll mit der Hilfe von zwei Sequenzdiagrammen veranschaulicht werden. In beiden Abbildungen werden anfangs Daten mit dem Server ausgetauscht. Anschließend soll der Roboter einen anderen Ort ansteuern. Die Abbildung 5.2 zeigt die Verwendung einer niedrigen Abstraktion. Hier werden die Motoren direkt durch Befehle vom PDA beeinflusst.

Die Abbildung 5.3 zeigt die Verwendung einer hohen Abstraktion. Hier wird dem Controller-Programm nur ein Befehl gegeben, zu einem bestimmten Ziel zu fahren. Die Ausführung wird direkt von dem Controller-Programm vorgenommen.

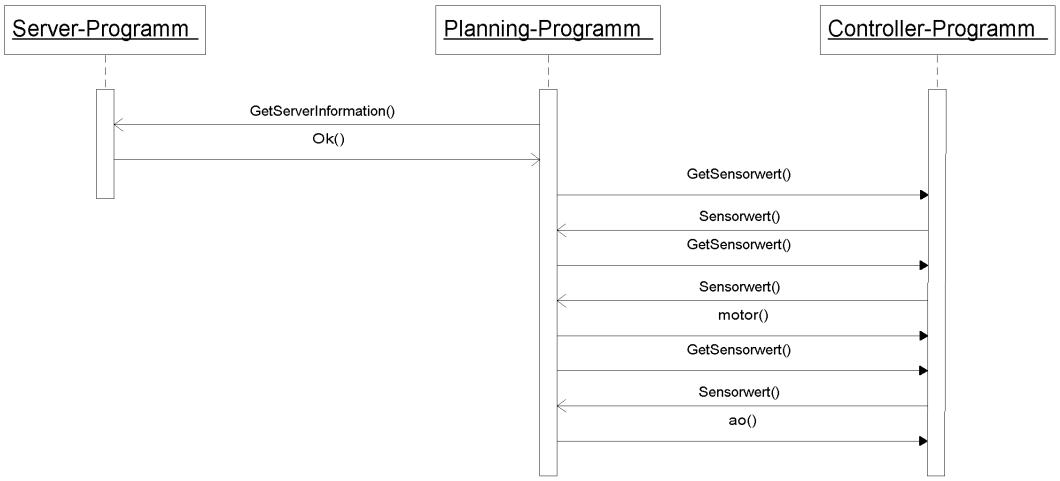

Abbildung 5.2: Sequenzdiagramm direkte Ansteuerung (Konkret)

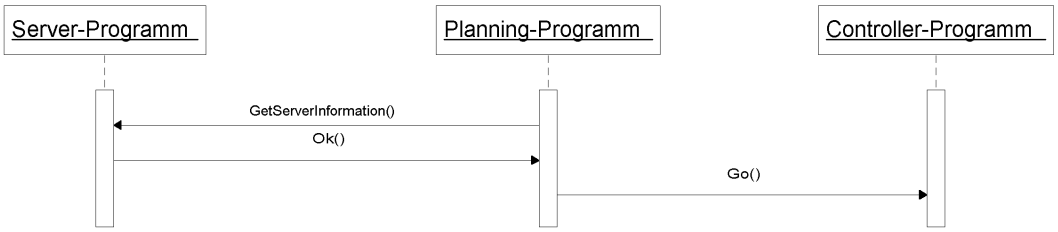

Abbildung 5.3: Sequenzdiagramm abstrakte Ansteuerung (Abstrakt)

5.3 Die Roboter-Klasse

Auf dem PDA wird der Roboter durch eine abstrakte Klasse repräsentiert. Die Klasse stellt alle Funktionen und Eigenschaften des Roboters zur Verfügung. Durch den Aufruf von Methoden der Klasse können Funktionen auf dem Controller-Board aufgerufen werden. Um die Eigenschaften des Roboters in der Klasse darzustellen, werden vom Controller-Board Daten in einer festgelegten Reihenfolge an den PDA gesendet und in entsprechende Member-Variablen der Klasse geschrieben.

Da die Kommunikation über eine serielle Verbindung erfolgt, wird ein Protokoll benötigt. Dieses Protokoll sollte in der Lage sein, eine schnelle und fehlertolerante Übertragung zu gewährleisten. Da die Geschwindigkeit besonders wichtig ist, wird auf ein aufwändiges Protokoll verzichtet. Die Überprüfung der Daten sollte durch eine Checksummenprüfung erfolgen.

Wird auf dem PDA eine Methode der Roboter-Klasse aufgerufen, so wird eine Folge von Bytes aufbereitet und über die serielle Schnittstelle an das Controller-Board übertragen. Um die Befehle auswerten zu können, ist ein Bytecode-Interpreter nötig. Dieser Bytecode-Interpreter wertet Byte für Byte aus und ruft dann die entsprechende Funktion auf dem Controller-Board auf. Der Bytecode-Interpreter besteht aus einem einfachen if-else-Konstrukt bzw. aus einem switch-case-Konstrukt.

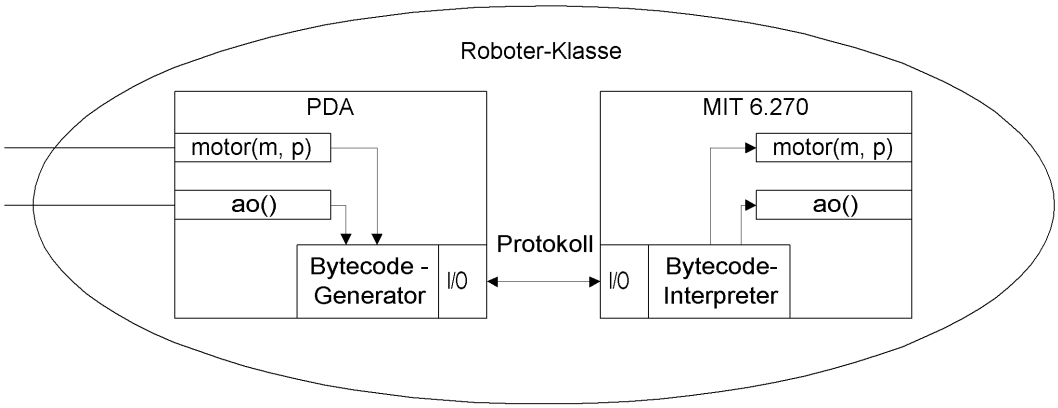

Abbildung 5.4: Roboter-Klasse: Übertragung zum Controller-Board

Die Klasse stellt, von außen betrachtet, eine abstrakte Schnittstelle als Vertreter für das Controller-Board zur Verfügung. Die Realisierung der Klasse ist auf den PDA und auf das Controller-Board verteilt.

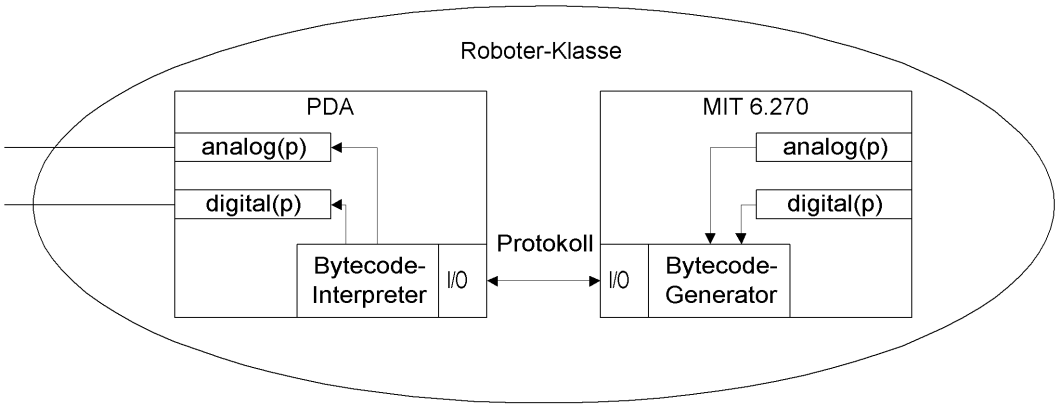

Abbildung 5.5: Roboter-Klasse: Übertragung zum PDA

Sicht von außen

Die Klasse stellt alle Funktionen der Interactive C-Bibliothek als öffentliche Methoden der Klasse zur Verfügung. Die Sensorwerte werden als Attribute der Klasse dargestellt. So ist es möglich ein Objekt der Klasse zu erzeugen und dieses Objekt zu manipulieren. Die Sensorwerte können über Methoden gelesen werden. Für jeden Sensorwert gibt es eine Methode zum Lesen. Die Sensorwerte werden als "private" deklariert, damit die Werte nicht manipuliert werden können.

Wie in Kapitel 5.1.1 bereits diskutiert, soll es möglich sein, die Interactive C-Bibliothek zu erweitern. So werden zu den durchgereichten Bibliotheksfunktionen eigene Erweiterungen ermöglicht. Sollten für eine Aufgabenstellung die Standardfunktionen der Interactive C-Bibliothek nicht ausreichen, so kann die Klasse um eigene Methoden erweitert werden. Zum Beispiel könnte die Klasse um eine Methode mit dem Namen wallfollow() erweitert werden. Ob die Methode auf dem PDA oder auf dem Controller-Board implementiert wird, ist variabel.

Sicht von innen

Serielle Kommunikation

Über die serielle Verbindung soll es möglich sein, einzelne Bytes zu senden und zu empfangen. Das Senden wird benötigt, um Befehle an den Roboter zu senden.

Das Empfangen wird benötigt um Sensorwerte und evtl. auch Rückgabewerte vom Controller-Board zu empfangen.

Bevor Daten gesendet und empfangen werden können, muss die serielle Schnittstelle geöffnet werden. Es müssen einige Einstellungen vorgenommen werden, wie z.B. die Übertragungsgeschwindigkeit und die Anzahl der Bits. Damit Daten empfangen werden können, wird ein separater Thread benötigt. Dieser Receive-Thread wartet auf das Eintreffen von Daten. Werden Daten empfangen, dann werden diese aus dem Empfangspuffer der seriellen Schnittstelle ausgelesen und weiterverarbeitet.

Die Methoden

Wie bereits erwähnt ist das Innere der Klasse auf den PDA und das Controller-Board verteilt. Auf der Seite des PDA befinden sich die Methoden der Klasse. Diese Methoden generieren einen Bytecode. Dieser besteht aus einem Byte, welches den Start der Übertragung signalisiert. Dieses Start-Byte dient auch der Synchronisation, falls ein Übertragungsfehler auftritt. Darauf folgt ein Byte für die eindeutige Identifizierung der gewünschten Funktion. Anschließend folgen eventuelle Parameter der Funktion. Abschließend folgt dann die Checksumme. Dieser Bytecode wird Byte für Byte an eine Funktion übergeben. Diese Funktion sendet die Daten über die serielle Verbindung an das Controller-Board. Diese Sende-Funktion verfügt über einen Timeout-Zähler. Sollte die Übertragung scheitern, so kehrt die Funktion nach Ablauf einer bestimmten Zeit mit einem Fehlerwert zurück.

Wird vom Controller-Board das Start-Byte empfangen, so springt der Bytecode-Interpreter in seinen Anfangszustand. Der Bytecode-Interpreter besitzt einen Zähler. Dieser Zähler wird mit jedem empfangenen Byte hoch gezählt. Der Zähler wird auf den Wert Null gesetzt, wenn das Start-Byte empfangen wurde. Wird das nächste Byte empfangen, so wird der Zähler um Eins erhöht. Auf diese Weise lässt sich feststellen, welche Bedeutung die jeweiligen Bytes haben. Eine abschließende Kontrolle der Checksumme dient zur Überprüfung, ob die Daten richtig angekommen sind. Sind alle Bytes korrekt angekommen, so wird die entsprechende Funktion auf dem Controller-Board aufgerufen.

Auf diese Weise wird z.B. bei dem Aufruf der Methode `motor(m, p)` ein Byte 0xff als Start-Byte, 0x03 für die Funktion `motor(m, p)`, m = 0 für die Auswahl des Motor-Ports und p = 100 für die Drehzahl des Motors übertragen.

Start-Byte	Ident-Byte	Parameter 1	Parameter 2
0xff	0x03	0	100

Tabelle 5.1: Bytefolge bei Aufruf einer Methode der Roboter-Klasse

Wird in diesem Fall eine negative Drehzahl angegeben, so wird statt einer 0x03 eine 0x04 übertragen, da auf die Weise keine negativen Zahlen übertragen werden. Dadurch wird auf dem Controller-Board ebenfalls die Funktion `motor(m, p)` aufgerufen. Der Unterschied ist jedoch, dass der Wert p für die Drehzahl negiert wird.

Die Sensorwerte

Die Aufbereitung der Sensorwerte ist ebenfalls auf den PDA und das Controller-Board verteilt. Die Sensorwerte werden von dem Controller-Programm bereitgestellt. Ein separater Sensor-Prozess speichert die Werte in einer festgelegten Reihenfolge in einem Array ab. Die Sensorwerte werden in dieser Reihenfolge an das Planning-Programm gesendet. Diese "Sensorleiste" kann ganz oder teilweise übertragen werden. Das heißt, es können alle analogen und digitalen Eingänge des Controller-Boards übertragen werden. Alternativ können z.B. nur die analogen oder nur die digitalen Eingänge übertragen werden. Eine weitere Unterteilung stellt die Übertragung der unteren oder der oberen Hälfte der Sensorleiste dar. Welches Spektrum von Sensorwerten übertragen wird, kann über Methoden beeinflusst werden.

Die Roboter-Klasse besitzt einen einfachen Bytecode-Interpreter, der in einer switch-case-Struktur gehalten ist. Es gibt ein Start-Byte zur Synchronisation und einen Zähler, der die Anzahl der übertragenen Bytes zählt. Das Start-Byte ist das erste Byte in dem Array. Da jedes Byte eine feste Position hat, können die Werte in die entsprechenden Variablen der Klasse geschrieben werden.

Eines ist zu beachten. Der Wert des Start-Bytes darf nicht als Sensorwert auftauchen. Ist das der Fall, dann setzt der Bytecode-Interpreter des Planning-Programms seinen Zähler zurück. Dadurch befindet sich der Bytecode-Interpreter wieder in seinem Anfangszustand. Aus diesem Grund müssen die Sensorwerte vor ihrer Übertragung auf ihren Wert überprüft werden. Hat ein Sensor den Wert 0xff, so wird er auf den Wert 0xfe gesetzt. Durch diese Maßnahme ist der Wertebereich der Sensoren um einen Wert eingeschränkt. Da insgesamt noch 255 Werte verbleiben, kann

Start-Byte	**Sensor 1**	**Sensor 2**	**Sensor n**
0xff	SensorFront	SensorBump	Sensor n

Tabelle 5.2: Bytefolge der Sensorwerte

dies vernachlässigt werden. Außerdem nutzen die meisten Sensoren in der Praxis den vollen Wertebereich gar nicht aus.

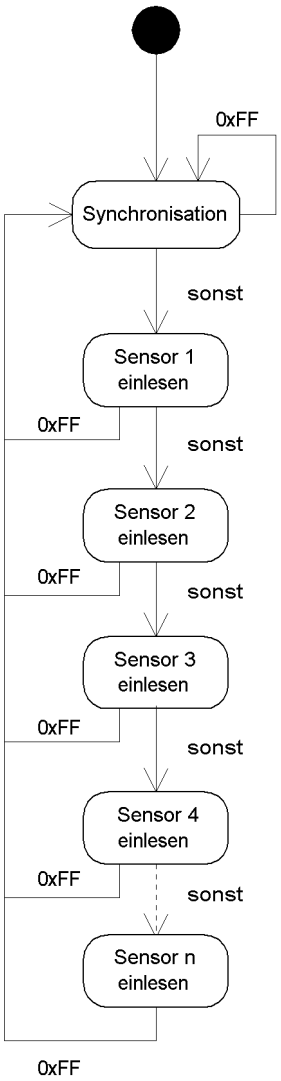

Abbildung 5.6: Speicherung der Sensorwerte in der Roboter-Klasse

Erweiterbarkeit der Klasse

Erweiterung um Methoden

Es gibt zwei unterschiedliche Möglichkeiten, die Klasse mit eigenen Methoden zu erweitern. Es gibt die Möglichkeit eine Methode in dem Planning-Programm zu implementieren. In diesem Fall kann man die Methode der Klasse hinzufügen und durch den Aufruf bereits vorhandener Methoden die neue Methode implementieren.

Die zweite Möglichkeit besteht darin, eine Methode in dem Controller-Programm auf dem Controller-Board zu implementieren. Für diese Möglichkeit sind drei Schritte nötig:

1. Die Methode muss der Klasse im Planning-Programm hinzugefügt werden. In der Methode wird der Aufruf der Funktion auf dem Controller-Board implementiert. Hierbei muss ein Bytecode ausgesucht werden, der noch nicht für eine andere Methode verwendet wird.
2. Der Bytecode-Interpreter des Controller-Programms muss angepasst werden, damit ihm die Methode bekannt ist.
3. Die Funktion muss im Controller-Programm implementiert werden.

Für welche Art der Implementierung man sich entscheidet, hängt von der Art der Anwendung ab. Geht es um zeitkritische Vorgänge, dann ist eine Implementierung in dem Controller-Programm vorzuziehen.

Erweiterung um Sensorwerte

Soll die Klasse um einen Sensorwert erweitert werden, dann muss der Klasse eine Member-Variable hinzugefügt werden. Diese sollte als "privat" deklariert werden. Bei dieser Vorgehensweise ist es nötig, eine öffentliche Methode zu erstellen. Die Methode gibt den Wert der Membervariablen zurück. Der Bytecode-Interpreter muss für den Empfang eines weiteren Bytes erweitert werden. Auf der Seite des Controller-Programms muss das Sensor-Array um einen Wert vergrößert werden. Im Sensor-Prozess muss der neue Sensorwert eingepflegt werden, damit der Sensorwert an die neue Stelle des Arrays geschrieben wird. Beim Senden der Sensorwerte muss die neue Größe des Arrays berücksichtigt werden. Dies kann durch eine dynamische Ermittlung der Arraygröße vernachlässigt werden.

5.4 Design des Controller-Programms

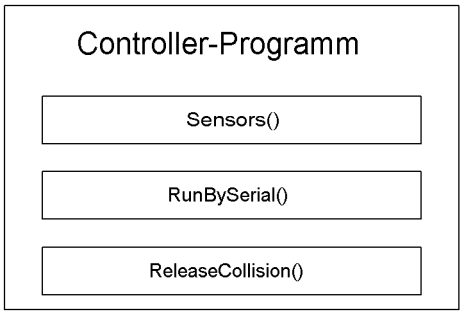

Abbildung 5.7: Prozesse auf dem Controller-Board

Auf dem Controller-Board laufen drei Prozesse parallel ab. Es gibt einen Sensor-Prozess, der die Sensorwerte permanent ausliest und in globale Variable schreibt. Die anderen zwei Prozesse laufen konkurrierend parallel, wie in der Subsumption-Architektur beschrieben. Der zweite Prozess ist ein Keep-Alive-Prozess. Er ist für die Kollisionen zuständig. Dieser Prozess wird benötigt, da die Kollisionen in Realzeit behandelt werden sollten. Er ist für die "lebenserhaltenden" Basisfunktionalitäten zuständig. Dieser Prozess greift nur ein, sofern eine Kollision registriert wird. Solange der Roboter keine Kollision registriert, befolgt er die Befehle des PDA. Trifft er auf ein Hindernis, so ignoriert er die Befehle des PDA und weicht dem Hindernis aus. Ist die Kollision nicht mehr präsent, so kommen wieder die Befehle des PDA durch.

Die Prioritäten zwischen PDA und MIT-6.270-Board werden durch eine Subsumption-Architektur realisiert. Abbildung 5.8 zeigt dieses Verhalten. Das MIT-6.270-Board hat die höhere Priorität. Es kann den Einfluss des PDA auf die Aktoren unterdrücken.

Der dritte Prozess besteht aus einem Bytecode-Interpreter. Er wertet die eingegangenen Befehle aus und ruft die entsprechenden Funktionen auf. Wird ein Byte empfangen, so wird im Gegenzug ein Sensorwert an das Planning-Programm geschickt. Wird der Sensorwert vom Planning-Programm empfangen, so wird ein weiteres Byte an das Controller-Programm geschickt.

Die Abbildung 5.9 zeigt die internen Zusammenhänge der Subsumption-Architektur auf dem MIT-6.270-Board. Der Keep-Alive-Prozess hat die höhere Priorität. Er ruft die Funktion `ReleaseCollision()` auf, wenn eine Kollision vorliegt. Der Prozess mit dem Bytecode-Interpreter ruft die Funktion `RunBySerial()` auf, sofern er nicht von dem Keep-Alive-Prozess unterdrückt wird.

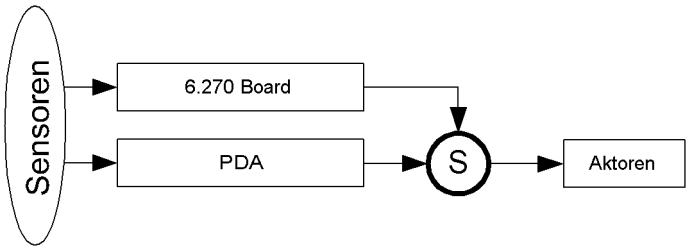

Abbildung 5.8: Subsumption-Architektur zwischen PDA und Controller-Board

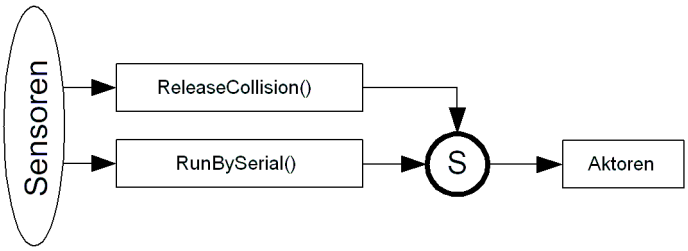

Abbildung 5.9: Subsumption-Architektur auf dem Controller-Board

Design der endlichen Automaten

Die Prozesse, die auf dem Roboter laufen, sind als einzelne Automaten implementiert. Sie kommunizieren über globale Variable, da Interactive C leider keine Prozesskommunikation zur Verfügung stellt. Das Prozess-Scheduling ist sehr einfach gehalten. Man kann beim Start eines Prozesses eine bestimmte Anzahl von Ticks an Rechenzeit zuweisen und bestimmen wie viel Stack dem Prozess zugewiesen werden soll.

