

Interaktives Lernen von Weltmodellen für einen Service-Roboter

Steen Kristensen, Volker Hansen, Sven Horstmann, Jesko Klandt, Konstantin Kondak, Frieder Lohnert und Andreas Stopp

DaimlerChrysler AG
Forschung und Technologie
Kognition und Robotik
Alt-Moabit 96A, 10559 Berlin

Der technische und wirtschaftliche Erfolg von Service-Robotern hängt maßgeblich davon ab, wie schnell sich diese Maschinen an neue Umgebungen und Aufgaben anpassen lassen. In diesem Beitrag wird gezeigt, wie ein Service-Roboter (mobiler Roboter mit Manipulator) durch interaktive Belehrung, d.h. durch Herumführen in einer a priori unbekanntem Umgebung sowie durch Zeigen und Benennen relevanter Objekte sein Weltmodell generiert, um komplexe Aufgaben ausführen zu können. Die hierfür angewandten Techniken und insbesondere die von uns entwickelte Systemarchitektur werden näher beschrieben.

1 Einleitung

Zweck von Robotern ist es, Menschen von Arbeitsaufgaben, die entweder zu gefährlich, zu schwierig, zu ermüdend oder einfach zu teuer sind, zu entlasten. Unsere Arbeiten im „Advanced Servicing Robot“-Projekt zielen auf die Entwicklung eines vielseitigen Service-Roboters für ein breites Spektrum von Aufgaben und Einsatzumgebungen.

Unter einem Service-Roboter verstehen wir folgendes (aus [IPA 1994]):

„Ein Serviceroboter ist eine frei programmierbare Bewegungseinrichtung, die teil- oder vollautomatisch Dienstleistungen verrichtet. Dienstleistungen sind dabei Tätigkeiten, die nicht der direkten industriellen Erzeugung von Sachgütern, sondern der Verrichtung von Leistungen an Menschen und Einrichtungen dienen.“

Die Tatsache, dass autonome Service-Roboter in Umgebungen, die unterschiedlicher, unstrukturierter und dynamischer sind als die von z.B. Industriemanipulatoren, eingesetzt werden, bedingt, dass fortschrittliche Methoden der Planung und Sensordatenverarbeitung für diese Systeme unverzichtbar sind. Andererseits bedeutet die Vielfalt der Einsatzumgebungen und -gebiete von Service-Robotern, dass eine gegebene, aufwendig entwickelte Funktionalität nicht generell einsetzbar ist, und deshalb nicht unverändert an viele potentielle Kunden verkauft werden kann. Die Herausforderung beim Erstellen von wirtschaftlich erfolgreichen Service-Robotern besteht deshalb darin, Systeme zu schaffen, die einfach an ihre spezifischen Einsatzbedingungen adaptierbar und dennoch so einheitlich sind, dass es wirtschaftlich und technisch möglich ist, sie zu produzieren und zu warten. Wir vertreten die Auffassung, dass es zum Erreichen dieses Ziels notwendig ist, eine standardisierte, für unterschiedliche Aufgaben einfach konfigurierbare Plattform mit entsprechend standardisierter Software zu entwickeln. Es wurde deshalb entschieden, eine Softwarearchitektur, basierend auf einer Menge von standardisierten aber spezialisierten Perzeptions- und Steuerungsmodulen, einzusetzen. Dabei muss darauf geachtet werden, dass

die von diesen Perzeptions- und Steuerungsmodulen verwendeten Modelle dem Roboter durch den Benutzer beigebracht, also vom Roboter erlernt werden können, und nicht von Experten eingegeben werden müssen.

In diesem Beitrag zeigen wir, wie eine modulare Architektur nicht nur die robuste Ausführung von Aufgaben unterstützen kann, sondern auch direkt für das Lernen der dafür erforderlichen Repräsentationen verwendet werden kann. In Abschnitt 2 werden thematisch verwandte Arbeiten kurz skizziert. Die von uns entwickelte Architektur wird in Abschnitt 4 vorgestellt, während im Abschnitt 3 erklärt wird, wie das System Modellwissen erlernt. In Abschnitt 5 wird anhand eines Beispiels die Funktionsweise des Gesamtsystems erläutert. Abschließend werden in den Abschnitten 6 und 7 die Ergebnisse zusammengefasst und diskutiert.

2 Bezüge zu anderen Arbeiten

Da jedes mobile Robotersystem in irgendeiner Weise eine Architektur hat, gibt es eine Fülle von Arbeiten zu diesem Thema. Allerdings scheinen die Architekturen vieler Systeme eher ad hoc entstanden zu sein, als dass sie mit besonderer Rücksicht auf Modularität, Flexibilität und Wiederverwendbarkeit entwickelt worden wären. Eine der ersten erwähnenswerten Ausnahmen sind Firby's „Reactive Action Packages“(RAPs) [Firby 1989], die einen systematischen Ansatz bieten, Aufgaben (Tasks) und deren Zerlegung in Task-Netze zu beschreiben. Vergleichbar dazu ist Simmons' „Task Control Architecture“ (TCA) [Simmons 1994], die darüber hinaus betriebssystemartige Bestandteile enthält und damit zugleich die Infrastruktur für die Ausführung der Aufgaben bereitstellt. Aus RAP hervorgegangen ist das „Conditional Sequencing“ von Gat [Gat 1994]. Bei dieser Architektur wurde versucht, ihren Gewinn hinsichtlich erhöhter Zuverlässigkeit und verringerter Entwicklungszeit zu quantifizieren.

Alle genannten Architekturen haben gemeinsam, dass sie streng aufgabenorientiert sind. So basieren z.B. die Aufgabenbäume bei RAPs und TCA auf einer bestimmten Aufgabe und müssen von einem Experten dafür erstellt werden. Unser Bestreben geht dahin, dass ein Benutzer in der Lage sein soll, dem Roboter neue Aufgaben beizubringen. Seine Aufgabe soll dabei nicht darin bestehen, ihm beizubringen, wie verschiedene Aufgaben (mit all ihren Ausnahmen) gelöst werden, vielmehr soll er der Maschine zeigen, wo und mit welchen Objekten oder Werkzeugen eine generische Aufgabe auszuführen ist. Dazu gehört z.B. das Zeigen einer neuen Umgebung mit Objekten, an die angedockt, oder die manipuliert werden sollen. Deshalb brauchen wir eine Architektur, in der der Aufgabenkontext die Systemstruktur bestimmt, z.B. durch automatische Generierung geeigneter Task-Netze.

Dies hat entscheidende Bedeutung für die zugrundeliegende Systemarchitektur und die zu verwendende Modellierung der Umwelt.

