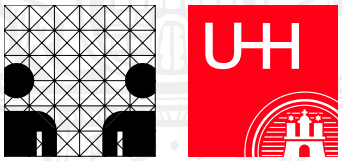
Entwurf rekonfigurierbarer Systemarchitekturen für mobile Roboter

Dissertation
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

vorgelegt
dem Fachbereich Informatik
der Universität Hamburg



von
Dipl.-Inform. Johannes Bitterling

Hamburg, Februar 2005

Gutachter:	Prof. Dr. Jianwei Zhang Prof. Dr.-Ing. Karl Kaiser
Vorsitzender des Prüfungsausschusses:	Prof. Dr. Klaus von der Heide
Dekan:	Prof. Dr.-Ing. H. Siegfried Stiehl
Tag der Einreichung:	31. August 2004
Tag der Disputation:	07. Februar 2005

Kurzfassung

Diese Arbeit befaßt sich mit der Konstruktion von Robotersystemen unter den Randbedingungen variierender Anwendungsprobleme und Einsatzumstände. Üblicherweise wird die Konstruktion unter Beachtung von Paradigmen vorgenommen, die strukturelle Regeln für den Systemaufbau implizieren. Aus der komparativen Systematisierung solcher Schemata geht der deskriptive Begriff der sog. Systemarchitektur hervor. Alle Architekturentwürfe weisen eine Verbindung zu Problemen bestimmter Klassen auf. Trotz einer nomothetischen Qualität innerhalb der jeweils beherrschten Problemdomäne sind solche Entwürfe in Bezug auf Allgemeinheit oder Adaptierbarkeit erheblichen Einschränkungen unterworfen. Es wird untersucht, wie diese aus dem problembezogenen Entwurf rührenden Einschränkungen durch Modifikation des Designprozesses aufgehoben werden können.

Eine Aufteilung des Entwurfs auf vier softwarebezogene Ebenen wurde vorgenommen, um die unerwünschten Einflüsse auf den Vorgang der Konstruktion zu eliminieren. Auf der ersten Ebene wird eine Abstraktion von der strukturellen Vernetzung der Hardware vorgenommen; die zweite Ebene dient der hierarchischen Modellierung von Hardwarekomponenten durch Softwarerepräsentationen. Die Implementierung von höherwertigen Fähigkeiten erfolgt bei Nutzung abstrakter Hardwarerepräsentationen auf generische Weise. Dieser Fähigkeiten bedienen sich auf der dritten Entwurfsebene unabhängige asynchrone Subsysteme, die wesentliche Merkmale konventioneller Architekturen aufweisen. Eine problemgerechte Kombination dieser Subsysteme erfolgt auf der vierten Ebene im Kontext flexibler Metaarchitekturen. Hier werden die Konfiguration von Robotersystemen und ihre angemessene Anpassung unterstützt.

An einem Demonstrator wurde der Aufbau eines von Hardwarestruktur und Geräteeigenschaften abstrahierenden Basissystems gezeigt. Dieses wurde zur exemplarischen und in Bezug auf Anwendung und Plattform unspezifischen Formulierung von Systemfähigkeiten eingesetzt. Da auf die Frage nach deren Kombination wegen der paradigmenspezifischen Beschränkungen keine sinnvolle Antwort in Form einer statischen Architektur gegeben werden kann, wurde mit der Metaarchitektur eine Struktur entwickelt, die den dynamischen Architekturentwurf gestattet. Zur Verifikation der Vollständigkeit wurde die Subsumption der klassischen Ansätze demonstriert. Neben der Vereinheitlichung treten so als neuartige Attribute von Systemarchitekturen dynamische Rekonfiguration, die Erweiterbarkeit, die Adaptierbarkeit, die Übertragbarkeit und die Austauschbarkeit von Komponenten auf. Dies ist ein wesentlicher Schritt in Richtung der Konstruktion allgemeiner Systeme.

Abstract

This study investigated the construction of robotic systems in the presence of conditions, like variations of the field and the kind of application. Usually, construction is carried out while considering one of a list of available paradigms which imply certain structural rules for construction. The descriptive notion of "system architectures" is a result of comparing and systematizing schemes related to such systems. All architectural variants seem strongly related to both certain classes of robot application areas and specific design rules. Thus, such variants are severely limited as regards universality and adaptability. In this study it was attempted to derestrict the systems' capabilities by altering the traditional design process.

The process of design was divided into four software-related layers. This allows eliminating unwanted effects on the process of construction. The first layer deals with the abstraction from given hardware structures and communication links used; the second layer provides hierarchical access to available hardware components by providing appropriate software representations. All implementation of high-level capabilities is performed in a generic way utilizing abstract representational objects. On the third layer, independent and asynchronous subsystems make use of these capabilities, and thus may operate detached from hardware specifics. A single subsystem may show properties of a conventional robot architecture, but merely implements the solution of a single aspect of a robot application problem. Finally, the combination of subsystems into an operational system is dealt with on the fourth layer. A flexible meta-architecture permits configuring and revising arbitrary robot architectures.

A demonstrator platform was utilized to show the construction of a basic system, which offers independence both from the hardware structure and the properties of specific sensor or effector devices. Using this system base, some high-level capabilities were designed which are still unrelated to both the robot's application area and the technical details of the platform. As the conceptual restrictions of architectures following classical paradigms still hold, a completely different mode was chosen to formulate the system superstructure. A meta-architecture allows the dynamic assembly of architectural schemes. To verify the functional completeness, the capability of subsuming the classical design variants was demonstrated. This study found that not only was a technical unification reached regardless of architectural decisions, but also novel attributes of robot architectures emerged: namely dynamic reconfiguration, extensibility, adaptability, exchangeability of components and an increased potential of transferability. This may be considered to be a significant step forward towards the construction of universal robotic systems.

Inhaltsverzeichnis

Abbildungsverzeichnis

Symbol- und Abkürzungsverzeichnis

$\mathbf{A}$	Matrix
$\mathbf{A}^\top$	transponierte Matrix
${}^A\mathbf{T}_B$	Transformation von Koordinatensystem A nach B
$\mathbf{f}()$	vektorwertige Funktion
$\mathbf{a}$	Vektor
3T	*3-tiered [architectures]* (BONASSO)
ADT	Abstrakter Datentyp
AFSM	*augmented finite state machine*
AI	*artificial intelligence*, vgl. KI
ATLANTIS	*A three-layer architecture for navigating through intricate situations*
AuRA	*autonomous robot architecture* (ARKIN)
CML	*concurrent mapping and localization* – vgl. SLAM
CORBA	*common object request broker architecture*
CRP	*captured robot problem*
CWA	*closed world assumption*: (p nicht deduzierbar) $\therefore \neg p$
DCE	*distributed computing environment*
DCOM	*distributed component object model*
DoF	*degree(s) of freedom*
DP	*distance propagation*
DR	*"dead reckoning"*; eigentl.: *deduced reckoning*
EERUF	*error eliminating rapid ultrasonic firing*
FSM	*finite state machine*
FWA	*flat world assumption*
IMA	Internes Modell der Außenwelt (Repräsentation)
IRM	*innate releaser mechanism*
JIRA	*Japan Industrial Robot Association* (gegr. 1968)
KI	Künstliche Intelligenz
LIDAR	*light detection and ranging*
LPS	*local perceptual space* (robozentrisch)

LRF	*laser range finder*, eine LIDAR-Anwendung
MCL	*Monte Carlo localization*
MPI	*message passing interface*
NHC	*nested hierarchical controller* (MEYSTEL)
pfield	Potentialfeld
pthread	POSIX thread (IEEE POSIX 1003.1c standard 1995)
PTU	*pan-tilt-unit*, Schwenk-/Neigeeinheit
PVM	*parallel virtual machine*
RAP	*reactive action package*
RCS	*real-time control system* (ALBUS)
RIA	*Robot Institute of America* (gegr. 1974)
RMI	*remote method invocation*
RPC	*remote procedure calls*
SFX	*sensor fusion effects*
SGH	*symbol grounding hypothesis*
SIP	*server information packet*
SLAM	*simultaneous localization and mapping*
SSS	*servo, subsumption, symbolic* (CONNELL)
TCA	*task control architecture*
TLA	*three-layer[ed] architecture* (GAT)
TTC	*time to collision*

Kapitel 1

Einleitung und Übersicht

Das Beseelte scheint sich vom Unbeseelten durch zweierlei hauptsächlich zu unterscheiden, durch Bewegung und Wahrnehmung.
ARISTOTELES, *De anima*

Dieses erste Kapitel wird in 1.1 den allgemeinen Kontext des gewählten Forschungsproblems – des systematischen Entwurfs mobiler Robotersysteme – vorstellen, und dazu in 1.2 die grundsätzliche Fragestellung dieser Arbeit vorstellen. Diese wird nach Betrachtung des Stands der Forschung, die im folgenden Kapitel 2 geschieht, in Kap. 3 zu einer spezifischen, im Rahmen dieser Arbeit bearbeiteten Problemstellung entwickelt werden.

Der Abschnitt 1.3 dieses Kapitels skizziert die Struktur der Arbeit und das beabsichtigte Vorgehen.

Der letzte Abschnitt dieses Kapitels 1.4 umreißt in Kürze den Beitrag der Arbeit.

1.1 Mobile Roboter

Gegenstandsbestimmung: Was sind Roboter? Zu der Frage, was genau ein Roboter sei, existiert eine Reihe von Definitionsversuchen, die vornehmlich von Forschungs- und Industriegruppen unternommenen wurden – beispielsweise der *Japan Industrial Robot Association* (JIRA) oder des *Robot Institute of America* (RIA). Erst bei dem Versuch, eine Vergleichbarkeit statistischer Daten über die Stückzahlen im Einsatz befindlicher Systeme herzu-

stellen, tritt mitunter die Unvereinbarkeit der Definitionen zutage [Lus81, S. 123]. Grundsätzlich hat sich in den letzten Jahrzehnten an diesem Zustand nicht viel geändert.

Als gemeinsamer Nenner anerkannt ist – nicht zuletzt durch die plakative Benennung des Gegenstands durch die Entlehnung des tschechischen Wortes für (Fron-)Arbeit in [Č61] –, daß es sich um technische Artefakte handeln soll, die eine komplexe und variable physische Arbeit durchführen sollen, ohne dabei unmittelbar von einem Menschen gesteuert zu werden. Solche "Roboter" hingegen, die keine eigenständige physische Existenz haben und beispielsweise als *web robots* in Erscheinung treten, sind in Anbetracht der explizit physischen Konnotation des Wortes *robota* – das eine Variante der mittelalterlichen *corvada* bezeichnet – streng genommen irreführend benannt und sollen hier nicht behandelt werden.

Robotik und mobile Systeme. Die noch immer im Aufbau befindliche Disziplin der "Robotik" tritt je nach Schwerpunkt eines jeweils betrachteten Projekts als Teildisziplin der Ingenieurswissenschaften oder der Informatik auf – erstere liefern (als "Robotertechnik") das technische Artefakt, die letztgenannte das "System" und somit die Methoden für die zu inkorporierende Kontrolle.

Im Fall dieser Arbeit soll eine Berücksichtigung des Bereichs der mobilen Robotiksysteme stattfinden. Obgleich kaum ein prinzipieller Unterschied zwischen allgemeinen Effektoren mit physischer Wirkung auf die "Außenwelt" (z.B. Roboterarmen) und solchen besteht, die eine Veränderung der Koordinaten des Roboters selbst in der Umwelt bewirken (wie einem Fahrwerk mit Antrieb), tritt doch bei mobilen Systemen eine verstärkte Notwendigkeit zur reflexiven Berücksichtigung des Roboters selbst als Teil der zu manipulierenden (Um-)Welt auf. Insofern ist durch die Wahl "mobiler Roboter" keine Einschränkung gegeben, sondern geschieht vielmehr die Berücksichtigung des allgemeinen Falls.

Erkenntnisinteresse. Robotik wird – mindestens – aus zwei verschiedenen Gründen betrieben:

- Zum einen ist die Suche nach einer pragmatische Lösung des Problems der Durchführung von (konkreter und spezifizierter) Arbeit ein Gegenstand der Forschung, der in der Vergangenheit die Entwicklung recht erfolgreicher spezialisierter technischer Systeme befördert hat. Wenn auch in "einfachen" Fällen oft ein Automat ausreichend sein

kann, ist unter bestimmten Randumständen der Einsatz eines Roboters wegen seiner spezifischen Eigenschaften erforderlich.

- Zum anderen besteht ein (zunächst theoretisches) Erkenntnisinteresse ohne Bezug auf konkrete Aufgabenstellungen an der Analyse der Bedingungen der erfolgreichen Konstruktion von Robotersystemen. Die gewonnenen Erkenntnisse dienen schließlich dem (praktischen) Ziel der Konstruktion leistungsfähigerer Systeme. Ein analoges Verhältnis besteht beispielsweise zwischen der Teildisziplin der Softwaretechnik und der Informatik (wobei allerdings die "Robotertechnik" nicht die Teildisziplin des systematischen Entwurfs von Robotern ist, sondern "nur" die physisch-technischen Grundlagen bereitstellt).

Die Frage nach einer allgemeinen Methodologie der Robotik ist berechtigt, aber höchstwahrscheinlich noch nicht zu beantworten: obgleich entsprechende Beiträge existieren,[1] können auch hier noch keine Antworten gegeben werden, die den wissenschaftstheoretischen Status eines Satzes hätten. Allein Einsicht in die impliziten Regeln des Designs "erfolgreicher Systeme"[2] scheint möglich.

Anforderungen an Systeme. Die genaue Zusammenstellung der an ein robotisches System gestellten Anforderungen ist offensichtlich vom jeweiligen Erkenntnisinteresse abhängig. Betrachtet man Roboter als eine spezielle, nämlich physische und mobile Art eines Agenten [Bra97] (dies ist gerade unter Hinblick auf die Analyse verbreiteter Taxonomien autonomer Agenten üblich [Log98]), stellen sich die geforderten Eigenschaften nach FRANKLIN derart dar: "Reaktivität, Autonomie, kollaboratives Verhalten, Fähigkeit zur Kommunikation, Fähigkeit zur Inferenz, Persistenz des (inneren) Zustands, Persönlichkeit, Lernfähigkeit und Mobilität" [FG96]. Einige Autoren gehen darüber hinaus soweit, Emotionen integrieren zu wollen [TPP03] oder gar eine gewisse Verantwortlichkeit des Systems für seine Entscheidungen zu fordern [KJ99]. Gemessen am derzeitigen Stand der Forschung kann allerdings nicht (in Abgrenzung zur Simulation) im eigentlichen Sinne von Robotern als Lebewesen gesprochen werden, und bis zu einer Rechts- und Schuldfähigkeit wäre auch dann noch ein unüberschaubarer Weg zurückzulegen. Tatsächlich überwiegen nicht erfüllte Anforderungen (ob "noch" oder "überhaupt" soll hier nicht diskutiert werden), und viele Lösungen sind noch prototypisch, aber keineswegs allgemeinen.

[1] Vgl. [Ioc99] und natürlich bestimmte Aspekte dieser Arbeit

[2] Dieser Terminus wird erstmals mit [KBM98] im Rahmen einer Fallsammlung verwendet.

Herausforderung. Die besondere Schwierigkeit der Robotik – über die algorithmische Umsetzung von Verfahren hinaus, wie sie Gegenstand der Informatik ist – rührt aus mehreren Quellen.

Zum einen ist eine Vielzahl von Teilsystemen zu integrieren, die ihrerseits aus aktuellen Forschungsgebieten hervorgehen: so wird (um nur ein wesentliches Beispiel anzuführen) im Ideal eine zufriedenstellende und robust funktionierende Bildverarbeitung von den ersten Vorverarbeitungsstufen bis hin zur Objekterkennung zur Lösung vieler Anwendungsprobleme vorausgesetzt, obwohl schon diese für sich allein für nicht absehbare Zeit Gegenstand weiterer Forschung sein wird. Zum anderen stellen die Unwägbarkeiten der "Umweltschnittstelle" an sowohl den Sensoren als auch den Effektoren ganz eigene Anforderungen an die in der zwischenliegenden "Verarbeitung" zu implementierenden Verfahren und Systeme; und dies selbst, wenn nicht einmal einsatztaugliche weitreichende Fehlertoleranz, sondern überhaupt nur grundlegende prototypische Funktion das Entwurfsziel wäre.

1.2 Einführung in die Fragestellung

Eine der wichtigsten gegenwärtigen Fragestellungen der Robotik resultiert aus den technischen Wurzeln dieser Disziplin und dem spezifischen strukturellen Beitrag der Informatik: wie sollen die Bedingungen von Konstruktion und Einsatz von Robotern vernünftigerweise aussehen?

Spezielle Aufgaben, problembezogener Entwurf. Die technischen Wurzeln der Robotik liegen in den technischen und naturbezogenen Wissenschaften, deren Theorien und Anwendungen die Komponenten des physischen Teils einer Roboterplattform[3] liefern.

In der von "Maschinen" bis zu "Robotern" reichenden Skala stellen Systeme zur Telepräsenz (vgl. [GS02]) eine erste Stufe dar. Sie sind "einfache" Maschinen mit vergleichsweise komplizierter Fernsteuerung und verfügen im typischen Fall nicht über Autonomie. Bereits autonom, aber weitgehend spezialisiert sind die heute verbreiteten Industrieroboter. Die Nachfolgerschaft zu gewöhnlichen Werkzeugen und Automaten ist in der Regel in äußerer Erscheinung und Grad der Spezialisierung gut zu erkennen – oft

[3] Begrifflich soll hier und im folgenden differenziert werden: die Roboterplattform als der gesamte Hardwareaufbau, der Roboter hingegen als die Software einschließendes hybrides Gesamtsystem.

handelt es sich um stationäre Systeme oder solche, die sich in einer speziell präparierten Umwelt bewegen (Schienen, Induktionsschleifen, synthetische Landmarken), und eine exakt umschriebene Aufgabe ohne oder mit nur wenigen variablen Parametern bearbeiten (z.B. bei robotisierten Demontagearbeiten [Gen94]). Entsprechend diffus verläuft in der technischen Literatur die Unterscheidung zwischen Robotern und ihren Vorstufen (erneut vgl. [Lus81]).

Mit dem Einsatz von Systemen dieser beiden Entwicklungsstufen fiel die Notwendigkeit der Präsenz eines menschlichen Bedieners im ersten Schritt am Platz der zu leistenden Arbeit, im zweiten Schritt überhaupt weg. Trotz der steigenden Komplexität der Aktionen, die solche Systeme auszuführen vermögen, werden Industrieroboter samt ihrer Arbeitsumgebung in aller Regel ingenieurmäßig für "die" zu erledigende Aufgabe konzipiert, konstruiert oder eingerichtet. Aus pragmatischen Gründen ist diese technische Spezialisierung eine zwar noch regelmäßig akzeptierte, aber sicherlich nicht optimale Lösung.

Das Problem der Entwicklung autonomer mobiler Robotersysteme erfordert die Lösung vergleichsweise vieler Subprobleme – so tauchen beispielsweise "Bildverarbeitung", "Objekterkennung", "Planung" und "Navigation" als Vorbedingungen für die Funktionalität des Gesamtsystems auf, obwohl sie von ihrer Komplexität gut als eigenständige Probleme betrachtet werden könnten. Jeder an einer bestimmten exemplarischen Anwendung orientierte Ansatz der "Einrichtung" auf ein Anwendungsproblem wäre, einmal abgesehen von der offensichtlichen Unzweckmäßigkeit wegen der hohen Zahl beteiligter Teilsysteme, auch in Bezug auf das Erkenntnisinteresse von weit geringerem Wert. Wesentlich interessanter und, wie bereits bemerkt, in seiner Systematik noch nicht annähernd durchleuchtet ist der Aspekt der sinnreichen Integration zu einem potentiell leistungsfähigen System für ein nicht vorab spezifiziertes Anwendungsproblem.