Dies bedeutet, dass z.B. der Sensorprozess sämtliche Sensorwerte permanent ausliest und sie in globalen Variablen speichert. Benötigt ein Prozess einen bestimmten Sensorwert, so liest er die globale Variable aus und wertet sie aus.

Die Automaten können vom Prinzip her als switch-case-Verteiler interpretiert

werden. Die Verteiler sind jedoch als if-else-Konstrukt implementiert, da es in Interactive C keine switch-case-Konstrukte gibt.

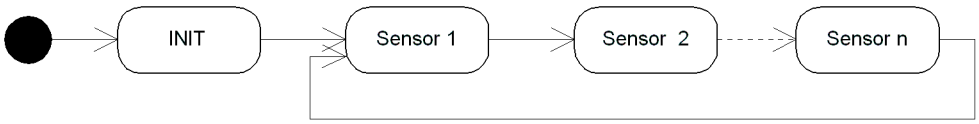

Abbildung 5.10: Automat Sensorprozess auf dem Controller-Board

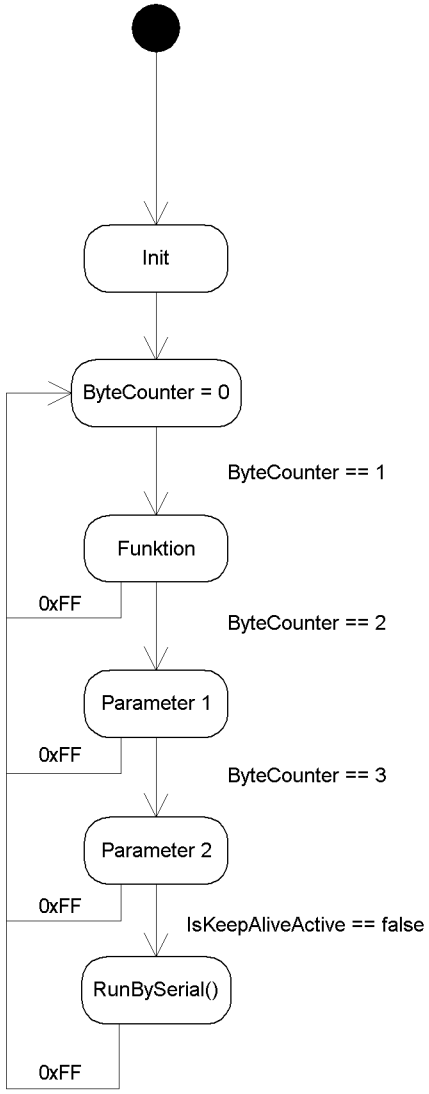

Abbildung 5.11: Automat Bytecode-Interpreter auf dem Controller-Board

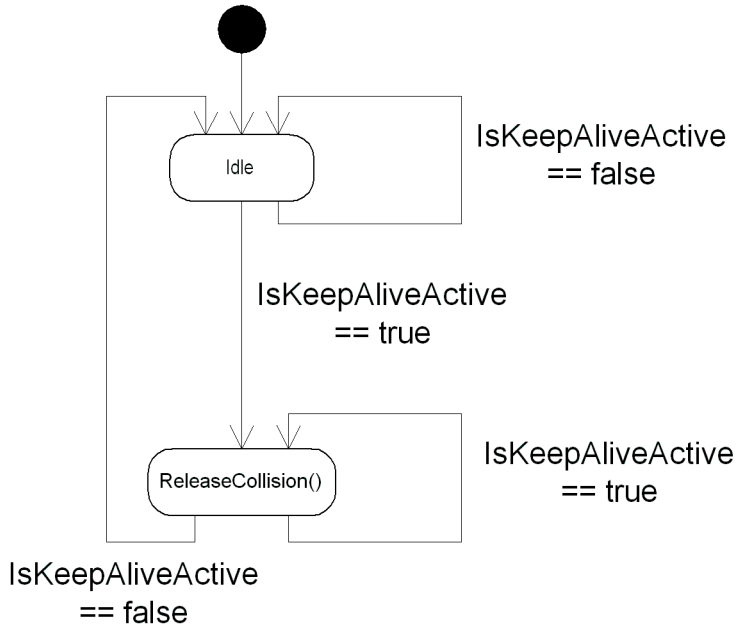

Abbildung 5.12: Automat Keep-Alive-Prozess auf dem Controller-Board

5.5 Die Server-Klasse

Das Planning-Programm soll die Möglichkeit haben, mit einem Server zu kommunizieren. Die Kommunikation soll schnurlos erfolgen, damit der Roboter mobil ist. Als Übertragungsmedium soll eine Funkverbindung dienen. Auf dieser Funkverbindung basierend soll eine TCP/IP-Verbindung erstellt werden. Auf dieser Grundlage ist es möglich über Sockets zu kommunizieren. Die Sockets ermöglichen dann eine direkte Kommunikation zwischen den Programmen auf dem PDA und dem Server.

Diese Kommunikationseinheit soll in eine Klasse gekapselt werden. Diese Server-Klasse stellt alle Methoden zur Verfügung, um mit einem bestimmten Server Kontakt aufzunehmen und mit ihm Daten auszutauschen.

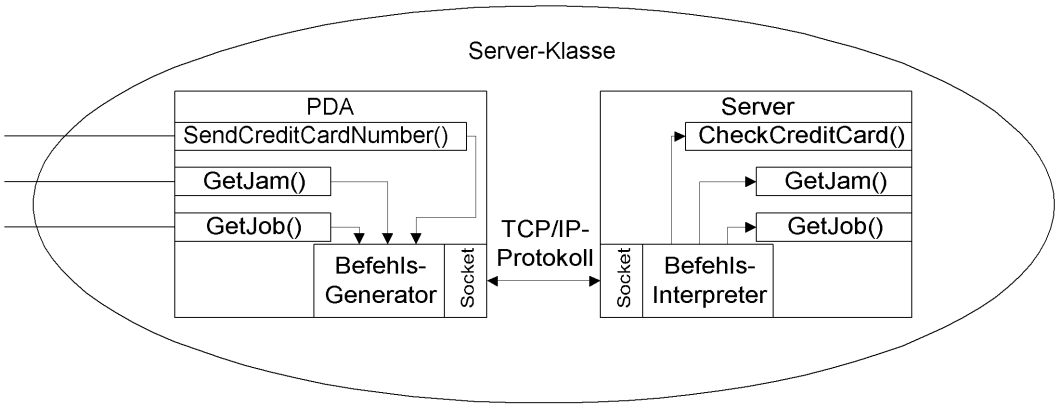

Abbildung 5.13: Server-Klasse: Übertragung zum Server

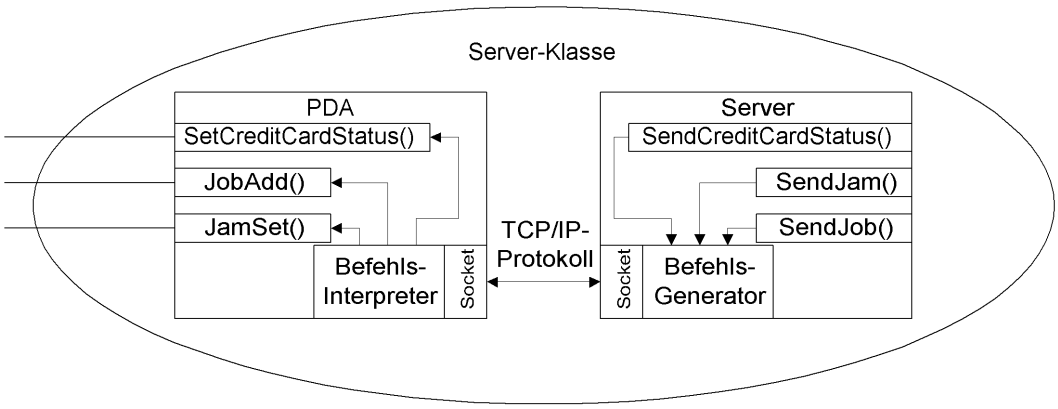

Abbildung 5.14: Server-Klasse: Übertragung zum PDA

Sicht von außen

Die Kommunikation mit dem Server-Programm

Die Klasse stellt Methoden zur Verfügung, um vom Server-Programm Dienste in Anspruch zu nehmen. Von dem Server-Programm können Stauinformationen und Fahraufträge abgerufen werden. Die Stauinformationen geben Auskunft über Stau auf einer der beiden Strecken des Parcours. Die Fahraufträge enthalten Informationen, welche Häuser angefahren werden sollen. Diese Informationen kann sich der Roboter holen, wenn er auf eine Tankstelle fährt. Damit die Dienste benutzt werden können, müssen zuerst Kreditkarten-Informationen übertragen werden. Diese Aufgabe übernimmt die Methode `SendCreditCardNumber()`. Die Kreditkartennummer ist als Membervariable gespeichert. Sie wird von der Methode `SendCreditCardNumber()` ausgelesen und an das Server-Programm übertragen. Von dem Server-Programm kommt eine Antwort zurück. Diese Antwort setzt sich aus einer Kennung und dem Inhalt zusammen. Die Kennung beschreibt, für welche Anfrage die Antwort bestimmt ist. Der Inhalt der Nachricht beschreibt, ob die Daten in Ordnung sind oder nicht.

Sind die Kreditkarten-Informationen in Ordnung, dann können von dem Server-Programm Informationen abgerufen werden. Die Methode `GetJam()` fordert eine Stauinformation vom Server an. Die Methode `GetJob()` ruft jeweils einen Fahrauftrag ab. Das Server-Programm antwortet entweder mit einem Fehler, mit einem Auftrag oder mit der Information, dass kein weiterer Auftrag vorliegt. Der Fehler entsteht, wenn die Methode aufgerufen wird, obwohl die Kreditkarten Informationen noch nicht überprüft wurden. Kommt als Antwort ein Auftrag zurück, so wird die Methode `GetJob()` erneut aufgerufen, um einen weiteren Auftrag anzufordern. Auf diese Weise werden alle anstehenden Aufträge abgerufen. Sind keine Aufträge mehr vorhanden, so kommt als Antwort die Meldung, dass kein Auftrag vorhanden ist. In diesem Fall wird die Verbindung getrennt.

Dieses Verhalten ist in der Methode `GetServerInformation()` implementiert. Die Abbildung 5.15 zeigt den entsprechenden Automaten der Methode.

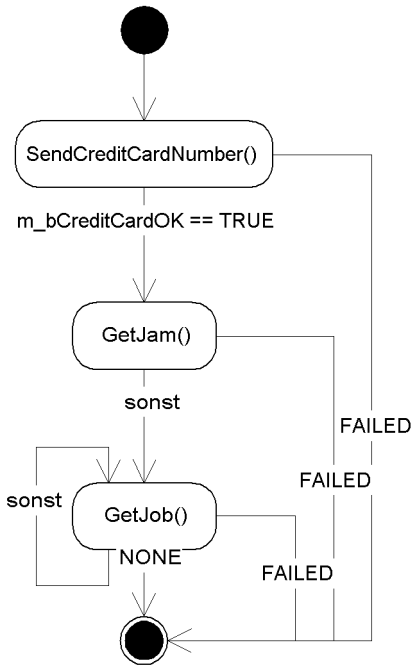

Abbildung 5.15: Automat GetServerInformation

Sicht von innen

TCP/IP-Kommunikation

Zum Erstellen einer TCP/IP-Verbindung werden Sockets benötigt. Sockets bilden die Endpunkte einer TCP/IP-Verbindung. Es sollen Stream-Sockets verwendet werden, da so sichergestellt ist, dass die Daten die gesendet wurden auch beim Empfänger ankommen. Ist dies nicht der Fall, so werden von den Sockets Ereignisse ausgelöst, die auf bestimmte Vorkommnisse hinweisen. Außerdem ist sichergestellt, dass keine Duplikate auftreten. Über die Verbindung sollen einfache Zeichenfolgen übertragen werden, die für bestimmte Befehle oder Informationen stehen.

Die Server-Klasse stellt Methoden zur Verfügung, um nach einem erfolgreichen Aufbau einer Funkverbindung, eine TCP/IP-Verbindung mit Sockets zu erstellen. Die Methoden ermöglichen einen Auf- und Abbau der TCP/IP-Verbindung und das Senden und Empfangen von Daten.

Sockets

In dieser Arbeit sollen spezielle MFC-Sockets von der Firma Microsoft verwendet werden, da die Betriebssysteme des PDA und des Servers, auf denen die Software laufen wird, ebenfalls von Microsoft sind. Diese Sockets bieten die beste Unterstützung für die Betriebssysteme. Die MFC-Sockets bieten einige spezielle Möglichkeiten. Dadurch werden einige Aufgaben vereinfacht. Leider kann es dadurch auch zu Inkompatibilitäten mit Anwendungen kommen, die auf herkömmlichen Sockets basieren.

Auf dem PDA läuft das Betriebssystem Pocket PC 2002 für mobile Geräte. Dieses System arbeitet mit einem Zeichensatz in Unicode. Die Standard-Betriebssysteme von Microsoft arbeiten in der Regel mit MBCS[1]. Daher ist eine Zeichenkonvertierung notwendig. Arbeitet man z.B. mit der Klasse CAsyncSocket, dann ist eine Zeichenkonvertierung nötig. Zur Konvertierung gibt es Funktionen die `wcstombs()` und `mbstowcs()`.

Diese Problematik lässt sich mit dem CSocket-Programmiermodell umgehen. Dieses Programmiermodell sieht die gemeinsame Verwendung von drei Klassen vor. Es wird ein Objekt der Klasse CSocket und ein Objekt der Klasse CSocketFile benötigt. Dazu wird je Übertragungsrichtung ein Objekt der Klasse CArchive benötigt. In diesem Modell wird die Zeichenkonvertierung durch das CArchive-Objekt erledigt. In der Abbildung 5.16 wird die Zusammenarbeit der Objekte dargestellt. Die Arbeit mit CArchive-Objekten bietet eine höhere Abstraktion. Der Umgang ist ähnlich wie der Umgang mit der Serialisierung von Daten in Dateien. Das CSocket-Programmiermodell ermöglicht sogar das Versenden von Objekten. Dies setzt allerdings voraus, dass auf der anderen Seite ebenfalls das CSocket-Programmiermodell verwendet wird. Um Daten zu senden, werden die Daten an ein CArchive-Objekt übergeben. Das CArchive-Objekt gibt die Daten weiter an das CSocketFile-Objekt und dieses übergibt die Daten an das CSocket-Objekt. Das CSocket-Objekt übernimmt die Kommunikation mit dem anderen Kommunikationspartner.

Auf dem PDA kommt die von der CSocket-Klasse abgeleitete Klasse CCeSocket zum Einsatz. Diese Klasse ist speziell für das Betriebssystem Pocket PC 2002 bzw. Windows CE bestimmt. Da Windows CE keine asynchrone Kommunikation unterstützt, bietet die Klasse eine eigene asynchrone Benachrichtigung für bestimmte Ereignisse. Werden Daten empfangen, so wird das Anwendungsgerüst informiert. Dieser Nachricht wird eine Behandlungsroutine zugeordnet. Diese Behandlungsroutine liest die Daten aus dem CArchive-Objekt aus, bis der Empfangspuffer leer

[1] Multibyte Character Set

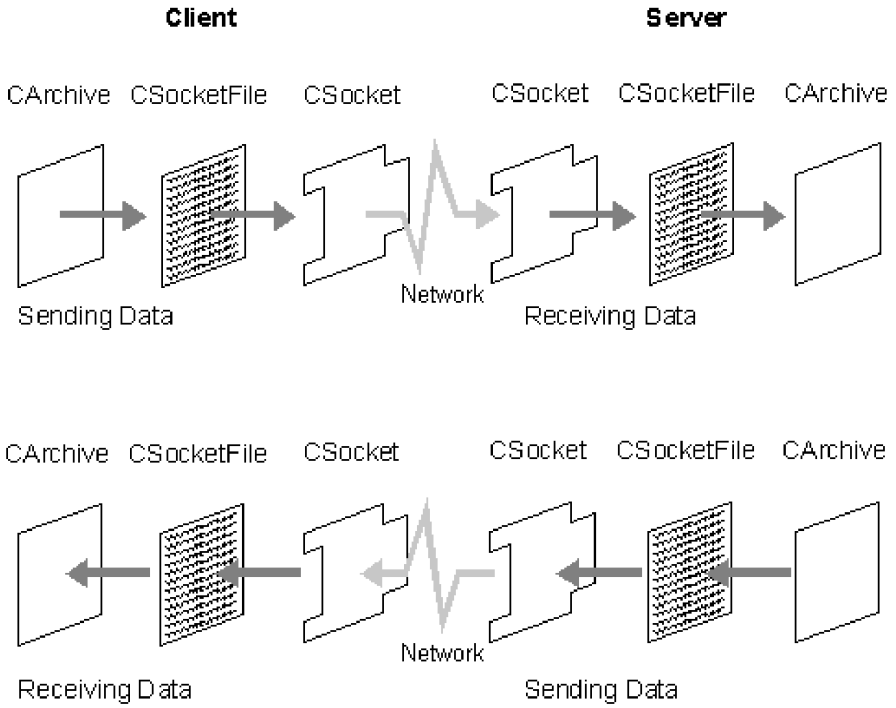

Abbildung 5.16: Windows-Sockets: Zusammenarbeit von Sockets mit Archiven

ist. Ohne die Benachrichtigungen wäre es notwendig, einen separaten Thread zu erzeugen, der auf ankommende Daten wartet.

Auf der Serverseite wird ein CSocket-Objekt erzeugt. Dieses Objekt wartet auf Verbindungswünsche. Geht ein Verbindungswunsch vom PDA ein, so wird auf der Serverseite ein weiteres CSocket-Objekt erzeugt. Dieses Objekt übernimmt, zusammen mit den CSocketFile- und CArchive-Objekten, die Kommunikation mit dem PDA. Das ursprüngliche Objekt wartet auf weitere Anfragen.

Bluetooth-Kommunikation

Die Funkverbindung wird per Bluetooth hergestellt. Bluetooth bietet die Möglichkeit, eine LAN-Verbindung über Funk aufzubauen. Hierfür wird das Bluetooth-Profil für den LAN-Access benutzt. Über diese Verbindung ist es möglich eine TCP/IP-Verbindung zu erstellen.

Die Server-Klasse bietet Methoden, um Geräte zu suchen, mit ihnen eine Verbindung einzugehen und diese wieder zu trennen. Die Klasse bietet die Möglich-

keit, bestimmte Geräte fest einzutragen. So wird verhindert, dass sich die Anwendung mit beliebigen Geräten verbindet. In dieser Arbeit sind zwei feste Stationen vorgesehen. Daher gibt es zwei Member-Variablen, in welche die festen Stationen eingetragen werden können. Um die Geräte eindeutig zu identifizieren, wird die Hardware- bzw. MAC-Adresse des gefundenen Gerätes ausgewertet. Diese Adresse ist weltweit eindeutig. So kann sichergestellt werden, dass es sich um das gewünschte Gerät handelt.

Wurde ein Gerät gefunden, dessen Adresse nicht mit der gewünschten übereinstimmt, wird die Suche fortgesetzt. Handelt es sich um die richtige Adresse, wird die Suche gestoppt. Nun ist es möglich, zu dem Gerät eine LAN-Access-Verbindung aufzubauen. Über die Verbindung kann dann eine TCP/IP-Verbindung hergestellt werden.

High Point BTAccess-Bibliothek

Um das Bluetooth-Modul des PDA ansprechen zu können, wird eine separate Bibliothek benötigt. Für den iPAQ H3970 gibt es die BTAccess-Bibliothek von der Firma High Point Software (BTAccess, 2003). Sie stellt zwei Klassen zur Verfügung. Die Klasse CBtStack bietet Methoden, um den Bluetooth-Stack des PDA anzusprechen. Die Klasse CBtDevice lassen sich gefundene Geräte ansprechen und deren Eigenschaften auslesen. Die Tabelle 5.3 zeigt die für die Realisierung benötigten Methoden der Klasse CBtStack.

Methode	**Beschreibung**
Connect	Initialize a connection to the stack
Disconnect	Terminate a connection to the stack
StartDeviceSearch	Start searching for Bluetooth devices
StopDeviceSearch	Stop searching for Bluetooth devices
GetBtAddress	Get Bluetooth address of the local device
IsBtRadioOn	Test whether Bluetooth radio is on or off
BtRadioOn	Turn the Bluetooth radio on
BtRadioOff	Turn the Bluetooth radio off

Tabelle 5.3: Benötigte Methoden der Klasse CBtStack

Die Methode `Connect()` initialisiert eine Verbindung zu dem Bluetooth Stack. Diese Verbindung wird mit der Methode `Disconnect()` wieder getrennt. Mit `StartDeviceSearch()` wird die Suche nach Geräten gestartet. Die Suche wird erst beendet, wenn `StopDeviceSearch()` aufgerufen wird. Man kann also die

Suche permanent eingeschaltet lassen[2]. Dabei muss allerdings beachtet werden, dass der PDA dabei für andere Geräte nicht erreichbar ist und der Energieverbrauch des PDA höher ist. Mit der Methode `GetBtAddress()` kann man die MAC-Adresse des PDA herausfinden. Mit `BtRadioOn()` und `BtRadioOff()` kann man das Bluetooth-Modul ein- bzw. ausschalten.