3 Lernen von Umweltmodellen

Abgesehen von bewusst einfach konstruierten, reaktiv arbeitenden Robotern, verfügen fast alle über irgendeine Form der Umgebungsbeschreibung. Verbreitet ist eine sub-symbolische Modellierung des Freiraums bzw. der Hindernisse der Umgebung in einem sogenannten Evidenzgitter (Evidence-Grid) [Elfes 1987] (s. Abbildung 4), die für Kollisionsvermeidung und/oder geometrische Planung genutzt werden kann. Die Repräsentation, die zu lernen wir den Roboter befähigen wollen, sollte die folgenden Anforderungen erfüllen:

- Sie sollte über längere Zeitspannen brauchbar sein, d.h. sie sollte nur dann ungültig werden, wenn sich die Umgebung "entscheidend" verändert, damit der Roboter nur dann eine neue Repräsentation erlernen muss. Außerdem sollte es möglich sein, diese (automatisch oder manuell) zu modifizieren, um die Leistung des Systems zu verbessern.

- Sie sollte die Spezifikation und Planung von Aufgaben unterstützen, die der Benutzer dem System aufträgt. Diese könnten z.B. lauten: "Bringe die Tasse in die Küche". Daraus folgt, dass die Repräsentation symbolische Anteile enthalten muss.
- Sie sollte die für die Ausführung der geplanten Missionen notwendige Information liefern, d.h., dass Information über Landmarken und Objekte enthalten sein muss.

Eine Repräsentation, die diese Bedingungen erfüllt, ist ein annotierter topologischer Graph. In dieser Graphendarstellung stellen Knoten bestimmte Orte und Kanten existierende befahrbare Verbindungen zwischen diesen Orten dar. Diese rein topologische Darstellung ist angereichert mit metrischer Information über die absolute Lage der Orte. Außerdem werden, wie oben bereits erwähnt, auch Landmarken (Türen, Wände) und Objekte (z.B. Tische) eingetragen. Auf diese Weise sind alle Punkte, an die der Roboter zur Erfüllung einer ihm gestellten Aufgabe fahren muss, entweder direkt gespeichert, oder können über Modellwissen aus sensorisch erkannten Objektpositionen abgeleitet werden. Wesentlich ist, dass die abgespeicherten Orte und Objektpositionen nicht genau sein müssen, da der Graph nur als Grundlage der groben Planung der Bewegung des Roboters dient. Auch ermöglicht das Graphenmodell z.B. das Überspringen von Knoten, falls diese nicht erreicht werden können, dies jedoch für die Erfüllung der Aufgabe nicht erforderlich ist und ein direktes Anfahren des nächsten Knotens möglich ist. Ausserdem kann, durch vorausschauendes Prüfen des global geplanten Weges, frühzeitig nach unerwarteten statischen Hindernissen, wie z.B. geschlossenen Türen, gesucht werden.

Topologische Karten sind aus der Kognitionswissenschaft [Piaget and Inhelder, 1967] bekannt, sind aber auch in der mobilen Robotik weit verbreitet [Kuipers & Buyn 1988][Kuipers & Buyn 1991][Koenig & Simmons 1998][Thrun et al. 1998][Gutmann & Nebel 1997][Jensfelt & Kristensen 1999]. Die meisten der hier referenzierten Untersuchungen zeigen darüber hinaus Möglichkeiten auf, topologische Graphen durch Explorationsstrategien automatisch zu erlernen; wir jedoch halten es für angebracht, dem Roboter die Umgebung in Interaktion mit dem Menschen beizubringen, da die autonome Exploration folgende Probleme aufwirft:

- Es kann nicht sichergestellt werden, dass der Roboter während der Exploration alle relevanten Umgebungsdetails erfassen kann; diese können z.B. durch Hindernisse, geschlossene Türen und umherlaufende Personen verdeckt werden.
- Es kann Regionen geben, in die der Roboter nicht eindringen darf, z.B. in bestimmte Bereiche von Industrieanlagen oder Büros.
- Die entstehenden topologischen Graphen sind "bezugslos", d. h. sie stellen für den menschlichen Benutzer willkürlich gewählte Punkte ohne symbolische Namen wie z.B. "Küche" oder "Druckerraum" dar. Natürlich könnte diese Information später hinzugefügt werden, jedoch ist dann nicht sichergestellt, dass die vom System vergebenen Knoten wirklich mit für den Benutzer aussagekräftigen Orten verbunden sind.

Obwohl auch mit unserer Systemarchitektur autonome Exploration möglich ist, haben wir uns aus den oben genannten Gründen für eine interaktive Form des Lernens der Umgebung entschieden. Das heißt, wir geben dem Benutzer die Möglichkeit, den Roboter durch die Umgebung zu dirigieren und ihm wichtige Orte oder Objekte zu zeigen und diese zu benennen. Das heißt nicht, dass der Benutzer dem Roboter Landmarken zeigen oder metrische Beschreibungen der Objekte eingeben muss. Der Erwerb dieses Wissens wird im Abschnitt 5 beschrieben. Obwohl ein Erweitern des Weltmodells jederzeit möglich ist (auch während des normalen Betriebs), haben wir für das Lernen einen Modus vorgesehen, der explizit aktiviert werden muss. Dies hängt damit zusammen, dass es z.B. in Fabrikumgebungen nicht zulässig ist, dass sich das Weltmodell des Roboters, und damit sein Verhalten, selbständig oder unbeabsichtigt ändert.

4 Eine Architektur zur Ausführung von Aufgaben unter Benutzung erlernter Weltmodelle

Wie bereits erwähnt, stellt die Tatsache, dass der Benutzer den Roboter dadurch an neue Aufgaben und Umgebungen anpasst, in dem er ihn herumführt, ihm Orte und Objekte zeigt und benennt, und der Roboter damit interaktiv Repräsentationen seiner Umgebung lernt, mehrere Anforderungen an die zugrunde liegende Systemarchitektur. Die wichtigste dieser Anforderungen ist die, dass das System in der Lage sein muss, sich selbst so zu konfigurieren, dass Aufgaben in einer a priori unbekanntem Umgebung, die nur durch die im Weltmodell der Maschine enthaltene Information beschrieben ist, gelöst werden können. Wir nennen diese Art der automatischen Konfiguration *kontextadaptiven Betrieb*.

Da die Zahl der zu erwartenden Arten von Umgebungen, und damit die der benötigten Sensorik- und Planungsmodule, nicht beschränkt ist, müssen wir in der Lage sein, den Roboter durch Hinzufügen solcher Module auf einfache Weise anzupassen. Auch dieses stellt Anforderungen an die Systemarchitektur.

4.1 Systemüberblick

Aus Sicht des Systementwurfs liegt das Hauptproblem eines kontextadaptiven Systems darin, es strukturiert und verständlich zu halten, obwohl die genaue Konfiguration zu einem bestimmten Zeitpunkt nicht im voraus festgelegt wird. Deshalb wurde entschieden, das System funktional in *Kompetenzen* zu partitionieren, von denen zur Zeit vier implementiert sind: die Navigationskompetenz NAV, die Manipulationskompetenz MAN, die Relokalisierungskompetenz REL und die Modellierungs-/Lernkompetenz MOD.

Die Aktivierung dieser Kompetenzen wird von einem Missionsplaner und einem Missionsausführer koordiniert. Ein funktionales Blockbild des Systems ist in Abbildung 1 gezeigt.