Der ideale Roboter als "unspezialisierte Maschine". Die auf bestimmte Aufgaben festgelegte Rechenmaschine hat eine sehr erfolgreiche Entwicklung zum allgemeinen speicherprogrammierbaren Rechner durchgemacht. Dieser kann erfolgreich zur Bearbeitung von Anwendungsproblemen eingesetzt werden, die zur Zeit der Konstruktion noch nicht einmal spezifiziert waren. Analog soll der hypothetische Vertreter der Klasse der idealen Roboter hier als *allgemeine unspezialisierte Maschine* betrachtet werden.

Die Tatsache, daß die Mehrzahl der existierenden Systeme zur Bewältigung eines oder weniger konkreter Probleme konstruiert wurde, tut diesem Po-

tential nicht notwendig Abbruch. Das von ARKIN auf die Robotik übertragene Konzept MCFARLANDs der "ökologischen Nische" [Ark95] ist allerdings ein deutlicher Hinweis darauf, daß im Allgemeinen das weniger spezialisierte Gerät unter "Selektionsdruck" einen Vorteil besitzt.

Zum gegenwärtigen Zeitpunkt dominiert allerdings noch immer ein anwendungsproblembezogenes Vorgehen des Systementwurfs: Roboter werden für eine bestimmte ökologische Nische konzipiert und sind meist außerhalb dieses fest umschriebenen Bereichs nicht einsetzbar. Dies schlägt sich – wenigstens außerhalb der industriellen Robotik – nicht (mehr) notwendig auf der Ebene der Entwicklung spezieller Hardwarekomponenten nieder, doch ist die Wahl der Kontrollarchitektur und die Durchführung der konkreten Programmierung scheinbar zwingend auf den jeweiligen Problemkontext zugeschnitten: Lösungsstrategien werden in der Regel zu einem bestimmten Hardwareaufbau passend codiert; und eine Systemdemonstration geschieht im Kontext eines gewählten Szenarios.

Die Adaption des so erhaltenen Systems zur Bearbeitung einer anderen, hinreichend unähnlichen Aufgabe erfordert nicht nur eine angemessene Parametrisierung, sondern meist ein vollständiges konzeptionelles Redesign.[4]

Systematisierung und Kontrollparadigmen. Als Ausweg aus der Notwendigkeit des problembezogenen Entwurfs haben sich in der Vergangenheit bestimmte Kontrollparadigmen als empirisch gewonnene Heuristiken für den Aufbau der inneren Struktur von Robotersystemen herauskristallisiert. Die daraus resultierenden Basissystemstrukturen (deren ausführlicher Betrachtung der Abschnitt 2.1 gewidmet ist) versprechen, einen Rahmen für angemessene technische Lösungsansätze zur Bearbeitung von Anwendungsproblemen zur Verfügung zu stellen. Faktisch haben diese in der Vergangenheit oft überhaupt erst zur Lösung bestimmter Aufgaben geführt.

Wie ARKIN bemerkt, besitzen allerdings diese Ansätze – die z.B. subsymbolische Methoden der Regelungstechnik bzw. symbolische Verfahren der künstlichen Intelligenz einsetzen – die durchaus unerwünschte Eigenschaft, daß sie jeweils nur für Anwendungsprobleme einer bestimmten Problemklasse ein effizientes Systemdesign versprechen [Ark95]. Dies ist nicht not-

[4] Der Vollständigkeit halber sei auf einen weiteren, völlig verschiedenen Ansatz hingewiesen: die automatische Konstruktion von Robotern, wie sie [LP00] beschreibt, und die selbständige Entwicklung von Kontrollarchitekturen [Ebn98]. Dies ist ein in guter Näherung dem *genetic programming* entsprechender Ansatz, der jedoch wenigstens derzeit nur in einem vergleichsweise kleinen Konfigurationsraum sinnvoll erscheint. Es bleibt daher abzuwarten, ob dieser Ansatz in naher Zukunft wertvolle Beiträge zur Lösung dieses Problems leisten kann.

wendig von Nachteil: geschieht der Entwurf mit Anspruch auf Simulation oder gar der (hier nicht auf Stichhaltigkeit überprüften) Absicht auf "Konstruktion" eines experimentellen "kognitiven Systems" oder mit Intention der Lösung eines konkreten technischen Anwendungsproblems, sind die Entwurfsheuristiken äußerst hilfreich.

In Bezug auf den Entwurf einer "unspezialisierten Maschine" bleiben allerdings Wünsche offen: die Selektion eines Kontrollparadigmas – und resultierend einer bestimmten Systemarchitektur samt Vernetzung der interagierenden Teilsysteme – ist in gewisser Weise abschließend und erschwert eine Anpassung der Maschine zum Zweck der Bearbeitung von Aufgaben eines abweichenden Typs.

Einfluß der Architektur. Prinzipiell läßt sich argumentieren, daß praktisch alle Systemarchitekturen in Bezug auf das grundsätzliche Potential zur Bearbeitung von Anwendungsproblemen gleich mächtig (wenn auch problembezogen unterschiedlich effizient) sind [Ark95]. Die einzige Ausnahme stellen zustandsfreie rein reaktive Systeme dar.

Mindestens aus Effizienzgründen (Rechenzeit, Eleganz der Formulierung) werden in der Praxis dennoch bestimmte Teilaspekte zweckmäßigerweise in reaktiven bzw. deliberativen Teilsystemen implementiert. Nur gelegentlich findet sich der Einwand, daß gewisse Teilprobleme – in der Regel die deliberativ/symbolisch zu lösenden – wegen mutmaßlich fehlender Nähe zu biologischen Vorbildsystemen ohnehin "falsch gestellte" Probleme seien. Auf diese Position soll im Rahmen dieser Arbeit nicht eingegangen werden. Letztlich ist die Zweckbestimmung, mit den gegebenen technischen Mitteln ein der Lösung eines Anwendungsproblems hinreichend dienliches Systemverhalten zu realisieren.

Übliche Problemkandidaten für die Bearbeitung durch reaktive Systeme sind beispielsweise die Kollisionsvermeidung oder einfache Tropismen (bereits in [Wal50]). Für Probleme wie beispielsweise kartengestützte Navigation oder die Durchführung von planbaren zugbasierten Spielen (vgl. [Bit00]) scheinen hingegen deliberative Systeme besonders geeignet. Im Allgemeinen gilt, daß ein "Verhalten" durch reaktive und eine "Funktion" durch deliberative Systembestandteile jeweils mit der einfacheren Struktur erzeugt werden kann.

Ob der Verdienste beider Ansätze in den jeweiligen Problemklassen und des Wunsches der Integration zu einem Gesamtsystem sind seit etwa 1985 Kombinationen in Form hybrider Kontrollarchitekturen – oft in geschichteter Form – üblich (mehr dazu in 2.1). Unter Hinzufügung einer zwischen

den Teilsystemen vermittelnden Zwischenschicht entstehen die in jüngster Zeit entwickelten dreischichtigen Architekturen (vgl. [Gat97]).

Der Preis der Verortung von Teilproblemlösungen in jeweils der Schicht, die die optimalen Voraussetzungen für die Umsetzung bietet, ist eine statische Kommunikationsvernetzung der Teilsysteme, die wiederum sehr leicht anwendungsproblemspezifisch ausfällt.

Bestreben der Lösung von Architektureinflüssen. Zur Annäherung an eine gewünschte Generizität innerhalb eines solchermaßen statisch strukturierten Systems – und natürlich zur Anpassung an Anwendungsaufgaben, ohne eine Rekonfigurierung des Systems vornehmen zu müssen – sind verschiedene Ansätze gebräuchlich, die sich jeweils innerhalb des reaktiven bzw. deliberativen Teils hybrider Gesamtentwürfe lokalisieren lassen.

Im Fall des Ansatzes auf deliberativ-symbolischer Ebene – in der Regel also zur Codierung von Planungsproblemen – werden beispielsweise Interpreter für spezielle roboterbezogene Programmiersprachen zur Verfügung gestellt [BJ83], bei denen eine vom Roboteranwender bereitgestellte Formulierung eines Algorithmus die explizite Kontrolle über das System ausübt. Im reaktiv-subsymbolischen Fall können beispielsweise weitere konkurrent aktivierte Verhaltensmuster wie z.B. die *behaviors* in [KM99] nachgeladen werden und den Roboter indirekt steuern, während die explizite Systemkontrolle (*action selection*) je nach Ausprägung bei einem der statischen Architektur zuzuschlagenden Mechanismus wie einem Arbiter, einem Summationsmechanismus oder einer internen Gewichtung verbleibt.

Ziel solcher Ansätze ist, eine unmittelbare Bindung der Roboterkonstruktion zum bekannten Anwendungsproblem zu vermeiden. Durch die architekturspezifische Verortung des dynamischen Potentials in einem reaktiv oder deliberativ konzipierten Systemteil wird allerdings erneut eine Einschränkung der Allgemeinheit hergestellt, die den potentiellen Raum effizient lösbarer Anwendungsprobleme verkleinert.

Die resultierende Fragestellung. Die Synthese der wissenschaftlichen Erkenntnisse und technischen Methoden robotiknaher Gebiete erlaubt die Konstruktion eines Artefakts, das bereits viele der gewünschten Eigenschaften in Bezug auf ausgesuchte Anwendungsprobleme aufweist.

Die Reichweite der Robotik wird unter anderem bestimmt durch die Qualität (insbesondere Performanz und Stabilität) der von Nachbardisziplinen – wie der Bildverarbeitung, Mustererkennung und KI – bereitgestellten

Verfahren. In diesen Bereichen ist ohne jeden Zweifel noch in nicht abschätzbarem Umfang Arbeit zu leisten. Dennoch ist es völlig unangemessen, die praktischen Einschränkungen der heute verfügbaren Roboter allein auf noch nicht vollständig entwickelte Leistung der benachbarten Disziplinen zurückführen zu wollen: Es geht darum, unter Verwendung der verfügbaren prototypischen Methoden das jeweils optimale Resultat zu erzielen.

Als genuin robotikspezifische strukturelle Frage erscheint also, wie diese Verfahren zweckmäßig zu einem System zusammengefügt werden sollten: Wie soll man konkret einen Roboter konstruieren, der das Potential hat, "etwas zu tun", wobei dieses "etwas" unter Umständen noch nicht einmal vorab spezifiziert ist?

Im Rahmen einer Untersuchung dieser Frage werden drei konstruktive Teilschritte erforderlich sein.

1. Die technische Integration.

   Als hybrides technisches System mit Hard- und Softwarekomponenten wird stets eine (allerdings plattformspezifische) Integration erforderlich sein, um für Hardwarekomponenten die Möglichkeit der Nutzung auf einer algorithmischen Ebene zu erschließen.

2. Die Bereitstellung von Lösungen für Teilprobleme.

   Essentielle Fähigkeiten eines mobilen Roboters betreffen die Abfrage der Sensoren, die Auswertung der so gewonnenen Daten und die Ansteuerung der Effektoren inklusive der Vorrichtungen zur Lokomotion. Größere Aggregationen dieser Basisfähigkeiten könnten die Erkennung von Merkmalen in Sensordaten (mittels z.B. Bildverarbeitung), die Planung von Handlungsabläufen (unter Einsatz z.B. von Planungsverfahren der Künstlichen Intelligenz) oder die Durchführung von Navigation (unter Einsatz aller genannter Mittel) sein.

   Jedes einzelne Teilproblem war und ist gegenwärtig Gegenstand der Forschung, jedoch nicht (allein) der Domäne der Robotik zuzuordnen. Da die genannten Fähigkeiten erforderliche Komponenten eines praktischen Entwurfs sind, wird die Notwendigkeit bestehen, mindestens exemplarisch Lösungen zu den entsprechenden Problemen zu erstellen.

3. Die Strukturierung des Systems.

   Im Fall des Rechnerentwurfs im Rahmen der technischen Informatik ist ein wesentliches Ziel, durch Modellierung eines "großzügigeren

virtuellen Verhaltens" [Lag87] für jede nächsthöhere Entwurfsebene eine immer weiter gehende Abstraktion von der fundamentalen technischen Implementierung zu schaffen und von der Hardware ausgehende *bottom-up*-Einflüsse zu minimieren (idealerweise völlig auszuschalten). Auch in der Softwaretechnik und im Betriebssystementwurf ist eine Schichtung von Abstraktionslayern verschiedener Komplexität übliches Vorgehen. Ob ein analoges Vorgehen in der Robotik, in der die Außenwelt betreffende Funktionseinheiten eine Leistung grundsätzlich nur mit gewisser Wahrscheinlichkeit oder mit unbekanntem Fehleranteil erbringen, möglich ist, wäre zu untersuchen.

Zur Bedeutung der Fragestellung. Roboter entstehen heute oft, indem man Hardwarekomponenten assembliert, Treibersoftware für die Geräte installiert und versucht, ein bestimmtes Gesamtsystemverhalten oder eine Systemleistung möglichst robust zu realisieren. In der Regel ist weder eine Generizität bezüglich der Plattform vorgesehen, noch sind Subsysteme gegen funktionsgleiche andere Implementierungen austauschbar (was Voraussetzung für ein vernünftiges *benchmarking* wäre), noch kann die durch die nicht orthogonale Systemarchitektur vorgegebene, oftmals detaillierte Struktur "von außen" angepaßt werden, um eine Adaption an ein verändertes Anwendungsproblem oder abweichende Rahmenbedingungen vorzunehmen.

All dies ist mit dem Ideal der unspezialisierten Maschine nur schwer vereinbar. Von diesem Standpunkt aus erscheint der Entwurf eines möglichst wenig spezialisierten Systems lohnend. Die Anforderung der Anpaßbarkeit an verschiedene Anwendungsprobleme ist natürlich, und die Existenz eines diesbezüglichen Problems bekannt. Spezielle Roboterprogrammiersprachen, nachladbarer Code, graduelle oder binäre Aktivierung von *skills* bzw. *reactive action packages* zur Laufzeit stellen partielle Lösungen dar, basieren aber auf jeweils einer vorgegebenen – und mitunter in einigen Details der Ausprägung willkürlich erscheinenden, mutmaßlich in der fachlichen Historie der Entwickler begründeten – festen Architektur. Somit läßt sich feststellen, daß das Problem durchaus für eine Vielzahl von speziellen Fällen gelöst ist – doch in keinem Fall allgemein.

1.3 Vorgehen und Lösungsansatz

Die Komponenten "Kontrolle" und "Weltschnittstelle" (Kapitel 2). In Kapitel 2 wird zunächst ein für die weitere Vertiefung der Fragestellung

nötiger Überblick über die Entwicklung und den gegenwärtigen Stand der Forschung in Bezug auf die zum Bau eines vollständigen Roboters benötigten Teilsysteme gegeben: Im Wesentlichen sind dies natürlich die Systeminfrastruktur ("Kontrolle"), die Sensorik mit der jeweils gerätespezifischen Sensordatenverarbeitung und die Motorik, letztere dargestellt im für mobile Systeme wesentlichsten Gesamtkontext der Navigation.

Da der Schwerpunkt der Arbeit auf dem Aspekt der Systemarchitektur liegt, erhält die Robotikkomponente "Kontrolle" einen besonderen Platz. Hier wurde im Laufe der letzten Jahre in der Forschergemeinschaft eine Vielfalt von Ansätzen – von impliziten, nicht als solchen entworfenen Architekturen bis zu der aktuellsten Ausprägung, den TLAs – präsentiert und diskutiert.

Der Hauptertrag des Kapitels 2 gliedert sich in zwei Teile. Zum einen werden die Kontrollmodelle und Systemarchitekturen, die bislang in der Literatur vorgeschlagen wurden, auf ihre spezifischen Eigenschaften untersucht, da sie für den Systementwurf einen Lösungsraum mit unvermeidbaren, aber jeweils für sich wählbaren Mengen von Einflüssen aufspannen. Zum anderen findet (kurz) eine Betrachtung der Teilsysteme statt, die an der Schnittstelle zur "Außenwelt" auftreten und das zentrale Charakeristikum von Robotiksystemen ausmachen. Sie führen Randbedingungen ein, die bei dem Entwurf der Systemkontrolle Berücksichtigung finden müssen.

Die Problemspezifikation (Kapitel 3). In Kapitel 3 werden die Existenz und die Relevanz des aufgeworfenen Problems der Konstruktion einer "unspezialisierten Maschine" unter Berücksichtigung der in Kapitel 2 vorgestellten Lösungsvorschläge erneut überprüft.

Die Bewertung des Problems aus Sicht der gegebenen Situation und die Feststellung der tatsächlichen Berechtigung der vorgestellten klassischen Ansätze zu jeweils mindestens einem konkreten Zweck führt für das weitere Vorgehen auf die Festlegung der folgenden Arbeits- und Testschritte der Erstellung von exemplarischen Subsystemen, um überhaupt in eine Kontrollarchitektur einzufügende Elemente zu besitzen, des Entwurfs einer abstrakten und unspezifischen Kontrollarchitektur und der Prüfung dieses Entwurfs auf die Möglichkeit der Subsumption der vorhandenen Ansätze und den Nachweis der spezifischen Leistungserweiterung gegenüber der Summe dieser Ansätze.

Der Systementwurf (Kapitel 4). Hier findet zunächst eine kurze Bestandsaufnahme der Ausgangssituation statt, die sich auf den in Kapitel 2 ange-

führten Forschungsergebnisse, die im Projektrahmen verfügbare Hardware und bereits geleistete Vorarbeiten stützt.

Der eigentliche mehrschichtige Lösungsansatz wird *bottom-up* präsentiert. Dieser Weg der Konstruktion wie Präsentation erscheint vorteilhaft, da gerade bei einem Versuch des Entwurfs einer "unspezialisierten Maschine" die *top down*-Einflüsse eher allgemeiner Natur und nur schwach ausgeprägt sein werden.

Die thematische Verbindung der auf der unteren Ebene getroffenen Entwurfsentscheidungen mit dem Thema dieser Arbeit scheint vordergründig paradox. Allerdings geht es (neben dem eher technischen Aspekt der Inbetriebnahme) entscheidend darum, alles zu vermeiden, was bereits auf dieser Ebene durch strukturelle Implikationen auf die übergeordneten Ebenen wirken und damit letztlich den Freiheitsgrad des Architekturentwurfs auf höherem Abstraktionsniveau einschränken könnte.

Die einzelnen Entwurfsschritte sind im groben Umriß:

1. Die mechanische, elektrische und informationstechnische Integration der Hardwarekomponenten als Bedingungen für eine Verwendbarkeit der Sensoren und Effektoren selbst,
2. die Bereitstellung einer angemessenen, möglichst abstrakten Softwareschnittstelle zu den spezifischen Roboterkomponenten,
3. die Bildung von Subsystemen, die unter Rückgriff auf die Schnittstellen spezielle Verfahren (zur Implementierung von einem Verhalten oder einer bestimmten Funktion) als Einheiten kapseln,
4. die Wahl einer zwischen den Subsystemen vermittelnden orthogonalen Kontroll- und Kommunikationsstruktur und deren Implementierung.

Die Präsentation der zu Evaluationszwecken modellierten Anwendungsszenarien und Verweise auf mit dem Gesamtsystem durchgeführte weitere Arbeiten schließen dieses Kapitel ab. Die Evaluation schließt ein:

- die Verifikation der verlangten Eigenschaften
  - der Orthogonalität
  - der Subsumption anderer Ansätze und Methoden,
  - der Austauschbarkeit der modularen Subsystemen;

- die Auswertung der zusätzlichen Eigenschaften, unter anderem
  - die strukturelle Fehlertoleranz,
  - die Adaptierbarkeit der so zu erstellenden Systeme.

Schluß: Kapitel 5. Im Schlußkapitel wird eine Zusammenfassung der Ergebnisse vorgenommen. Der Ertrag wird unter Rückgriff auf die mit der Fragestellung konnotierten Anforderungen erneut einer abschließenden Bewertung unterzogen.

1.4 Beitrag dieser Arbeit

Grundlage. Diese Arbeit führt in Kap. 2 die Kontrolle als wichtigste und einzig strukturbildende Komponente eines Robotersystems ein und präsentiert in 2.1 eine Analyse der wichtigsten Systemarchitekturen und ihrer spezifischen Leistungen. Dieser deskriptive Teil ist als Fortführung des Systematisierungsbestrebens mit dem Ziel der Herstellung einer Ordnung der wachsenden Vielfalt an Architekturentwürfen zu sehen.