Methode	**Beschreibung**
LAPConnect	Connect to a Lan Access Point
LAPDisconnect	Disconnect from Lan Access Point
GetSignalStrength	Get RSSI signal strength of any connection to this device from the local device (low, good, high)

Tabelle 5.4: Benötigte Methoden der Klasse CBtDevice

Für jedes gefundene Gerät wird ein Objekt der CBtDevice-Klasse erzeugt. Ein CBtDevice-Objekt stellt sämtliche Eigenschaften und Dienste eines gefundenen Gerätes zur Verfügung. Die Tabelle 5.4 zeigt die für die Realisierung benötigten Methoden der Klasse CBtDevice. Die Methoden `LAPConnect()` und `LAPDisconnect()` werden benötigt, um mit einem gefundenen Gerät die gewünschte LAN-Access-Verbindung aufzubauen. Mit `GetSignalStrength()` ist es möglich, die Signalstärke eines gefundenen Gerätes zu ermitteln. Diese Methode wird für die Realisierung nicht unbedingt benötigt. Sie dient nur zur Information.

Die Member-Variable `m_cDeviceAddr[]` der Klasse CBtDevice enthält die MAC-Adresse des gefundenen Gerätes. Diese Membervariable ist nicht in der Dokumentation der Klasse aufgeführt. Sie ist jedoch ein Element der Klasse. Bei der Membervariablen handelt sich um ein Array vom Typ unsigned char. Es enthält 6 Elemente, welche die MAC-Adresse wiedergeben.

Die Abbildung 5.17 zeigt das Verhalten bei der Suche nach einem Server. Die Suche wir durch `SearchStart()` angestoßen. Wurde der richtige Server gefunden, so wird mit `LAPConnect()` der Verbindungsvorgang zum LAN-Access angestoßen. Damit ist der Vorgang vorerst abgeschlossen. Die BTAccess-Bibliothek

[2] Laut der Dokumentation für die Bibliothek werden bei dauerhafter Suche auch noch Geräte gefunden, die später auftauchen. In der Realisierung hat dies nicht funktioniert. Zur Lösung dieses Problems ist die Suche mit einem Timeout verbunden. Wird nach einer bestimmten Zeit nicht das gewünschte Gerät gefunden, wird die Suche gestoppt und neu gestartet.

schickt eine Nachricht an das Anwendungsgerüst, wenn der Verbindungsaufbau abgeschlossen oder misslungen ist.

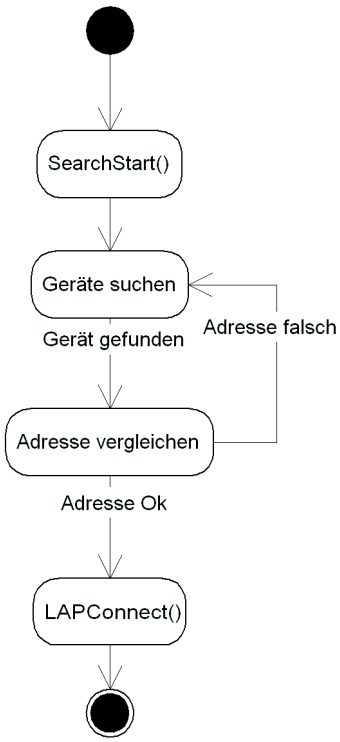

Abbildung 5.17: Server-Klasse: Suche nach Server

Wenn das Anwendungsgerüst über den Verbindungsaufbau benachrichtigt wird, wird die Methode `OnConnectionStatus()` aufgerufen. Die Methode wertet die Nachricht aus. Als erstes wird überprüft auf welche Art von Verbindung sich die Nachricht bezieht. Für den LAN-Access wird die Variable `eSvc` auf den Wert `BTSVC_LAP` überprüft. Ist dies der Fall, dann bezieht sich die Nachricht auf den LAN-Access. Wenn der Verbindungsaufbau erfolgreich war, dann enthält die Nachricht der Wert `BT_CONNECTION_COMPLETE`[3]. Anschließend wird die Methode `ClientConnect()` aufgerufen. Dies geschieht solange, bis die Verbindung steht. Der Aufruf ist mit einem Timeout[4] versehen. Nach einer bestimmten

[3] In der Praxis steht die Verbindung zu diesem Zeitpunkt noch nicht. Beim Aufruf von der Connect-Methode des CCeSocket-Objekts wird daher eine Abfrage vorgenommen, ob der Aufbau der TCP/IP-Verbindung funktioniert hat.

[4] Der Timeout ist als Schleife realisiert. Bei jedem Versuch wird ein Zähler erhöht und daraufhin mit der Funktion Sleep() einen Moment gewartet. Der Versuch den Timeout mit einem Windows-Timer zu realisieren hat nicht funktioniert, da die Timer bei starker Auslastung des Programms

Anzahl von misslungenen Verbindungsversuchen wird der Vorgang abgebrochen. Nach einem erfolgreichen Aufbau der Verbindung ist es möglich Daten zwischen den Programmen auszutauschen. Ist der Austausch abgeschlossen, wird die Methode `ClientDisconnect()` aufgerufen, um die Socket-Verbindung zu schließen. Zum Abschluss wird `LAPDisconnect()` aufgerufen um die LAN-Access-Verbindung zu trennen.

nicht ausgelöst werden.

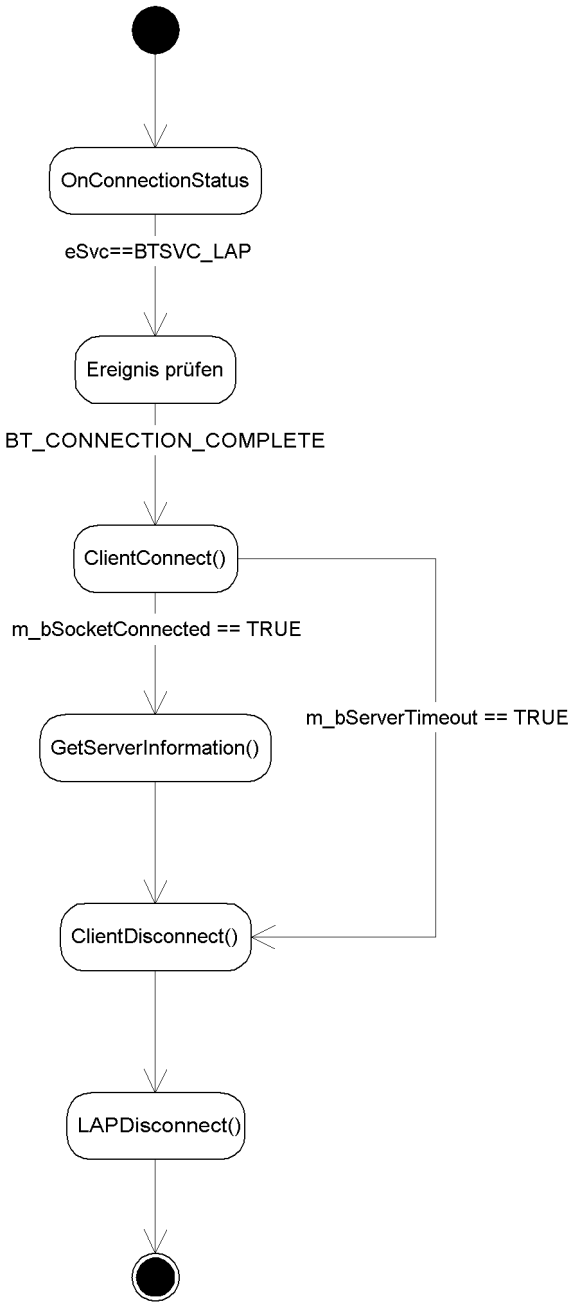

Abbildung 5.18: Server-Klasse: Austausch mit Server

Auf dem PC läuft ein Bluetooth LAN-Access Server Treiber. Dieser Treiber stellt einen LAN-Access zur Verfügung. Über diesen Zugang kann der PDA bei Bedarf mit dem Server-Programm Kontakt aufnehmen. Dieser Netzwerkzugang erhält eine eigene IP-Adresse. Diese wird standardmäßig auf die private Adresse 192.168.0.1[5] gesetzt. So existiert auf jedem Server die gleiche feste IP-Adresse, mit der sich der PDA verbinden kann. Dadurch braucht man nicht von jedem Server die IP-Adresse zu ermitteln.

5.6 Design des Clients (PDA)

Um das Szenario mit dem PDA realisieren zu können, benötigt der PDA die Möglichkeit, mit dem Controller-Board zu agieren. Dies wird durch die Roboter-Klasse realisiert. Durch die Roboter-Klasse besteht auf dem PDA die Möglichkeit, den Roboter aktiv zu steuern und den aktuellen Zustand zu erfragen. Das Szenario erfordert außerdem die Möglichkeit, mit einem Server kommunizieren zu können. Für das Szenario ist nur eine schnurlose Kommunikation sinnvoll. Die Server-Klasse erfüllt diese Anforderungen.

Der PDA bildet das Herzstück zwischen Controller-Board und Server. Das Controller-Board wird durch die Roboter-Klasse repräsentiert. Die Roboter-Klasse stellt eine Schnittstelle zu dem Controller-Board dar. Sie ermöglicht die Abfrage von Sensorwerten und das Aufrufen von Funktionen auf dem Controller-Board. Der Server wird durch die Server-Klasse repräsentiert. Die Server-Klasse ermöglicht den Aufbau einer Funkverbindung, um mit dem Server Informationen auszutauschen.

Nun stehen zwei Schnittstellen zur Verfügung, mit denen zum einen das Controller-Board beeinflusst und zum anderen der Server angesprochen werden kann. Nun stehen zwei Bausteine zur Verfügung, die sinnvoll verbunden werden müssen. Dies soll durch ein separates Roboter-Programm erfolgen.

Zur Visualisierung und Bedienung des Programms auf dem PDA wird eine grafische Oberfläche eingesetzt. Die grafische Oberfläche besteht aus einer Dialog-Klasse.

[5] Leider lässt sich keine andere Adresse als die 192.168.0.1 einstellen. Es ist darauf zu achten, dass keine anderen Geräte diese Adresse verwenden.

Die grafische Oberfläche

Zur Information z.B. während der Entwicklung, werden die Sensorwerte des Roboters in der Anwendung angezeigt. In der Dialog-Klasse befinden sich Steuerelemente, die unter anderem die Sensorwerte anzeigen. Für die Anzeige der Sensorwerte in der grafischen Oberfläche wird ein Timer benötigt. Dieser Timer wird beim Start des Programms initialisiert. Er löst in einem bestimmten Zyklus ein Ereignis aus. Jedes Mal, wenn der Timer ausgelöst wird, werden die Sensorwerte aus dem Objekt der Roboter-Klasse ausgelesen. Anschließend wird die Methode `UpdateData(FALSE)` aufgerufen, damit die Anzeige aktualisiert wird.

Zusätzlich zu den Sensorwerten wird angezeigt, welche Stauinformation und welche Aufträge das Planning-Programm gerade vorliegen hat. Die Aufträge werden in einer List Box gespeichert und angezeigt. Die Station mit der sich das Planning-Programm als nächstes verbinden möchte, wird ebenfalls angezeigt. Für die Anzeige von Meldungen dient eine List Box. Hier werden z.B. Informationen beim Verbindungsaufbau und Fehlermeldungen ausgegeben. Die Verwendung einer List Box hat gegenüber einer Messagebox den Vorteil, dass man nicht ständig "Ok" klicken muss, wenn eine Meldung kommt. Da es sich um eine autonome Anwendung handelt, sind Messageboxen nicht angebracht.

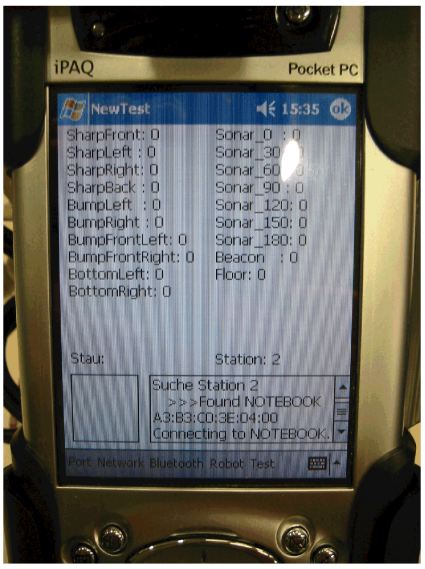

Abbildung 5.19: Planning-Programm: Verbindungsaufbau mit dem Server (1)

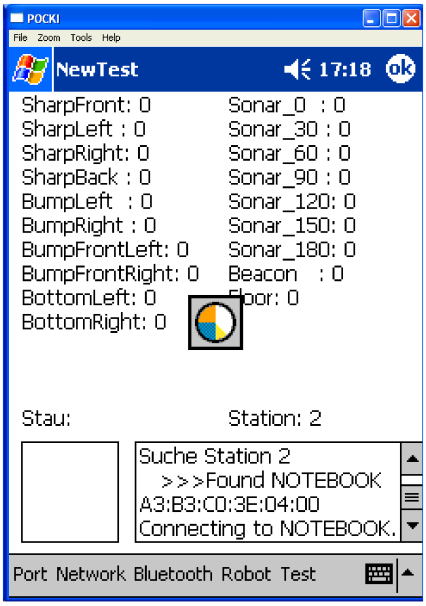

Abbildung 5.20: Planning-Programm: Verbindungsaufbau mit dem Server (2)

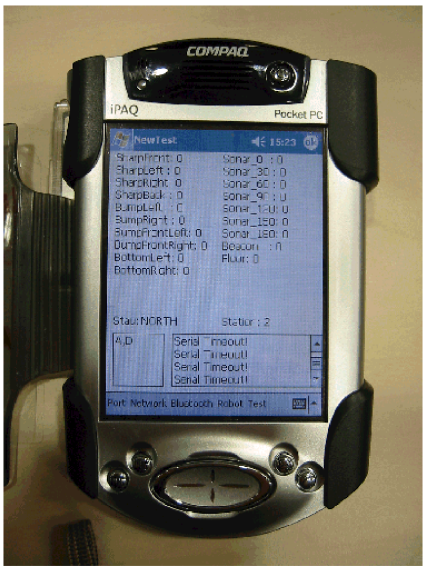

Abbildung 5.21: Planning-Programm: Nach Verbindung mit dem Server (1)

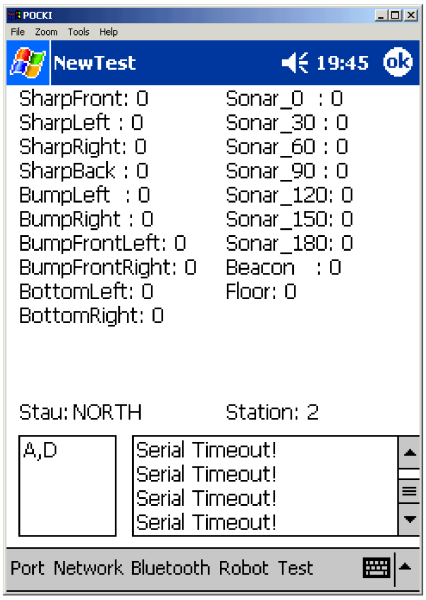

Abbildung 5.22: Planning-Programm: Nach Verbindung mit dem Server (2)

Die Abbildung 5.23 zeigt die Zusammenhänge zwischen den Klassen und dem Roboter-Programm.

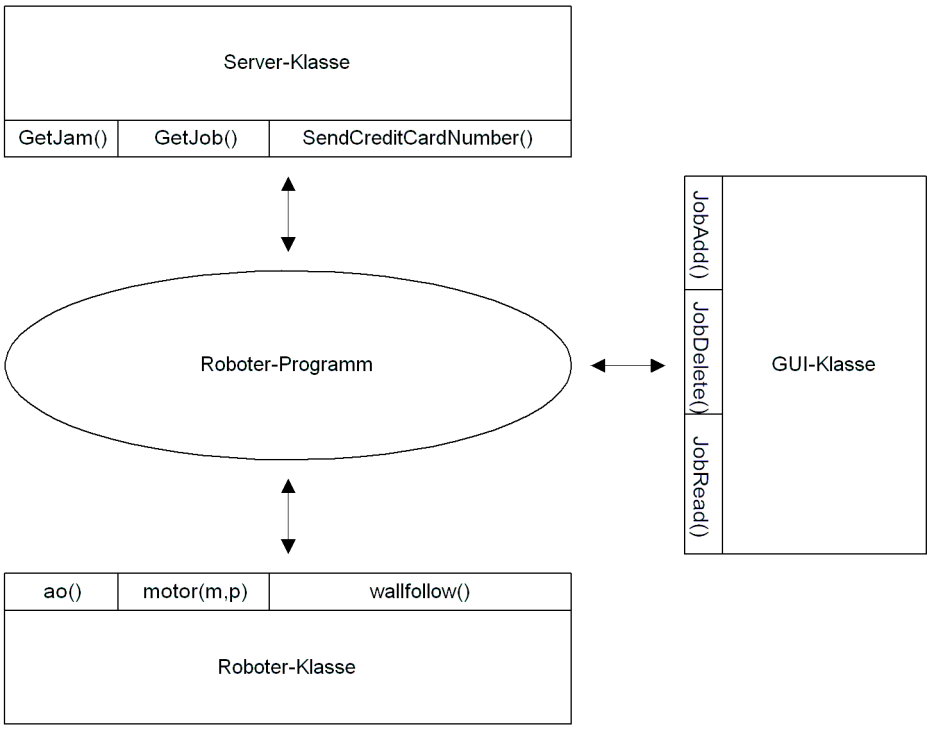

Abbildung 5.23: Zusammenspiel der Klassen auf dem PDA

Roboter-Programm

Auf dem PDA läuft ein Roboter-Programm, das die Steuerung des Roboters übernimmt. Das Programm übernimmt das Planning. Es merkt sich, in welchem Zustand sich der Roboter gerade befindet und entscheidet, was er als nächstes tun wird. Das Roboter-Programm arbeitet mit einem Objekt der Roboter-Klasse und einem Objekt der Server-Klasse. Das Roboter-Programm wird als separater Thread realisiert. Das bedeutet, dass das Roboter-Programm parallel zur Anwendung läuft.

Um den Roboter zu steuern liest das Roboter-Programm die Sensorwerte aus dem Roboter-Objekt aus und ruft daraufhin Methoden auf, die dann den Roboter beeinflussen. In dem Roboter-Programm wird ein Automat implementiert, der das gewünschte Verhalten des Roboters erzeugt. Der Automat wird als switch-case-Konstrukt implementiert.

Um Informationen auszutauschen ruft das Roboter-Programm Methoden der Server-Klasse auf. Mit der Server-Klasse kann eine dynamische Funkverbindung zum Server aufgebaut werden um anschließend mit dem Server-Programm zu kommunizieren. Weitere Details über das Roboter-Programm sind in Kapitel 5.9 zu finden.

5.7 Design des Servers (PC)

Das Server-Programm läuft auf einem stationären PC. Es ermöglicht die Eingabe von Informationen für den Roboter. Es können Stauinformationen und Transportaufträge eingegeben werden. Für die Realisierung wird Microsoft Visual C++ 6.0 zusammen mit den Microsoft Foundation Classes (MFC) verwendet. Das Server-Programm besteht aus drei Komponenten, der Server-Klasse, der Applikation und der GUI-Klasse.

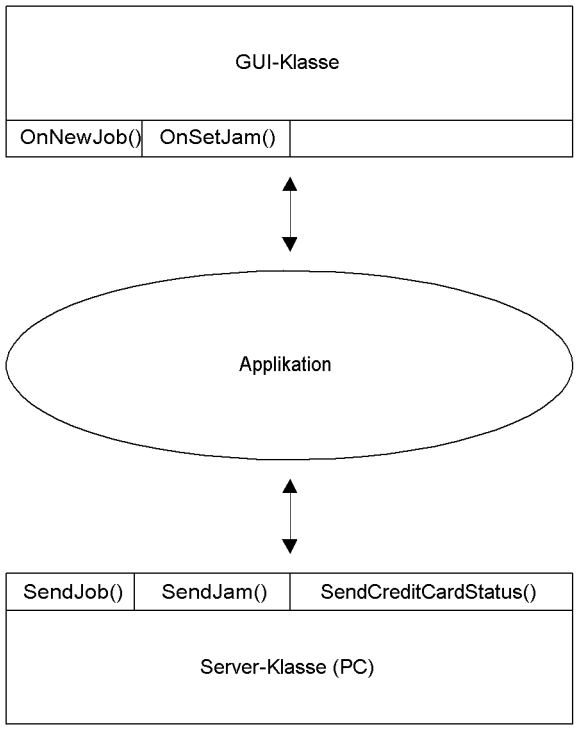

Abbildung 5.24: Zusammenspiel der Klassen des Server-Programms

Die Applikation

Die Applikation bildet das Grundgerüst des Server-Programms. Realisiert ist die Applikation als MDI-Projekt. Bei der Erstellung eines MDI-Projekts wird bereits Code für die Serialisierung erzeugt. Die Serialisierung ist bei der Verwendung des CSocket-Programmiermodells von nutzen. Die Applikation erzeugt je ein Objekt der GUI-Klasse und der Server-Klasse.

Die grafische Oberfläche

Das Server-Programm verfügt über eine grafische Oberfläche. In der grafischen Oberfläche ist der Parcours dargestellt. Es sind die zwei Stationen und die vier Plätze, die der Roboter ansteuern kann, zu sehen. Die vier Plätze bestehen aus Schaltflächen, über die Transportaufträge ausgewählt werden können. Für die beiden Strecken die der Roboter fahren kann, gibt es ebenfalls zwei Schaltflächen. Durch das Klicken einer der beiden Schaltflächen kann auf der jeweiligen Strecke Stau signalisiert werden. Ein Textfeld zeigt Informationen über den Verbindungsstatus mit dem Roboter an.