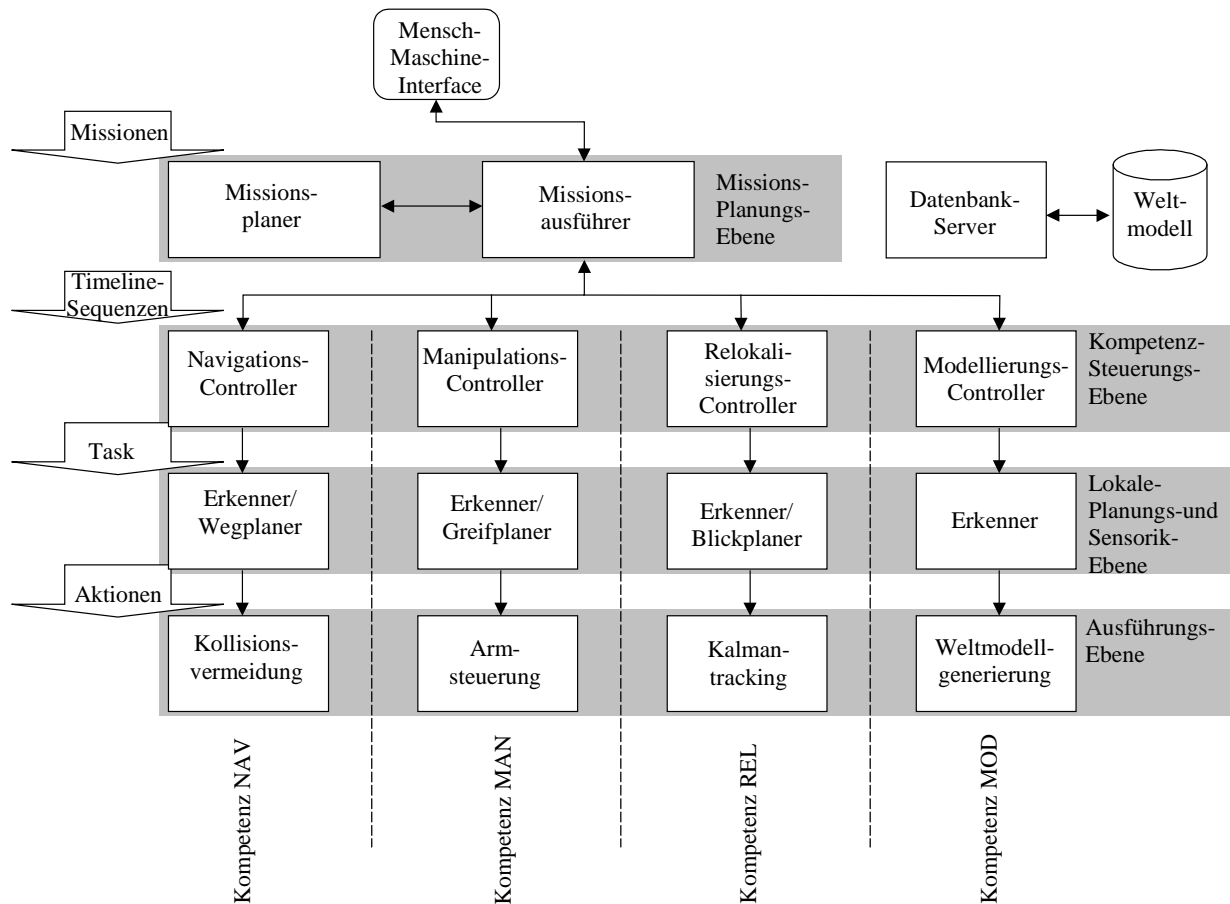


Abbildung 1: Funktionaler Überblick über die ASR-Systemarchitektur. Zu beachten ist, dass ein Block eine einzelne, abstrakte Funktionalität, und nicht einen Prozess (ein Programm) beschreibt. Das getrennt gezeichnete Weltmodell steht allen Kompetenzen und Ebenen zur Verfügung, wird aber hauptsächlich von der Kompetenz-Steuerungs- und der Planungsebene benutzt. Links sind die Typen der Informationen angegeben, die zwischen den Ebenen ausgetauscht werden.

Wie in diesem Bild gezeigt, gibt es vier Ebenen:

1. Missions-Planungs-Ebene

Auf dieser Ebene werden die Missionen symbolisch mittels eines STRIPS-artigen Planers geplant. Eine Mission, die vom Benutzer mittels des Mensch-Maschine-Interface (MMI) ausgewählt wird, enthält den gewünschten Zielzustand und evtl. einige Zwischenzustände, während der momentane Zustand des Systems den Ausgangszustand des Plans darstellt. Zustandsübergänge werden durch die Ausführung von Tasks realisiert. Die zur Verfügung stehenden Tasks, bzw. Zustandsübergänge, sind entweder direkt in der Datenbank des Systems gespeichert (z.B. spezielle Manipulations-Tasks) oder werden aus dem topologischen Graphen abgeleitet (Navigations-Tasks, generische Armbewegungen). Dadurch steht neu hinzugelernete Information sofort dem Planer zur Verfügung. Planungsergebnis ist eine Missions-Timeline, die die Abfolge der auszuführenden Tasks durch die verschiedenen Kompetenzen beschreibt, so dass die gegebene Mission erfüllt wird. Der Missionsausführer gruppiert die direkt aufeinanderfolgenden Tasks gleicher Kompetenzen zu Paketen (Timeline-Sequenzen). Diese Pakete werden dann sequentiell an die Kompetenzen übergeben und von diesen ausgeführt. Falls ein Fehler oder ein

unerwartetes Ereignis auftritt, wird der Missionsplaner beauftragt, eine Neuplanung vorzunehmen, ohne dabei den Task zu benutzen, der den Fehler ausgelöst hat. Das Weltmodell ist, aufgrund seiner engen Anbindung an den Planer, auf der Missions-Planungs-Ebene dargestellt, es ist jedoch auch von den anderen Ebenen aus erreichbar (z.B. zum Abruf metrischer Information).

2. Kompetenz-Steuerungsebene

Diese Ebene stellt die Schnittstelle zur symbolischen Missions-Planungs-Ebene dar und steuert die internen Abläufe in den jeweiligen Kompetenzen. Hier wird die rein symbolische Information aus der Missionsplanung mit metrischer Information aus dem Weltmodell angereichert, wobei Tasks entstehen, die von den unteren Ebenen ausgeführt werden können. Ein Task T ist definiert als diejenige Funktionalität, die durch eine Kompetenz ausgeführt werden kann; daher stellt die Menge aller Tasks den Befehlssatz des Systems auf der Timeline-Ebene dar. Zu beachten ist, dass eine Kompetenz in der Lage sein kann, mehrere Tasks auszuführen. Ein typischer Task für die NAV-Kompetenz wäre z.B. "fahre zum Punkt $\langle X,Y \rangle$ " oder "Docke an das Objekt $\langle A \rangle$ an". Eine Liste der zur Zeit implementierten Tasks ist in Tabelle 1 zu finden.