Entwurfsebenen. Diese Analyse der Architekturen weist das grundsätzlichen Problem der mangelnden Allgemeinheit der vorgestellten Ansätze auf (Kap. 3) und soll Existenz und Relevanz der Fragestellung dieser Arbeit bestätigen, wie ein möglichst von technischen Randbedingungen gelöster Systementwurf vorgenommen werden kann, der anschließend auf verschiedenartige Anwendungsprobleme angepaßt werden kann.

Abstraktion von der Hardwarekonfiguration. Zur Bewältigung des Problems wird in Kapitel 4 ein Lösungsansatz auf insgesamt vier verschiedenen Ebenen vorgestellt. Die unterste Ebene befaßt sich allein mit der Abstraktion von den konstruktionsbedingten Randbedingungen konkreter Plattformen und wird am Beispiel der Inbetriebnahme einer Demonstratorplattform (Abb. 4.1(b), S. 96) einer näheren Untersuchung unterzogen. Auf dieser Ebene ist kaum ein wesentlicher Designfreiraum zu erwarten, doch ist die Eliminierung von Einflüssen der oft akzidentellen Struktur der Hardwarevernetzung erforderlich.

Geräte- und Protokollabstraktion. Die folgende, nächsthöhere Ebene ist mit der Abstraktion von Protokoll und Leistung eingesetzter Geräte verbunden. Auf diese Weise soll ein unangemessener *bottom up*-Einfluß von der gewählter Roboterhardware auf die Implementierung konkreter Verfahren weitestgehend ausgeschaltet werden. Darauf aufbauender Software soll eine generische Schnittstelle zur Verfügung gestellt werden, die eine angemessene Unterstützung für weitere konkrete wie simulierte Systeme bietet. Nach kurzer Betrachtung der Alternativen stellt sich die Verwendung einer Hierarchie von objektorientierten Hardware-*proxy*-Klassen als optimale Schnittstelle und Abstraktionsmittel der Wahl dar, um Algorithmen ohne verzichtbaren Bezug auf Protokolle und andere Spezifika der konkreten Sensoren und Effektoren formulieren zu können.

Formulierung komplexer Systemfähigkeiten. Da die Konfiguration und Anordnung vorhandener Sensoren und Effektoren grundsätzlich plattformspezifisch ist, sollten auf mehrere Geräte oder die räumliche Lage bestimmter Geräte bezogene Softwaresystemkomponenten nicht zweckmäßig einer allgemeinen Bibliothek hinzugefügt werden. Dennoch ist die koordinierte Nutzung mehrerer Roboteraggregate bei praktisch allen vorhandenen Lösungsentwürfen erforderlich, von einfachen reaktiven Entwürfen einmal abgesehen. Die Bearbeitung dieser Schwierigkeit wird auf der dritten Ebene vorgenommen: Hier werden Subsysteme formuliert, die jeweils eine solche komplexe Verarbeitungsfähigkeit des Gesamtsystems in möglichst abstrakter und weitgehend von der Hardwareauswahl unabhängiger Weise implementieren sollen. Auf dieser Ebene erweist sich, daß Subsysteme bei Betrachtung ihrer internen Verarbeitung grundsätzlich bestimmten Architekturschemata zugeordnet werden müssen, und sie daher mitunter trotz allen Strebens nach Allgemeinheit nur einen beschränkten Anwendungskontext besitzen.

Dynamische Architekturkomposition. Die Bereitstellung einer Reihe von komplexen Systemfähigkeiten macht grundsätzlich noch keine Lösung eines konkreten Anwendungsproblems aus. Die Struktur, die durch Koordination dieser Systembestandteile ein angemessenes Gesamtverhalten entstehen läßt, ist die Systemarchitektur, und an dieser Stelle besteht ein erheblicher Gestaltungsspielraum.

Aus den Erfahrungen mit der Entwicklung von Roboterarchitekturen (Abschnitt 2.1) wird klar, daß keine effiziente Lösung für die Gesamtheit denkbarer Anwendungsprobleme zu erwarten ist. Aus diesem Grunde wird kein

weiterer konkreter Architekturentwurf vorgeschlagen. Als Ansatz zur Abstraktion von bestimmten Anwendungsproblemklassen wird stattdessen eine Struktur konzipiert, die einen einfachen Aufbau von beliebigen Systemarchitekturen unter Verwendung der Subsysteme gestattet. Zur Demonstration der Vollständigkeit und Eleganz dieses Ansatzes (der sich, wie bei allen Architekturentwürfen üblich, einer sinnvollen quantitativen Analyse entzieht) werden für ein gemeinsames klassisches Problem drei den vorherrschenden Hauptparadigmen des Entwurfs entsprechende Prototypen vorgestellt und diskutiert.

Der Beitrag auf dieser Gestaltungsebene geht erheblich über die Möglichkeit zum Aufbau von Architekturschemata unter Verwendung eines einheitlichen strukturbildenden Mittels hinaus. Die Organisation der Subsysteme zu einem System entsprechend einer Architektur wird von einer besonderen, unveränderlichen Systemkomponente – einer Metaarchitektur – besonders unterstützt. Diese Komponente sorgt für eine angemessene Trennung der Subsysteme, damit diese von der expliziten Kommunikation abgesehen möglicht voneinander entkoppelt ablaufen können, und so keine nicht zweckdienliche gegenseitige Beeinflussung entsteht. Die internen von der Metaarchitektur verwalteten Kommunikationswege gestatten das transparente nachträgliche Einfügen, das Herauslösen und das Austauschen von Subsystemen, sowie das nachträgliche Überladen von Subsystemausgaben. Diese Eigenschaften gestatten umfassende nachträgliche funktionale und strukturelle Änderungen an einer Architekturkonfiguration, ohne daß Eingriffe in die Subsysteme (oder ihren Code) vorgenommen werden müssen. Auf diese Weise werden nicht nur Austauschbarkeit, Wartbarkeit und Testbarkeit, sondern auch die Wiederverwendbarkeit von Subsystemen unterstützt; überdies wird die Adaptierbarkeit von Architekturentwürfen erheblich vereinfacht. Der Nutzen für Wiederverwendung und Erweiterung wird in auf natürliche Weise bei dem Entwurf der drei – an sich konzeptionell völlig verschiedenen – prototypischen Lösungen (4.4.4) deutlich.

Publikationen im Kontext. Einige spezielle Verfahren, die in dieser Arbeit Verwendung finden, wurden separat veröffentlicht; so in [BBBM01] die Erfahrungen mit der generischen Anbindung an simulierte und physische Roboter (als Koautor; die Darstellung geschieht aus Sicht der Simulatorentwickler), in [BM00] die Entwicklung einer Kombination aufgabenspezifischer Bildverarbeitung mit deliberativer Systemkontrolle, in [BM01] die Integration eines LRF in dieses System mit einer Behandlung gewisser Probleme der Sensorfusion, in [BM02] die Lösung eines Abgleichproblems zwischen Kinematik und Sensorik zur Vereinfachung der Sensorfusion zwi-

schen Kameras und Laserentfernungsmessung, in [BKM03b] die Optimierung der Selbstlokalisierungsfähigkeiten durch Nutzung weiterer Sensoren und in [BKM03a] schließlich neue Ansätze zur inkrementellen Selbstlokalisierung und Kartierung. Diese Verfahren treten in Abhängigkeit von ihrer Funktion innerhalb der Systemarchitektur auf Komponenten- bzw. Subsystemebene auf.

Unter Benutzung des vorgestellten Systems, seiner Vorläufer oder seiner Komponenten sind am Fachbereich Informatik der Universität Hamburg die Arbeiten [BB01, Küp01, BSK02, GS03, KM03, Küp03, SB03, Wil03] entstanden, an der Universität der Bundeswehr Hamburg die Arbeiten [Fis03, Hol03, San03]. Allen Verwendern sei an dieser Stelle noch einmal herzlich für den Einsatz bei der Erprobung und die ausgezeichnete Zusammenarbeit gedankt.

Zusammenfassung. Kern dieser Arbeit ist die Vorstellung eines neuartigen Entwurfs zur Konstruktion von Roboterarchitekturen

- als Beitrag zur Diskussion der Möglichkeit und Notwendigkeit des Paradigmenwechsels vom problemgetriebenen Konstruktionsansatz zur problemunspezifischen Roboterkonstruktion,
- zur orthogonalen Subsumption der wesentlichen klassischen Ansätze unter Nutzung der errungenen Fortschritte,
- zur Beibehaltung des vormaligen Architekturgedankens pro Subsystem, doch bei gleichzeitiger Generizität der Architektur,
- zur Einführung neuartiger Eigenschaften wie dynamischer Rekonfigurierbarkeit auf Architekturebene, Interoperabilität und Benchmark durch die neu geschaffene Option zum Modulaustausch.

Kapitel 2

Komponenten und Ansätze der Konstruktion mobiler Roboter

> *There has always been some scepticism in the community that there is any generality to be derived from using software architectures with robots (...).*
> R. PETER BONASSO, [KBM98]

Dieses Kapitel behandelt die Komponenten von Robotersystemen und den Stand der diesbezüglichen Forschung. Entsprechend der Zielsetzung dieser Arbeit steht die Art der durch die Wahl der Architektur vermittelten "Kontrolle" an prominentester Stelle und wird in Abschnitt 2.1 entsprechend differenziert untersucht. Eine Analyse der Implikationen der spezifischen Schwächen, aber auch jeweiligen Leistungen folgt in Kapitel 3.

Da im Gegensatz zu simulierten Systemen – also solchen, die allein durch Softwarekomponenten realisiert sind – kein Roboter sich nur aus "Architektur" konstituieren kann, muß auch auf die anderen Komponenten eingegangen werden: mindestens im Sinne eines "*bottom up*"-Einflusses ist von den sensor- bzw. effektorseitig eingesetzten Verfahren eine erhebliche Einschränkung des Gestaltungsspielraums der umschließenden Systemarchitektur zu erwarten. Abschnitt 2.2 befaßt sich in Berücksichtigung dieses Sachverhalts mit der "Schnittstelle zur physischen Welt", die die Robotik so interessant, aber auch anspruchsvoll macht. In Abschnitt 2.2.1 wird eine entsprechende Beziehung zwischen dem Kontrollmodell und der Behandlung von Sensordaten hergestellt.

Andere Verbindungen wie die zwischen Kontrollmodell und der Art der Integration des "Lernens" – deliberativ-symbolisch vs. reaktiv-subsymbolisch

– könnten auf vergleichbare Weise untersucht werden. Dies liegt allerdings jenseits des von dieser Arbeit abgedeckten Bereichs. Einmal davon abgesehen, daß die Integration des Lernens grundsätzlich wünschenswert sein mag, erscheint es im Gegensatz zur Handhabung von Sensoren und Effektoren nur als optionaler Systembestandteil. Speziell die Verarbeitung und Interpretation sensorischer Daten wird in der Regel als Basis für robotisches Lernen angenommen (obgleich mitunter genau der umgekehrten Argumentation der Vorzug gegeben wird) und damit hier als notwendige Grundlage behandelt.

Im hier gegebenen Fall der Durchführung einer Untersuchung zu *mobilen* Robotern dient die Schnittstelle zur Außenwelt in Abschnitt 2.2 als exemplarisch zu untersuchender Fall, da sie Anforderungen an die sensor- wie effektorbezogene Steuerung in sich vereinigt.

2.1 Architektur und Kontrolle

In diesem Abschnitt soll eine Übersicht über die in der einschlägigen Literatur behandelten Lösungen für Architekturen und ihrer Varianten für die Kontrolle von Robotersystemen gegeben werden. In den einleitenden Bemerkungen (2.1.1) findet nach ersten begrifflichen Vereinbarungen eine Motivation des Ordnungsschemas statt, das bei der Vorstellung des Stands der Forschung in den Abschnitten 2.1.2–2.1.4 Verwendung findet.

2.1.1 Vorbemerkungen und Begriffsbestimmungen

Funktionale Dekomposition des Systems "Roboter". Eine allgemeine Zerlegung in strukturelle Komponenten kann unter verschiedenen Gesichtspunkten und mit ganz verschiedenem Detaillierungsgrad erfolgen. Auf einer abstrakten Ebene liegt eine funktionale Dekomposition in ein Sensor-, ein Kontroll- und ein Effektorenteilsystem intuitiv nahe:

> *"Programming a robot involves integrating many functions, including perception of the world around it, formulation of plans of action, and monitoring of the execution of these plans."* – [Nil82, Kap. 7, S. 275]

Diese von NILSSON vorgetragene Behauptung – inbesondere seine Komponentenwahl – impliziert einige besondere Bedingungen, die durchaus

nicht in allen Fällen gegeben sind; in zahlreichen Fällen können sogar Gegenbeispiele angegeben werden (darauf wird in 2.1.3 bei der Betrachtung von Architekturen des sog. reaktiven Typs zurückzukommen sein). In der Tat sind beinahe alle Komponenten unter gewissen Umständen verzichtbar oder können in stark abgewandelter Form zum Einsatz kommen – die einzige Ausnahme stellt die *Kontrolle* dar. Es sei also unterstellt, daß jedes sinnreich und hinreichend komplex in Abhängigkeit von den Umweltbedingungen agierende Artefakt die Integration irgendeiner Form von "Kontrolle" erfordert, sei sie (wie in NILSSONs Aussage) explizit oder implizit realisiert.

In aller Regel wird die technische Umsetzung der Kontrollkomponente durch Software vorgenommen (auch hier gibt es Ausnahmen, wie die von WALTER entwickelten Roboter zeigen [Wal50]). Dies fällt somit in die Domäne der Informatik.

Bestimmung der "Kontrolle". Da ein Roboter *kausale* und *situationsbezogene* Aktionen durchführen soll, ist eine wie auch immer geartete Vermittlung zwischen Sensoreingabe und Effektoraktion (Ausgabe) erforderlich. Besteht die Anforderung weitergehend darin, auch nicht ausschließlich auf die *aktuell erfaßbare* Situation bezogene Aktionen durchführen zu müssen, oder einen bestimmten temporalen Ablauf (wie eine Kette von Einzelaktionen) ohne Einwirkung von außen kommender Stimuli vorzunehmen, so wird die Integration eines internen "Zustands" innerhalb der vermittelnden Instanz erforderlich sein.

Begrifflich ist unter der "Kontrolle" genau die Instanz zu verstehen, die unter Berücksichtigung von Sensorinformationen eine Wirkung auf die Effektoren vermittelt, und somit ein ggf. auch zeitliches "Verhalten" des Gesamtsystems herstellt.

So klar also ist, daß ein autonomes System über eine solche Kontrolle verfügen muß, so wenig zwingend ist die spezifische Art ihrer Umsetzung gegeben.

Freiheitsgrade der Umsetzung. Die oben angeführte Definition des Begriffs "Kontrolle" ist bewußt unscharf gehalten, da eine Reihe von wichtigen Freiheitsgraden bestehen (was im Folgenden den Ansatz zur Entwicklung einer Taxonomie liefern wird): Die Kontrolle kann zentral oder verteilt auftreten, oder solchermaßen in der Gesamtstruktur aufgehen, daß nachträglich kein bestimmter Ort der Implementierung mehr bezeichnet werden

kann. Der Modus der Kontrollausübung könnte steuernd oder regelnd sein oder in Ausführung eines symbolischen Plans bestehen, oder (zweckmäßigerweise) Aspekte aller Varianten aufweisen.

Im Kontext von Roboterarchitekturen hat die Wahl eines "angemessenen" Kontrollmodells – neben in der Regel nicht verrückbaren Leistungsgrenzen durch die Schnittstelle zur Außenwelt, wie sie durch die Leistung der Sensoren und Effektoren bestimmt werden – maßgeblichen Einfluß auf die überhaupt zu erreichende Performanz des Gesamtsystems. Die Berücksichtigung der problemspezifisch betrachteten Angemessenheit führt also auf natürliche Weise zu ganz unterschiedlichen Kontrollmodellen.

Modelle der Kontrolle. Um zu verschiedenen Kontrollmodellen zu gelangen, ist es am einfachsten, die Suche bei (möglichst verschiedenen) Anforderungen an Systeme zu beginnen, und dann solche Systeme zu analysieren, die diesen Anforderungen gerecht werden.

Eine der am einfachsten zu stellenden, wenn auch im allgemeinen Fall schwierig umzusetzenden Anforderungen an Roboter ist die der angemessenen Reaktivität. Ein Roboter soll auf bestimmte Umweltsituationen, evtl. nur einfache Reize, nach möglichst geringer bauartbedingter Verzögerung mit einer ggf. vorab bestimmten Aktion (re-)agieren. Die erste prototypische Umsetzung eines Systems in [Wal50] weist überhaupt nur diese Fähigkeit auf. Ein klassisches Beispiel für eine Anforderung dieses Typs ist die Kollisionsvermeidung: unterschreitet ein anderer Körper eine kritische Distanz zur Plattform, wird die Bewegung des Roboters gestoppt; eine andere Art der Lösung berechnet die zulässige Geschwindigkeit als Funktion der kritischen Distanz. Dies setzt eine möglichst direkte Kopplung von – im weitesten Sinne – Perzeption und Aktion, in der Praxis: Sensoren und Effektoren voraus. Ob eine bestimmte Aktion nur (parameterfrei) "ausgelöst" werden soll oder als Funktion der wahrgenommenen wertkontinuierlichen Reize betrachtet wird, ist für eine Entscheidung zugunsten eines angemessenen Modells praktisch irrelevant: entscheidend ist eine zügige Transformation der Eingabedaten in geeignete Steuersignale der Effektoren. Dies impliziert insbesondere, daß keine zeitlich ausgedehnten Berechnungen unter Berücksichtigung einer gespeicherten Vorgeschichte oder einer geplanten Zukunft vorgenommen werden dürfen. Das *reaktive* Kontrollmodell entspricht genau dieser Beschreibung.

Eine andersgeartete Anforderung betrifft die Durchführung deutlich komplexerer Aufgaben, bei denen die einer Situation jeweils angemessene Aktion nicht oder nicht zweckmäßig allein als Funktion der aktuellen Sensorda-

ten zu ermitteln ist. Dies ist generell der Fall in (bei subsymbolischer Diktion) entsprechend anders konditionierten Problemräumen, bei denen ein Gradientenaufstieg nicht zum gewünschten Ziel, sondern möglicherweise zu lokalen Maxima führt, und (symbolisch) bei Problemen, bei denen erst eine beharrliche – d.h. vom unmittelbaren Sensoreindruck entkoppelte! – Verfolgung eines mehrschrittigen Plans über für sich genommen wenig attraktive Zwischenziele zum eigentlichen Ziel führt. Als typische Vertreter dieser Problemklasse erscheinen alle Explorations- und Navigationsprobleme, wie die Kartierung einer unbekannten Umwelt, die Bewegung vom aktuellen Standort zu einem gegebenen Ziel bei bekannter Umgebungskarte, oder die Durchführung zugbasierter Spiele (vgl. [Bit00]). In derartigen Fällen ist eine Berücksichtigung der Vorgeschichte und/oder der geplanten Zukunft erforderlich, um den entsprechend einer angenommenen Kosten- und Nutzenfunktion mutmaßlich geeignetesten nächsten "Schritt" zu *wählen*. Aufgrund des durch die Planung bedingten vergleichsweise hohen Rechenaufwands und die oftmals diskretisierte Planung liefert dieses Kontrollmodell wiederum zeitdiskrete Steuerinformation. Das *deliberative* Modell leistet genau dies und erscheint solchen Problemen angemessen.