Es gibt ein Gruppenfeld, in dem die Aufträge angezeigt und bearbeitet werden könnten. Es gibt eine Schaltfläche zum Erstellen eines neuen Auftrags. Wird die Schaltfläche betätigt, werden die vier Schaltflächen für die Plätze aktiviert. Nun ist es möglich den ersten Platz, der angefahren werden soll, z.B. mit der Maus zu wählen. Wurde der erste Platz gewählt, wird die gewählte Schaltfläche deaktiviert, damit für Start und Ziel nicht der gleiche Platz gewählt werden kann. Nun sind noch drei Schaltflächen auswählbar. Wird die zweite Schaltfläche, also das Ziel gewählt, werden alle Schaltflächen deaktiviert und der Auftrag erscheint in einer List Box. Auf diese Weise können beliebig viele Aufträge eingegeben werden. Die Aufträge werden in der List Box abgelegt und für den Roboter bereitgehalten. Die Aufträge werden nach dem FIFO-Prinzip eingereiht. Wird ein Auftrag vom Roboter abgerufen, wird er aus der List Box entfernt, da dieser nun von dem Planning-Programm verwaltet wird. Eine weitere Schaltfläche ermöglicht das Löschen einzelner Aufträge z.B. bei fehlerhaften Eingaben.

Ein zweites Gruppenfeld beinhaltet die Elemente für die Anzeige und Bearbeitung der Stauinformation. Hier wird über ein Textfeld die aktuelle Stausituation angezeigt. Über eine Schaltfläche kann der Stau zurückgesetzt werden.

Die Abbildung 5.25 auf Seite 72 zeigt einen Screenshot des Server-Programms. In der Abbildung ist ein Auftrag von Platz A nach Platz D zu sehen. Die Stauinformation ist auf Nord gesetzt.

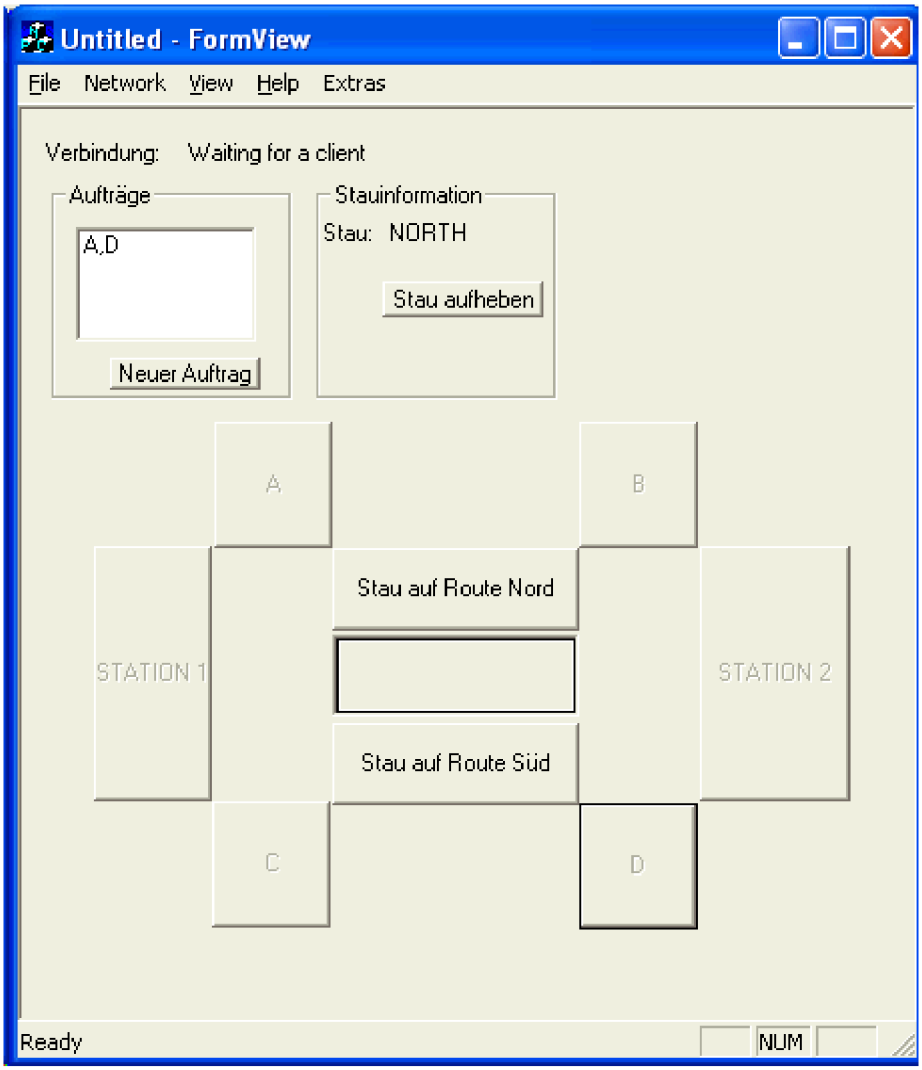

Abbildung 5.25: Screenshot des Server-Programms

5.8 Realisierung des Szenarios

Bisher wurde das Roboter-Programm allgemein beschrieben. In diesem Kapitel soll die Realisierung des Szenarios beschrieben werden. Um die Aufgaben den Szenarios zu erfüllen, müssen Automaten entwickelt und implementiert werden. Die Automaten beschreiben das Verhalten für die benötigten Abläufe.

Der Startpunkt des Roboters ist die Station 1. Nach dem Start des Planning-Programms baut der Roboter eine Verbindung zum Server der Station 1 auf und tauscht Informationen aus. Ist der Austausch abgeschlossen, so fährt der Roboter los. Stehen Aufträge an, dann wird der erste Auftrag bearbeitet. Der Roboter fährt zu dem ersten Platz, der im Auftrag steht. Wenn Stau vorhanden ist, dann muss der Roboter eventuell einen Umweg fahren. Der Stau existiert immer nur auf einer der beiden Strecken zwischen den Plätzen (siehe Abbildung 5.26). Das heißt, es sind auf jeden Fall alle Plätze und Stationen erreichbar. Wenn kein Auftrag ansteht, fährt der Roboter zur Station 2. Hier baut er eine Verbindung zum Server der Station 2 auf, denn hier können auch noch Aufträge vorliegen.

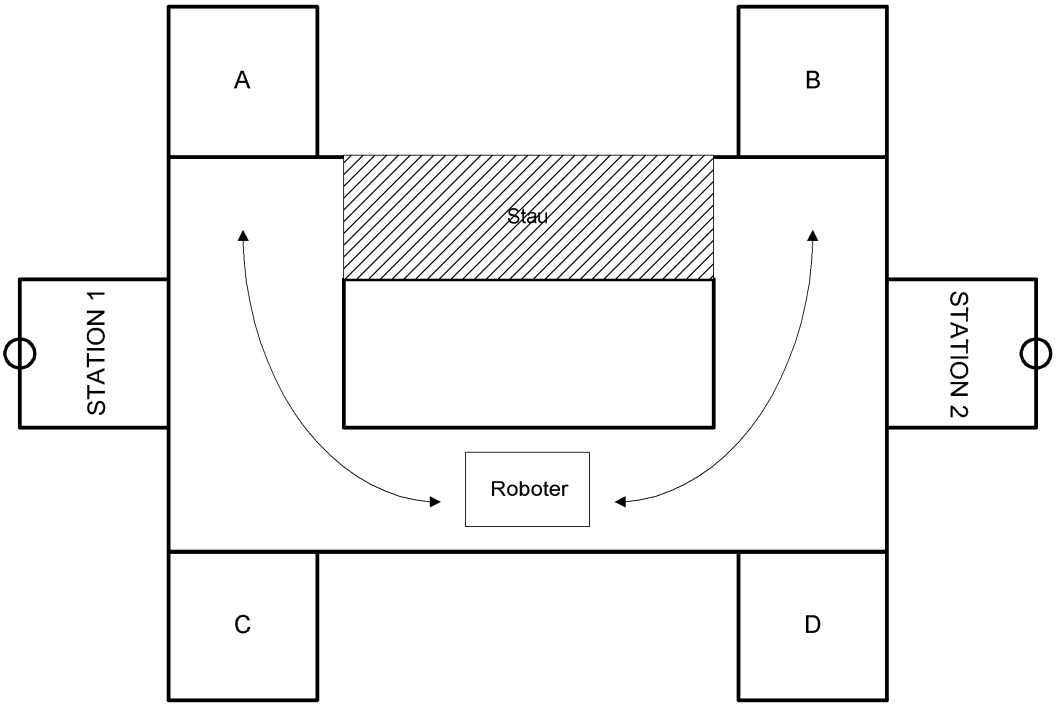

Abbildung 5.26: Szenario mit Stau

Abbildung 5.27: Roboter im Szenario

Der Parcours besteht aus 6 Holzkisten mit Metallboden. Diese Kisten sind an einer Seite geöffnet, damit der Roboter hineinfahren kann. Zwei Kisten stellen die Stationen und 4 Kisten die Plätze dar. Um die Kisten zu unterscheiden, verfügen die beiden Stationen über jeweils ein Infrarot-Leuchtfeuer. Diese Leuchtfeuer werden als Beacon bezeichnet. Die Station 1 strahlt mit einer Frequenz von 100Hz und die Station 2 mit einer Frequenz von 125Hz[6]. Dadurch lassen sich die Stationen eindeutig voneinander unterscheiden. Durch die Beacons kann unterschieden werden, ob sich der Roboter auf einem der Plätze A-D oder in einer Station befindet. Befindet sich der Roboter in einer Station, so kann anhand der Frequenz des Beacons ermittelt werden, ob es sich um die Station 1 oder Station 2 handelt. Die Plätze unterscheiden sich nicht voneinander. Sie sind in der inneren Karte des Roboters vermerkt. Durch die Bestimmung der aktuellen des Roboters Position kann ermittelt werden, um welchen Platz es sich gerade handelt.

Die Nord- und Südroute sind durch eine Kiste getrennt. Diese Kiste befindet sich

[6]Die Frequenz ist auf eine Trägerfrequenz von 40kHz moduliert.

in der Mitte des Parcours. Der äußere Rand des Parcours ist durch Trennwände begrenzt. Die Trennwände sind aus Holz und haben kleine Füße aus Aluminium, damit sie nicht umfallen. Die Trennwände verbinden die 6 Kisten. Soll einer der Plätze angefahren werden, so könnte der Roboter z.B. an den Außenwänden entlang fahren.

5.9 Realisierung des Roboters

Damit der Roboter sich in dem Szenario zurechtfindet, verfügt er über diverse Sensoren. Dabei sollten nur so viele Sensoren wie nötig benutzt werden, um herauszufinden, wo sich der Roboter gerade befindet. Mit welchen Sensoren der Roboter ausgestattet ist, wurde bereits in Kapitel 4.2.2 besprochen.

Um die Entfernungen zu den Wänden zu messen, werden die Abstandssensoren von Sharp verwendet. Eine weitere Hilfe ist der Sonarsensor. Dieser ist so hoch montiert, dass er die niedrigen Trennwände nicht registriert. Allerdings kann er die Kisten "sehen", da diese höher als die Trennwände sind. Um eine Kollision zu erkennen, werden die Bumper verwendet. Um eine Kiste zu registrieren, werden die zwei Drahtkontakte verwendet. Sie werden kurzgeschlossen, wenn sie den Metallboden einer Kiste berühren. Damit sie eine Kiste registrieren, bevor der Roboter hineinfährt, sind sie vorne am Roboter montiert. Zur Erkennung der Stationen wird der Infrarot-Empfänger für die Beacon-Erkennung ausgewertet.

Das Roboter-Programm

Bei der Realisierung des Roboter-Programms stellt sich die Frage nach der Aufgabenverteilung zwischen Controller-Programm und Planning-Programm. Diese Fragestellung wurde bereits in Kapitel 5.1.1 dikutiert. Generell gilt, dass zeitkritische Vorgänge auf das Controller-Programm verlagert werden sollten. An dem Beispiel des Szenarios sollen einige sinnvolle Varianten erläutert werden.

Um das Roboter-Programm zu realisieren, wird ein Automat benötigt. Dieser Automat wird in einzelne Zustände gegliedert, die den jeweiligen Teilaufgaben entsprechen. Es gibt zum Beispiel einen Zustand, der für den Datenaustausch mit dem Server steht oder einen Zustand, der für das Folgen einer Wand steht.

Das Roboter-Programm läuft auf dem PDA, basiert auf dem Betriebssystem Windows Pocket PC 3.0. Auf dem Betriebssystem liegt die Bibliothek Pocket PC 2002. Sie stellt die Schnittstelle zwischen Roboter-Programm und Betriebssystem dar. Das Roboter-Programm bedient sich der Funktionalitäten der Server-Klasse und der Roboter-Klasse. Dieser Zusammenhang wird in Abbildung 5.28 dargestellt. Das Roboter-Programm gibt einige Informationen, wie z.B. aktuelle Sensorwerte an die grafische Oberfläche weiter.

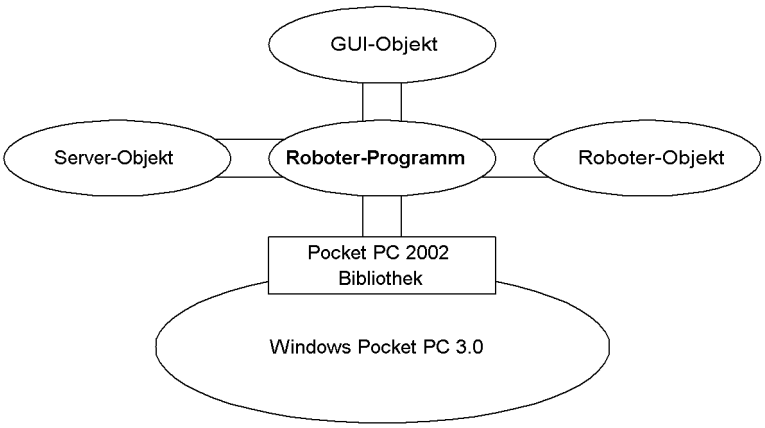

Abbildung 5.28: Komponenten des Roboter-Programms

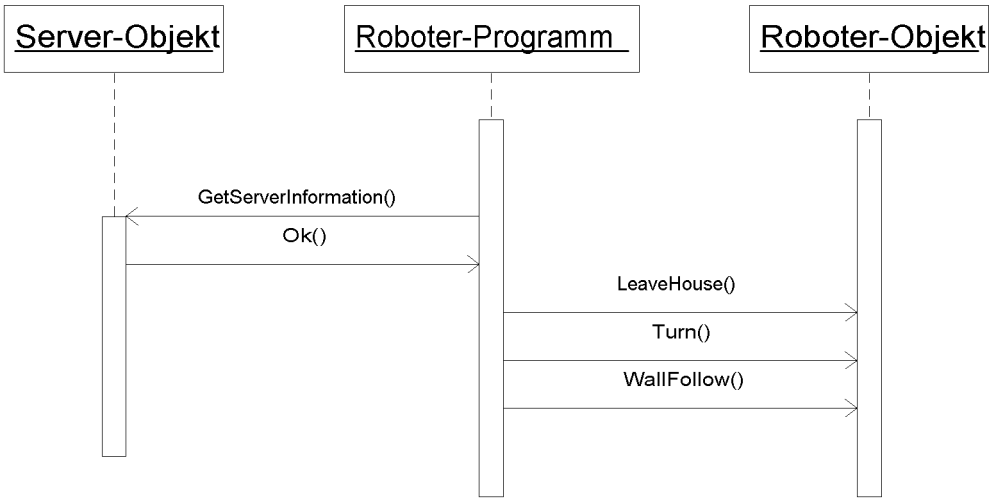

Abbildung 5.29: Sequenzdiagramm des Roboter-Programms

Der Automat

Das Schwierige ist die Navigation innerhalb des Parcours.

Damit der Roboter nicht ungewollt in eine Kiste fährt, befinden sich die Bodenkontakte vorne. Wird eine Kiste registriert, die nicht angesteuert werden soll, fährt der Roboter gar nicht erst hinein.

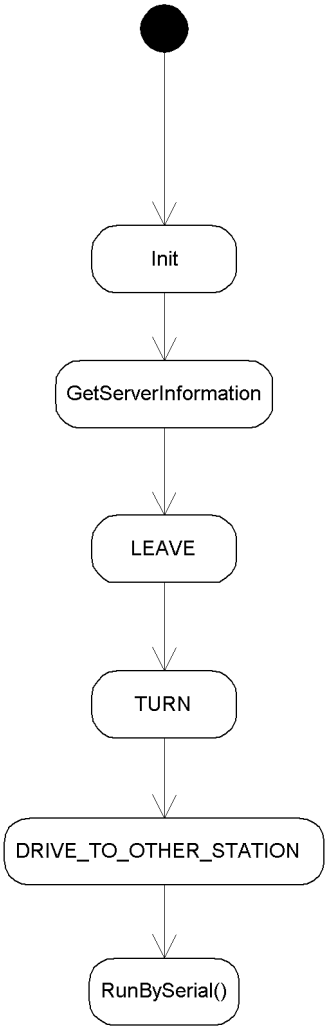

Abbildung 5.30: Automat des Roboter-Programms

Wallfollow

Eine wichtige Funktionalität für das Roboter-Programm ist die Möglichkeit einer Wand zu folgen. In dem Parcours gibt es zwei Möglichkeiten. Es kann der Abtrennung zwischen Nord- und Südroute gefolgt werden oder der Außenwand. Das Verfolgen der Abtrennung ist die einfachste Möglichkeit direkt von einer Station zur anderen Station zu fahren, ohne eventuell ungewollt in einen Platz zu fahren. Folgt der Roboter der Außenwand, trifft er auf die Plätze. Da ein Platz bzw. eine Kiste registriert wird, bevor der Roboter hineinfährt, kann der Roboter auch an einer Kiste vorbei fahren.

Ausparken

Da der Roboter in einer Kiste nicht wenden kann, muss er rückwärts heraus fahren. Daher gibt es einen Programmteil, der für das Ausparken zuständig ist.

Haus-Erkennung

Die Haus-Erkennung wird benötigt, um eine Station oder einen Platz zu detektieren. Fährt der Roboter durch den Parcours und erkennt einen Metallboden, so wurde eine Station oder ein Platz erkannt. Nun kann entschieden werden, ob der Roboter in die Kiste hineinfahren soll oder nicht. Befindet er sich in einer Kiste, so kann beim Ausparken durch die Haus-Erkennung festgestellt werden, ob er die Kiste verlassen hat.

Beacon-Erkennung

Um die beiden Stationen zu erkennen, gibt es die Beacon-Erkennung. Diese wird von dem Roboter-Programm verwendet, um zwischen den beiden Stationen zu unterscheiden.

5.10 Evaluation

Die Ergebnisse dieser Arbeit sollen an praktischen Tests überprüft werden. Die Steuerung des Roboters über den PDA läuft recht zuverlässig. Es ist sogar möglich das serielle Kabel zu entfernen und später wieder zu verbinden. Wird das Kabel entfernt, führt der Roboter die aktuelle Tätigkeit weiter aus. Kollidiert er mit einem Hindernis, kommt der Keep-Alive-Prozess zum Einsatz. Er behebt die Kollision und überlässt die Steuerung wieder dem PDA. Von dem PDA kommen jedoch keine Befehle an. Daher bleibt der Roboter einfach stehen. Wird nun die serielle Verbindung wiederhergestellt, kommen wieder Befehle vom PDA an. Nun wird der aktuelle Vorgang fortgesetzt.

Beim Wallfollow hat sich ergeben, dass die Implementierung auf dem MIT-6.270-Board erfolgen sollte. Anfangs wurde der Wallfollow auf dem PDA implementiert. Der Roboter folgte zwar der Wand, jedoch gab es bei besonders zeitkritischen Situationen Probleme mit den Reaktionszeiten. Das war zum Beispiel bei Drehungen auf der Stelle der Fall. Bis die Sensorwerte das Programm auf dem PDA erreicht haben und die entsprechende Reaktion das Controller-Programm erreicht, hat sich der Roboter schon weiter gedreht als gewünscht.

Die verwendete Plattform ist sehr robust und bietet viel Platz für die Unterbringung der Komponenten. Die Plattform ist jedoch zu sperrig. Das liegt einerseits an der Größe und andererseits an der Form. Da die Plattform rechteckig ist, eckt sie gelegentlich an. Eine abgerundete Form wäre da deutlich besser. Ein Beispiel für eine solche Plattform ist der Roboter von Michael Manger (Manger, 2003). In seiner Diplomarbeit entwickelt er eine Plattform, die speziell für Roboterfußball gut geeignet ist. Die Plattform (siehe Abbildung 5.31) hat drei Antriebsräder, die in einem Winkel von 120° zueinander stehen. So ist es möglich, durch den Antrieb von jeweils zwei Rädern, die Richtung zu bestimmen. Die Räder sind so genannte Allseitenräder. Das dritte Rad, welches nicht angetrieben wird, dreht frei mit. Die Plattform basiert auf dem fallonbot (Torrone, 2003), der in Abbildung 5.32 zu sehen ist.