Kompetenz	Task
NAV	Fahre zum Punkt X,Y
NAV	Docke am Objekt A
MAN	Nimm Objekt A von Objekt B
MAN	Lege Objekt A auf Objekt B
MAN	Bewege den Greifer zu Position X,Y,Z mit Orientierung Φ_x, Φ_y, Φ_z
REL	Starte Relokalisierung
REL	Stoppe Relokalisierung
REL	Relokalisier einmalig
MOD	Starte Lernphase
MOD	Beende Lernphase
MOD	Optimiere Weltmodell

Tabelle 1: Gegenwärtig zur Verfügung stehende Tasks

3. Lokale-Planungs-und-Sensorik-Ebene (LOPAS)

Auf dieser Ebene wird komplexere Sensordatenverarbeitung und sub-symbolische Planung durchgeführt. Für einen Docking-Task besteht z.B. die Sensordatenverarbeitung in der Erkennung und Lokalisierung des im *Task* spezifizierten Dockingobjekts und die Planung im Erstellen eines Satzes von Wegpunkten, um das Dockingobjekt mittels einer kontinuierlichen Bewegung in der richtigen Orientierung erreichen zu können. Die Ausgaben der LOPAS-Ebene sind sub-symbolische Kommandos, sog. *Aktionen*. Erläuterungen und Referenzen zu einem der Erkennungsverfahren sind im Abschnitt 5.2 zu finden.

4. Ausführungsebene

Auf dieser Ebene werden die Aktionen, typischerweise von echtzeitfähigen, reaktiven Modulen, die direkt die Hardware steuern, ausgeführt. Dazu gehört z.B. die Kollisionsvermeidung, die auf Basis eines aus den aktuellen Sensordaten aufgebauten Evidence-Grids (also einer dynamischen, lokalen Karte, Abbildung 4) Kollisionen mit dynamischen Hindernissen vermeidet, die von der (langsameren) übergeordneten Ebene im lokal vorgeplanten Weg noch nicht berücksichtigt wurden. Hier wird der „Dynamic

Window Approach“ [Fox et al 1997] eingesetzt, bei dem aus einem Aktionsraum (Linear- und Rotationsgeschwindigkeit des Roboters) anhand mehrerer lokaler Kriterien zyklisch eine geeignete Aktion selektiert und ausgeführt wird.

Eine genauere Beschreibung der Erkennungs- und Sensordatenverarbeitungsalgorithmen würde den Rahmen dieses Artikels sprengen.

Besonders beachtenswert an der Systemstruktur ist ihre Uniformität. Dies mag auf den ersten Blick künstlich und unzweckmässig erscheinen, z.B. bei der REL-Kompetenz, die ja eigentlich keine Aktuatoren hat. Trotzdem kann man ihr eine Ausführungs-Ebene in Gestalt des Kalman-Filters, der die Resultate der Sensor-Module fusioniert und kontinuierlich in Positionskorrekturen für die Plattform umwandelt, zuordnen. Es hat sich herausgestellt, dass das Festhalten an dieser strengen funktionalen Struktur die Verständlichkeit des Designs erheblich verbessert, und nicht zuletzt die Anzahl der von mehreren Kompetenzen gemeinsam genutzten Module erhöht hat.

Aus Sicht der Systemstruktur ist offensichtlich, dass Verbindungen zwischen den Kompetenzen nicht nur - wie in der funktionalen Sicht in Abbildung 1 - auf der Timeline-Ebene, sondern auch auf darunter liegenden Ebenen bestehen. Zum Beispiel müssen die Kompetenzen NAV und REL Informationen über die Position des Roboters austauschen. Der offensichtlichste Unterschied zwischen funktionaler und struktureller Sicht liegt in der LOPAS-Ebene, in der es einen Pool von Sensormodulen, genannt *Erkenner*, gibt, die dynamisch aktiviert und von den Kompetenzen gemeinsam benutzt werden. Unsere generische Prozessstruktur für die LOPAS-Ebene ist in Abbildung 2 gezeigt. Für jeden Kontext-Typ C gibt es eine Instanz dieser Prozessstruktur. Einen Kontext-Typ definieren wir als ein Tupel $C=(E,M,T)$, wobei E ein Umgebungstyp, M die zur Verfügung stehende Modellinformation und T der Typ des auszuführenden Tasks ist. In unserem System gibt es für jeden Task-Typ T einen lokalen Planer, während der Satz von verwendeten Erkennern auch von M abhängt. Wie dieser Satz von Erkennern instanziiert wird, wird in Abschnitt 4.2 beschrieben. Zwei Beispiele für Konfigurationen der LOPAS-Ebene der NAV-Kompetenz sind in Abbildung 3 zu sehen.

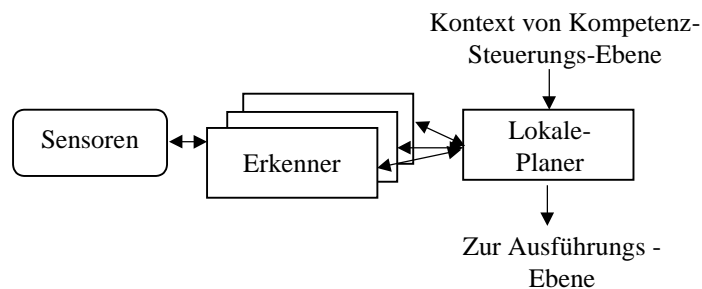


Abbildung 2: Generische Prozessstruktur auf der LOPAS-Ebene. Mehrere Erkennen können in einer parallelen oder seriellen Anordnung arbeiten.

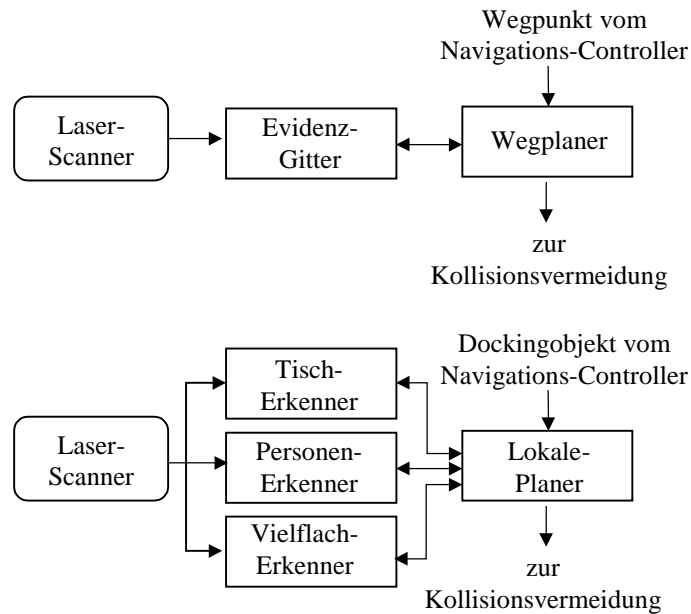


Abbildung 3: Zwei Ausprägungen der generischen Prozessstruktur (oben: freie Navigation, unten: Docking). Es arbeiten jeweils nur die Erkener, die zum gesuchten Objekt tatsächlich ein Modell haben.

Durch Verwendung der generischen Prozessstruktur haben wir erreicht, dass auch die LOPAS-Ebene, obwohl kontextspezifisch, trotzdem einheitlich, erweiterbar sowie einfach verständlich und wartbar ist.