Architektur als Option für Kontrollintegration. Die Festlegung auf ein ganz bestimmtes Kontrollmodell für die Bearbeitung von Problemen aus solch verschiedenen Kategorien (wie sie für eine unspezialisierte Maschine allerdings wenigstens prinzipiell möglich sein sollte, vgl. 3.2) führt unter geeigneten Umständen überraschend weit, hat aber deutliche Grenzen [Nol02]. Eine Integration mindestens der beiden genannten in ihren Anforderungen extremalen Modelle und (vor allem) ihre Abstimmung ist erforderlich:

> *[We] "must reconcile hard real-time systems with systems which cannot meet real-time deadlines"* – [Jones93]

Dies ist die wesentliche Funktion der Architektur. Die Begriffe "Architektur" und "Kontrolle" scheinen untrennbar miteinander verknüpft und sind nur unscharf gegeneinander abgegrenzt. Begrifflich soll hier vereinbart werden, daß die "Architektur" nicht nur die jeweilige modulare Struktur eines Entwurfs bezeichnen soll, sondern allgemeiner für die Art der Vernetzung von Funktionseinheiten einer Kommunikations- und somit Kontrollstruktur stehen soll. Als Funktionseinheiten in der äußersten Vergröberungsstufe sind dabei die Sensorschnittstelle, die explizite oder implizite Ablaufsteuerung und die Effektorenschnittstelle aufzuführen.

Die (Software-)*Architektur* ist also das Mittel zur Integration der Kontrolle (und der anderen Elemente, der Systemkomponenten aus Sensoren bzw. Effektoren mit den jeweiligen Schnittstellen) in das Gesamtsystem. Mit der abstrakten Architektur korrespondiert ein technisches Rahmenwerk, das eine dem jeweils gewählten Kontrollmodell entsprechende Knüpfung von Verbindungen zwischen den Systemkomponenten gestattet.

Anforderungen an Architekturen. Nach der o.a. Aussage von JONES sind einige allgemeine Anforderungen an Systemarchitekturen für Roboter indirekt ableitbar. Grundsätzlich werden an das Gesamtsystem folgende Anforderungen gestellt, was sich in Anforderungen an die die Architektur entsprechend niederschlägt:

1. Das Gesamtsystem soll die Fähigkeit zur "Erfüllung der Aufgabe" besitzen, die die außerhalb des Roboters liegende physikalische Welt betrifft.

   Das Spektrum der Ausgestaltung dieser Fähigkeit kann von einem einfachen reaktiven "Verhalten"[1] bis zur symbolischen (deliberativen) Planung während der Laufzeit reichen: Substrukturen beider Varietäten können als in die Architektur zu integrierende Elemente auftreten.

2. Zu diesem Zweck wird eine Kontrolle zeitlich ausgedehnter Prozesse nötig sein.

   Dies kann (vgl. BONASSO in [KBM98, S. 193f]) auf zwei im Abstraktionsniveau verschiedenen Ebenen erforderlich werden, nämlich

   - als Regelung von *kontinuierlichen* Größen, wie z.B. Geschwindigkeit und Beschleunigung der Plattform,
   - als Kontrolle zeit- oder wert*diskreter* Prozesse, wie sie beispielsweise in der strategischen Planung auftreten.

   Diese Zweiteilung korrespondiert wiederum mit der von JONES festgestellten Notwendigkeit der Integration reaktiver bzw. deliberativer Systemkomponenten.

[1] Je nach Reichweite der jeweiligen Definition des Begriffs "Verhalten" müßte unter Umständen sogar nur von einer *Simulation* von Verhalten durch z.B. Gradientenabstieg gesprochen werden. Der Begriff des Verhaltens (*behavior*) wird hier und im Folgenden, wie in der Robotik üblich, synonym zur programmierten, umwelt- bzw. plangemäßen Reaktion verwendet.

3. Eine interne Kommunikation der Komponenten wird auch im einfachsten Fall erforderlich.

   Als unter Umständen für das jeweilige Kontrollmodell typische Optionen der Verknüpfung im Datenfluß können (u.a.) *m*-zu-*n*-Kommunikation mit interner Deliberation, die Summation (vektorielle Addition oder Mittelwertbildung) von Ausgabekanälen, die Gewichtung von Ausgabekanälen, die Selektion von Ausgabekanälen (durch z.B. einen *arbiter*), die Inhibition von Ausgabekanälen, und die "Aktivierung" und die Inhibition von vollständigen Architekturkomponenten auftreten. Viele der in den folgenden Abschnitten vorgestellten Architekturen verwenden nur eines dieser Elemente, andere eine größere Teilmenge. Prinzipiell sollten in einer allgemeinen Architektur im Sinne der Erweiterbarkeit des Aufgabenbereichs des Systems (die sog. *taskability*) möglichst viele Optionen offen bleiben. Die Frage, welche minimale Menge von Verknüpfungen funktionale Vollständigkeit sichert, ist zunächst nicht relevant.

4. Aus der bereits zu 1. aufgeführten Anforderung entstehen möglicherweise zusätzliche Anforderungen durch den Umgang mit einem fehlerbehafteten *world interface*, das unter sensorseitiger Unschärfe wie verschiedenen effektorseitigen Problemen leidet.

Taxonomien für Architekturen. Zur Einordnung vorhandener Roboterarchitekturen kann eine Reihe von Kriterien herangezogen werden (vgl. auch [Ros97, S. 31]), doch größte beschreibende Kraft und demzufolge Verbreitung hat ein vergleichsweise einfaches Modell erlangt, das lediglich die prinzipielle Anordnung der bereits identifizierten Hauptkomponenten, d.h. von Sensor-, Kontroll- und Planungssubsystemen und ihrer Kommunikationsstruktur beschreibt. Den Veröffentlichungen zu diesem Thema entsprechend werden die Teilsysteme als "*sense*", "*plan*" bzw. "*act*" bezeichnet.

Nach heutigem Verständnis muß allerdings darauf hingewiesen werden, daß die Komponente *"plan"* durchaus irreführend benannt ist, da nicht in allen Architekturvarianten das geschieht, was gemeinhin als begrifflich mit KI assoziierter expliziter "Planung" bezeichnet wird – der Begriff *control* träfe den Funktionsaspekt präziser; und daß die Komponente *"act"* nicht genau das tut, was handlungstheoretisch als "Handlung" bezeichnet wird (weshalb im Deutschen oft nur von "Effektoren" die Rede ist). Zur Erhaltung der Kompatibilität mit der Mehrheit der Veröffentlichungen auf dem Gebiet der Robotik werden die eingeführten Begriffe hier beibehalten, obgleich eine Reihe von Alternativen zur Verfügung steht; so spricht beispiels-

weise NILSSON aus Sicht der KI von *perceptual processing* bzw. der *action function*, vgl. [Nil98, S. 23].

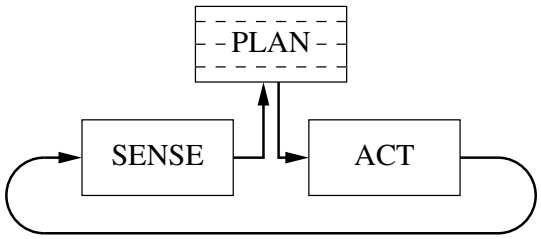
Eine einfache Taxonomie, das Kontrollmodell berücksichtigend. Zu den verschiedenen Kontrollmodellen werden entsprechend der unterschiedlichen *Anordnung* der Module *sense*, *plan* und *act* drei Basis-"Architekturen" (auch die zugehörigen "Entwurfsparadigmen") unterschieden. Stark vereinfacht können Robotersysteme als einer der Klassen zugehörig betrachtet werden:

1. deliberative Systeme (vgl. 2.1.2),
2. reaktive Systeme (vgl. 2.1.3) und
3. hybride Systeme (vgl. 2.1.4).

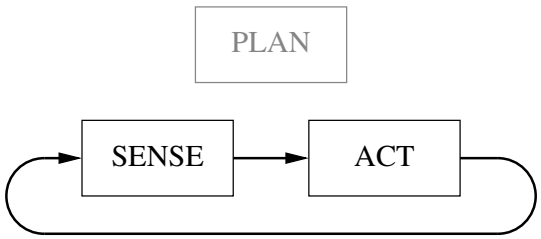
Die oft als grundsätzlich verschieden dargestellten Paradigmen unterscheiden sich – einmal abgesehen von den präferierten Methoden der systeminternen Kommunikation – im Wesentlichen in der jeweiligen Modalität der Kopplung von Perzeption und Aktion [Nei74], d.h. in der Art, wie Kontrolle in der Architektur lokalisiert ist (Abb. 2.1.1).

Eine auf den ersten Blick ganz ähnliche Unterteilung wird in [Mur00] vorgeschlagen. Ein Mangel dieser Darstellung für den hier beabsichtigten Zweck ist jedoch, daß uneinheitlich statt des deliberativen Modells auch vom "hierarchischen" Kontrollparadigma gesprochen wird. Da allerdings, wie in 2.1.2.3 dargelegt werden wird, das symbolisch-hierarchische als Spezialfall des deliberativen Modells auftritt, hingegen deliberative Systeme nicht notwendig eine interne Hierarchie aufweisen müssen, und zudem hybride Systeme, die Züge der ersten beiden genannten Modelle tragen, auch in [Mur00] als "deliberativ/reaktiv" bezeichnet werden, ist es für die Aufstellung einer Typtaxonomie günstiger, nicht von einem (wenn auch verbreiteten) Spezialfall auszugehen, und grundsätzlich vom deliberativen Modell zu sprechen, um ggf. auf die hierarchische Ausprägung als Untertyp hinzuweisen.

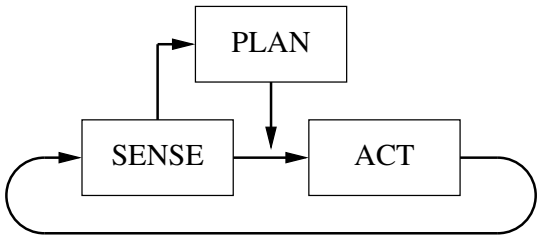
In den folgenden Abschnitten 2.1.2 bis 2.1.4 werden diese drei Klassen von Ansätzen mit ihren charakteristischen Eigenschaften und ihren wesentlichen Untertypen (Abb. 2.2) vorgestellt werden. Zur Diskussion der das Verhalten des Roboters beschreibenden Funktionen seien eingeführt $\mathbf{i}_t$ für den Eingabevektor (Sensordaten), $\mathbf{o}_t$ für den Ausgabevektor (Effektorsteuerung) und $\mathbf{s}_t$ für den Zustandsvektor (*state*) jeweils zum diskreten Zeitpunkt



(a) Schema deliberativer Systeme



(b) Schema reaktiver Systeme



(c) Schema hybrider Systeme

Abb. 2.1: Schemata der Basissystemarchitekturen (nach [Mur00], ergänzt). Deliberative Systeme (a) führen sich wiederholende Perzeptions-Aktions-Zyklen mit eingeschobener Planungsphase (oft in einem geschichteten Planer) aus; reaktive Systeme (b) sind gedächtnislos, die Planungsphase ist zur Zeit der Systementwicklung abgeschlossen; hybride Systeme (c) parametrisieren ihren reaktiven Teil durch Planungsdaten.

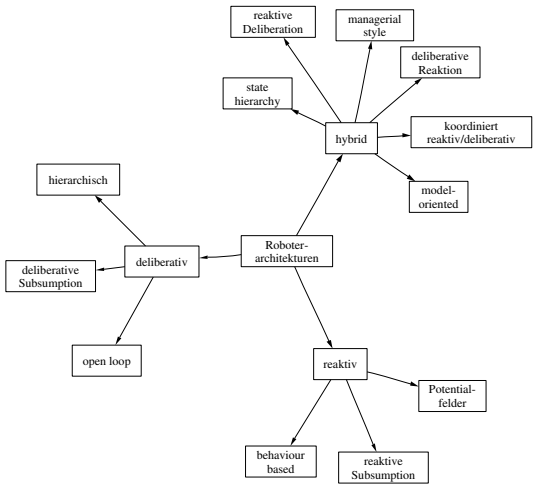


Abb. 2.2: Taxonomie der Architekturtypen: Artunterscheidung nach dem Kontrollparadigma, Gattungsunterscheidung nach weiteren Spezifika.

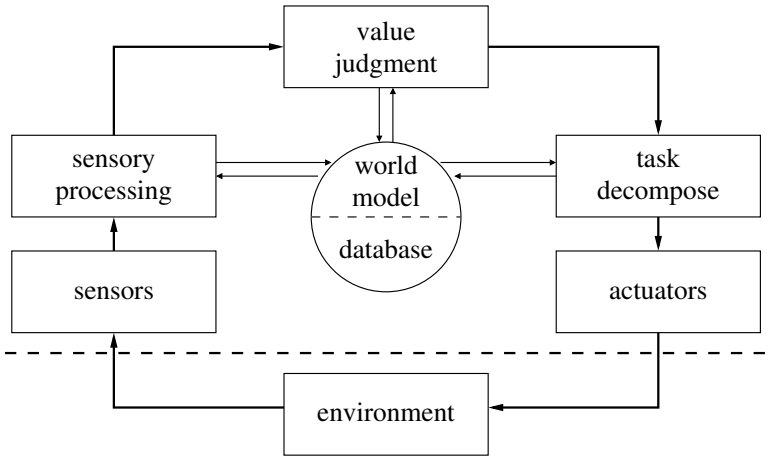
t. Eine Beschränkung auf diskrete t erscheint statthaft, da in der Regel Standardrechner mit getakteter Verarbeitung eingesetzt werden.

2.1.2 Das deliberative Modell

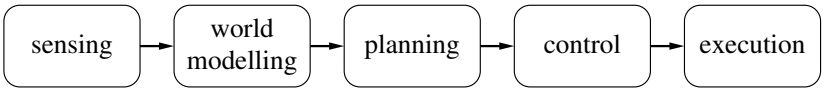
Als *deliberativ* sollen hier solche Systeme bezeichnet werden, die ihre Reaktion auf gemessene Umweltparameter auf Basis einer programmgesteuerten "Wahl" zwischen zwei oder mehr Alternativen treffen können. Diese Eigenschaft ist, da eine Betrachtung des Modells von der Konstruktionsseite ausgehend erfolgt ("*white box*") leicht entscheidbar.

Mit der Möglichkeit einer tatsächlich freien Wahl im Sinne des Wortes korrespondiert die Erfordernis, daß die Reaktion des Systems auf gleiche Stimuli unterschiedlich ausfallen kann. Hierfür sollen allerdings keine Nichtdeterminismen bestimmend sein,[2] sondern die Einbeziehung eines stets implizit angenommenen internen *Zustands* – die Sensoreingabe wirkt also nicht direkt auf die Effektoransteuerung, sondern durchläuft ein Teilsystem, das aus Sensoreingabe *und* internem Zustand die Systemreaktion berechnet.

[2]Systematisch und in Bezug auf die Roboterarchitektur wäre eine Quelle für echte Zufallszahlen als externe Entität zu betrachten, deren Ablesung zum Stimulusvektor beiträgt.



(a) nach [Alb89]



(b) nach [KBM98]

Abb. 2.3: Schemata deliberativer Systemarchitekturen (a) nach ALBUS und (b) nach KORTENKAMP et al.

2.1.2.1 Allgemeines Schema

Die Basisstruktur (vgl. Abb. 2.1(a)) wird in der Literatur in verschiedenen Stufen der Verfeinerung dargestellt; Abb. 2.3 zeigt zwei exemplarische Diagramme. Das Schema von ALBUS bezieht die Außenwelt, die nicht Komponente des Roboters ist, explizit ein und bildet so eine Rückkopplungsschleife (was nicht regelungstechnisch zu interpretieren ist, da üblicherweise auch vom Systementwickler keine Angaben über die Totzeiten der Glieder gemacht werden können). Dieser Aspekt der Darstellung ist eher untypisch und wird eher bei reaktiven Systemen verwendet, sofern sie als "situierte Agenten" interpretiert werden.

Steuerung der Abfolge. Eine grundsätzliche Gemeinsamkeit ist die Abfolge der Verarbeitung in den Blöcken *sense*, *plan* und *act*: die sensorische Eingabe wirkt in keinem Fall direkt auf die Effektoren. Generell wird ein

sequentiell-zyklisch organisiertes Systemverhalten nach dem Schema

$$\text{sense} \rightarrow \text{plan} \rightarrow \text{act} \tag{2.1}$$

angenommen:

1. zunächst werden Sensordaten abgerufen und entsprechend der Sensormodalität (vor-)verarbeitet,
2. auf Basis dieser Daten und des Systemzustands findet eine Planung der Aktion statt.
   $$\mathbf{o}_{t+1} = \mathbf{f}(\mathbf{m}_t, \mathbf{i}_t) \tag{2.2}$$
   In diesem Zuge kann der Systemzustand verändert werden.
   $$\mathbf{m}_{t+1} = \mathbf{u}(\mathbf{m}_t, \mathbf{i}_t) \tag{2.3}$$
3. Die aus dem Planungsprozeß resultierenden Anweisungen $\mathbf{o}_{t+1}$ werden unmittelbar an die Effektoren weitergeleitet.

Die (Vor-)Verarbeitung der Sensordaten soll dabei unter Anwendung von mehr oder weniger ausgeklügelten Interpretationsstrategien ein abstraktes und symbolisches Bild der Umwelt liefern, das in ein internes Modell der Außenwelt, internes (IMA, [Ste65, S. 245]) eingeht. Eine interne Karte wäre als Modell der Umwelt eine solche symbolische Interpretation, selbst wenn sich der Symbolgehalt auf die Entscheidung *belegt/nicht belegt* beschränken sollte.

Elemente dieses Modells sind in der Regel über mindestens einige Zyklen der Ausführung persistent und liefern die Grundlage für das Planungsverfahren. Aus rein pragmatischen Gründen (Rechenzeit, Aufwand) wird in der Regel auch die Planung nicht in jedem Zyklus vollständig neu durchgeführt, sondern auch Planinformation (Grad der Komplettierung, Abbruchbedingung für den Teilschritt) im Zustand $\mathbf{m}$ gespeichert. Was die Effektoraktion als "Ausgabe" angeht, wird von einer exakten Durchführung zunächst einmal ausgegangen.

Speicher, Zustand und Repräsentation. Die Existenz eines internen Zustands setzt die Integration eines *Speichers* voraus. Die Präsenz von Speicher wird oft als wichtigstes Unterscheidungsmerkmal zu den reaktiven Modellen herangezogen, ist streng genommen aber nicht genügend, wenn er nicht auch tatsächlich an wenigstens einer Stelle zum Zwecke der Deliberation

verwendet wird (einige reaktive Modelle besitzen über die in ihnen auftretenden AFSMs Speicher und implizite lokale Zustandsinformation).

Ein wesentliches Merkmal deliberativer Systeme ist die Benutzung eines internen Modells der Außenwelt, das ebenfalls als Systemzustand in Erscheinung tritt. Das Weltmodell, das im einfachsten Fall nicht mehr als eine Aufbereitung von historischen wie aktuellen Sensordaten zusammen mit statischen Interpretationsregeln sein muß, erfüllt drei Funktionen:

1. es leitet die Bewertung der aktuellen Sensordaten (*value judgment*), um deliberativ der angenommenen Situation und dem "Wissen" angemessen erscheinende Schritte zur Lösung einer (u.U. impliziten, fest codierten) Aufgabe zu wählen,

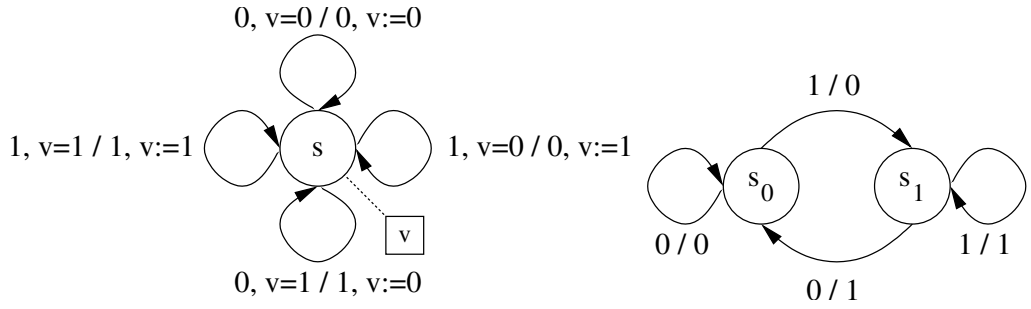
2. es ist an der Planung von auszuführenden Schritten beteiligt (*task decomposition*),

3. es wirkt optional auf die Auswertung von Sensordaten (*sensory processing*) – beispielsweise, um eine Adaption (Rauschen, Beleuchtungsverhältnisse, im Modell erfaßte Kalibrierungsdaten) vorzunehmen (sofern im Block *sense* Vorverarbeitung vorgenommen wird, ist also zusätzlich eine Kommunikation "*sense*←*plan*", d.h. in Rückrichtung vozusehen).