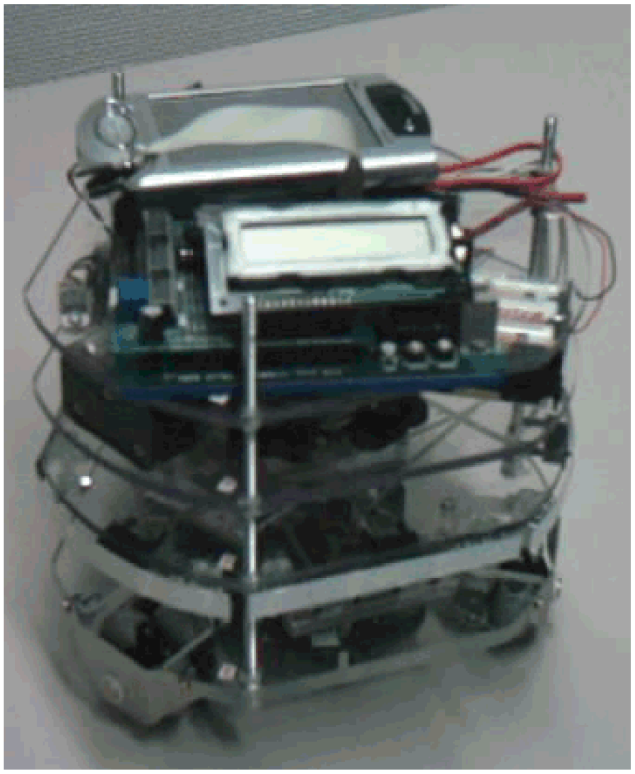

Abbildung 5.31: Plattform mit Allseitenrädern

Abbildung 5.32: Roboter fallonbot

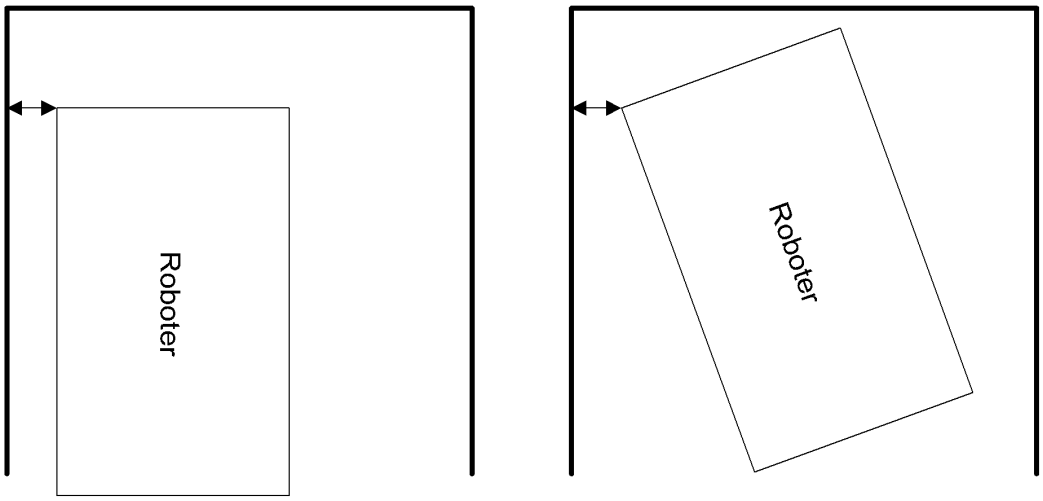

Abbildung 5.33: Problematik beim Ausparken

Problem Ausparken

Da der Roboter in einer Kiste nicht wenden kann, muss er rückwärts heraus fahren. Das ist nicht immer einfach, da sich die Sensoren für den seitlichen Abstand vorne befinden. Steht der Roboter nicht gerade in einer Kiste, kann nicht unterschieden werden, ob z.B. nur das Vorderteil oder der ganze Roboter zu dicht an einer der Wände steht. Die Abbildung 5.33 verdeutlicht diese Problematik. Steht der gesamte Roboter, wie in der Abbildung, zu dicht an der Wand, dann sollte das linke Rad stärker als das rechte Rad rückwärts angesteuert werden. Steht jedoch nur das Vorderteil zu dicht an der Wand, sollte das rechte Rad stärker als das linke Rad angesteuert werden. Daher ist es wichtig, dass der Roboter schon beim Hineinfahren möglichst mittig in die Kiste fährt.

Kapitel 6

Resümee

6.1 Ergebnisse

In dieser Arbeit ist ein System entstanden, das es ermöglicht, die Steuerung eines Roboters über einen PDA zu realisieren. Durch den PDA ergeben sich viele neue Entwicklungsmöglichkeiten. Das kommt einerseits durch die erhöhte Rechenleistung und die größeren Ressourcen. Andererseits ergeben sich auch viele neue Ansätze zur Entwicklung durch die Kommunikationsmöglichkeiten des PDA. Da der PDA über eine funkbasierte Kommunikation verfügt, ist der Roboter jetzt in der Lage, mit anderen Systemen ohne eine mechanische Verbindung Informationen auszutauschen.

Bisher war es nicht möglich komplexere Programme zu erstellen, da die Möglichkeiten durch Interactive C eingeschränkt sind. Durch die Verwendung einer objektorientierten Programmiersprache ist es nun möglich objektorientierte Software für einen Roboter zu erstellen.

Nun ist es einem Benutzer möglich auf den Roboter Einfluss zu nehmen. Es findet eine Benutzerinteraktion über einen Server statt. Der Roboter kann sich bei Bedarf von dem Server Informationen abholen, um diese zu verarbeiten.

Das System wurde in einer 3-Schicht-Architektur realisiert. Dadurch gibt es eine klare Trennung der Aufgabenbereiche. Die unterste Schicht befindet sich auf dem Controller-Board. Sie ist für die lebenserhaltenden Funktionen zuständig. Sie hat die höchste Priorität. Sie funktioniert unabhängig von den anderen beiden Schichten. Die mittlere Schicht befindet sich auf dem PDA. Auf dem PDA läuft das Planning-Programm. Es entscheidet über die nächsten Aktionen, die der Roboter ausführt. Das Verhältnis zwischen Controller-Board und PDA ist als Subsumption-Architektur realisiert.

6.2 Bewertung der Ergebnisse

In diesem Abschnitt werden die Vor- und Nachteile des entstandenen Systems bewertet.

Durch die mehrschichtige Architektur ist ein flexibles System entstanden. Es ist möglich, je nach den Anforderungen Aufgaben zu verschieben. Da das System in mehrere Komponenten unterteilt ist, lassen sich einzelne Komponenten austauschen. Soll z.B. ein anderes Controller-Board verwendet werden, so ist es möglich, die Roboter-Klasse durch eine speziell angepasste zu ersetzen. Wenn die schnurlose Kommunikation nicht über Bluetooth, sondern über WLAN realisiert werden soll, kann die Server-Klasse ausgetauscht werden. Mit der verwendeten Architektur ist es außerdem möglich, das System relativ einfach zu erweitern.

Auf dem PDA ist der Empfang von seriellen Daten in einem separaten Thread realisiert. Der Thread ist unabhängig von der Roboter-Klasse implementiert. Leider ist es nicht gelungen, einen Weg zu finden, die seriellen Funktionalitäten in die Roboter-Klasse zu kapseln. Der Thread wartet permanent auf eingehende Zeichen und verarbeitet diese. Besser wäre eine Benachrichtigung, ähnlich wie bei den Sockets, da der separate Thread Ressourcen kostet.

Die Einbettung der BTAccess-Bibliothek in die Server-Klasse ist nicht komplett gelungen. Die Bibliothek ist nicht für autonome Anwendungen ausgelegt. Sie ist für Anwendungsprogramme mit grafischer Benutzeroberfläche konzipiert. Daher ist es nicht gelungen, einige Behandlungsroutinen in die Server-Klasse zu integrieren. Wenn Beispielsweise in Gerät gefunden wird, dann wird zuerst die Dialog-Klasse benachrichtigt. Von dort wird die Information an die Server-Klasse durchgereicht.

6.3 Aussichten

Bisher war die Programmierung des Roboters auf eine funktionsorientierte Programmiersprache begrenzt. Durch die Verwendung einer objektorientierten Programmiersprache für den PDA und den Server ergeben sich neue Möglichkeiten. Es ist denkbar, statt die Sensorwerte in Variablen zu speichern, Sensor-Objekte zu verwenden. Die Sensor-Objekte könnten für jeden einzelnen Sensor Minimal-, Maximal- und Durchschnittswerte usw. enthalten. Weitere Informationen über objektorientierte Sensoren gibt es in der Diplomarbeit von Hartmut Koschnitzke (Koschnitzke, 2003).

Durch die Verwendung eines PDA mit einem Funkmodul ist es nun auch möglich, dass die Roboter untereinander oder mit einem Server kommunizieren können.

Eine weitere Anwendung, die auf dieser Arbeit aufbauen könnte, wäre die Kombination mit einem GPS-Empfänger. So könnte der Roboter seine aktuelle Position dem Server mitteilen. Dies funktioniert jedoch nur im Freien, da ein GPS-Empfänger in Gebäuden keine Signale empfangen kann. Außerdem wäre eine Kombination mit einem Kameraserver denkbar. Hier könnte mit der Hilfe einer Kamera die Position des Roboters bestimmt werden. Diese Technik könnte für den Einsatz im Roboterfußball genutzt werden. Den Robotern könnte mitgeteilt werden, wo sich gerade der Ball und/oder die anderen Mitspieler befinden.

Die Diplomarbeit von Rainer Balzerowski (Balzerowski, 2002) befasst sich mit der "Realisierung eines Webcam basierten Kamera Systems für mobile Roboter". Diese Arbeit wurde mit einem Roboter, gebaut aus dem LEGO-MINDSTORM-Paket, realisiert.

Die Studienarbeit von Ilia Revout (Revout, 2003b) beschreibt die "Konzeption eines Frameworks für Kamerabildanalyse". Zeitgleich zu dieser Arbeit entsteht die Diplomarbeit von Ilia Revout (Revout, 2003a). In der Arbeit entsteht eine Schnittstelle für die Verwendung der Kamerabildanalyse.

Aufbauend auf den Arbeiten von Rainer Balzerowski, Ilia Revout und dieser Arbeit wird von Michael Gottwald (Gottwald, 2003) ein weiteres System entwickelt. Es soll ermöglicht werden, dass ein Roboter über eine CORBA-Schnittstelle auf die Informationen eines Kameraservers zugreift.

Die Verwendung des CSocket-Programmiermodells bietet interessante Möglichkeiten. Das Programmiermodell ermöglicht das Versenden von Objekten. So ist die Verwendung von Sensor-Objekten denkbar. Auf diese Weise könnten die Objekte ohne großen Aufwand über die TCP/IP-Verbindung übertragen werden.

Da die Realisierung dieser Arbeit in Module gekapselt ist, können einzelne Module ausgetauscht werden. Es wäre denkbar die Kommunikation der Server-Klasse statt mit Bluetooth über WLAN zu realisieren. WLAN bietet eine größere Reichweite und hat eine höhere Bandbreite. Leider ist der Energieverbrauch von WLAN höher. Da die Arbeit mit Bluetooth in dieser Arbeit oft Probleme bereitet hat, liegt die Hoffnung darin, dass WLAN einfacher zu handhaben ist.

Die neueren PDA's, wie z.B. der HP 5550, sind mit WLAN und Bluetooth ausgestattet. So könnte z.B. die Kommunikation mit einem Server über WLAN realisiert werden und die Kommunikation zwischen den Robotern mit Bluetooth. So könnten

die Roboter, die sich in der näheren Umgebung befinden per Bluetooth miteinander kommunizieren und somit Energie sparen.

Da der Roboter über eine TCP/IP-Verbindung verfügt, wäre es auch denkbar eine Verbindung zu anderen Computern in einem Netzwerk oder sogar im Internet aufzubauen. Es könnte Benutzern ermöglicht werden, auf einer Internetseite Aufträge oder andere Informationen für den Roboter zu hinterlegen. Eine weitere Variante wäre die Möglichkeit, dem Roboter E-Mails mit Aufträgen zu schicken, die er sich dann abruft und abarbeitet.

Neue Chancen ergeben sich auch durch die Möglichkeit, mit Dateien zu arbeiten. Es könnten Daten aufgezeichnet und später ausgewertet werden. Es könnten auch Informationen über einen längeren Zeitraum gespeichert werden. Bisher waren Informationen nach dem Beenden eines Programms verloren. Zusätzlich könnten Werte für die Initialisierung und Speicherung von Einstellungen eines Programms verwendet werden.

Anhang A

Serielle Schnittstelle des MIT-6.270-Boards

Sending/receiving serial from 6.270 board

When IC runs, the 6.270 board's serial port is set for 9600 baud, no parity, 1 stop bit. Normally the serial port is used for communication between the pcode on the board and IC running on the host. However, if you disconnect the board from IC on the host, it is possible for a program on the board to take over the serial port.

Transmitting Characters

Transmitting characters is pretty easy; you basically just poke directly to the 6811's UART hardware. Here's how:

```
void serial_putchar(int c)
{
 /* wait until serial transmit empty */
   while (!(peek(0x102e) & 0x80));
   poke(0x102f, c);  /* send character */
}
```

Remember to stop IC (or disconnect the board) before you send characters out, since any characters you send will most likely confuse IC quite a bit.

Also, if you connect the board to a terminal emulator or other program/device which might send characters to the board, be sure to disable the pcode's handling of incoming serial (described in the next section). Otherwise, as soon as you send a character to the board, the board will wedge waiting for the rest of the packet (in the format IC would send). (One sign this is happening is that the heartbeat stops.)

Receiving Characters

Receiving characters is only slightly more hard. You first have to tell the pcode running on the board to not pick up the characters itself. Note that this will disable downloading and other interaction if you hook back up to IC. This means you have to either reset the board or re-enable the pcode serial to get IC to interact with the board.

Here's the code:

```
/* necessary to receive characters using
 serial_getchar */

void disable_pcode_serial()
{
   poke(0x3c, 1);
}
/* necessary for IC to interact with board again */
void reenable_pcode_serial()
{
   poke(0x3c, 0);
}
int serial_getchar(int c)
{
 /* wait for received character */
   while (!(peek(0x102e) & 0x20));
   return peek(0x102f);
}
```

Randy Sargent rsargent@media.mit.edu

Anhang B

FAQ

Bei der Entwicklung dieser Arbeit sind viele Probleme aufgetaucht. So gab es z.B. Probleme mit der Entwicklungsumgebung für den PDA oder Probleme mit der Verwendung von Bluetooth. Der folgende Abschnitt soll eine Hilfestellung für nachfolgende Arbeiten sein. Die Fragen und Antworten sind an Personen gerichtet, die sich selbst ebenfalls mit der Thematik oder Teilbereichen dieser Arbeit auseinandersetzen möchten. Die Probleme sind auf die hier benutzte Soft- und Hardware bezogen. Einige Probleme sind eventuell schon hinfällig geworden, da es teilweise schon wieder neuere Versionen von Soft- und Hardware gibt. Der aktuelle Stand ist September 2003.

B.1 MIT-6.270-Board

Warum gehen bei der seriellen Kommunikation mit dem MIT-6.270-Board Bytes verloren?

Die serielle Schnittstelle des MIT-6.270-Boards bietet leider keine Handshake-Leitungen. Es gibt eine Leitung zum Senden, eine zum Empfangen und zwei Masse-Leitungen. Daher können bei einer Übertragungsgeschwindigkeit von 9600 Baud/s Bytes verloren gehen. Daher empfiehlt sich ein Handshake-Verfahren auf Software-Ebene. Mit folgendem Code kann man die Schnittstelle auslesen und warten bis der Empfangspuffer wieder leer ist. Ist der Puffer wieder leer, kann an den Sender eine Bestätigung geschickt werden, dass das nächste Byte gesendet werden kann.

```
int serial_getchar()
```

```
{
   int Temp;

   /*wait for received character*/
   while (!(peek(0x102e) & 0x20));

   Temp = peek(0x102f);

   /*wait for free receive-buffer*/
   while ((peek(0x102e) & 0x20));

   return Temp;
}
```

B.2 Bluetooth und TCP/IP

Warum läuft ActiveSync nicht über Bluetooth mit dem Acer Adapter?

Mit dem Acer Bluetooth-Adapter gab es unter Windows XP Probleme beim Einrichten einer ActiveSync-Verbindung. Das Problem lag in der Bluetooth-Software. Mit einem TDK Bluetooth-Adapter und der entsprechenden Software hat der Aufbau auf dem gleichen System funktioniert. Ein späterer Versuch den Acer Bluetooth-Adapter mit der TDK-Software zum Laufen zu bekommen hat funktioniert. Die Ursache scheint der sogenannte "Bluetooth-Stack COM-Server" zu sein. In der Firewall von Zonealarm waren nun zwei COM-Server zu sehen. Es war jedoch nur der TDK COM-Server aktiv.

Nachfolgend ist die Vorgehensweise beschrieben:

1. Deinstallieren der Acer-Software.
2. Installieren der TDK-Software.
3. Anschluss den TDK-Adapters.
4. TDK Bluetooth-Adapter über Bluetooth Manager auf iPAQ suchen. Wenn vorher schon eingetragen, dann aus Liste löschen und neu suchen.

5. Wenn Gerät gefunden, dann auswählen und "ActiveSync Partner" anhaken und im Menü Bindung mit Gerät erstellen. Es wird nach Kennwort gefragt. Hier kann man zum Testen einfach z.B. eine "1" angeben. Nun müsste am PC eine Meldung kommen, wo dieses Kennwort bestätigt werden muss. (Achtung: Eventuell kommt die Meldung am PC nicht hoch, dann Bluetooth-Symbol in der Taskleiste klicken.)

6. Aufbau einer ActiveSync-Verbindung vom iPAQ aus, indem "ActiveSync starten" im BT-Menü aufgerufen wird. (Achtung: Vorher muss evtl. eine Verbindung direkt über das Cradle per USB oder seriellem Anschluss hergestellt werden.)

7. Entfernen des TDK-Adapters

8. Anschluss des Acer-Adapters (in denselben USB-Port) und einlegen der Acer-CD nach Aufforderung durch das Betriebssystem.

9. Punkt 4 und 5 jetzt mit Acer-Adapter wiederholen.

10. Aufbau einer ActiveSync-Verbindung vom iPAQ aus, indem ActiveSync starten im BT-Menü aufgerufen wird.

Wieso erscheint eine Fehlermeldung, wenn die LAN-Access-Verbindung vom PDA aus getrennt wird?

Diese Meldung erscheint leider nicht nur bei einer fehlerhaften Trennung der Verbindung, sondern auch bei einer korrekten Trennung der Verbindung. Diese Meldung ist für den Betrieb im Zusammenhang mit einem autonomen Roboter sehr hinderlich. Das Programm läuft zwar nach einer Trennung weiter, jedoch stürzt das Programm nach mehreren Trennungen und den entsprechenden Meldungen ab. Auch nach Anfrage bei der Firma High Point Software (BTAccess, 2003) ist keine Lösung bekannt, wie man diese Meldung vermeiden kann. Hier ein Ausschnitt aus der E-Mail:

> You are right, the message is not coming from BTAccess but from the PocketPC operating system. I do not know how to eliminate this message. I believe the OS is using the RAS (Remote Access Server) subsystem to manage this kind of connection, so you may be able to read up on this on the web somewhere. Good luck!

Warum lässt sich ActiveSync über Bluetooth nicht einrichten?

Um eine Verbindung mit ActiveSync über Bluetooth zu erstellen, ist es unbedingt notwendig, zuvor eine Verbindung über das serielle oder das USB-Kabel herzustellen. Die Verbindung über Bluetooth wird vom PDA aus über das Bluetooth-Menü gestartet. Eventuell muss zwischen den Geräten eine Partnerschaft eingerichtet werden. Dies hängt von den Einstellungen am PDA und am PC ab.

Wieso verschwindet nach dem Aufbau einer LAN-Access-Verbindung vom PDA das Fenster nicht?

Damit der LAN-Access funktioniert, ist eine Bluetooth-Nullmodem-Verbindung nötig. Weshalb diese benötigt wird ist nicht ganz klar. Die Verbindung wird benötigt, um eine Internetverbindungsfreigabe für den LAN-Access bereitzustellen. Genauere Informationen für die Installation unter Windows 2000 und Windows XP sind auf der CD zu dieser Arbeit zu finden.

Wie kann man testen, ob überhaupt eine Verbindung besteht?

Es gibt ein Programm mit dem Namen vxUtil. Mit dem Programm kann man einige Diagnosen durchführen. So kann man z.B. einen Ping-Befehl an den Kommunikationspartner schicken. Das Programm kann unter www.cam.com kostenlos heruntergeladen werden.

B.3 Embedded Visual C++ 3.0

Warum gibt es einen Fehler, wenn man einen String benutzen möchte?

Windows CE arbeitet auf Systemebene mit Unicode. Daher ist teilweise eine Zeichenkonvertierung notwendig. Mit dem Makro `TEXT("COM1:")` bzw. der Kurzform `_T("COM1:")` werden wie hier der String "COM1:" übergeben. Es wird eine Konvertierung in einen TCHAR Typ vorgenommen.

Warum erscheint keine Menüleiste in meiner erstellten Anwendung?

Hier sind zwei Zeilen Code notwendig, die in OnInitDialog() aufgerufen werden und die Menüleiste sichtbar machen. Die Menüleiste erscheint unten, da das Startmenü beim Pocket PC oben ist.

```
CCeCommandBar *pCommandBar =
(CCeCommandBar)m_pWndEmptyCB;

pCommandBar->InsertMenuBar(IDR_MENUBAR);
```

Warum lassen sich den Steuerelementen keine Member-Variablen zuweisen?

Möchte man ein Steuerelement in einem Dialog unterbringen und diesem Steuerelement mit dem Class-Wizard eine Member-Variable zuordnen, so kann es zu Problemen kommen. Es erscheint eine Fehlermeldung, dass es für das Control keine Typen Informationen gäbe. Das Problem hängt mit einer Einstellung des Projektes zusammen. Man darf das Projekt nicht mit den MFC als statisch verknüpfte Bibliothek einbinden. Man muss es dynamisch machen. Dies kann man auch nachträglich noch in den Einstellungen des Projekts ändern. Es gibt keinen ersichtlichen Grund für dieses Verhalten. Scheinbar ist dies ein Bug. Es ist nämlich möglich die Einstellung auf dynamisch zu stellen, dann die Member-Variablen hinzuzufügen und anschließend die Einstellung wieder auf statisch zu ändern. Per Hand ließ sich die Membervariable auch einbetten. Also scheinbar ein Bug!

Auf welchem Betriebssystem läuft Embedded Visual Tools 3.0?

Auf Windows 95 und Windows 98 (Erste Ausgabe) ist die Installation nicht möglich. Auf Windows 98 SE lässt sich die Entwicklungsumgebung installieren, jedoch ohne Emulatoren. Die Installation unter Windows NT 4.0 funktioniert, jedoch ohne den benötigen Emulator für die Pocket PC 2002 SDK. Diese wird für den iPAQ benötigt. Die vernünftigsten Möglichkeiten sind Windows 2000 und Windows XP.