4.2 Kontextadaptiver Betrieb

Unter kontextadaptivem Betrieb verstehen wir, wie bereits erwähnt, die Tatsache, dass das System seine Struktur im Betrieb in Abhängigkeit des aktuellen Kontextes C ändert.

Eine Voraussetzung für die kontextadaptive Nutzung unterschiedlicher Module ist, dass diese Module standardisierte Schnittstellen haben, so dass es keine Rolle spielt, welches Erkennungsmodul benutzt wird. Solche standardisierten Schnittstellen haben wir definiert.

Diese Schnittstellen sind von Hendersons *logical sensor systems* [Henderson & Shilcrat 1984] inspiriert, bei denen jede sensordatenverarbeitende Einheit als *logischer Sensor* angesehen wird, von dem andere Prozesse ihre Eingaben erhalten können. In dieser Hinsicht sind alle Erkener logische Sensoren und produzieren einen standardisierten *Ausgangsvektor*, bestehend aus einem Kopf und einem Datenbereich. Die Struktur des Kopfes ist für alle Erkener identisch und sagt etwas über den Typ, die Herkunft und den Status der Daten aus, während der Datenbereich sensorspezifisch ist. Die Idee dahinter ist, dass jeder Klient (dies kann auch ein weiterer Erkener sein) den Kopf auswerten kann, um festzustellen, ob die im Datenbereich enthaltenen Daten für ihn relevant sind, um sie entsprechend zu verarbeiten.

Im Prinzip kann jeder Erkener Daten- und Modell-getrieben arbeiten. Wenn der Erkener mit einem Satz von Modellen aktiviert wird, wird er versuchen, diese mit extrahierten Objekten abzugleichen; bekommt er jedoch keine Modelle, wird er einfach nur Objekte extrahieren und Hypothesen für die Objekte liefern, für deren Erkennung er programmiert wurde. Ausserdem kann jeder Erkener sowohl ein Server, als auch ein Klient sein, so dass er sich bei einem anderen Erkener anmelden kann, um Sensor- oder Objektdaten zu bekommen. Diese Anmeldung wird vom System automatisch mittels einer einfachen Konfigurationsdatei, in der vermerkt ist, welche Erkener welche Art von Objekten erkennen können, vorgenommen.

Dadurch meldet sich ein Klient für eine bestimmte Art von Objekt (bei allen Erkennern, die dieses Objekt liefern können) an, und nicht bei bestimmten, ihm namentlich bekannten Erkennern. Dies hat den Vorteil, dass das Hinzufügen oder Entfernen von Prozessen im System für die Klienten völlig transparent ist, und außerdem erleichtert es die Implementierung redundanter Prozesse, und erhöht somit die Fehlertoleranz.

Der vereinfachte Algorithmus eines Erkenners sieht wie folgt aus:

```
Warten auf Aktivierung
Anmeldung/Aktivierung relevanter Sub-Erkennenner
solange aktiviert
    warte auf Eingangsdaten
    extrahiere und schicke Objekte an Klienten
Abmeldung/Deaktivierung der Sub-Erkennenner
```

Aus diesem simplen Algorithmus kann man ersehen, dass sich die Aktivierung der Erkennenner im System baumartig ausbreitet, wobei die Sensor-Server die äußersten Äste und die Klienten die Wurzel des Aktivierungsbaums darstellen. Auf diese Weise wird die Entscheidung, welche Erkennenner wann zu aktivieren sind, d.h. die Struktur des Systems, implizit durch den auszuführenden Task T , und die Art der zur Verfügung stehenden, und deshalb zu suchenden Modelle M , bestimmt.

Nachdem die Systemarchitektur beschrieben wurde, mit der die gelernte Weltmodellinformation verarbeitet wird, folgt im nächsten Abschnitt eine Beschreibung des Belehrungsvorganges selbst.

5 Interaktives Belehren

Grundlage des Konzepts, dem Roboter durch Belehrung Weltmodellinformation zu vermitteln, ist, dass dies interaktiv (mit der in Abschnitt 3 gegebenen Begründung) mit der Maschine am Einsatzort stattfinden soll. Letzteres hat den Vorteil, dass der Benutzer sofort Rückmeldung darüber erhält, ob das System ausführen kann, was er von ihm erwartet. Ausserdem ist sichergestellt, dass die Beschreibung der Umwelt genau zum System passt, das heißt, dass z.B. jede Landmarke, die ins Weltmodell eingetragen wird, später von mindestens einem Erkennenner erkannt werden kann, nämlich wenigstens von dem, über den sie beim Erlernen ins Weltmodell kam.

5.1 Das Weltmodell

In Abschnitt 3 wurde beschrieben, dass ein annotierter topologischer Graph eine brauchbare Modell-Repräsentation darstellt, da er die Forderung nach Langzeitgültigkeit erfüllt, und ausserdem die Planung und Ausführung von Missionen unterstützt. Im folgenden werden die von uns gewählten Komponenten unseres topologischen Graphen erläutert:

Knoten

Die Knoten beschreiben Orte in der Umgebung. Da die Knoten sowohl für die Missionsplanung als auch für die Navigation benutzt werden, können diese Orte Räume, Positionen von Objekten sowie für die Navigation wichtige Punkte wie Türen oder Kreuzungen von Gängen sein. Die Knoten tragen als Beschreibung eine 2D-Position sowie eine Kovarianzmatrix, um die Unsicherheit ihrer Position auszudrücken.

Kanten

Kanten stellen befahrbare Verbindungen zwischen Knoten dar, d.h. existiert eine Kante zwischen zwei Knoten, kann in der Missionsplanung davon ausgegangen werden, dass der Roboter von einem zum anderen Knoten gelangen kann.

Features

Wir definieren Features als Eigenschaften der Umgebung, die für die Navigation genutzt werden können, da sie *on the fly*, also im Vorbeifahren, von den entsprechenden Erkennern erkannt werden können. Das sind z.B. Türen und Wände. Features sind, genau wie den Knoten, Position und Positionsunsicherheit zugeordnet.

3D-Objekte

Hierbei handelt es sich um Objekte, bei denen die dreidimensionale Gestalt wesentlich ist. In unserem System sind das Objekte an die angedockt, oder die gegriffen werden sollen. 3D-Objekte unterscheiden sich von Features dadurch, dass sie in einem dedizierten Erkennungsvorgang erkannt werden müssen, da die Aufnahme von 3D-Daten mit den verwendeten Sensoren nicht *on the fly* möglich ist.

Verbindungen (Links)

Diese Links verbinden Features und 3D-Objekte mit Knoten. Dadurch wird zum einen eine Sichtbarkeit ausgedrückt, zum anderen wird eine räumliche Zuordnung hergestellt.

Die Modellierung der Sichtbarkeit dient dem effizienten Einsatz der Erkennung für die Landmarkenerkennung. Nur wenn der Roboter sich auf Kanten bewegt, die an Knoten grenzen, von denen ein Feature sichtbar ist, wird auch der entsprechende Erkennung aktiviert.