Dem Sensor ist keine nicht modellgetriebene Eigenaktivität gestattet (*active perception*, Aufmerksamkeit).

Für KORTENKAMP und MEYSTEL ist die Modellierung der Welt offenbar eine die Auswertung von Sensordaten betreffende Angelegenheit; statt des direkt die Effektoren steuernden *task decomposition*-Moduls von ALBUS wird hier ein *control*-Modul vorgesehen. Inwieweit dieses eine (im Prinzip sinnvolle) lokale Rückkopplung ermöglichen soll, wird nicht spezifiziert. Beide Varianten können dennoch als Verfeinerung des bereits in Abb. 2.1.1 dargestellten Schemas einer linearen, im einfachsten Fall sogar sequentiellen Verarbeitung von *sense*, *plan* und *act* angesehen werden; wobei ALBUS das *plan*-Subsystem verfeinert, während BONASSO die Schnittstellen *sense/plan* und *plan/act* näher spezifiziert.

Integration der Deliberation. Nicht in allen Fällen ist die Verwendung eines Weltmodells direkt nachzuweisen, da es in einfachen Fällen implizit in der Wahl der diskreten Zustände des Systems codiert sein kann. Über die vom System ausgeführte Aktion wird also abhängig davon, ob ein Reiz



(a) explizites Weltmodell (b) implizites Weltmodell

Abb. 2.4: Explizite und implizite Weltmodelle, Beispiel für eine AFSM mit (a) einer Variablen und (b) eine äquivalente FSM: die Ausgabe ist nicht direkt an die Eingabe gekoppelt; die Eingabe des letzten Zeittakts wird ausgegeben. Deliberation findet in Bezug auf die Ausgabe bzw. die Zustandstransition statt. Das System ist deliberativ, da der Systemzustand in die Ausgabe eingeht.

präsent ist oder nicht, an "verschiedenen Stellen im Code" entschieden (s. Abb. 2.4). Die Deliberation kann also sowohl als konditionale Zustandstransition ohne Pflege eines explizites Weltmodells wie als konditionale Ausgabe unter Beibehaltung des aktuellen Zustands bestehen; nur im letzteren Fall wird tatsächlich ein explizites Weltmodell verändert.

Das grundsätzliche Konzept deliberativer Systeme läßt eine Reihe von Optionen für die Realisierung des *plan*-Subsystems, die in 2.1.2.2–2.1.2.4 behandelt werden.

2.1.2.2 Der *open loop*-Typ

Schema. Die *open loop*-Kontrolle ist ein direktes Verfahren zur Steuerung eines Roboters, bei dem sämtliche Aktion durch einen einfachen monolithischen Planer veranlaßt wird und folgende Randbedingungen als vereinbart gelten:

1. sensorseitig ist nicht mit wesentlichen Meß- oder Verarbeitungsfehlern zu rechnen,

   dies könnte durch Verwendung eines kalibrierten Systems in bekannter Umwelt (CWA) angenommen werden,

2. das Planungssystem arbeitet korrekt und auf einem (wegen Bedingung zu 1.) fehlerfreien Weltmodell,
3. effektorseitig werden alle geplanten Aktionen mit für Erfüllung der Aufgabe hinreichender Präzision ausgeführt,

   dies ist bei Aufgaben, die keine große Genauigkeit erfordern, unter Umständen gewährleistet.

Im Ablauf wird mindestens ein Zyklus der sequentiell auszuführenden Aktivitäten *sense/plan/act* stattfinden, wobei das System in den letzten beiden Teilen des Zyklus ohne Sensoreingabe ("*closed eyes*") vorgeht. Sind die Daten der initialen *sense*-Ausführung in der Vollständigkeit zur Problemlösung genügend – d.h. beispielsweise ein zu suchendes und greifendes Objekt ist nicht verdeckt –, kann unter der Einführung der zusätzlichen Annahme einer statischen Umwelt (den Roboter und die von ihm bewegten Dinge einmal ausgenommen) der gesamte, nach der ersten Durchführung von *plan* vollständige Plan ohne Rückgriff auf weitere Sensordaten bis zum Ziel ausgeführt werden.

Motivation. Unter diesen Annahmen scheint das Problem in der Hauptsache reduziert auf

> *(...) "synthesizing a sequence of robot actions that will (if properly executed) achieve some stated goal, given some initial situation. The action synthesis part of the robot problem can be solved by a production system. The global database is a description of the situation, or state, of the world in which the robot finds itself, and the rules are computations representing the robot's actions."* – [Nil82, Kap. 7, S. 275f]

Damit würde die Robotik, soweit es nicht um die Teilaspekte der Vorverarbeitung von Sensordaten ginge, den Methoden der KI vollständig zugänglich. In der Realität sind die eingangs genannten Annahmen allerdings keineswegs erfüllt.

Randbedingungen. Deliberative Architekturen des Typs *open loop* können unter geeigneten Umständen mit gewissem Erfolg funktionieren. Im wesentlichen treten diese Umstände in genau zwei Fällen auf: zum einen in der *Simulation*, denn hier treten übliche Fehlerquellen – wie ungenaue Sensordaten oder der nicht genügend präzisen Ausführung von Effektorkommandos – nicht auf, zum anderen bei Verwendung vollständig *kalibrierter Systeme*, sofern die erreichte Präzision zur Erfüllung der Aufgabe ausreichend

ist. Spätestens bei mehrschrittigen, komplexeren Plänen neigen jedoch auch solche Systeme zu nicht mehr akzeptablem Verhalten, was durch hinreichend hochfrequentes Durchlaufen der *sense/plan/act*-Zustände mit Neuplanung allerdings in Grenzen verbessert werden kann.

Fazit. Wenige Systeme entsprechen dem *open loop*-Schema. Vorzug dieser Konstruktionsweise ist eine große strukturelle Ähnlichkeit zur gut erforschten klassischen Arbeitsweise in der Informatik, in der perfekte Datenquellen und fehlerfreie Umsetzung von Instruktionen grundsätzlich vorausgesetzt werden dürfen. Da Systeme dieses Architekturtypus in einem gutmütigen Simulationsmodell funktionieren, besteht eine praktische Lösung oft darin, die Ausführungsbedingungen in der realen Welt denen der Simulation nach Möglichkeit anzunähern: Durch geeignete Auslegung und Kalibrierung von Sensoren und Effektoren sowie ggf. Modifikation der Einsatzumgebung kann für bestimmte Aufgaben eine brauchbare Lösung erreicht werden. Als allgemeine Lösung ist dieser Ansatz allerdings wegen der zeitlich ausgedehnten "*closed eyes*"-Intervalle untauglich. Eine gewisse Verbesserung – zum Preis eines komplexeren Planers – stellt die Verwendung hochfrequenter "sense/plan/act"-Zyklen dar, in denen sowohl der (genügende) Erfolg der laufenden Aktion als auch Änderungen der Umwelt überwacht werden können.

Die feinere Granularität der vormals elementaren komplexen Aktionen, die nun zyklisch Überwachung und ggf. sensorisch geregelte Korrekturen enthalten, wird ohne weitere strukturelle Verbesserung zu einem deutlich komplexeren, im Grenzfall nicht mehr praktikablen Planungssubsystem mit zahlreichen sog. *micro-states* führen. Einen Lösungsansatz für dieses Problem versprechen Systeme des hierarchischen Typs zu liefern, die eine entsprechende Partitionierung des Planers unterstützen.

2.1.2.3 Der hierarchischer Typ

Schema. Im deliberativ-*hierarchischen* Architekturtyp werden die Hauptblöcke in der allgemeinen für deliberative Systeme charakteristischen Anordnung *sense/plan/act* belassen. Eine Spezialisierung findet durch den hierarchischen Aufbau der Planungskomponente statt (vgl. Abb. 2.5).

Diese Verfeinerung hat vorrangig das Ziel, eine im Mittel höhere Frequenz des Durchlaufens vollständiger *sense/plan/act*-Zyklen zu erreichen. Rechenzeitintensive "höhere" Planungsverfahren werden hier nur bei Bedarf aktiv.

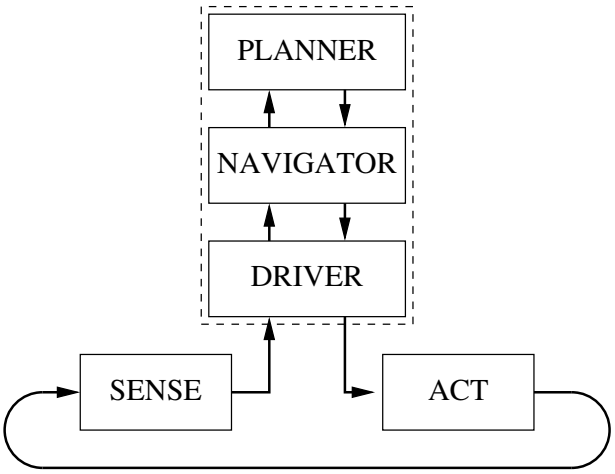


Abb. 2.5: Schema der deliberativ-*hierarchischen* Kontrollstruktur. Die Hierarchieebenen sind aufgabenspezifisch und hier dargestellt für eine Navigationsaufgabe.

Die Häufigkeit der Interaktion und die Menge ausgetauschter Daten nimmt in den höheren Ebenen ab.

Motivation. Zentrale Behauptung ist, daß ein hierarchischer Planer im Mittel eine geringere Zeit zur Verarbeitung der periodisch eintreffenden Sensordaten benötigt als ein monolithischer Planer. Dies gilt, da nur die unteren Schichten tatsächlich regelmäßig aktiviert werden müssen und die oberen Schichten nur bei besonderen Bedingungen – z.B. bei Übergang zum nächsten Teilziel – Rechenzeit für sich beanspruchen. Die durchschnittlich geringere Verzögerung, die nur unter besonderen Bedingungen die des monolithischen Planers erreicht, ermöglicht kürzere Zyklen und kann so die Reaktivität des gesamtem Systems erhöhen.

Als klassische Aufgabe für hierarchische Planer wurde von MEYSTEL das Problem der Navigation eingeführt; der Entwurf des *nested hierarchical controller* (NHC, vgl. Abb. 2.6) stellt einen entsprechenden Lösungsansatz dar. Im NHC folgt die Dekomposition des Planers der Zerlegung des Problems:

1. Der *mission planner* bestimmt das im Rahmen der strategischen Aufgabe zu erreichende (Teil-)Ziel und teilt dies dem *navigator* mit.

2. Das taktische Subsystem *navigator* wählt einen Weg (u.U. unter Nutzung von bekannter Routeninformation) zu diesem Ziel aus.

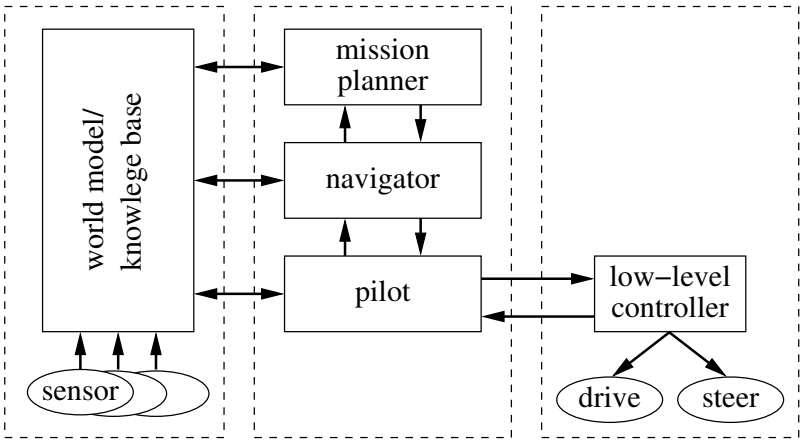


Abb. 2.6: Beispiel einer deliberativ-*hierarchischen* Kontrollstruktur: Implementierung als NH-(*nested hierarchical-*)Controller (MEYSTEL in der Darstellung nach [Mur00]) für den Aspekt der Fortbewegung. Mit zunehmender Hierarchieebene sinkt i.A. das Kommunikationsvolumen; Echtzeitanforderungen bestehen nur für die unteren Schichten.

Seine Kommunikation mit dem *mission planner* beschränkt sich auf die Meldung von Erfolg oder Fehlschlagen der angeforderten Aktion.

3. Als unterste Komponente hat das *pilot*-Subsystem die geeignete Bedienung der Effektoren unter Berücksichtigung sensorischer Daten (Korrektur von effektorseitigen Fehlern, Vermeidung von Kollisionen etc.) zu erledigen.

   An den *navigator* werden wiederum der Erfolg (Abschluß der Fahrt einer Teilstrecke) oder Mißerfolg gemeldet.

Im Mittel fällt so die *closed eyes*-Zeit im Vergleich zu der Verwendung eines monolithischen Planers geringer aus, da das *driver*-Subsystem für sich eine geringere Verzögerung als ein monolithisches Planungssystem aufweist. Ein Nebeneffekt ist, daß durch die höhere Rate der Durchläufe vollständiger *sense/plan/act*-Zyklen eine schnellere Reaktion auf sich verändernde Umgebungsverhältnisse erreicht wird.

Ein Ansatz von ALBUS, das *real-time control system* (RCS), nimmt die zentrale Idee in NHC auf und nutzt Hierarchien auch innerhalb der Subsysteme *sense* und *act*. In RCS sind die Schichten der Planungskomponente horizontal mit entsprechenden Schichten der Sensorik- und Effektorikkom-

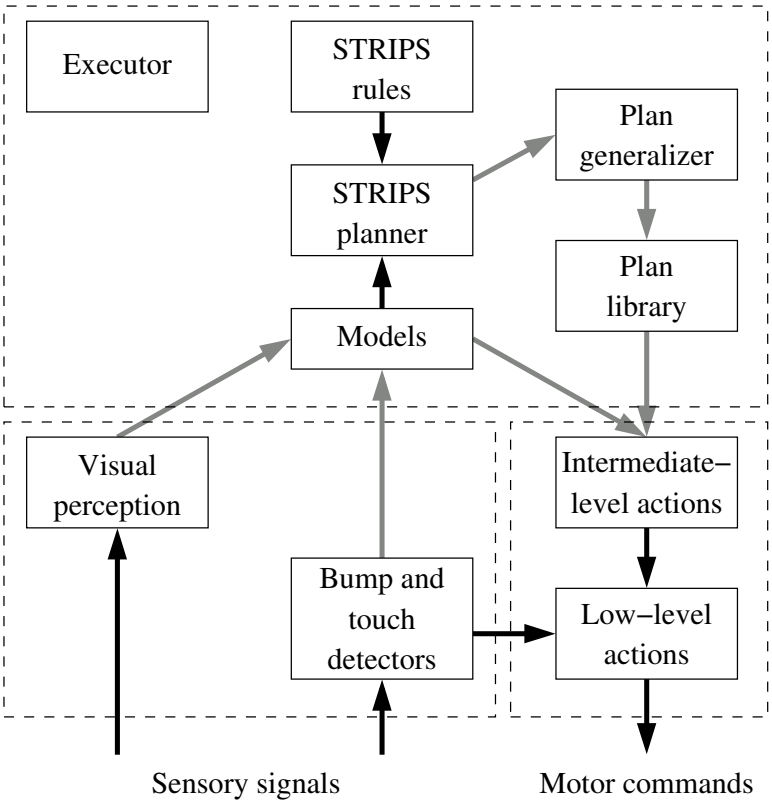
ponenten vernetzt. Dies bewirkt, daß aufwendig zu ermittelnde Merkmale tatsächlich auch nur bei Bedarf berechnet werden brauchen.

Randbedingungen. Eine effiziente Lösung kann genau dann erreicht werden, wenn eine geeignete Partitionierung des zu lösenden Anwendungsproblems gefunden werden kann, die auf eine hierarchische Struktur abzubilden ist. Dies ist in der Regel der Fall, wenn eine Dekomposition der Aufgabe so möglich ist, daß

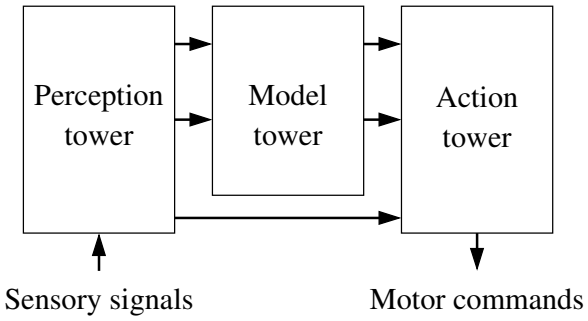
1. der *Regelfall* eine einfache Verarbeitung erfordert, sodaß sie in einer der unteren Schichten des Planers vollständig und schnell bearbeitet werden kann,
2. der *Sonderfall*
   (a) von den unteren Planerschichten zuverlässig und schnell erkannt werden kann, und
   (b) ein Zeitverzug durch die Aktivierung von übergeordneten Planerschichten akzeptabel ist.

Fazit. Wohl der früheste Repräsentant dieser Architekturform ist das am SRI entwickelte System Shakey, dessen Schema in Abb. 2.7(a) dargestellt ist. Das rein deliberative Schema konnte aus pragmatischen Gründen allerdings nicht durchgehalten werden: Die direkte Verbindung von Perzeption und Aktion – bei SHAKEY zur Vermeidung von Schäden an der Plattform im Fall der bereits erfolgten Kollision mit Hindernissen genutzt – nimmt Teile des reaktiven Schemas vorweg.

Das Beispielproblem der "Navigation" zerfällt auf natürliche Weise so, daß es durch die in NHC implementierten Komponenten bearbeitet werden kann: Die hierarchische Variante erlaubt also eine sinnvolle und effiziente Implementierung einer Lösung für das Navigationsproblem, ohne daß der Rahmen deliberativer Systeme verlassen wird. Dieses spezielle Anwendungsproblem ist wichtig, doch nicht das einzige gestellte Problem. Für einige Aufgaben ist das hierarchische Schema unangemessen: bei einer Erweiterung des als Beispiel angeführten Problems um Kollisionsvermeidung in dynamischer Umwelt ist die Reaktionszeit des Systems aus zwei Gründen suboptimal. Erstens ist oft ist gerade der Sonderfall der einer schnellen Reaktion bedürftige Fall. Die Verlagerung der Bearbeitung des besonderen Falls in höhere Schichten des hierarchischen Planers ist wegen der zusätzlichen Kommunikation zwischen den Schichten und der so eingeführten Verzögerung also unter Umständen leistungsverschlechternd. Zweitens



(a) Architekturschema Shakey



(b) *Triple-Tower*-Architektur

Abb. 2.7: Architekturen nach [Nil98, S. 446ff]: oben das Architekturschema des 1967 am SRI entwickelten Roboters Shakey (ergänzt um die Gruppierung), unten das abstrahierte Schema der *Triple-Tower*-Architektur. Die direkte Verbindung von Perzeption und Aktion stellt bereits einen Vorgriff auf reaktive Systeme dar.