Wenn man den Emulator startet, stürzt das gesamte System ab. Gibt es eine Lösung?

Dieses Problem ist bei der Verwendung von Windows XP aufgetreten. Leider ist keine Lösung bekannt.

Wie bekommt man einen Pointer auf eine Dialogklasse?

Mit `AfxGetMainWnd()` holt man sich einen Pointer auf das Dialogfenster. Die Methode `AfxGetMainWnd()` gibt einen Pointer auf ein CWnd-Objekt zurück. Deshalb muss man einen Typecast auf die entsprechende Dialogklasse machen.

Warum stürzt die Anwendung ab, wenn UpdateData(FALSE) aufgerufen wird?

In dieser Arbeit war es notwendig einen Thread zu erstellen, der den Empfang der seriellen Schnittstelle übernimmt. Sind Daten angekommen so sollte von dem Thread eine Methode der Dialogklasse aufgerufen werden, um die empfangenen Zeichen auszuwerten bzw. anzuzeigen. Wurde in dieser Methode `UpdateData(FALSE)` aufgerufen, um die Anzeige zu aktualisieren, so stürzte das Programm mit einer Meldung ab. In der Datei wincore.cpp ist der Fehler in einer Fehlerbehandlungsroutine ausgelöst worden. Der Beschreibung nach darf die Methode `UpdateData(FALSE)` nicht von einem Thread aus aufrufen. Um das Problem in den Griff zu bekommen wurde die Aktualisierung der Anzeige mit einem Timer realisiert. Auf diese Weise wird die Methode `UpdateData(FALSE)` zyklisch aufgerufen. Nachfolgend der Kommentar aus der Datei wincore.cpp:

```
Note: if either of the above asserts fire and you are
writing a multithreaded application, it is likely that
you have passed a C++ object from one thread to another
and have used that object in a way that was not intend-
ed.(only simple inline wrapper functions should be
used)

In general, CWnd objects should be passed by HWND from
one thread to another.  The receiving thread can wrap
the HWND with a CWnd object by using CWnd::FromHandle.
```

```
It is dangerous to pass C++ objects from one thread to
another, unless the objects are designed to be used in
such a manner.
```

Wie lassen sich Strings konvertieren?

Es gibt Funktionen um von Multibyte-Character nach Unicode (Wide-Character) und umgekehrt zu wandeln. Die Funktionen heißen `wcstombs()` und `mbstowcs()`. Von Multibyte Character nach Unicode kann man auch recht einfach konvertieren, indem man einen char* einer CString Variablen zuweist.

Welche Vorteile hat die Klasse CCeSocket?

Die Klasse ist abgeleitet von der CSocket-Klasse. Um asynchrone Kommunikation unter Windows CE zu verwenden muss man die Klasse CCeSocket benutzen. Windows CE unterstützt in sich keine asynchrone Socket-Kommunikation, aber die Klasse CCeSocket bietet asynchrone Benachrichtigung für die folgenden Typen von Socket-Events.

- Benachrichtigung, dass der Socket zum Lesen bereit ist (`FD_READ`)
- Benachrichtigung, von eingehenden Verbindungen (`FD_ACCEPT`)
- Benachrichtigung, von hergestellten Verbindungen (`FD_CONNECT`)

Bemerkungen

Man sollte die Klasse CCeSocket bzw. eine von CCeSocket abgeleitete Klasse für Windows CE Anwendungen bevorzugen, bevor man die Klassen CSocket oder CAsyncSocket verwendet. Die CSocket- und CAsyncSocket-Klassen arbeiten auf den Standard-Windows Message-basierten Benachrichtigungen von Netzwerk-Events. Ohne Zugriff auf die asynchrone Event-Benachrichtigung muss eine Anwendung im Polling-Betrieb die gewünschten Events abfragen. Das ist aber sehr

uneffizient. Die CCeSocket-Klasse erreicht den Effekt der asynchronen Event-Benachrichtigung in dem zwei Threads erzeugt werden die Socket-Events darstellen. Wenn ein Thread eines der drei Events entdeckt, für welche die CCeSocket-Klasse Benachrichtigungen anbietet, sendet er eine Nachricht an den entsprechenden Handler. Alle Instanzen der Klasse CCeSocket teilen sich die gleichen zwei Threads zur Benachrichtigung.

Ein bedeutender Vorteil der Klasse CCeSocket ist die Verwendung der Klassen CSocketFile und CArchive. Dies macht den Aufbau zwar etwas komplexer, jedoch übernimmt die Klasse CArchive die Zeichenkonvertierung zwischen Standard-Windows und Windows CE bzw. von MBCS nach Unicode. Außerdem ist es durch diese Kombination möglich nicht nur Strings, sondern auch Objekte zu senden.

Wieso hängt sich mein Programm auf, wenn OnReceive() der Klasse CCeSocket aufgerufen wird?

Es kann vorkommen, dass alle Funktionalitäten der CCeSocket-Klasse funktionieren bis auf eine. Man kann eine Verbindung aufbauen und abbauen. Das Senden von Daten sollte auch funktionieren. Es kann jedoch zu einem Problem beim Empfangen von Daten kommen. Wenn `OnReceive()` der CCeSocket-Klasse aufgerufen wird, dann wird es so lange aufgerufen, bis sich der PDA aufhängt.

Die Methode `OnReceive()` der Klasse CCeSocket arbeitet nur vernünftig, wenn man sie mit einer CArchive-Klasse kombiniert. Die Klasse CArchive benötigt wiederum ein Objekt der Klasse CSocketFile. Informationen über dieses CSocket-Programmiermodell findet man in der Beschreibung der Klasse CSocket.

Wieso funktioniert die Serialisierung nicht?

Habe letzte Versuche gemacht, ein für mich brauchbares laufendes Programm mit CSocket zu erstellen. Wie es aussieht, ist es nur sinnvoll, die CSocket-Architektur anzuwenden, wenn man ein SDI- oder MDI-Projekt verwendet, da diese Projekte schon mit den nötigen Ressourcen für die Serialisierung ausgestattet sind.

Warum läuft die BTAccess-Bibliothek nicht?

Beim Versuch ein eigenes dialogbasiertes Programm mit der BTAccess-Bibliothek zu schreiben kann es zu Problemen kommen. Es kam trotz eingebundener Bibliothek und Header-Dateien vor, dass beim Aufruf von Methoden der Bibliothek nichts passiert. Das Programm ließ sich normal ausführen, es konnte ein CBtStack-Objekt der CBtStack-Klasse erzeugt werden. Beim Versuch eine Verbindung zum Bluetooth-Stack aufzubauen wollte, passierte nichts. Es kommen keine Fehlermeldungen bzw. Laufzeitfehler. Aufgehängt hat sich die Anwendung scheinbar auch nicht, da man noch andere Funktionen aufrufen konnte. Wenn man einen Pointer mit dem "`->`" Operator auf das Objekt benutzen wollte, so hat die Entwicklungsumgebung keine Methoden usw. angezeigt. Nach einer Anfrage per E-Mail an die Firma High Point Software (BTAccess, 2003) kam folgende Antwort. Hier ein Auszug aus der E-Mail:

> As for getting your application to work, I'm not sure what's different. If it compiles correctly you must have the include-directories, etc specified correctly. One problem may be if you're compiling your program as a Debug build you may have runtime errors (try compiling as a Release build). When you purchase the product we also supply a debug-version of the BTAccess dll for use with a Debug build application.
>
> Tim Johnson High Point Software (503) 312-8625

Beim Versuch das Projekt wie in der E-Mail beschrieben als "Release" zu kompilieren, funktionierte das Programm immer noch nicht wie gewünscht. Nach Überprüfung der Projekteinstellungen für die Art der Einbindung der MFC-Klassen hat es plötzlich funktioniert. Bisher waren die MFC statisch eingebunden. Nachdem die MFC nun einmal gar nicht eingebunden wurden und anschließend kompiliert wurde funktionierte es selbstverständlich nicht. Anschließend wurden sie als shared Dll eingebunden und kompiliert. Danach hat es funktioniert. In Visual SourceSafe war es nun möglich die Veränderungen des Projektes zu vergleichen. Vermutlich handelt es sich hier um dieselbe Ursache wie bei dem Problem, Steuerelementen eine Membervariable zuzuweisen. Die Problematik wird in diesem Abschnitt unter der Frage "Warum lassen sich den Steuerelementen keine Membervariablen zuweisen?" behandelt. Dies scheint ein Bug zu sein. Die Screenshots aus Soure Safe zeigen die Veränderungen in der Projekt-Datei, die nach der Veränderung der Projekteinstellungen aufgetreten sind.

Merke: MFC wenn möglich als shared Dll einbinden.

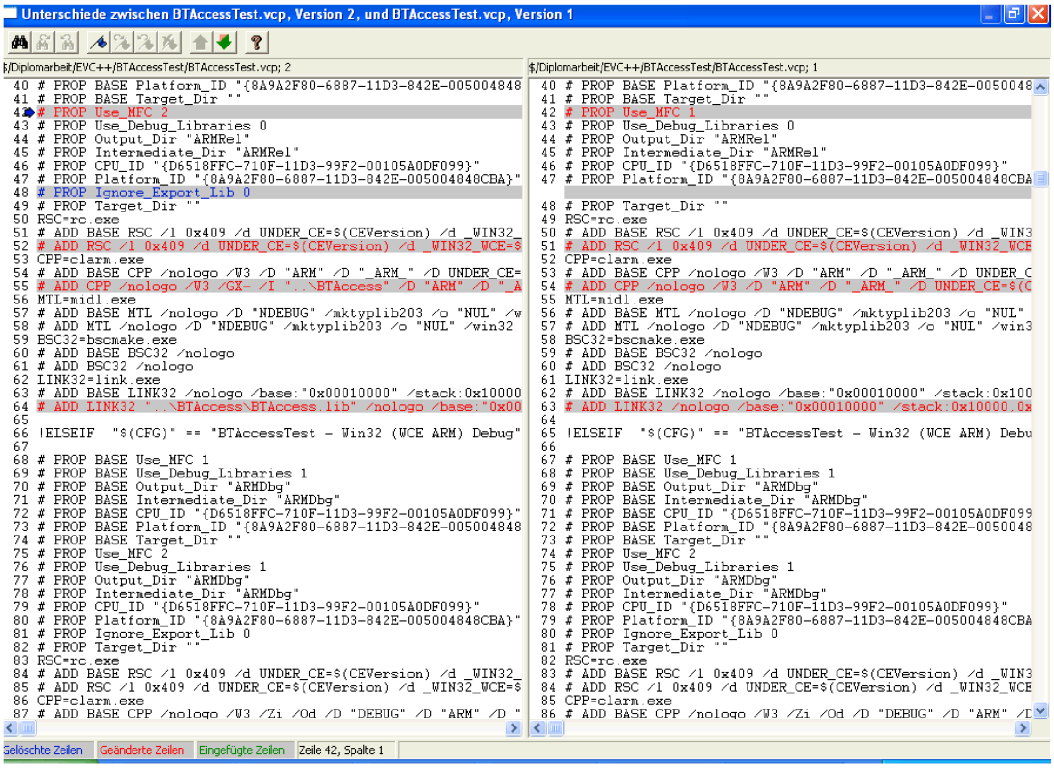

Abbildung B.1: Screenshot aus Visual SourceSafe (1)

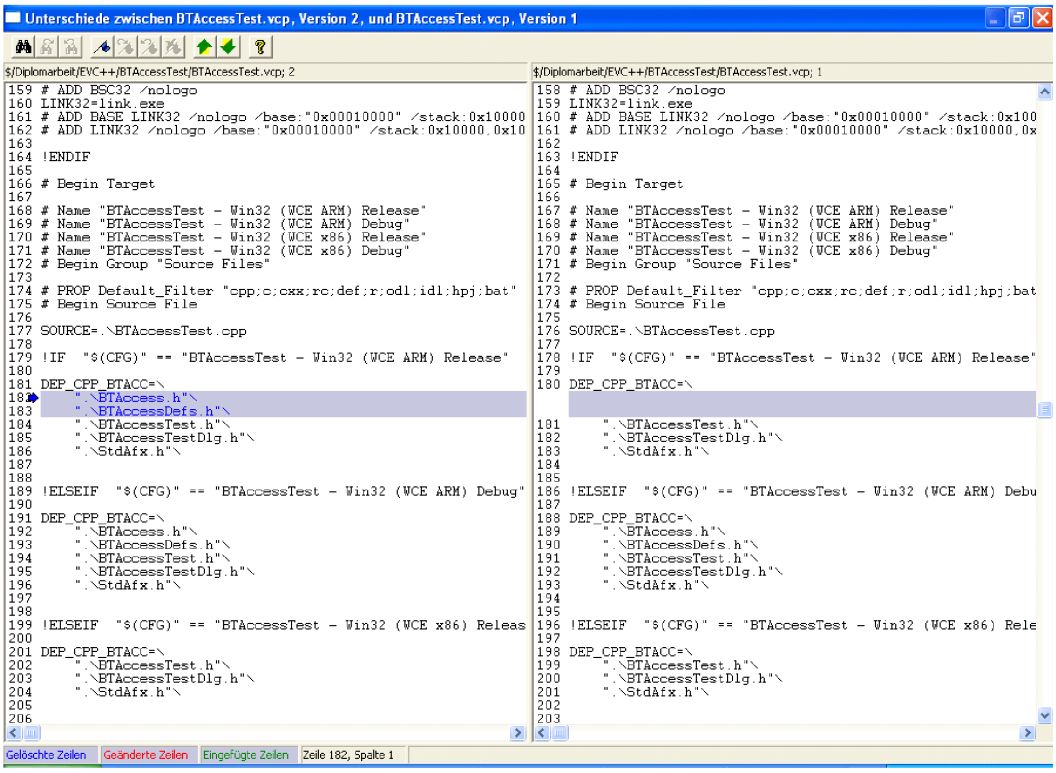

Abbildung B.2: Screenshot aus Visual SourceSafe (2)

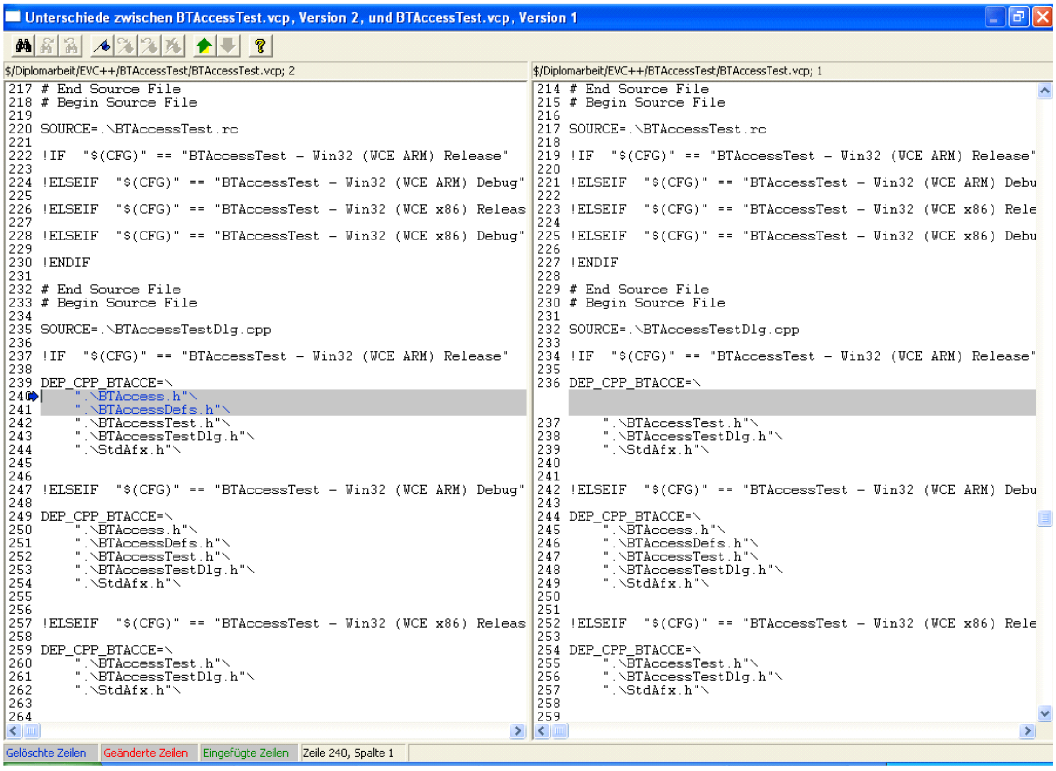

Abbildung B.3: Screenshot aus Visual SourceSafe (3)

Hier noch ein paar Tipps für die Projekteinstellungen. Folgende Einstellungen sollten unter dem Menüpunkt Project->Settings überprüft werden.

- Reiter *C/C++* unter *Category: "Preprocessor"* auswählen. Unter *Additional include Directories:* sollte `..\BTAccess` eingetragen sein. Voraussetzung dafür ist, dass sich der Ordner BTAccess im selben Ordner befindet, wie der Ordner des Projektes.

- Reiter *Link* unter *Category: "General"* auswählen. Unter *Object/library modules:* sollte `"..\BTAccess\BTAccess.lib"` eingetragen sein.

Damit die BTAccess-Bibliothek funktioniert, muss z.B. per ActiveSync unbedingt die Datei BTAccess.dll in das Verzeichnis `/Windows` auf dem iPAQ kopiert werden. Ist dies nicht der Fall, so kann es zu uneindeutigen Fehlermeldungen kommen.

Ist es möglich per Code eine ActiveSync-Verbindung aufzubauen?

Nach einer Anfrage per E-Mail an die Firma High Point Software (BTAccess, 2003) kam folgende Antwort. Hier ein Auszug aus der E-Mail:

> After you create a serial port Bluetooth connection to the other device you can launch the ActiveSync program as follows:
>
> ```
> PROCESS_INFORMATION  pi;
> DWORD rc =
> CreateProcess (_T("\\Windows\\repllog.exe"),
> _T("/auto /c:\"BlueToothActiveSync\""),
> NULL, NULL, FALSE, 0, NULL,NULL, NULL, &pi);
> ```
>
> p.s. you may need to use this alternative command-line argument to start ActiveSync:
>
> ```
> DWORD rc =
> CreateProcess (_T("\\Windows\\repllog.exe"),
> _T("/AppRunAtRs232Detect"),
> NULL, NULL, FALSE, 0, NULL, NULL, NULL, &pi);
> ```

Der Aufbau einer ActiveSync-Verbindung hat funktioniert. Jedoch war es nicht unbedingt notwendig, vorher eine serielle Verbindung mit der BTAccess-Bibliothek aufzubauen. Dies machte sogar Probleme. Wir die Datei repllog.exe auf dem iPAQ gestartet, so wird ActiveSync gestartet, als ob man in dem Bluetooth-Menü des PDA's "ActiveSync starten" auswählt. Leider ist es nicht gelungen die ActiveSync-Verbindung wieder zu trennen.

Wieso werden von der BTAccess-Bibliothek keine Geräte gefunden?

Bei der Suche von Geräten mit BTAccess gibt es folgendes zu beachten.

- Der Energieverbrauch ist höher

- Während der Suche kann der iPAQ nicht von anderen Geräten entdeckt werden
- Während der Suche können andere Geräte keine Verbindungen zum iPAQ aufbauen
- Es kann vorkommen, dass ein und dasselbe Gerät mehrfach gefunden wird
- Suche findet nach ca. 30 Sekunden nichts mehr. Neustart der Suche erforderlich.

Welcher Anwendungstyp ist zu bevorzugen?

Beim Übergeben der Sensorwerte an die Oberfläche ist das SDI-Projekt immer abgestürzt. Der Versuch das Fenster mit `SendMessage()` zu benachrichtigen hat aber leider nicht funktioniert. Der Versuch, einen Funktionspointer zu verwenden, war ebenfalls nicht erfolgreich. Da es in einer dialogbasierten Anwendung bisher funktioniert hat, habe ich beschlossen die Socket-Funktionen in eine dialogbasierte Anwendung zu übernehmen und die Anzeige mit Hilfe eines Timers zu aktualisieren. Dieser Weg hat funktioniert.

Wie kann man die Registrierung des PDA bearbeiten?

Die Entwicklungsumgebung Embedded Visual C++ bietet einen Remote Registry Editor. Damit ist es möglich z.B. bei bestehender ActiveSync-Verbindung vom PC aus die Registrierung zu bearbeiten.

Wie kann man Screenshots vom PDA erstellen?

Von Microsoft gibt es die Windows Mobile Developer Power Toys. Diese beinhalten unter anderem das Programm "ActiveSync Remote Display". Mit diesem Programm kann man den Bildschirm des PDA auf einem PC darstellen. Durch die Tastenkombination "Alt + Druck" kann man den aktuellen Bildschirm in den Zwichenspeicher laden und z.B. in ein Zeichenprogramm einfügen.

Wie bekommt man die IP-Adresse des PC's heraus?

Wenn eine Verbindung über ActiveSync besteht, kann man die IP-Adresse der ActiveSync-Verbindung mit der Funktion `gethostbyname("ppp peer")` herausfinden. Wenn man ein CCeSocket-Objekt verwendet, kann man statt der IP-Adresse den String `"ppp_peer"` direkt beim Aufruf von `Connect()` angeben.

Wird der LAN-Access verwendet, so sollte man die Adresse des PC's kennen. So kann man z.B. die Adresse des LAN-Access Server Treibers auf "192.168.0.1" einstellen. So hat man eine Feste Adresse mit der man sich verbinden kann.

Wieso erscheint eine Fehlermeldung, wenn ich mein Programm ausführen möchte?