Ein Beispiel für ein Weltmodell ist in Abbildung 7 gezeigt. Die Knoten sind als kleine Kreise, die Kanten als Linien zwischen den Knoten dargestellt. Features sind z.B. Türen (Schwarze Balken) und Wände (Linien). 3D-Objekte sind als Linienumrisse dargestellt. Die Links sind in dieser Darstellung nicht eingezeichnet.

5.2 Belehrung über die Modellierungs-Kompetenz

Aufgabe der MOD-Kompetenz ist, dem Benutzer das einfache Belehren des Roboters zu ermöglichen, um ihm die oben beschriebenen Weltmodelle beizubringen. Die Grundlage der Weltbeschreibung, der topologische Graph, wird dem Roboter durch einfaches Herumführen in der Umgebung beigebracht. Dazu werden dem Roboter in einem Evidenzgitter (Abbildung 4), das die lokale Umgebung des Roboters in einer Draufsicht zeigt und aus Daten seiner Laserscanner gewonnen wird, Zielpunkte vorgegeben, die er dann autonom ansteuert. Der Vorteil dieser Zielpunktvorgabe gegenüber einer Handsteuerung der Maschine (z.B. mittels Joystick) besteht darin, dass das Ansteuern der Zielpunkte mit den gleichen Verfahren erfolgt, die später in der Missionausführung benutzt werden. Tatsächlich wird die Zielvorgabe in Kommandos an die NAV-Kompetenz umgesetzt, die keine Information darüber hat, ob gerade eine Belehrung stattfindet oder eine Mission ausgeführt wird. Auf diese Weise ist sichergestellt, dass der Roboter den gelernten Graphen später auch für die autonome Navigation benutzen kann. Die vorgegebenen Punkte werden automatisch als Knoten in den Graphen eingetragen und durch Kanten verbunden, nachdem der Roboter sie erreicht hat. Um die dabei auftretenden Positionsfehler, die insbesondere bei geschlossenen Wegen zu Inkonsistenzen führen können, zu eliminieren, benutzen wir eine kontinuierliche Relokalisierung mittels Laserscandaten, sowie ein Optimierungsverfahren [Lu & Miliotis 1997], das unter Berücksichtigung der zwischen den Weltmodellknoten entstandenen relativen Positionsunsicherheiten den Fehler, der während des Teachens eines geschlossenen Weges auftritt, auf die Knotenpositionen verteilt, und dadurch den entstandenen Graphen entzerrt. Das Schließen des Graphen kann manuell oder, unter günstigen Bedingungen (kleiner Offset), automatisch durch Wiedererkennung der gelernten Features erfolgen.



Abbildung 4: Evidenz-Gitter, das zur Zielpunktvorgabe benutzt wird. Weiß zeigt Hindernisse an, schwarz bedeutet Freiraum, grau sind Bereiche, die mit den Sensoren (noch) nicht erfasst wurden.

Die Features werden automatisch gelernt, während der Roboter herumgeführt wird. Dies wird von der MOD-Kompetenz dadurch erreicht, dass alle Erkenner für Features ohne Modellvorgabe gestartet werden. Die von ihnen gelieferten Features werden dann, nachdem sie mindestens n-mal gesehen wurden, durch den MOD-Controller ins Weltmodell integriert. Zur Zeit lassen wir den Benutzer die erkannten Türen interaktiv im Display bestätigen, da auch Fehl-Erkennungen von Tür-artigen Strukturen wie z.B. Fenstern auftreten können. Die Links zu Features werden automatisch vom MOD-Controller erzeugt, der protokolliert, zwischen welchen Knoten die Features von den Erkennern gesehen wurden.

3D-Objekte werden ebenfalls interaktiv vom Benutzer gezeigt und benannt. Objekte, deren Beschreibungen dem System bereits bekannt sind, werden dem Roboter einfach durch Anklicken im Evidenzgitter gezeigt, und dann durch die entsprechenden Erkenner verifiziert. Danach wird ihre Position mit der entsprechenden Beschreibung im Weltmodell eingetragen.

Für Objekte, die dem System noch unbekannt sind, kann eine Beschreibung ebenfalls interaktiv erzeugt werden. Je nach Eigenschaften des Objekts kommen dazu verschiedene Verfahren zum Einsatz. Beispielsweise wird für durch Kombinationen planarer Flächen beschreibbare Objekte (Vielfläche) zunächst ein 3D-Bild der Szene, in der sich das unbekannte Objekt befindet, aufgenommen, und in planare Flächen segmentiert [Jiang & Bunke 1994]. Das segmentierte Bild wird dem Benutzer in einer Teachbox präsentiert (s. Abbildung 5). Hier kann der Benutzer durch Anklicken die zum Objekt gehörenden Flächen markieren, die Flächenkombination benennen und mögliche Docking-Positionen oder Greifpunkte definieren. Das Ergebnis wird dann im Weltmodell als Objekt-Prototyp abgespeichert, und kann bei Bedarf referenziert werden, wenn an anderer Stelle ein Objekt von diesem Typ auftritt.

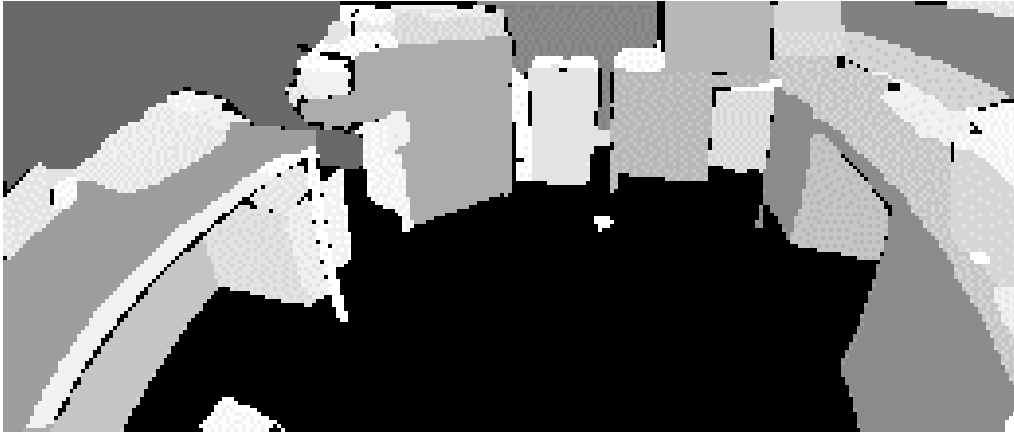


Abbildung 5: In planare Flächen segmentiertes Entfernungsbild

Die hier beschriebenen interaktiven Belehrungsmethoden haben sich für das schnelle und zuverlässige Erzeugen von Weltmodellen als geeignet erwiesen. Ein Beispiel für ein so erzeugtes Modell wird im nächsten Abschnitt beschrieben.

6 Ergebnisse mit dem Roboter „Clever“

In diesem Abschnitt werden wir ein Beispiel für ein dem Roboter „Clever“ (s. Abbildung 6) durch Herumführen in einer bisher unbekanntem Büroräumung beigebrachtes Weltmodell zeigen.