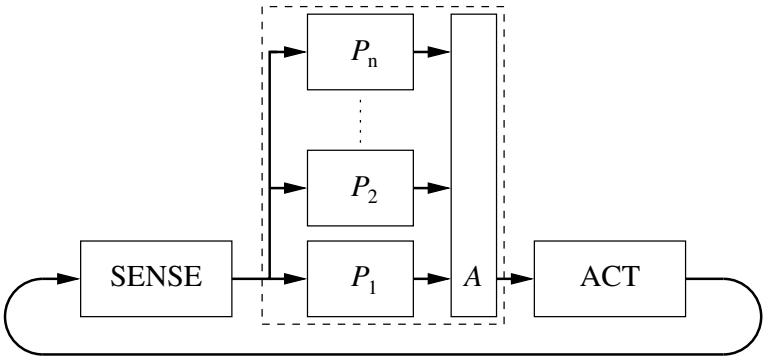


Abb. 2.8: Beispiel einer deliberativ-*subsumptiven* Kontrollstruktur mit einem Arbiter A, der gemäß Priorisierung der Antworten der Planungsmodule P_i genau *eine* Aktion zur Ausführung kommen läßt.

kann für hierarchische Systeme keine zuverlässige Vorhersage der Zykluszeit mehr getroffen werden. Die bedarfsweise Aktivierung einer Kaskade von "höheren" Planungsinstanzen führt zu Verzögerungen, die nicht ratsam erscheinen lassen, die volle Leistung der jeweils unteren Schicht tatsächlich in Anspruch zu nehmen: So darf beispielsweise die *time to collision* (TTC) nicht mehr allein die Höchstgeschwindigkeit bestimmen, da unklar ist, zu welchem Zeitpunkt überhaupt wieder die Effektorkontrolle *act* wirksam wird.

Da eine allgemeine Systemarchitektur nicht allein auf die Klasse der Probleme eines speziellen Typs wie der der gut hierarchisch zerlegbaren eingestellt werden sollte, besteht weiterer Optimierungsbedarf. Der Gedanke der vertikalen Aufgabendekomposition anhand der Analyse "häufiger" und "seltener" Fälle stellt hingegen eine wesentliche Bereicherung dar.

2.1.2.4 Deliberative Subsumption

Schema. Wesentliches strukturelles Merkmal für deliberative Subsumption ist eine dem hierarchischen Planer ähnliche vertikale Aufteilung, doch handelt es sich hier nicht im eigentlichen Sinne um in einer Rangfolge angeordnete Schichten. Im Subsumptionsplaner werden mehrere Steuerungsdaten generierende und Steuerung anfordernde Planungsmodule, die verschiedene Aspekte des Anwendungsproblems bearbeiten, gleichzeitig tätig: Teilprobleme werden grundsätzlich als konkurrent auftretende Probleme verstanden. Keine dieser Instanzen ist direkt mit dem die Effektoren steuernden Teilsystem verbunden. Die unter Umständen im Detail oder im Gan-

zen widersprüchlichen Kommandos der einzelnen Planungsmodule werden von einer für diesen Architekturtyp spezifischen Substruktur verarbeitet und zu eindeutiger Steuerinformation für die *act*-Komponente verarbeitet.

Die Abstimmung zwischen den divergierenden Ausgaben der einzelnen Planermodule kann auf unterschiedliche Weise geschehen:

1. für die Selektion genau *einer* Modulausgabe
   (a) durch Priorisierung der Module und *Selektion* der Steuernachricht des "wichtigsten" Moduls (*arbitration*),
   (b) durch wechselseitige *Inhibition* von Modulen,
2. für die Abstimmung zwischen den Modulausgaben durch relative (und zeitlich variante) *Gewichtung* der Module ("Summation").

Aus naheliegenden Gründen kommt ein *Scheduling* der Module grundsätzlich nicht in Frage. – In der Praxis wird gegenüber Abb. 2.8 zur Verringerung der Komplexität zusätzlich ein Informationsfluß zwischen den Planungsinstanzen P_i stattfinden, um beispielsweise intern eine freiwillig-hierarchische Aufteilung in *navigator* und *driver* (vgl. 2.1.2.3) vornehmen zu können.

Motivation. Der wesentliche Mangel hierarchischer Planer, daß durch bedarfsorientierte Aktivierung übergeordneter Schichten unter Umständen nicht vorhersehbare Verzögerungen eintreten, soll durch Parallelisierung der Planungsmodule im Subsumptionsplaner behandelt werden. Dies geschieht um den Preis eines höheren Rechenaufwands, da stets sämtliche Module aktiv sein werden; doch ist erst so ein zuverlässiger Zeittakt erreichbar, der bei Überlegungen wie der Wahl einer maximalen sicheren Fahrgeschwindigkeit eine wesentliche Rolle spielt.

Ein weiterer strukturbedingter Vorzug des Subsumptionsmodells wird erkennbar, wenn die Module des Planers untereinander nicht in einer hierarchischen Beziehung stehen: so lassen sich durchaus zwei miteinander nicht verbundene Parameter der Lagebewertung des Roboters je nach Abschlußstruktur des *plan*-Subsystems entweder sequentiell oder parallel optimieren, ohne daß die entsprechenden Module überhaupt voneinander wissen müssen.

Randbedingungen. Das Subsumptionsmodell kann mit Gewinn eingesetzt werden, wenn das vollständige Anwendungsproblem eine Zergliederung in logisch unabhängige Teile zuläßt, die konkurrent von den Planungsmodulen bearbeitet werden. Da die Überwachung von bestimmten effektorbezogenen Betriebsparametern (z.B. max. Geschwindigkeit und Sicherheitsabstände) häufig unabhängig geschehen kann, ist die Annahme gerechtfertigt, daß eine derartige Zerlegung entsprechend oft zu finden sein wird. Die nachträgliche Integration von Modulen ist zudem recht einfach.

Die Priorisierung ist von der Hierarchie des entsprechenden Modells insofern verschieden, als daß die Module im Subsumptionsmodell konkurrent ablaufen. Die Leistungssteigerung durch die garantierte obere Schranke für die Bearbeitungszeit und damit schnellere Abfolge der *sense/plan/act*-Zyklen erfordert eine Parallelisierung; ein entsprechend leistungsfähiger Rechner ist also Vorbedingung für den Einsatz dieses Kontrollmodells.

Fazit. Je nach Typ der Lösung der effektorseitigen Abschlußstruktur des Planers können spezifische Schwierigkeiten entstehen. Die Verwendung einer modulweisen Priorisierung (was bereits einer impliziten Hierarchie entspricht) impliziert, daß eine "neutrale" Nachricht vereinbart werden muß, die z.B. von einem Kollisionsvermeidungsmodul generiert wird, wenn keine Kollision zu befürchten ist. Abgesehen vom Mehrbedarf an Kommunikation gegenüber dem hierarchischen System würde die Zykluszeit nun vom *langsamsten* Modul bestimmt. Im Fall wechselseitiger Inhibition wird zudem lokales Wissen um Strukturinformation voraus: den inhibierenden Modulen muß bekannt sein, welche anderen Module inhibiert werden müssen. Ein Modul, das eine Kollision abwenden soll, müßte beispielsweise wissen, welche anderen Module Bewegungskommandos – und damit auch die für das unmittelbare Problem ursächlichen – generieren könnten. Auch bei Einsatz einer Gewichtung der Modulausgaben durch "Summation" mit anschließender Normierung werden zahlreiche möglicherweise sogar zeitlich variante Parameter in das System eingeführt. Diese sind nicht nur schwer festzulegen, sondern setzen auch einer Erweiterbarkeit wenigstens in der Praxis schnell Grenzen. Zudem ist nicht jedes Problem überhaupt in parallelisierbare Teile zerlegbar; oft ist die parallele Verarbeitung bereits dadurch eingeschränkt, daß eine höher priorisierte Schicht Informationen aus der Verarbeitung der niedriger priorisierten Schicht benötigt (z.B. bei einer Inhibition, deren Vorbedingung eine bestimmte Parameterkonditionierung des zu inhibierenden Moduls ist).

Der Subsumptionstyp des Planers läßt immerhin eine Revision des Plans durch prinzipiell jedes Modul zu. Dies *kann* generell zu einem großen

Problem werden, wenn die schrittweise Abwicklung der Sequenz eines "Hauptplans" durch Überlagerung von "dringenden" Modulen unmöglich gemacht wird: der Fehlerzustand wird unter Umständen nicht einmal entdeckt.

Der hauptsächliche Unterschied des Subsumptionsmodells zur rein hierarchischen Variante ist eine wenigstens prinzipiell bessere *Erweiterbarkeit* um Planungsmodule, die hier nicht zwingend in eine Planungshierarchie integriert werden müssen (dennoch ist sorgfältig auf eine problemangemessene Priorisierung der Modulantworten zu achten, um das gewünschte Systemverhalten zu erzeugen). Die Subsumptionsarchitektur vermag aus einer *Parallelisierung* größeren Nutzen zu ziehen. Der gemeinsamen Systemtakt wird allerdings durch das *langsamste* Modul bestimmt. Bietet sich keine feingranulare Parallelisierung an, bleibt die Reaktivität des Systems weiterhin erheblich eingeschränkt.

2.1.2.5 Zusammenfassung

Das deliberative Paradigma entspricht weitgehend einer direkten Umsetzung der Methoden der klassischen Programmierung: Es entstehen Zyklen von Eingabe, Verarbeitung und Ausgabe (vgl. 2.1). Wegen der vergleichsweise zeitintensiven Auswertung der Sensordaten und der nötigen Planungsprozesse neigen hierarchische Systeme zu wohldurchdachten, aber deutlich verzögerten Reaktionen auf eine dynamische Umwelt, und kommen so nur in einem eingeschränkten Einsatzgebiet überhaupt in Frage. Durch Anpassung der inneren Struktur der *plan*-Systemkomponente an bestimmte zu bearbeitende Anwendungsproblem lassen sich immerhin erhebliche Leistungsverbesserungen gegenüber der einfachsten Variante erzielen. Für einige Anwendungsprobleme ist diese Anpassung allerdings nicht möglich. Diese Einschränkung der Menge lösbarer Probleme ergibt sich zwangsläufig aus der (im Vergleich zu noch vorzustellenden Alternativen) großen Zykluszeit und des hohen Zeitanteils planender/umsetzender Tätigkeit ohne aktive Sensorik. Diese Einschränkung hat insbesondere Konsequenzen

- in dynamischer Umgebung,

  da sich Umweltparameter auch in den *closed eyes*-Intervallen verändern. Dies könnte noch prädiktiv modelliert werden, doch führt ein solcher Ansatz spätestens dann nicht mehr zum Erfolg, wenn eine *Änderung* z.B. eine Geschwindigkeits- oder Richtungsänderung eines beweglichen Objekts eintritt;

- unter sensorisch unscharfer Wahrnehmung und mechanischer Ungenauigkeit,

  da die unerwünschte Wirkung eines auf Basis fehlerbehafteter Sensordaten erstellten Planes oder eine mechanisch fehlerbehaftete Aktionsumsetzung nicht unmittelbar, sondern frühestens im nächsten Zyklus festgestellt werden kann.

Der einzige im Rahmen strikt deliberativer Architekturen zulässige Lösungsansatz für diese Probleme ist der Versuch der Verringerung der Dauer eines vollständigen *sense/plan/act*-Zyklus.

Eine weitere systemimmanente Einschränkung ist durch die Konstruktion des symbolisch operierenden Planers und des Weltmodells gegeben: die für eine geeignete Interpretation der Sensordaten gemachte *closed world assumption* darf durch neuartige Bedingungen während des Einsatzes nicht verletzt werden. Die Frage nach den Einsatzmöglichkeiten deliberativer Kontrollmodelle ist also – über die oben genannten grundsätzlichen Probleme hinaus – mit der Frage verknüpft, ob und wie überhaupt unverarbeiteten Sensordaten eine angemessene "Bedeutung" zugeschrieben werden kann (*symbol grounding hypothesis*), auf deren symbolische Repräsentation ein Planungsmodul zurückgreifen kann. Eine weitere Verbindung besteht (allerdings jenseits des von dieser Arbeit abgedeckten Bereichs) zum Lernproblem: müßte ein deliberativ konzipiertes System lernen, wären *symbolische* "Regeln" zu ergänzen. In einem nicht lernenden System bleibt der finite Regelsatz notwendig zu unvollständig, um in neuartigen Situationen – je nach Systemdesign könnte das bereits eine veränderte Beleuchtungssituation sein – absichtsvoll angemessen zu handeln.

2.1.2.6 Implementierungen

Das **deliberative** oder – entsprechend der vorherrschenden strukturellen Verknüpfung benannte – hierarchische ([PHTO+00]: "traditionelle") Paradigma tritt vermutlich erstmalig 1967 mit dem am SRI entwickelten Roboter *Shakey* auf. Es ist nach wie vor häufig anzutreffen [Alb81, Alb89, Mey91, Edl96] und ist unverändert für bestimmte Aspekte auch der hybriden Architekturmodelle (s. 2.1.4) von tragender Bedeutung.

2.1.3 Das reaktive Modell

Ein wesentlicher Anlaß zur Weiterentwicklung der reaktiven Systeme war, daß die deliberativen Modelle durch die Integration expliziter und symbo-

lischer Planung unter Umständen nicht hinreichend performant sind. Dies galt sicherlich um so mehr auf den um 1980 verfügbaren Rechnern.

Motivation. Trotz erheblicher Leistungssteigerung lassen nicht alle Anwendungsprobleme die Integration eines symbolischen Planers zu. Problematisch ist nicht nur sein Rechenzeitbedarf, sondern auch, daß die Umsetzung geplanter Aktionen oft aus mechanischen Gründen nicht mit der erforderlichen Präzision geschehen kann: dies macht u.U. eine schnelle Nachregelung erforderlich, die mit der durch einen expliziten Planer eingeführten Verzögerungszeit nicht angemessen geleistet werden kann. Darüber hinaus wurden Probleme identifiziert, die nicht durch eine bloße Steigerung der Verarbeitungsgeschwindigkeit des Planers behoben werden können: unter gewissen Umständen ist eine vollständige Planung auch ohne zeitliche Randbedingungen wegen unbekannter Umweltgrößen nicht praktikabel. Diese Situation ist bereits gegeben, wenn während der Laufzeit Unvorhergesehenes geschehen kann, da in diesem Fall die Annahme CWA nicht gültig ist. Grundsätzlich lassen sich Wahrnehmungen nur unter der Voraussetzung korrekt in symbolisch verarbeitbare Repräsentationen übersetzen, wenn eine entsprechende Transformation durch explizite Konstruktion vorgegeben ist (oder erlernt wurde). In der Sicht von BROOKS war damit bereits der Einsatz "klassischer", auf Symbolsystemen basierender KI bei der Konstruktion von Robotersystemen eine Fehlentwicklung [Bro90c]. Ein weiterer Wunsch war die Erprobung von biologisch inspirierten Modellen [Bro91a, Bro90b, Bro91c].

Die zentrale Idee. In Konsequenz wurde der rechenintensive symbolische Planer aus dem System entfernt und versucht, eine möglichst unmittelbare Abbildung von Sensoreingaben auf die Effektorsteuerung vorzunehmen: die Planung wurde gewissermaßen von der taktischen Phase des Systemeinsatzes in die strategische Phase des Systemdesigns vorgezogen, um vorab eine angemessene Reaktion auf eine bestimmte Qualität der zur Laufzeit gemachten Messungen festzulegen. Diese Festlegung geschieht allerdings jeweils nur lokal im Rahmen eines nicht näher in Umfang und Zweck bestimmten Verarbeitungsmoduls (*"processing module"*, wiederum s. [Bro90c]); das komplexe Gesamtverhalten ergibt sich emergent durch eine wie auch immer geartete "Kombination" der Ausgabe der jeweils einen Problemaspekt bearbeitenden Module.

Bei Anwendung des in dieser Form selten anzutreffenden *rein* reaktiven Paradigmas ist zudem, da eine direkte Kopplung von Stimulus und Antwort

des Systems vorgenommen werden soll, die Verwendung eines internen Systemzustands nicht zugelassen. Alles "intelligente" Verhalten (*behavior*) des Systems liegt also in einer vorab getroffenen geeigneten Auswahl von Paaren der Eingabe- und Ausgabevektoren begründet (2.4).

$$\mathbf{o}_{t+1} = \mathbf{f}(\mathbf{i}_t) \quad \text{für alle } t \tag{2.4}$$

Dieses Modell ist von einfachen biologischen Organismen inspiriert (*stimulus-response model*), ohne allerdings eine ernstgemeinte äquivalente Umsetzung darzustellen: Organismen verfügen definitiv über einen intern implementierten Zustandsraum – in den einfachsten Fällen werden die Reaktionen des Systems über die Konzentration von beispielsweise Botenstoffen oder Hormonen parametrisiert – eine Option, die einem strikt nach dem reaktiven Paradigma entworfenen System zunächst nicht offen steht.

Als *reaktive Systeme* sollen allgemein die Systeme bezeichnet werden, bei denen kein internes Modell der Umwelt generiert wird, die Reaktion in der Hauptsache als Funktion des Stimulus berechnet wird, und idealerweise kein Zustand (oder Gedächtnis) des Systems vorhanden ist.

Die Einschränkung rührt daher, daß rasch erkannt wurde, daß in Anwendungsproblemen durchaus perzeptuell *un*unterscheidbare Situationen auftreten können, in denen *verschiedene* Reaktionen gefordert sind. Diese Unterscheidung kann von reinen Stimulus-Response-Systemen nicht geleistet werden (hier gilt stets: "aus den Augen, aus dem Sinn"); das Problem ist nur durch Einführung eines persistenten internen Zustands lösbar. BROOKS führt zu diesem Zweck Modul-lokale AFSMs ("*augmented finite state machines*", s. [Bro90c]) ein, die im Wesentlichen einen endlichen und strikt lokalen Zustand implementieren. Als Einschränkung bleibt bestehen, daß verschiedene *behavior*s keine Zustandsinformation teilen sollen, die dem "Modell" der deliberativen Systeme entsprechen würde.

2.1.3.1 Allgemeines Schema

Die in Abb. 2.1(b) dargestellte Basisstruktur deutet an, daß kein Planungsprozeß Einfluß auf die Verarbeitung der Sensordaten oder die Steuerung der Effektoren nimmt. Das Verhalten des Gesamtsystems ist emergent und kann typischerweise keiner bestimmten Komponente allein zugeschrieben werden. Der Baustein reaktiver Systeme ist das sog. "Verhalten" (*behavior*, gelegentlich: *schema*), das eine *sense*- und eine *act*-Komponente besitzt. Gehören mehrere *behavior*s zu einem System, so arbeiten sie konkurrent. Ein weiteres Kennzeichen reaktiver Systeme ist die in Bezug auf die Bausteine

jeweils *lokale* Wahrnehmung: es findet keine Sensorfusion statt, da auch kein Weltmodell erzeugt werden muß. Idealerweise ist ein Sensor direkt einem *behavior* zugeordnet (*sensor decoupling*), das ihn exklusiv und in spezifischer Weise (*action-oriented*) einsetzt [Ark93, S. 139]. Die Perzeption ist auf modellfreie Verfahren limitiert, da symbolische Verarbeitung nicht zugelassen ist; dies schließt beispielsweise eine Objekterkennung explizit aus. Im weitesten Sinne "erkannt"Erkennen werden können hingegen Merkmale, deren implizite "Bedeutung" für das System sich durch angemessene Codierung einer entsprechenden Reaktion ergibt.

Innerhalb eines Verhaltens wird – idealerweise zustandsfrei – eine Abbildung vorgenommen, die Perzeption auf eine Aktion abbildet:

$$\text{(perceptual schema)} \rightarrow \text{(motor schema)} \tag{2.5}$$

Konkurrieren mehrere *behavior*s um die Steuerung der Effektoren (das sog. *coupling problem*), so muß eine Koordination vorgesehen werden. Die übliche Bezeichnung *behavior selection* trifft nicht den Kern, da nicht *behaviors* selbst, sondern lediglich ihre Ausgaben in die Verarbeitung eingehen; nach außen wird allerdings der Eindruck erweckt, daß kurzzeitig ein *behavior* die Aktion des Systems determiniert. Technische Möglichkeiten hierfür sind beispielsweise

- die vektorielle Addition der Steuerungsdaten,

  so bei den in 2.1.3.2 behandelten Potentialfeldverfahren und den einfachen verhaltensbasierten Ansätzen in 2.1.3.3, oder

- die laterale Inhibition der Steuerungsdaten untergeordneter *behavior*s,

  so in der in 2.1.3.4 behandelten reaktiven Subsumptionsarchitektur.

Eine sequentielle Abarbeitung der Steuerungsdaten ist aus naheliegenden Gründen nicht sinnvoll, obgleich ein *round robin*-Scheduling sinnvoll sein kann.