Möchte man sein Programm von der Entwicklungsumgebung aus ausführen oder kompilieren, so kann es zu einer Fehlermeldung kommen. Eine Meldung kann sich darauf beziehen, dass die .exe Datei nicht auf das Gerät heruntergeladen werden kann. Die Meldung bezieht sich eventuell auf den Downloadpfad. Auf dem PDA befindet sich das Startmenü in der Regel im Pfad `\Windows\Startmenü`. In der Entwicklungsumgebung ist im Normalfall der Pfad `\Windows\Start Menu` eingetragen. Es ist auch schon vorgekommen, dass die Fehlermeldung keinen Hinweis auf den falschen Downloadpfad enthielt. In dem Fall wurden nur gemeldet, dass die Datei nicht heruntergeladen werden konnte.

Wieso lässt sich die serielle Schnittstelle des iPAQ nicht öffnen?

Die serielle Schnittstelle wird mit dem Befehl `CreateFile()` geöffnet. Als Dateiname wird der Name der serielle Schnittstelle angegeben. Dieser muss mit dem Makro `TEXT("COM1:")` übergeben werden. Besonders wichtig ist die Angabe des Doppelpunktes. In einigen Dokumentationen ist der Doppelpunkt nicht angegeben, bzw. in Standard-Windows ist dies nicht notwendig.

Anhang C

LAN-Access Server Konfiguration

Ein wichtiger Bestandteil dieser Arbeit ist ein funktionierender LAN-Access über Bluetooth. Hier kam es immer wieder zu Problemen, da nicht klar war, wie dieser Zugang zu konfigurieren ist. Es muss nicht nur der LAN-Access Server des Bluetooth-Adapters konfiguriert werden, sondern es muss eine Bluetooth-Nullmodem-Verbindung erstellt werden. Diese stellt eine Intnetverbindungsfreigabe für den LAN-Access Server zur Verfügung. Die Bluetooth-Nullmodem-Verbindung ist unbedingt erforderlich. Ohne diese Verbindung ist es nicht gelungen, eine LAN-Verbindung über Bluetooth zu erstellen. Nachfolgend wird die Installation für Windows 2000 und für Windows XP beschrieben. Die Installation ist bei beiden Systemen sehr ähnlich, jedoch war nicht immer klar, welche Einstellungen vorzunehmen sind.

C.1 Anleitung für Windows 2000

Bluetooth-Nullmodem

Sofern noch keine Bluetooth-Nullmodem-Verbindung vorhanden ist, muss diese erstellt werden. Hierbei handelt es sich um eine Direktverbindung. Die Verbindung für Bluetooth-Nullmodem auswählen und die Einstellungen aufrufen. Die Verbindung muss als gemeinsam genutzte Internetverbindung freigegeben werden. Sie sollte automatisch eine IP-Adresse aus dem Bereich 192.168.0.x zugewiesen bekommen.

Unter "Gemeinsame Nutzung der Internetverbindung" den Namen der Verbindung für den Bluetooth LAN-Access Server auswählen.

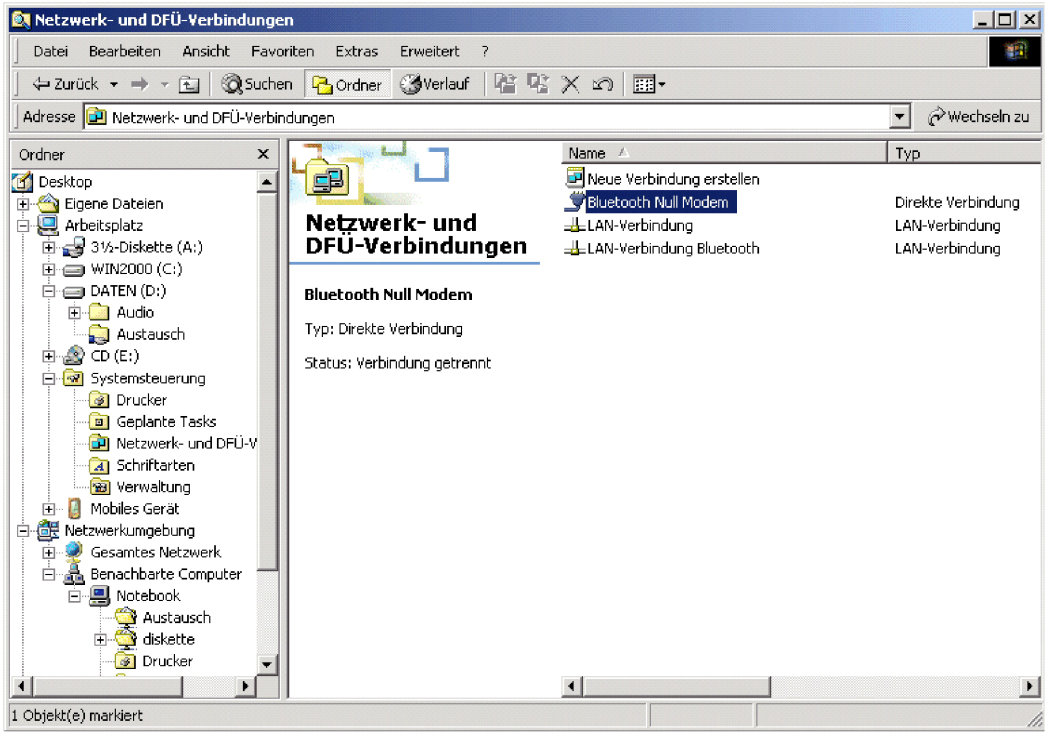

Abbildung C.1: Windows 2000: Bluetooth-Nullmodem auswählen

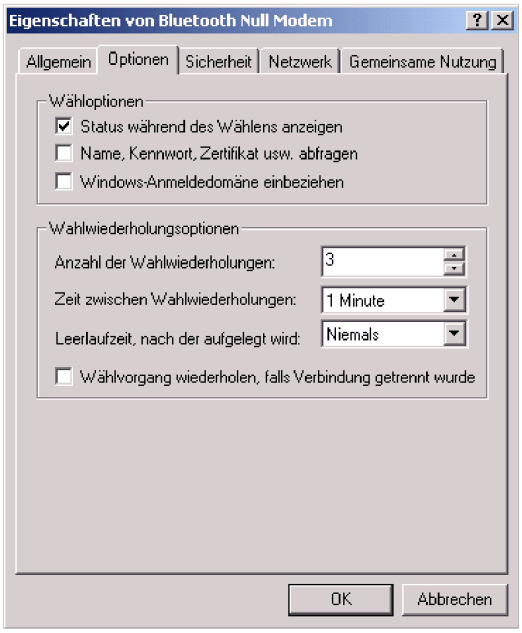

Abbildung C.2: Windows 2000: Optionen von Bluetooth-Nullmodem

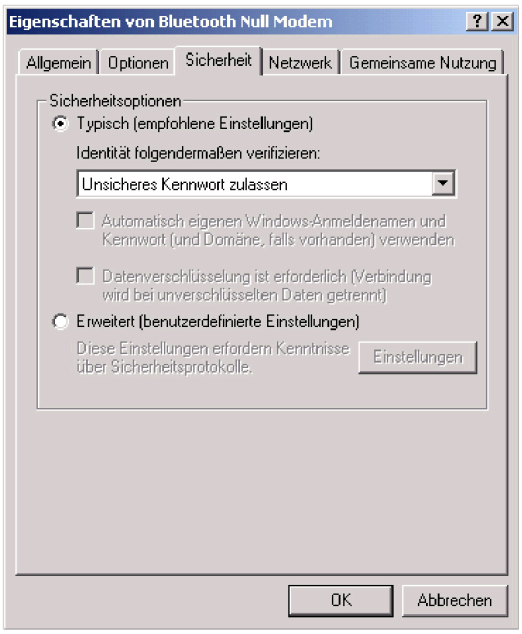

Abbildung C.3: Windows 2000: Sicherheitseinstellungen von Bluetooth-Nullmodem

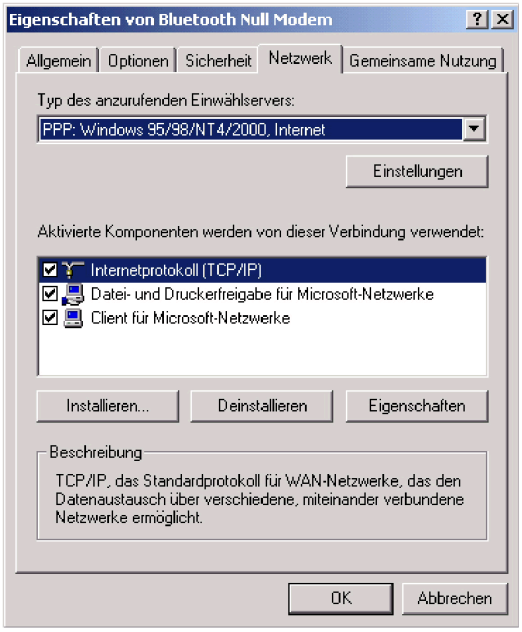

Abbildung C.4: Windows 2000: Netzwerkeinstellungen von Bluetooth-Nullmodem

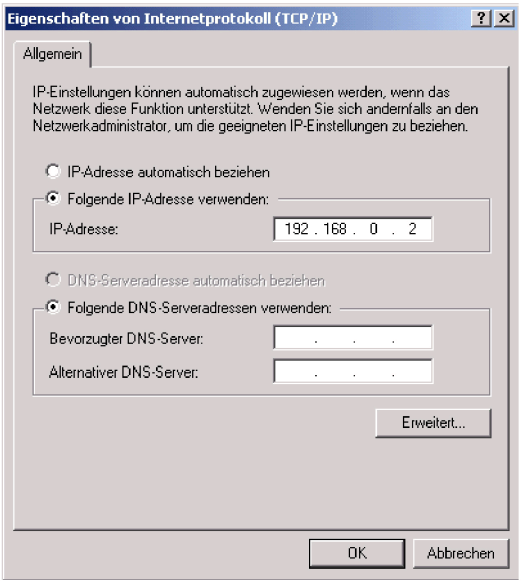

Abbildung C.5: Windows 2000: TCP/IP-Einstellungen von Bluetooth-Nullmodem

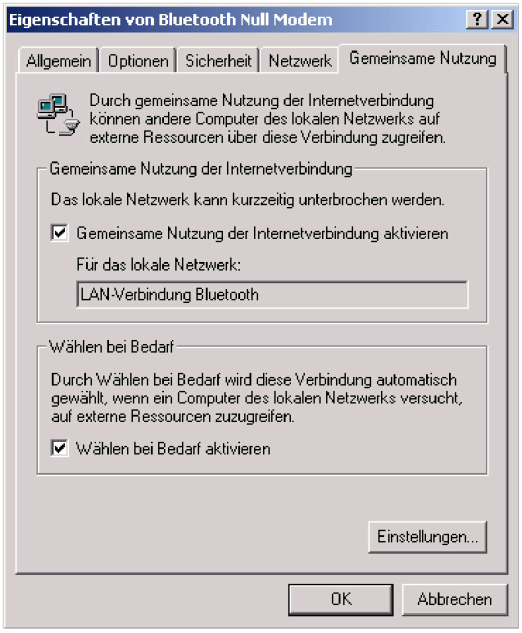

Abbildung C.6: Windows 2000: Gemeinsame Nutzung von Bluetooth-Nullmodem

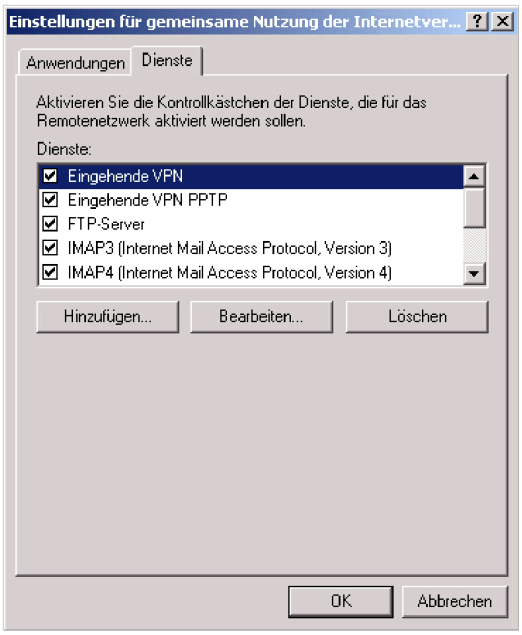

Abbildung C.7: Windows 2000: Dienste von Bluetooth-Nullmodem (1)

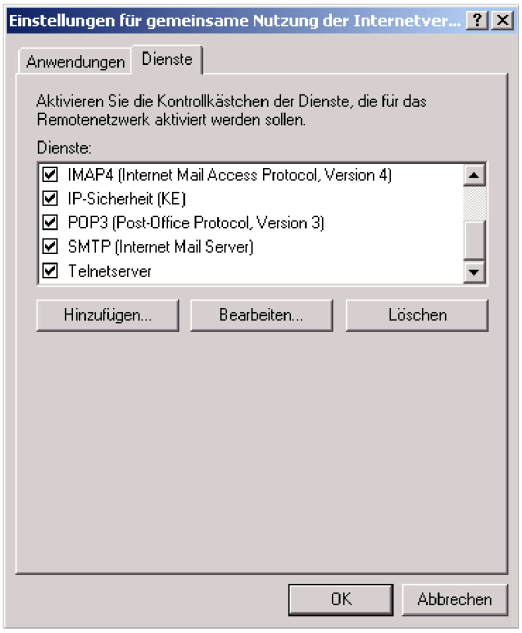

Abbildung C.8: Windows 2000: Dienste von Bluetooth-Nullmodem (2)

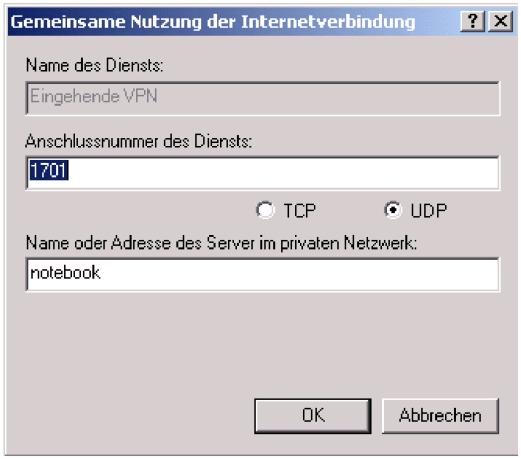

Abbildung C.9: Windows 2000: Dienst für Eingehende VPN hinzufügen

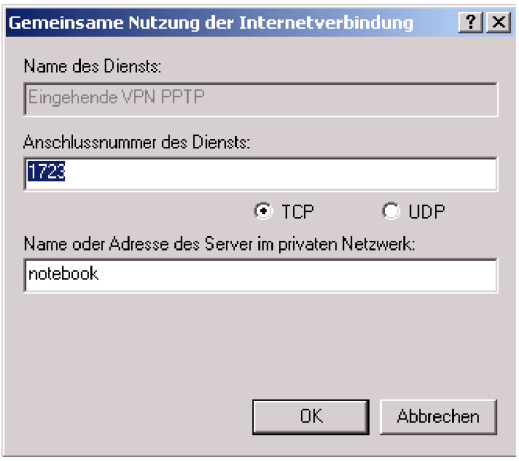

Abbildung C.10: Windows 2000: Dienst für Eingehende VPN (PPTP) hinzufügen

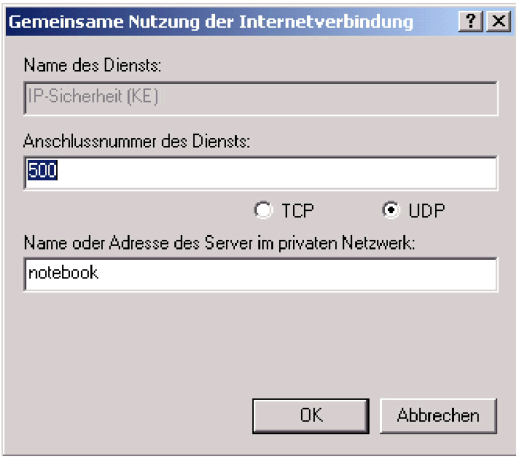

Abbildung C.11: Windows 2000: Dienst für IP-Sicherheit hinzufügen

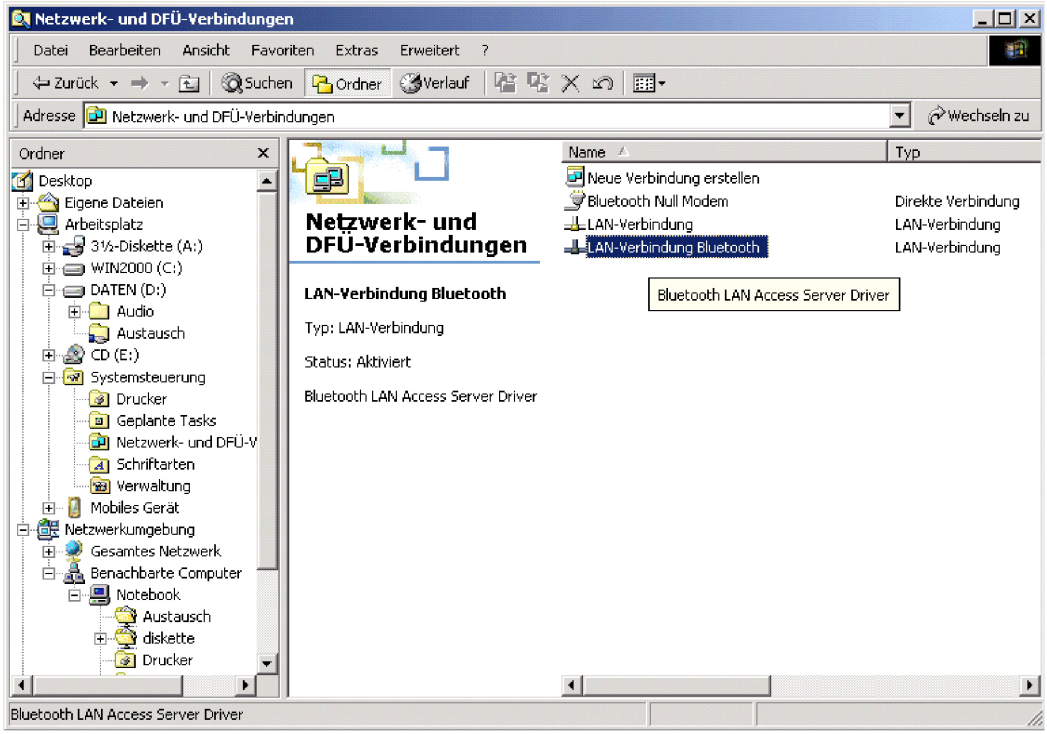

Abbildung C.12: Windows 2000: LAN-Verbindung für Bluetooth auswählen

LAN-Verbindung für Bluetooth

Die Einstellungen für das Internetprotokoll kontrollieren. In der Regel sollte die IP-Adresse auf "192.168.0.1" eingestellt werden. Unter dieser Adresse ist dann der LAN-Access für den PDA zu erreichen.

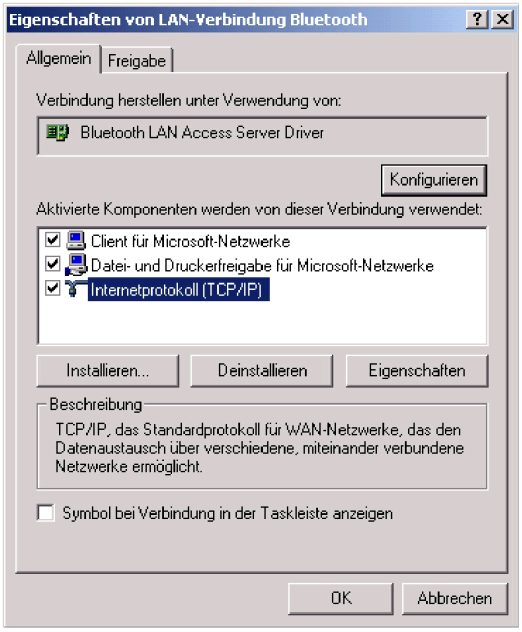

Abbildung C.13: Windows 2000: Netzwerkeinstellungen

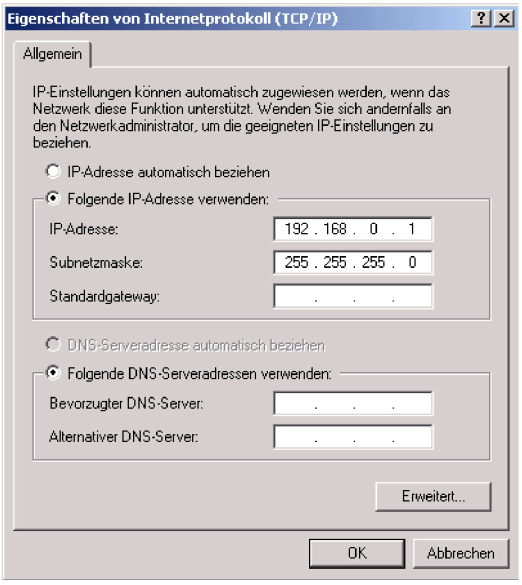

Abbildung C.14: Windows 2000: TCP/IP-Einstellungen

Abbildung C.15: Windows 2000: Freigabe

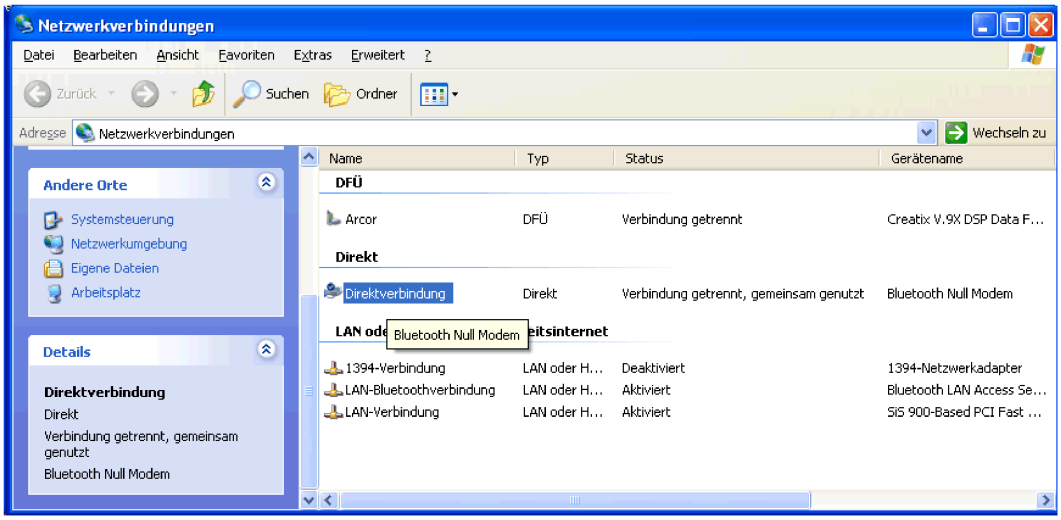

Abbildung C.16: Windows XP: Verbindung für Bluetooth-Nullmodem auswählen

C.2 Anleitung für Windows XP

Sollte unter Windows XP eine Meldung erscheinen, dass z.B. die Adresse "192.168.0.1" schon in Benutzung ist, so sollten alle Netzwerkverbindungen so eingestellt werden, dass die IP-Adresse automatisch zugewiesen wird. Dies gilt auch für die Netzwerkkarte. Dann sollte die Adresse für den LAN-Access richtig übernommen werden. Danach können für die anderen Geräte die IP-Adressen wieder fest eingestellt werden.