Abbildung 6: Die Experimentalplattform „Clever“. Der Roboter besteht aus einer mobilen Plattform, einem Arm mit 7 Freiheitsgraden, 2 Pentium-PC, 2 Sick-Laserscannern, 2 Farbkameras und Funk-Ethernet Anbindung.

Zur Erstellung dieses Modells wurde der Roboter von einem Startpunkt in unserem Labor in den Flur, der ein Rechteck bildet, gefahren. Im Labor wurden dem Roboter zwei 3D-Objekte gezeigt. Während des Lernens hat der Roboter ausserdem autonom Linien und Türen in der Umgebung erkannt und in das Weltmodell eingetragen. Die Zeit für die Aufnahme des Weltmodells betrug jeweils etwa 5 Minuten für die Fahrt über den Korridor und das Zeigen der 3D-Objekte (Tisch und Kaffeemaschine), wobei der Roboter die entsprechenden Modelle bereits vorher kannte, und nur deren Positionen ermitteln musste. Die Weglänge des Modells beträgt etwa 72 Meter. Die Erfahrung zeigt, dass mit auf diese Weise gewonnenen Weltmodellen in der Regel sofort gearbeitet werden kann, da die modellierten Verbindungen und Knoten bereits im Lernprozess für die Navigation des Roboters benutzt werden.

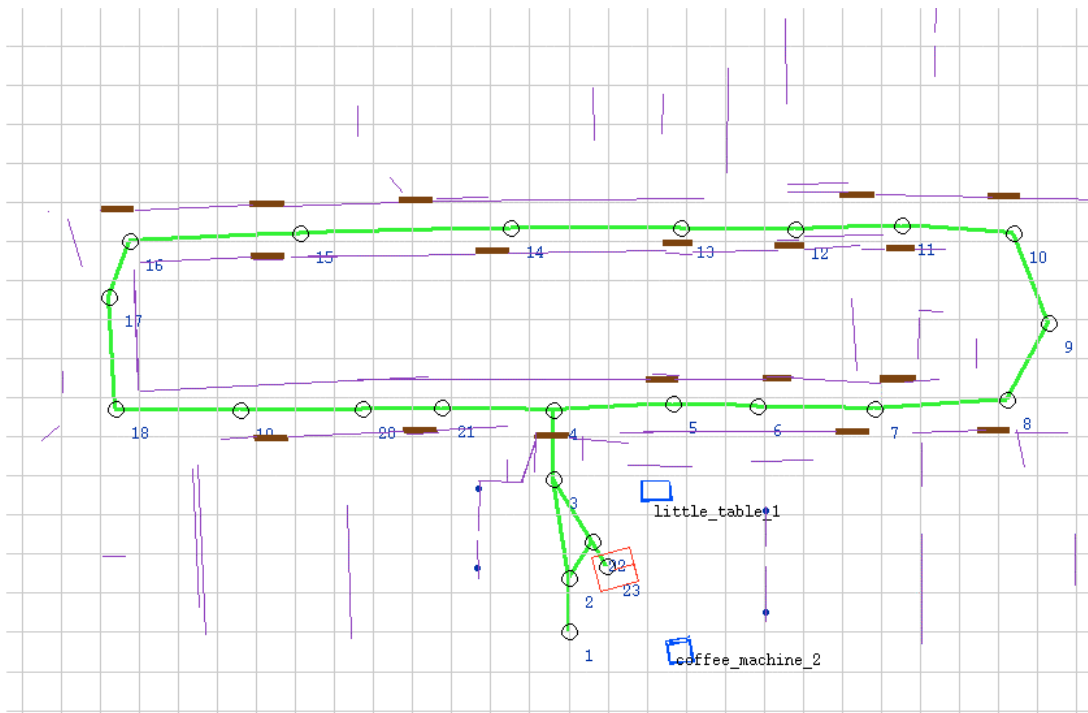


Abbildung 7: Durch Herumführen des Roboters erzeugtes Weltmodell einer ihm bisher unbekanntem Büroumgebung. Die Rastergröße beträgt 1m.

7 Diskussion und Zusammenfassung

In diesem Artikel haben wir die bei der DaimlerChrysler Forschung im Rahmen des Projektes „Advanced Servicing Robot“ entwickelten interaktiven Verfahren zur Gewinnung von Weltmodellinformation für mobile Roboter beschrieben. Ein wichtiges Kriterium beim Design von Service-Robotern ist, dass diese extrem flexibel sein müssen, um kommerziell attraktiv zu sein, und gleichzeitig sehr spezialisiert, um ausreichend sicher und robust arbeiten zu können. Wir haben behauptet, dass ein wesentlicher Schritt auf dem Weg zu diesem Ziel darin liegt, dem Roboter zu ermöglichen, die Repräsentationen zu lernen, die für die Erfüllung seiner Aufgaben nötig sind. Die Tatsache, dass nur die Art der Repräsentation, nicht aber die konkrete Umgebung und ihr Modell a priori bekannt sind, stellt mehrere Anforderungen an die Systemarchitektur in Hinblick auf die Flexibilität bezüglich der Modellinformation und der gegebenen Umgebung. Das erste Problem wurde durch die Einführung des Konzepts des *kontextadaptiven Betriebs*, das letztere durch die Verwendung eines hoch standardisierten, modularen, client-server basierten Designs gelöst.

Nach der Erläuterung der Systemarchitektur haben wir Methoden zum Vermitteln des Modellwissens an die Maschine gezeigt. Die Grundidee ist, dass der Benutzer den Roboter

durch die Umgebung dirigiert, als ob dieser autonom fahren würde, während dieser automatisch Informationen über Orte, Wege und Landmarken sammelt. Außerdem muss der Benutzer dem Roboter interaktiv relevante 3D-Objekte zeigen und benennen, die für die Erfüllung der späteren Aufgaben erforderlich sind. Dadurch entstehen kompakte Modelle, die zugleich verifiziert sind, da der Roboter an allen Orten gewesen ist, alle Wege gefahren, alle Landmarken erkannt, und alle Objekte überprüft hat, bevor sie in sein Weltmodell aufgenommen wurden.

An einem realen Beispiel haben wir gezeigt, dass die Information, die der Benutzer dem Roboter beigebracht hat, zusammen mit dem *kontextadaptiven Betrieb* ausreichend ist, um Missionen zu planen und den Roboter robust in einer zuvor unbekanntem aber standardisierten Büroumgebung navigieren zu lassen. Mit hier nicht dargestellten Experimenten haben wir ferner gezeigt, dass das vorgestellte Rahmensystem auch auf andere Umgebungen wie Fabrik- und Restaurantszenarien übertragbar ist.

Teile der Arbeiten im Projekt „Advanced Servicing Robot“ sind durch das BMBF-Projekt NEUROS (Neuronale Skills Intelligenter Roboter) gefördert worden, das unter Federführung der DaimlerChrysler Aerospace durchgeführt wurde.

Die Ergebnisse dieser Arbeiten dienen als Grundlage für weitere Forschung auf dem Gebiet der Mensch-Roboter-Interaktion im laufenden BMBF-Projekt MORPHA (Intelligente Anthropomorphe Assistenzsysteme).