Steuerung der Abfolge. Würde die Aktion stets allein vom aktuellen Perzept bestimmt, so wäre jede Aussicht auf Bearbeitung von Anwendungsproblemen dahin, die einen "Problemkontext" besitzen: die Reaktion des Roboters auf einen gleichen Stimulus würde zu beliebigen Zeitpunkten (im Rahmen mechanischer Wiederholgenauigkeit) gleich ausfallen. Interessanterweise liefert bereits dieses sehr einfache Modell die Lösung für wichtige Probleme (vgl. 2.1.3.2).

Zustand und Speicher. Das rein reaktive Modells kann beispielsweise durch Hinzunahme von

- *behavior*-lokalen (A)FSMs,
- *behavior*-lokalen Interpretern (*script*-Verarbeitung) oder durch
- Codierung von Aktivierungsfolgen (*skills*)

entsprechend erweitert werden [HSDK97]. Wenn NOLFI sich bemüht zu zeigen, daß auch Systeme ohne jeden *internen* Zustand eine (wörtlich: marginale) Datenspeicherung in der Codierung ihrer physischen Position und Orientierung in der Umgebung vornehmen können [Nol02], ist dies jedoch nicht anders als im Fall des Zählens unter Zuhilfenahme der Finger: ein Zustand wird externalisiert und als Stimulus genutzt, um wenigstens kurzfristige Datenpersistenz zu erlangen.

In den folgenden Abschnitten sollen drei wesentliche Vertreter reaktiver Architekturen vorgestellt werden.

2.1.3.2 Potentialfeldverfahren

Begrifflich bezeichnen Potentialfelder ("*pfields*") nicht eigentlich eine Architektur, sondern lediglich das charakteristische Mittel, dessen sich bestimmte reaktive Systeme bedienen: Potentialfelder ermöglichen eine einfache Art, eine Kopplung zwischen Ein- und Ausgabe herzustellen. Sie wurden von KROGH und KHATIB erfolgreich im Zusammenhang mit der Vermeidung von Hindernissen bei sowohl der Navigation mobiler Roboter als auch der Bewegung von mehrgliedrigen Effektoren eingesetzt [Kro86, Kha86].

Schema. Die Struktur, die diesen Systemen zugrundeliegt, entspricht Gleichung 2.4 und besteht nur aus einer einzigen Komponente, die durch Berechnung genau diese Verbindung herstellt. Als *rein* reaktive Architektur ist das Verfahren gedächtnislos und besitzt keinerlei persistenten Zustand.

Die Verwendung von Potentialfeldalgorithmen ist mit großem Nutzen beispielsweise im Kontext der Navigation mobiler Systeme bei der Hindernisvermeidung einsetzbar. Voraussetzung für eine korrekte Funktion ist, daß sowohl die zu erreichende Zielmarke als auch die relevanten Hindernisse zu jedem Zeitpunkt sensorisch wahrgenommen und ihre Positionen in Bezug auf das jeweilige lokale ("robozentrische") Koordinatensystem ermittelt werden können.

Die Gemeinsamkeit aller Modellvarianten ist, daß ein in Bezug auf die Zielkoordinaten positioniertes Attraktorfeld mit einer Reihe von repulsiv wirkenden Feldern, die auf den Hinderniskoordinaten plaziert werden, überlagert wird. Diese Berechnung geschieht dabei ausschließlich für die Position $(0,0)$, die der Roboter einnimmt, insbesondere also nicht für das gesamte "Feld", und ist damit nicht sonderlich rechenzeitaufwendig. In [Kro86] werden die repulsiven Felder mit der momentanen Geschwindigkeit des Roboters so parametrisiert, daß die Plattform bei maximaler Verzögerung noch sicher vor dem Hindernis anhalten könnte. Bei Abstraktion von der Masse des Roboters können, um zur Illustration an einem Beispiel eine gültige Visualisierung zu erhalten, statische Felder verwendet werden. Für eine Darstellung des zeitlichen Verlaufs der Bewegung muß zudem ein einheitliches globales Koordinatensystem gewählt werden; Koordinaten mit diesem Bezugssystem werden durch den Präfix ${}^{W}()$ gekennzeichnet. Abbildung 2.9(a) stellt ein mögliches Attraktorfeld für die Zielposition bei ${}^{W}\mathbf{A} = (4,4)$ dar. Der Feldvektor an einer Position ${}^{W}\mathbf{P}$ des Roboters codiert in diesem Beispiel den strategischen Richtungsvektor s der Bewegung (und nicht die Geschwindigkeit $\dot{s}$ oder die Beschleunigung $\ddot{s}$). Für die Wirkung dieses Feldes auf den Roboter bei initial ${}^{W}\mathbf{P} = (-5,-5)$ wurde mit

$$\mathbf{F}_{\text{attr}}(\mathbf{A},\mathbf{P}) = c \cdot \frac{\mathbf{A}-\mathbf{P}}{||\mathbf{A}-\mathbf{P}||} \quad \text{mit } 0 < c < 1 \tag{2.6}$$

ein homogenes Feld gewählt, da die Geschwindigkeit der Plattform bei Abwesenheit von Hindernissen unabhängig von der Entfernung vom Ziel sein soll.

Ein einzelnes repulsives Feld $\mathbf{F}_{\text{rep}}$ um ein Hindernis in $\mathbf{R}$ mit einem maximalen Wirkradius r kann entsprechend als

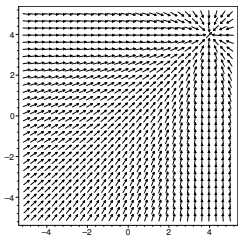
$$\mathbf{F}_{\text{rep}}(\mathbf{R},r,\mathbf{P}) = \begin{cases} \frac{r-||\mathbf{R}-\mathbf{P}||}{r} & \text{wenn } ||\mathbf{R}-\mathbf{P}|| < r \\ 0 & \text{sonst} \end{cases} \tag{2.7}$$

modelliert werden (die Überlagerung zweier Felder mit $\mathbf{R}_1 = (0,-2), r_1 = 3$ und $\mathbf{R}_2 = (0,3), r_2 = 2$ zeigt Abb. 2.9(b) dargestellt). Die Wahl einer derart einfachen linearen Funktion vereinfacht die Berechnung, da eine obere Grenze für die Fernwirkung eines Hindernisses gesetzt werden kann.

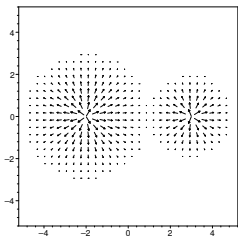
Zur Laufzeit wird durch Addition

$$\mathbf{F} = \mathbf{F}_{\text{attr}} + \sum_i \mathbf{F}_{\text{rep}_i} \tag{2.8}$$

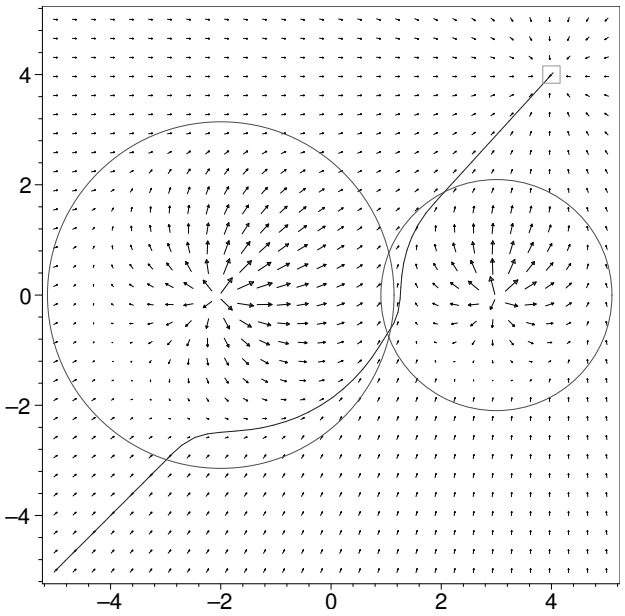
der Feldvektoren in $\mathbf{P}$ die angemessene Änderung der Roboterkoordinaten bestimmt. Abb. 2.9(c) zeigt die Überlagerung der Felder und die durch



(a) Attraktion



(b) Repulsion



(c) Feldkombination und Route (die Achsen zeigen globale Koordinaten)

Abb. 2.9: Potentialfelder zur reaktiven Steuerung der Bewegung eines Roboters: die Route von ${}^W\mathbf{P} = (-5, -5)$ nach ${}^W\mathbf{Q} = (4, 4)$ ergibt sich durch Einwirkung von Attraktion und Repulsion (s. Text).

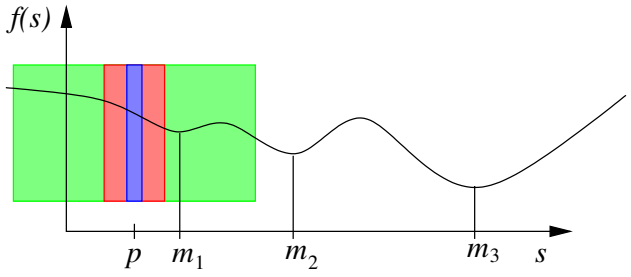


Abb. 2.10: Beispiel eines rein reaktiv nicht lösbaren Problems in einem eindimensionalen Problemraum: ab der initialen Position p sei nach dem Punkt in s zu suchen, in dem $f(s)$ ein globales Minimum annimmt. Bei Verzicht auf einen internen Zustand werden je nach Sensorapertur $m_1 \ldots m_3$ detektiert.

das Verfahren gewonnene Trajektorie von ${}^W\mathbf{P}$ nach ${}^W\mathbf{Q}$. In der praktischen Anwendung finden die relativen Positionen von Hindernissen und Ziel in robozentrischen Koordinaten Verwendung, wie sie ohne Umrechnung von einigen Sensoren sogar unmittelbar geliefert werden ($\mathbf{P} = (0, 0)$ in Gl. 2.6 und 2.7).

Motivation. Die Nutzung von Potentialfeldern (bzw. die Berechnung einer vektoriellen Summe für den Punkt $(0, 0)$) ermöglicht, mit minimalem Aufwand die lokal optimale Aktion für die Erreichung eines Ziels auszuwählen. Spezielle Feldtypen wie tangentiale Felder ermöglichen eine weitergehende Kontrolle der Bewegung, wie die Bevorzugung bestimmter Verkehrsrichtungen an Hindernissen.

Randbedingungen. Potentialfeldverfahren unterliegen drei grundsätzlichen Einschränkungen.

1. Nur bestimmte Anwendungsprobleme sind lösbar.

   Das Verfahren ist nur auf solche Probleme anwendbar, bei denen eine zu optimierende stetige Parameterfunktion $\mathbb{R}^n \rightarrow \mathbb{R}$ berechnet werden kann. Sind mehrere Parameter zu optimieren, ist zusätzlich eine Bewertungsfunktion erforderlich.

2. Die Lösung ist oft nicht optimal.

   Eine suboptimale Alternative wird genau dann statt einer global optimalen Lösung gewählt werden, wenn sie nur unter Berücksichtigung

ausschließlich lokaler Daten attraktiver scheint. Dazu KROGH: Das Prinzip der Anwendung ist

> "(...) *to choose the control at each instant based on the available* local *information so as to guarantee the* acceptability *(but not necessarily the optimality) of the resulting global trajectory.*" – [Kro86, S. 1666]

Ob dies tatsächlich "akzeptabel" ist, muß fallweise entschieden werden.

3. Die Lösung wird unter Umständen überhaupt nicht gefunden.

   Da in jedem Schritt nur die *lokal* optimale Entscheidung getroffen wird, gibt es weder einen sicheren Weg, lokalen Minima zu entgehen, noch eine Entscheidungshilfe bei lokal flachen Gradienten. Die Addition eines *noise*-Vektors hilft möglicherweise weiter, doch nicht mit letzter Sicherheit.

Fazit. Bei Anwendung auf geeignete Probleme (Problemart, keine lokalen Minima) können Potentialfeldverfahren eine mit geringem Aufwand berechenbare und sehr elegante Lösung darstellen.

Es besteht allerdings keine Äquivalenz zur Klasse der deliberativen Systeme: Rein reaktive Systeme werden, da gedächtnislos, stets auf einen Stimulus mit der gleichen Reaktion antworten. Ein reaktives System ohne Zustand ist nicht in der Lage, ein lokales Minimum im Problemraum zu überwinden. Läßt sich die (auch temporale) Apertur für Eingabedaten dergestalt erweitern, daß die Lokalität des Minimums einschrittig erkennbar wird, erscheint das Problem eventuell lösbar (Abb. 2.10) – bei einer anderen Skalierung des Problemraums versagt dieser Ansatz allerdings. Da weiterhin kein explizites Umweltmodell existiert (es ist immanent in der Konstruktion, in diesem Fall: der Feldfunktion, verborgen), geschieht kein Lernen. Auch die Lösung zeitabhängiger Probleme ist nicht umsetzbar.

2.1.3.3 Das verhaltensbasierte Modell

Das *behavior based*-Modell – wie in [ARH87] vorbereitet, in [Ark93] vorgestellt und beispielsweise in [KMRS97] implementiert – kann konzeptionell als eine Erweiterung der Potentialfeldansätze betrachtet werden, die eine parallele Nutzung verschiedenartiger Sensoren zuläßt.

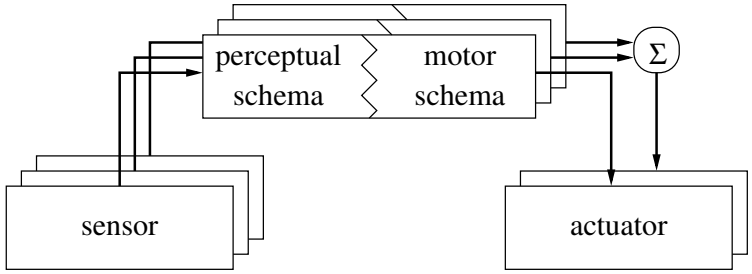


Abb. 2.11: Reaktive *behavior based*-Architektur nach ARKIN

Schema. Im hier behandelten Modell kommen eine Reihe von *behaviors* konkurrent zur Ausführung, die jeweils aus einem sog. Sensor- und einem Effektor*schema* bestehen: hier werden unter Einhaltung der geforderten Lokalität Perzepte auf Steuerungskommandos abgebildet (Abb. 2.11). Dabei ist zulässig, daß ein Verhalten keine (d.h. eine neutrale) Steuerungsinformation liefert. Dies ermöglicht, durch geeignete Formulierung des Sensorschemas eine Aktivierungsbedingung zu integrieren. Da auf diese Weise eine Aktion von bestimmten Wahrnehmungen reaktiv, d.h. praktisch unmittelbar ausgelöst werden kann, wird eine solche Vorbedingung in Anlehnung an den biologischen Reflexbegriff mitunter als *innate releaser mechanism* (IRM) bezeichnet. Die verschiedenen *behavior*s besitzen keine Kenntnis voneinander und treten nicht in Wechselwirkung, wie dies beispielsweise bei der reaktiven Subsumptionsarchitektur (2.1.3.4) der Fall ist. Die Reaktion des Gesamtsystems ergibt sich aus der Steuerung der idealtypisch ebenfalls *behavior*-lokalen Effektoren, in Fällen von durch mehrere *behaviors* gemeinsam zu nutzenden Effektoren durch vektorielle Summation mit anschließender Normierung.

Motivation. Das Verfahren kann die Leistung der in 2.1.3.2 beschriebenen Potentialfeldverfahren vollständig erbringen, indem die dort durchgeführte Berechnung in nur einem *behavior* mit interner Summation vorgenommen wird (die Verwendung mehrerer Instanzen ist nicht sinnvoll, da sich die Zahl der zu berücksichtigenden Hindernisse dynamisch ändern kann, die Zahl der Instanzen jedoch nicht). Hier besteht allerdings die zusätzliche Möglichkeit, durch Anwendung normaler Programmiertechniken – die Formulierungsmöglichkeiten einer Programmiersprache sind reicher als die von Feldgleichungen und erleichtern vor allem die Implementierung diskontinuierlicher Verläufe – das System um weitere Verhaltensaspekte zu

bereichern. Neben der leichten Erweiterbarkeit profitiert das System bei geeigneter Hardwaregrundlage von der stattfindenden Parallelisierung.

Eine entscheidende zusätzliche Option ist hier erstmalig durch die Einführung von *behavior*-lokalen Zuständen gegeben (so werden in [Ark93, S. 145] drei Zustände genutzt, um ein komplexeres zeitliches Verhalten bei der Implementierung eines kooperierenden Verhaltens zu erreichen). Die Erweiterung des *rein reaktiven* Modells durch Integration von Zuständen zur Oberklasse der *reaktiven* Architekturen gestattet die Bearbeitung deutlich komplexerer Anwendungsprobleme durch Einführung der lokal-deliberativen Entscheidung, den Systemzustand zu wechseln.

Randbedingungen. Das *sensor decoupling* als Lokalitätsforderung unterstellt implizit, daß nicht zwei *behaviors* die Perzepte eines Sensors nutzen. Diese Einschränkung ist durch die vorherrschende Betrachtung des Sensoreinsatzes als *action-oriented perception* bedingt – kein Sensor soll zwei verschiedenen *behaviors* dienen. Das Verfahren ist also nur unter der Voraussetzung sinnvoll einsetzbar, daß das Anwendungsproblem so auf perzeptuelle Schemata abgebildet werden kann, daß tatsächlich keine Mehrfachnutzung von Sensoren stattfinden muß.

Fazit. Als früher Vertreter dieser Architektur sind die *perceptual/motor schemas* [ARH87] zu nennen, die gewissermaßen zur Entwicklungszeit vorgeplante elementare Bedienungsstratregien für Systemkomponenten als *behavior* zur Verfügung stellen. Im Kontext der angeführten Arbeit kam allerdings auch die These auf, Perzeption müsse grundsätzlich aktionsorientiert konfiguriert sein; und genau die *dynamische* Rekonfiguration der Schemata zur Laufzeit wurde von ARKIN einige Jahre später durch Entwicklung einer hybriden *managerial style*-Architektur (s. 2.1.4.2) nachgereicht.

Der Verzicht auf exklusive Effektorsteuerung und die Einführung der Summation kann in Inaktivität des Systems bei sich aufhebenden Steuerinformationen resultieren (ARKIN schlägt erneut die additive Kombination von kleinen Zufallswerten vor). Da weder ein *arbiter* vorhanden ist noch die *behaviors* voneinander Kenntnis haben, wird eine Ausgabe eines *behaviors* indirekt per Summation oder direkt auf die zugeordneten Effektoren wirken, auch wenn dies nicht zweckmäßig ist, und eine andere Instanz innerhalb der Architektur dies (sogar reaktiv) feststellen könnte: ein einfaches Beispiel wäre, daß in Gegenwart eines bestimmten Reizes die Bewegung einer mobilen Plattform unterbleiben solle.

Zur Lösung dieser Probleme wäre eine nochmalige Erweiterung nötig, wenn die Abhilfe nicht unter Nutzung grundsätzlich unerwünschter Systemzustände, die einem rudimentären Weltmodell (Reiz präsent bzw. nicht präsent) entsprechen würden, erfolgen soll. Diese Erweiterung wird mit dem reaktiven Subsumptionstyp vorgestellt.

2.1.3.4 Reaktive Subsumption

Wie in den deliberativen Architekturen existiert auch in den reaktiven ein Subsumptionsverfahren. Die reaktive Variante ist untrennbar mit der Arbeit von BROOKS verbunden. Aus der Forderung, daß die Erweiterung eines reaktiven Systems um weitere *behaviors* eine Steigerung der gesamten Systemleistung ("*level of competence*") hervorrufen soll, leitet BROOKS in [Bro86a] die Notwendigkeit einer wechselseitigen Priorisierung der *behaviors* (in Form einer totalen Ordnung) ab. Hierbei treten die Teilsysteme des Planers erstmalig in Interaktion. Dies motiviert das Schema der Subsumption, in der ein *behavior* eine Form von Kontrolle über andere *behaviors* ausüben und somit das emergente Verhalten des Gesamtsystems beeinflussen kann.