Bluetooth-Nullmodem

Sofern noch keine Bluetooth-Nullmodem-Verbindung vorhanden ist, muss diese erstellt werden. Hierbei handelt es sich um eine Direktverbindung. Die Verbindung für Bluetooth-Nullmodem auswählen und die Einstellungen aufrufen. Die Verbindung muss als gemeinsam genutzte Internetverbindung freigegeben werden. Sie sollte automatisch eine IP-Adresse aus dem Bereich 192.168.0.x zugewiesen bekommen.

Im Register "Sicherheit" die Einstellung auf "Typisch" belassen. Im Register "Netzwerk" die IP-Adresse kontrollieren. Eventuell die Adresse auf 192.168.0.2 einstellen.

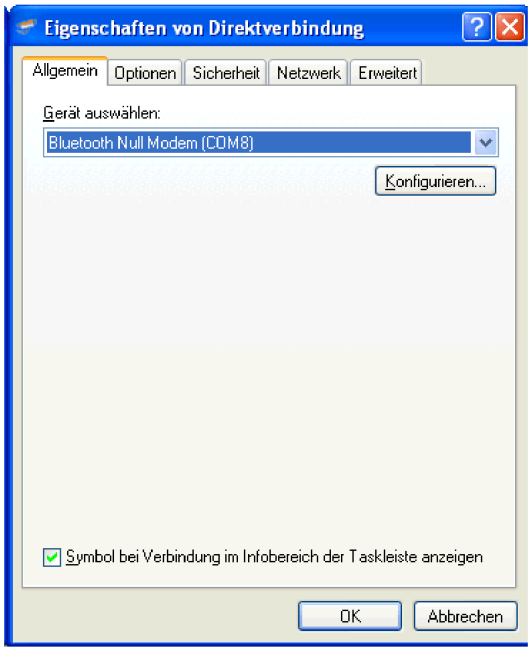

Abbildung C.17: Windows XP: Allgemeine Einstellungen von Bluetooth-Nullmodem

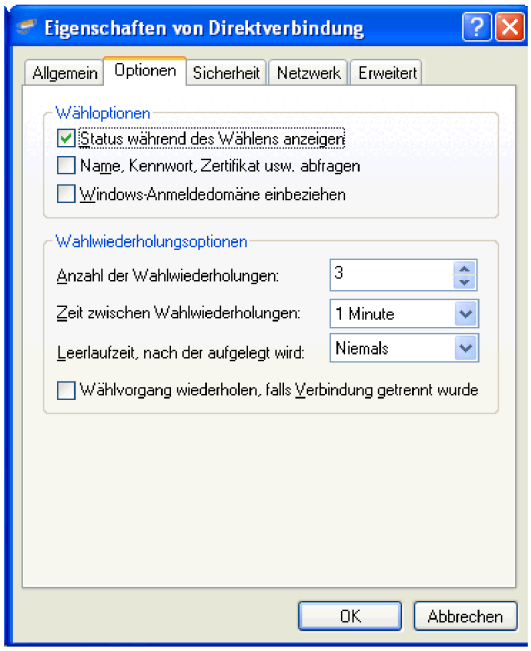

Abbildung C.18: Windows XP: Optionen von Bluetooth-Nullmodem

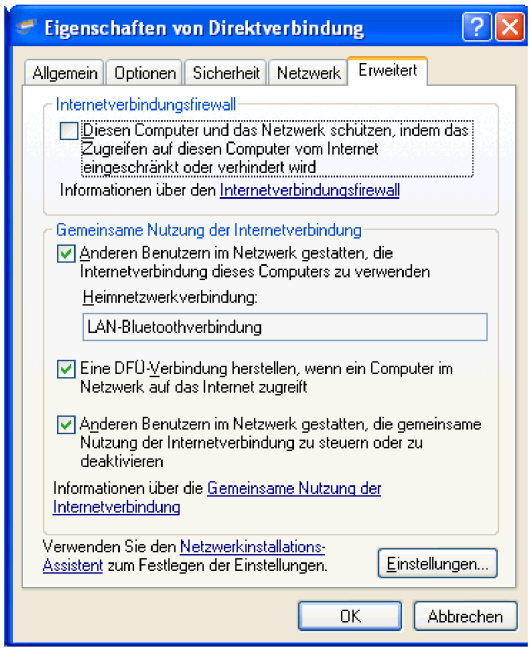

Abbildung C.19: Windows XP: Gemeinsame Nutzung von Bluetooth-Nullmodem

Unter "Gemeinsame Nutzung der Internetverbindung" den Namen der Verbindung für den Bluetooth LAN-Access Server auswählen.

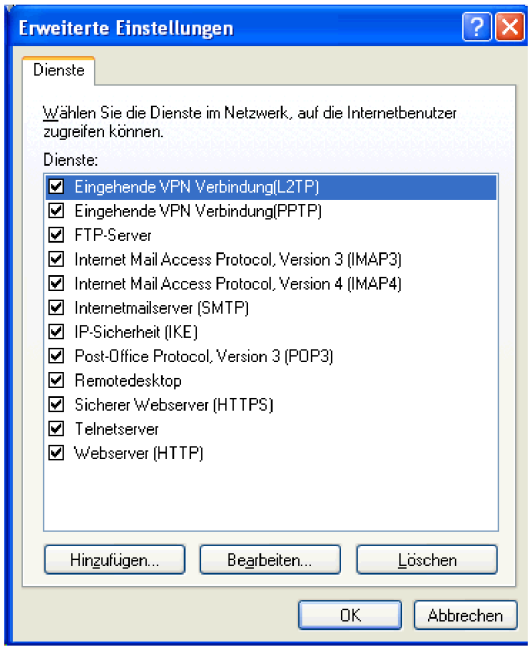

Abbildung C.20: Windows XP: Dienste von Bluetooth-Nullmodem

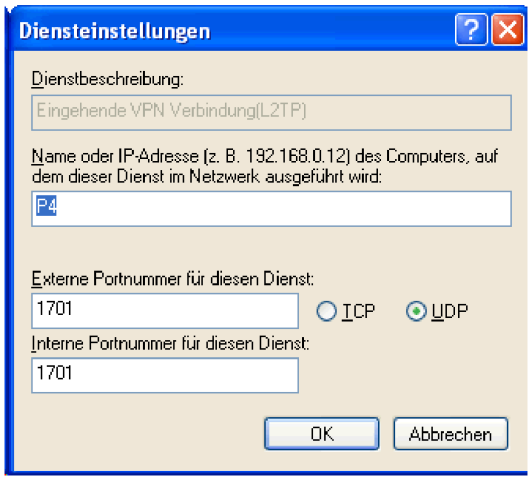

Abbildung C.21: Windows XP: Dienst für Eingehende VPN hinzufügen

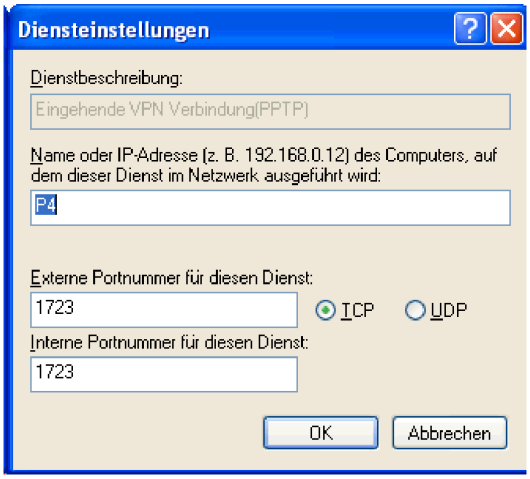

Abbildung C.22: Windows XP: Dienst für Eingehende VPN (PPTP) hinzufügen

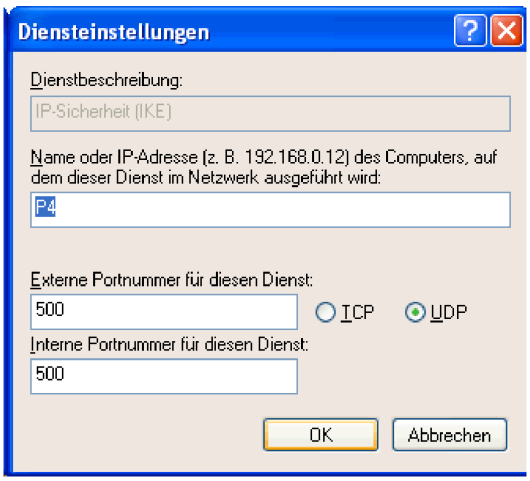

Abbildung C.23: Windows XP: Dienst für IP-Sicherheit hinzufügen

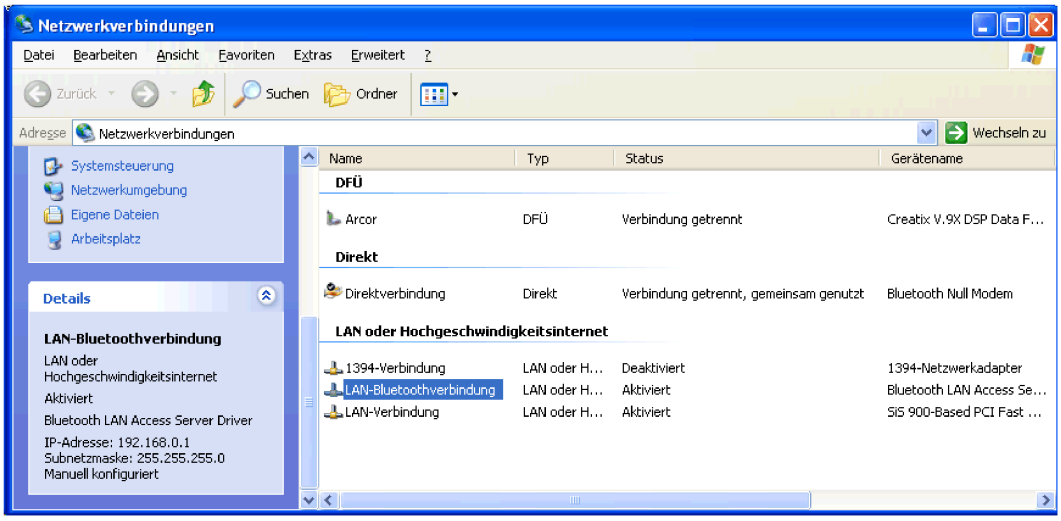

Abbildung C.24: Windows XP: LAN-Verbindung für Bluetooth auswählen

LAN-Verbindung für Bluetooth

Die Einstellungen für das Internetprotokoll kontrollieren. In der Regel sollte die IP-Adresse auf "192.168.0.1" eingestellt werden. Unter dieser Adresse ist dann der LAN-Access für den PDA zu erreichen.

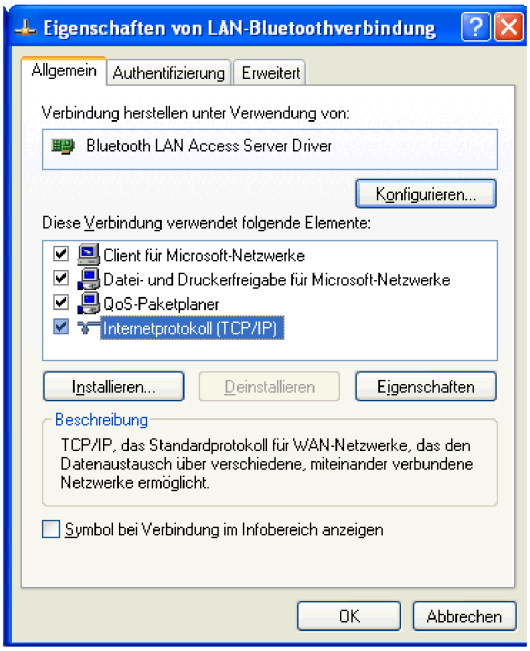

Abbildung C.25: Windows XP: Netzwerkeinstellungen

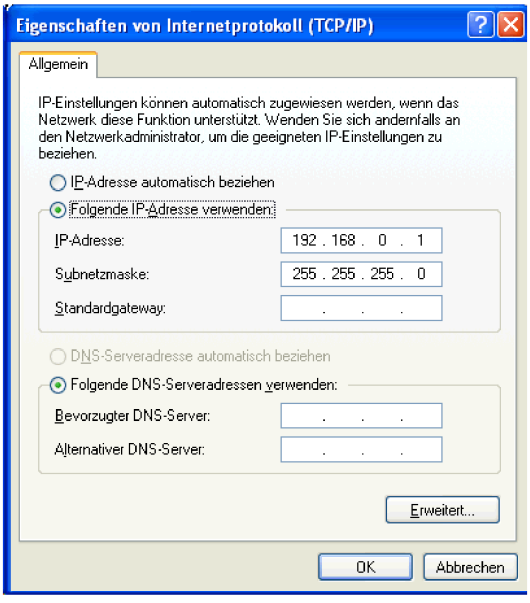

Abbildung C.26: Windows XP: TCP/IP-Einstellungen

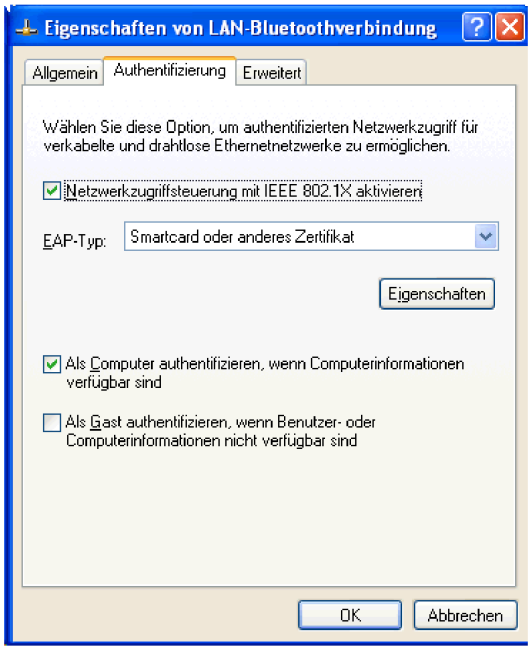

Abbildung C.27: Windows XP: Authentifizierung

Anhang D

Inhalt der CD-ROM

Verzeichnis	**Bemerkung**
Diplomarbeit	Enthält diese Diplomarbeit als Datei *Diplomarbeit.pdf*.
Server-Programm	Ausführbare Datei des Server-Programms und den Quellcode als Visual C++ 6.0-Projekt.
Planning-Programm	Ausführbare Datei des Planning-Programms und den Quellcode als Embedded Visual C++ 3.0-Projekt.
Controller-Programm	Quellcode-Dateien die für den Download des Controller-Programms auf das Controller-Board benötigt werden.
Tools	Enthält die Installationsdateien für Embedded Visual Tools 3.0, Pocket PC 2002 SDK, ActiveSync 3.7 und Windows Mobile Developer Power Toys.
Movies	Enthält einige Videosequenzen in denen sich der Roboter im Parcours bewegt.
Pictures	Enthält diverse Bilder, die im Rahmen dieser Diplomarbeit entstanden sind.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Activmedia 2002] ACTIVMEDIA: *ActivMedia Robotics*. online. 2002. – URL http://www.activrobots.com. – Zugriffsdatum: 02.12.2002

[Arkin 1998] ARKIN, Ronald C.: *Behavior-Based Robotics*. Massachusetts : Bradford Book, 1998. – ISBN 0-262-01165-4

[Balzerowski 2002] BALZEROWSKI, Rainer: *Realisierung eines Webcam basierten Kamera-Systems für mobile Roboter*. online. 2002. – URL http://www.informatik.haw-hamburg.de/~robots/papers.html. – Zugriffsdatum: 11.03.2003

[Braitenberg 1993] BRAITENBERG, Valentin: *Vehikel, Experimente mit kybernetischen Wesen*. Reinbek bei Hamburg : Rowohlt Taschenbuch Verlag GmbH, 1993. – ISBN 1290-ISBN 3 499 19531 3

[Brooks 1985] BROOKS, Rodney A.: *A ROBUST LAYERED CONTROL SYSTEM FOR A MOBILE ROBOT*. online. 1985. – URL http://www.ai.mit.edu/people/brooks/publications.shtml. – Zugriffsdatum: 29.07.2003

[BTAccess 2003] BTACCESS: *BTAccess*. online. 2003. – URL http://www.highpoint.com. – Zugriffsdatum: 01.01.2003

[Electrolux 2003] ELECTROLUX: *Trilobite*. online. 2003. – URL http://www.electrolux.de. – Zugriffsdatum: 16.06.2003

[Gerling 2002] GERLING, Mirco: *Agentenarchitekturen*. online. 2002. – URL http://www.informatik.haw-hamburg.de/~robots/gerling/studien.pdf. – Zugriffsdatum: 16.06.2003

[Goerz u. a. 2000] GOERZ, G. ; ROLLINGER, C.R. ; SCHNEEBERGER, J.: *Handbuch der Künstlichen Intelligenz*. 3. Auflage. München : Oldenbourg Wissenschaftsverlag, 2000. – ISBN 3-486-25049-3

[Gottwald 2003] GOTTWALD, Michael: *CORBA-Schnittstelle*. Persönliches Gespräch. November 2003

[Honda 2003] HONDA: *Honda ASIMO Humanoid Robot*. online. 2003. – URL http://www.honda-p3.com. – Zugriffsdatum: 10.06.2003

[Husqvarna 2003] HUSQVARNA: *Auto Mower*. online. 2003. – URL http://www.automower.com. – Zugriffsdatum: 16.06.2003

[Klemke und Koeckritz 2002] KLEMKE, Gunter ; KOECKRITZ, Oliver: *Robbe-Servo und Sonar-Support*. Persönliches Gespräch. Mai 2002

[Koschnitzke 2003] KOSCHNITZKE, Hartmut: *Ein Framework für ein universelles objektorientiertes Messwerterfassungssystem mit Hilfe von Design Pattern*. 2003

[Labs 2002] LABS, Newton R.: *Interactive C*. online. 2002. – URL http://www.newtonlabs.com/ic. – Zugriffsdatum: 02.11.2002

[LEGO 2002] LEGO: *LEGO MINDSTORMS*. online. 2002. – URL http://mindstorms.lego.com. – Zugriffsdatum: 02.12.2002

[von Luck 2002] LUCK, Kai von: *IKS-project (Integration Kognitiver Systeme)*. online. 2002. – URL http://www.informatik.haw-hamburg.de/~robots/. – Zugriffsdatum: 06.12.2002

[Manger 2003] MANGER, Michael: *Plattform für Roboterfußball*. Persönliches Gespräch. Dezember 2003

[Martin 2002] MARTIN, Fred: *The MIT LEGO Robot Design Competition*. online. 2002. – URL http://www.media.mit.edu/people/fredm/projects/6270. – Zugriffsdatum: 02.12.2002

[Microsoft 2003] MICROSOFT: *Microsoft Foundation Classes*. online. 2003. – URL http://www.microsoft.com. – Zugriffsdatum: 10.03.2003

[MIT 2003] MIT: *MIT´s Autonomous Robot Design Competition*. online. 2003. – URL http://web.mit.edu/6.270. – Zugriffsdatum: 10.03.2003

[Pfeifer und Scheier 1999] PFEIFER, Rolf ; SCHEIER, Christian: *Understanding Intelligence*. Bradford Book, 1999. – ISBN 0262161818

[Porschke 2002] PORSCHKE, Jan P.: *Genetische Programmierung von verhaltensbasierten Agenten*. online. 2002. – URL http://www.informatik.haw-hamburg.de/~robots/papers.html. – Zugriffsdatum: 06.10.2003

[Revout 2003a] REVOUT, Ilia: *Design und Realisierung eines Frameworks für Bildverarbeitung*. online. Dezember 2003. – URL http://www.informatik.haw-hamburg.de/~robots/papers.html. – Zugriffsdatum: 10.12.2003

[Revout 2003b] REVOUT, Ilia: *Konzeption eines Frameworks für Kamerabildanalyse*. online. 2003. – URL http://www.informatik.haw-hamburg.de/~robots/papers.html. – Zugriffsdatum: 01.12.2003

[RoboMow 2003] ROBOMOW: *RoboMow*. online. 2003. – URL http://www.robomow.com. – Zugriffsdatum: 06.10.2003

[Roomba 2003] ROOMBA: *Roomba Vertrieb Deuschland*. online. 2003. – URL http://www.roomba.de. – Zugriffsdatum: 06.10.2003

[Torrone 2003] TORRONE, Phillip: *fallonbot*. online. 2003. – URL http://dev.fallon.com/phillipttorrone/fallonbot/. – Zugriffsdatum: 01.06.2003

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum	Unterschrift des Studenten