Literatur

[Elfes 1987] A. Elfes. Sonar-Based Real-World Mapping and Navigation. IEEE Journal of Robotics and Automation, Vol. 3, No. 3, S. 249-265, 1987.

[Firby 1989] R.J. Firby, Adaptive Execution in Dynamic Domains. Yale University, Department of Computer Science (1989)

[Fox et al 1997] D. Fox, W. Burgard, S. Thrun, The Dynamic Window Approach to Collision Avoidance, IEEE Robotics & Automation Magazine, März 1997, S. 23-33

[Gat 1994] E. Gat. Three-Layer Architectures. In D. Kortenkamp, R.P. Bonasso, R. Murphy: Artificial Intelligence and Mobile Robots. AAAI Press/The MIT Press (1998) 195-210

[Gutmann & Nebel 1997] J-S. Gutmann und B. Nebel. Navigation mobiler Roboter mit Laserscans. In: Autonome Mobile Systeme. Springer 1997.

[Henderson & Shilcrat 1984] T. Henderson und E. Shilcrat. Logical Sensor Systems. Journal of Robotic Systems, Vol. 1, No. 2, S. 169-193, 1984.

[IPA 94] Serviceroboter – ein Beitrag zur Innovation im Dienstleistungswesen. Fraunhofer-Institut für Produktionstechnik und Automatisierung (IPA). 1994.

[Jensfelt & Kristensen 99] P. Jensfelt und S. Kristensen. Active Global Localisation for a Mobile Robot using Multiple Hypothesis Tracking. In: Proceedings of Reasoning with Uncertainty in Robotics '99, Stockholm, Schweden, 1999.

[Jiang & Bunke 1994] X. Jiang und H. Bunke. Fast Segmentation of Range Images into Planar Regions by Scan Line Grouping. Machine Vision and Applications, Vol. 7, No. 2, S. 115-122,

1994.

[Koenig & Simmons 1998] S. Koenig und R.G. Simmons. Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models. In: D. Kortenkamp et al.: Artificial Intelligence and Mobile Robots, S. 91-122. AAAI Press/The MIT Press, 1998.

[Kuipers & Buyn 1988] B. J. Kuipers und Y-T. Buyn. A Robust, Qualitative Method for Robot Spatial Learning. In: Proceedings of the Seventh National Conference on Artificial Intelligence, S. 774-779. AAAI Press, Menlo Park, 1988.

[Kuipers & Buyn 1991] B. J. Kuipers und Y-T. Buyn. A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations. Robotics and Autonomous Systems, Vol. 8, S. 47-63, 1991.

[Lu & Milios 1997] F. Lu und E. Milios. Globally Consistent Range Scan Alignment for Environment Mapping. Autonomous Robots, Vol. 4, No. 4, S. 333-349, 1997.

[Piaget & Inhelder, 1967] J. Piaget and B. Inhelder. The Child's Conception of Space, New York: Norton, 1967.

[Simmons 1994] R.G. Simmons. Structured Control for Autonomous Robots. IEEE Transactions on Robotics and Automation, Vol. 10(1) (1994) 34-43

[Thrun et al. 1998] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hoffmann, M. Krell und T. Schmidt. Map Learning and High-Speed Navigation in RHINO. In: D. Kortenkamp et al.: Artificial Intelligence and Mobile Robots, S. 91-122. AAAI Press/The MIT Press, 1998.



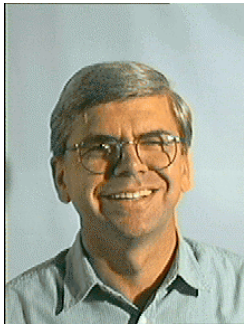
Dr. Frieder Lohnert leitet die Abteilung "Kognition und Robotik" bei der DaimlerChrysler-Forschung in Berlin, die sich mit Computational Intelligence, Intelligent Control, kognitivem Verhalten sowie mit autonomen Robotern für Produktion und Dienstleistungen befasst. Er studierte Elektrotechnik an der TU Berlin, war wissenschaftlicher Mitarbeiter am Heinrich-Hertz-Institut und am Institut für Technische Informatik der TU Berlin, bevor er 1984 in die Industrieforschung ging.



Steen Kristensen erwarb 1992 sein M.Sc.EE. an der Universität Aalborg, Dänemark. Er promovierte 1996 an der selben Universität mit Arbeiten zum Thema Sensorplanung für autonome mobile Roboter mit Gastaufenthalten an der University of Pennsylvania und am Oak Ridge National Laboratory. Seit Ende 1996 arbeitet er bei DaimlerChrysler Forschung und Technologie in Berlin an mobilen Robotern für Service und Produktion. Seine primären Forschungsinteressen sind Sensorik und Planung für mobile Roboter sowie deren Systemarchitekturen.



Sven Horstmann schloss 1997 sein Studium der Elektrotechnik an der TU-Berlin mit einer Diplomarbeit zum Thema „Situationsadaptive Kollisionsvermeidung für mobile Roboter“ ab. Seitdem arbeitet er bei der DaimlerChrysler Forschung auf dem Gebiet der mobilen Robotik. Interessenschwerpunkte sind die Navigation und das Hardwaredesign mobiler Roboter.



Andreas Stopp erwarb sein Diplom 1973 an der Technischen Universität Dresden (Technische Kybernetik) und promovierte 1987 an der Humboldt-Universität Berlin. Er arbeitete zunächst am Institut für Regelungstechnik Berlin, danach am Zentralinstitut für Kybernetik und Informationsprozesse der Akademie der Wissenschaften in Berlin an Architekturen für Robotik und KI. Seit 1991 ist er bei der DaimlerChrysler Forschung in Berlin auf dem Gebiet "Architekturen für intelligente Systeme" tätig. Er ist Leiter des Fachgebietes "Mobile Robotik".



Volker Hansen schloss 1994 von der Universität Karlsruhe als Diplom-Informatiker ab und erwarb im gleichen Jahr das DEA Informatique des Institut National Polytechnique Grenoble, Frankreich. Von 1995 bis 1999 arbeitete er bei DaimlerChrysler Forschung und Technologie an mobilen Robotern für Service und Produktion. Seit Ende 1999 arbeitet in der PKW-Entwicklung im Bereich Telematikdienste.



Jesko Klandt studiert an der TU-Berlin Informationstechnik im Maschinenwesen, Studienrichtung Prozess- und Systemtechnik. Seit 1995 arbeitet er bei der DaimlerChrysler Forschung an neuronalen Verfahren zur Robotersteuerung sowie an Weltmodellgenerierung und Relokalisierung.



Konstantin Kondak beendete 1999 den Studiengang Informationstechnik im Maschinenwesen, Studienrichtung Prozess- und Systemtechnik, an der Technische Universität Berlin. Zur Zeit promoviert er am Institut für Technische Informatik (Prof. G. Hommel), an der TU Berlin. Zu seinen Forschungsinteressen gehören Systemtheorie, nicht-lineare Optimierung und Robotik.