Schema. Die Bauelemente reaktiver Subsumptionsarchitekturen sind die aus dem *behavior based*-Modell bekannten aus perzeptuellen und motorischen Schemata bestehenden *behaviors*. Der wesentliche Unterschied gegenüber der *behavior based*-Variante ist jedoch, daß die einzelnen *behaviors* interagieren können. Zu diesem Zweck werden über die aus 2.1.3.2 und 2.1.3.3 bereits bekannte Summation hinaus drei besondere Verknüpfungen verwendet:

1. der Transport von Daten,

   der gestattet, Berechnungsergebnisse nicht nur an Effektoren, sondern auch an andere *behaviors* weiterzuleiten, die diese Daten quasi als virtuelle sensorische (immer noch subsymbolische) Eingabe betrachten,

2. die Inhibition von Ausgängen,

   die die Möglichkeit schafft, nur die Steuerausgabe eines anderen *behaviors* in geeigneten Situationen zu unterdrücken, während das davon betroffene *behavior* weiter arbeitet und einen ggf. vorhandenen lokalen Zustand auch weiter angemessen aktualisiert,

3. die Subsumption von Daten an Eingängen,

   die erlaubt, auf Transportwegen befindliche Daten durch die Resultate eines diesbezüglich übergeordneten *behaviors* gegebenenfalls zu überlagern.

Im Gegensatz zu einigen Varianten der deliberativen Subsumption (vgl. Abschnitt 2.1.2.4) ist kein *arbiter* vorgesehen.

Motivation. Die Einführung der Subsumption erlaubt durch Organisation in *layers of competence* [Bro89] vor allem, unter besonderen Bedingungen unangemessenes Verhalten, das von untergeordneten *behaviors* angeregt wird, mit der Steuerung eines übergeordneten *behaviors* zu überlagern. Diese Eigenschaft teilt es konzeptuell mit der deliberativen Subsumption (vgl. 2.1.2.4). Die einzelnen *behavior*s arbeiten teilweise ohne unmittelbaren Zugriff auf Sensor- und Effektorschnittstellen als interne Subsysteme, sind allerdings dennoch auf die reaktive Verarbeitungsweise beschränkt.

Durch geschickte Vernetzung von einzelnen *behaviors* (vgl. Abb. 2.12) und Modellierung der *behaviors* selbst als *augmented finite state machines* (AFSMs, vgl. [Bro89]) sind die Einschränkungen der *behavior based*-Architektur zu überwinden.

Randbedingungen. Die entscheidende Randbedingung für den Einsatz dieser Architekturvariante ist erneut eine angemessene Zerlegbarkeit des Anwendungsproblem in *behavior*-Aspekte, damit das emergente Verhalten eine brauchbare Lösung darstellt. Hierfür existiert kein konstruktives Verfahren, obgleich eine Orientierung an biologischen Vorbildern für bestimmte Aufgaben in der Vergangenheit oft Anregungen geliefert hat.

Jede erst einmal aufgebaute und funktionierende Vernetzung ist so spezifisch, daß eine über Änderung von Parametern hinausgehende Variation der Anwendungsaufgabe – beispielsweise eine Änderung von Handlungssequenzen – erhebliche Anpassungen an der Innenstruktur einzelner *behaviors* und/oder der Vernetzungsstruktur erfordert.

Fazit. Das Verfahren ist sehr leistungsfähig, erfordert aber unter Umständen eine komplizierte und vollständig aufgabenspezifische Vernetzung der internen Komponenten. Die Möglichkeit zur Zusammenfassung in sinnvolle Hierarchieebenen ist dabei nicht grundsätzlich gegeben, sofern keine totale Ordnung der *behavior*s in Bezug auf ihre Priorisierung hergestellt wer-

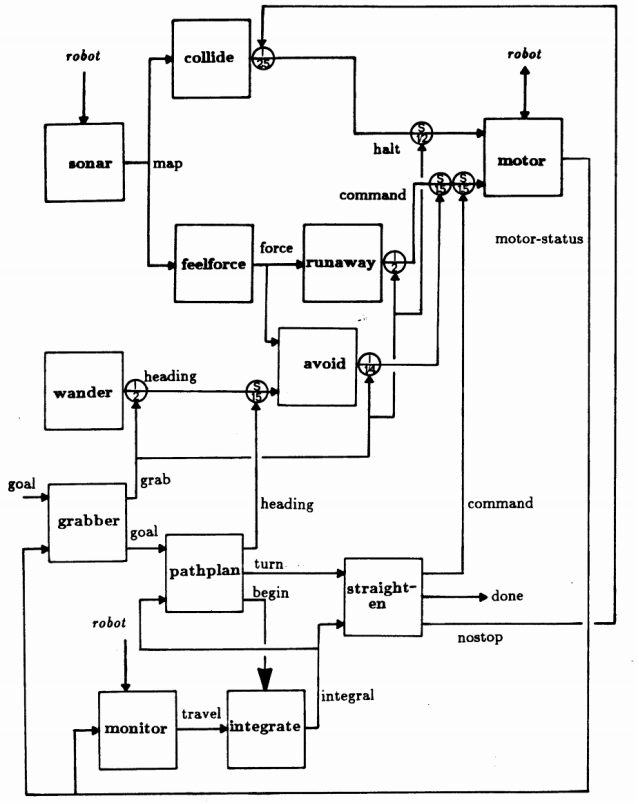


Abb. 2.12: Beispiel einer reaktiven Subsumptionsarchitektur aus [Bro86b] (Ausschnitt).

den kann, was dem oft zu Vergleichen herangezogenen biologischen Vorbild nicht entspricht. In [Ros95, Ros97] wird Informationsverlust als weiterer Nachteil von Subsumption identifiziert. Eine Nebenwirkung der tiefen Staffelung der Vernetzung ist, daß eine größere Latenz zwischen dem Eintreffen eines Sensorereignisses und der erfolgenden Reaktion entstehen kann. Genau die Vorteile, die die Einführung des reaktiven Schemas und damit der direkten Kopplung von Perzeption und Aktion veranlaßten, werden also zumindest graduell aufgegeben, während der Nachteil des Verzichts auf symbolische Weltrepräsentation und Planung erhalten bleibt. Als letzter Einwand sei bemerkt, daß eine unbeabsichtigte Implementierung von nicht-lokalem Zustand erfolgen kann, wenn zwei (oder mehr) *behaviors* in subsumptiver oder kommunikativer Rückkopplung zueinander angeordnet sind.

2.1.3.5 Zusammenfassung

Die Vorzüge reaktiver Architekturen liegen in der Robustheit des Verfahrens gegenüber vor allem sensorseitig auftretenden Problemen und der optimalen Arbeitsgeschwindigkeit. Ihre grundsätzliche Limitation liegt in der erheblichen Schwierigkeit bei der Konstruktion großer Systeme oder solcher, die komplexe Anwendungsprobleme mit kontextabhängig geforderten Reaktionen bearbeiten können – das "deliberative" Fällen einer Entscheidung ist deliberativ unter Einsatz eines Planers mit deutlich geringerem Aufwand zu implementieren. Viele Anwendungsprobleme sind überhaupt nicht mit Potentialfeldern zu lösen, und die Vernetzung mit Inhibition und Suppression kann schon bei einfachen Anwendungsproblemen äußerst kompliziert werden (vgl. Abb. 2.12). Reaktive Bearbeitung eines Anwendungsproblems erfordert eine dedizierte Konstruktion, deren Resultat im Anschluß schlecht auf andere Probleme adaptierbar ist.

Trotz des Verzichts auf symbolische Planung sind reaktive System wenigstens prinzipiell auch zur Bearbeitung von vielen Aufgaben einsetzbar, die nach allgemeinem intuitiven Verständnis Planung erfordern; im Gedankenexperiment könnten bei einem endlichen (d.h. diskreten) Problemraum alle Eingabevektoren mit den zur Konstruktionszeit vorausberechneten optimalen Aktionen zu Stimulus/Reaktion-Paaren verkoppelt werden. Gerade in Bezug auf die Konstruktion einer möglichst allgemeinen Maschine macht die Codierung einer solchen "Lösungsnormalform" allerdings keinen Sinn.

Umsetzungen. Das reaktive Systemmodell läßt sich technisch recht einfach umsetzen. Je nach Definition des Begriffs des "mobilen Roboters" kön-

nen unterschiedliche Systeme für sich beanspruchen, der erste Vertreter dieser Gattung zu sein. Rechnet man die (nicht programmierbaren) ab 1949 gebauten Schildkröten Elmer und Elsie des W. G. WALTER [Wal50, Wal51] ein, stellen wohl diese *rein* reaktiven Systeme die ersten Roboter. Diese fest verdrahteten und aus diskreten Bauteilen aufgebauten Plattformen nahmen (in der späteren Version) die technische Simulation von nur zwei Neuronen vor. Trotz der scheinbaren Anspruchslosigkeit demonstrierte der Aufbau sehr eindrucksvoll, daß bereits einfach verknüpfte technische Strukturen das an den Tag legen können, was als "emergentes reaktives Verhalten" bezeichnet wird. Während über Potentialfeldverfahren eine verblüffend einfache Lösung für bestimmte Navigationsprobleme gefunden werden konnte, gelangte mit der Arbeit von BROOKS, der die vorherrschende deliberative Dekomposition "*sense/plan/act*" für unangemessen hielt und einen Gegenentwurf vorlegte, auch deutlich komplexeres Verhalten bei bis zu diesem Zeitpunkt unerreichter Robustheit in Reichweite. Die Arbeiten von ARBIB und BRAITENBERG [Bra84] lieferten ein entsprechendes theoretisches Fundament.

Einschränkungen. Eine übliche Aufgabenstellung in der Robotik ist nicht, das emergente Verhalten einer Vernetzung von Verhaltensmustern zu beobachten, sondern vorrangig, ein *bestimmtes* Verhalten möglichst präzise zu synthetisieren. Dies bleibt unter Nutzung reaktiver Verfahren problematisch, da zwar aus dem Problem möglicherweise eine adäquate Menge von Stimulus/Response-Paaren abgeleitet werden kann, jedoch nicht unbedingt die Vernetzungsstruktur einer Subsumptionsarchitektur: der *layer of competence*-Gedanke hilft nur, sofern tatsächlich eine statische Priorisierung festzulegen ist. Der Verzicht auf symbolische Planung ist auch insofern ein Nachteil, als daß zur Laufzeit keine (symbolischen) Zielvorgaben an das System übermittelt werden können (das sog. *deliberation problem*): es existiert nicht einmal eine Repräsentation von Zielen innerhalb des Systems. Das System ist weitgehend für *eine* bestimmte Aufgabe konstruiert, ohne daß eine Aufgabenbeschreibung lokalisier- und änderbar wäre.

Auch die Zusicherbarkeit von Aktionsinvarianten komplexer Systeme hat praktische Relevanz (für die Prozeduren zur TÜV-Zulassung und CE-Zertifizierung von Robotersystemen s. [GED98, ED99, BZD01]). Die Einhaltung von nur in Modellform notierten kritischen Parameterbereichen und Randbedingungen kann wegen der kombinatorischen Explosion interagierender Subsysteme (das sog. *scaling problem*) nur schwer sichergestellt oder verifiziert werden, was zukünftig verstärkt von Bedeutung sein mag.

2.1.4 Die hybriden Modelle

Hybride Systeme sind dadurch gekennzeichnet, daß sie sowohl reaktive als auch deliberative Systemteile enthalten. An bestimmten Stellen findet eine direkte Perzeptions-Aktions-Kopplung statt, an anderer Stelle nimmt eine explizite Planung auf Basis eines symbolischen Weltmodells (ggf. indirekt) Einfluß auf die Steuerung der Effektoren.

Im Jahre 1989 veröffentlichte BROOKS einen Beitrag, in dem er eine reaktive Subsumptionsarchitektur beschrieb, die einem sechsbeinigen Roboter (starre Beine mit zwei DoF) die Fortbewegung erlaubt. Die Architektur enthielt nicht weniger als siebenundfünfzig AFSMs mit einer nicht genannten Zahl von lokalen Zuständen [Bro89]. Nicht ohne offensichtlichen Grund enthält der Titel dieser Arbeit, "*A Robot that Walks – Emergent Behavior from a Carefully Evolved Network*", den Hinweis auf sorgfältige Handarbeit. Das Design erfolgreicher komplexer reaktiver Architekturen wird noch heute wegen des bestehenden Mangels eines konstruktiven Entwurfsverfahrens mehr als Kunst denn als Wissenschaft verstanden.

Letztlich wurde die Entwicklung reaktiver Systeme durch den Wunsch nach reaktiverem Verhalten mit geringer Latenz initiiert. Reaktive Kontrolle war wegen der bereits in 2.1.3 angeführten mechanisch bedingten Ungenauigkeiten und einer wenig strukturierten und möglicherweise dynamischen Umwelt als unverzichtbar erkannt worden [AV90]. Spätestens nach einem Jahrzehnt der Entwicklung reaktiver Systeme war allerdings klar, daß relevante Anwendungsprobleme existieren, die von der Einführung von Zuständen, einem Weltmodell und einem Planer erheblich profitieren können, sofern die zeitlichen Randbedingungen einen Einsatz dieser Mittel zulassen – auch wenn, so CONNELL, ein zentrales Weltmodell vielleicht nicht mehr als eine Bequemlichkeit darstellt [Con92]. Die unmittelbare Kopplung von *sense*- und *act*-Komponenten hatte sich in den reaktiven Systemen für den Einsatz im Kontext bestimmter Probleme bewährt; nun wurde also der Versuch unternommen, der Planung zur Laufzeit wieder einen Platz im System zu geben – jedoch nicht mehr zwischen Aktion und Perzeption.

Der Entwurf hybrider Systeme wurde einige Zeit als unzulässige konzeptuelle Verquickung betrachtet [Mur00]. Aus pragmatischen Gründen haben sich diese Systeme dennoch durchgesetzt, da sie gestatten, die nicht akzeptablen Nachteile der bereits vorgestellten Systemparadigmen zu kompensieren: eine Planung darf symbolisch durchgeführt werden, ohne aufwendig reaktiv durch Transformation sämtlicher denkbarer Eingabevektoren emuliert zu werden; und eine überlegene Reaktionsgeschwindigkeit und Robustheit kann reaktiv erreicht werden, ohne den langsamen deliberativen

und symbolischen Planer unter Verlust jeder Übersichtlichkeit (*micro-states*) auf maximale Zyklusrate trimmen zu müssen. Bereits das nachträgliche Einbringen von Planung *on demand* durch deliberative Selektion oder Priorisierung von *behaviors*, oder die Parametrisierung eines *behaviors* durch deliberative Komponenten führt unmittelbar zu den hybriden Systemen (für beide Alternativen s. [RK03]).

Bereits der 1967 vorgelegte deliberative Architekturentwurf für Shakey weist eine *direkte* Kopplung von Sensor- und Effektorsubsystem auf (wenn auch nur zwischen den *bump and touch*-Sensoren, vgl. Abb. 2.7(a) auf S. 36). Die Integration dieser im deliberativen Konzept streng genommen nicht denkbaren Verbindung war allerdings nicht zum Zweck der Implementierung regulären Verhaltens geschehen, sondern um im Fall einer bereits erfolgten Berührung von Hindernissen weiteren Schaden zu verhindern. Auch die immer ausgedehntere Verwendung von Zuständen durch AFSMs in reaktiven Architekturen (vgl. [Bro90c]) stellt einen gewissen Schritt in Richtung hybrider Architekturvarianten dar. In beiden Fällen war das mit der Nichteinhaltung der klassischen Designvorgaben verbundene Ziel die Schaffung einer Möglichkeit zur Kombination von schneller Reaktivität mit langsamer Deliberation. Einen ähnlichen Ansatz verfolgten bereits GEORGEFF und LANSKY mit der Agentenarchitektur PRS-1 (*procedural reasoning system*) [GL87]. In diesem "*belief-desire-intention*"-System, wurden Regeln (*beliefs*) als logische Klauseln, und reaktive Teilsysteme (*desires*) als dynamisch parametrisierten *behaviors* codiert. Mit der variablen Aktivierung von sog. *knowledge areas* mit reaktiver oder deliberativer Innenstruktur über auslösende Umweltbedingungen wechselte das Verhalten des Roboters. Im Kontext der TCA (*task control architecture*) von LIN, SIMMONS et al. [LSF89] wurden erstmalig die Mechanismen benannt, die für die verschiedenen Formen von Kontrolle erforderlich sind, und Annahmen über den systematischen Aufbau von Roboterarchitekturen aus reaktiven und deliberativen Komponenten (allerdings in einem statischen Architekturrahmen) entwickelt.

Im eigentlichen Sinne hybride Architekturmodelle versuchen, die offensichtlicheren Vorteile sowohl der deliberativen Modelle (2.1.2) – die Option zur symbolischen Abstraktion und internen Modellierung der für die Aufgabe wesentlichen Umweltaspekte – als auch der reaktiven Modelle (2.1.3) – eben die Geschwindigkeit der Reaktion des Systems auf ungeplante Ereignisse – auf strukturierte Weise in einem System zu vereinen [Neh00].

2.1.4.1 Allgemeines Schema

Die in Abb. 2.1(c) auf S. 25 dargestellte Basisstruktur soll andeuten, daß die Kopplung von *sense* und *act* der reaktiven Systeme grundsätzlich beibehalten wird, aber einem zusätzlichen Einfluß eines Planers unterliegt, der ebenfalls auf die sensorischen Daten Zugriff besitzt. Dieses sog. *eavesdropping on sensors* ist ein besonderes Kennzeichen aller hybrider Architekturen. Überdies ist die reaktive Basisstruktur nicht statisch, sondern wird von einer deliberativen Systemkomponente so parametrisiert, daß eine komplexere Aufgabe bearbeitet werden kann.

$$\begin{aligned} \mathbf{m}_{t+1} &= \mathbf{u}_t(\mathbf{m}_t, \mathbf{i}_t) \\ \mathbf{o}_{t+1} &= \mathbf{f}_t(\mathbf{m}_t, \mathbf{i}_t) \end{aligned} \tag{2.9}$$

Zu diesem Zweck geht in die reaktive Berechnung der Systemreaktion ein globaler Zustand ein, der deliberativ unter Berücksichtigung der Sensorwahrnehmung und des vorherigen Systemzustandes bestimmt wird (2.9).

Im einfachsten Fall können hybride Systemdesigns durch Erweiterung bestehender reaktiver oder deliberativer Architekturen gewonnen werden: so kann die Integration von Gedächtnis und rudimentärer Planung in reaktiven Systemen dem Verlassen lokaler Minima der Bewertungsfunktion dienen. In ähnlicher Weise ist die Erweiterung deliberativer Systeme um eine "kurze" Rückkopplungsschleife möglich, die beispielsweise bei drohender Kollision unmittelbar auf die Effektoren einwirkt und natürlich den Planungsprozeß über eine im Weltmodell nicht erfaßte Zustandsänderung der Umwelt informiert, damit eine (ggf. partielle) Neuplanung eingeleitet werden kann. Beide Erweiterungen befinden sich klar jenseits des von jeweils dem reaktiven bzw. dem hierarchischen Paradigma umschriebenen Rahmens.

Einteilung. Eine feinere Unterteilung der Klasse der hybriden Systeme ist nach verschiedenen Kriterien denkbar. Einen Anhaltspunkt bietet die Art, in der Kontrolle durch die deliberative Komponente vorgenommen wird; das Schema von PIRJANIAN geht von diesem Punkt aus [PHTO$^+$00] und weist koordiniert deliberativ-reaktive Systeme, die deliberative Reaktion mit übergeordneter Planungskomponente und die wohl eher theoretische Option einer reaktiven Deliberation auf. In diesem Fall soll wegen des im Vordergrund stehenden Architekturaspekts jedoch die Art der strukturellen Integration von Kontrolle das Kriterium liefern. Es werden somit

1. der *managerial style* (2.1.4.2),

der eine recht unmittelbare Erweiterung der reaktiven Architekturen darstellt,

2. der *model-oriented style* (2.1.4.3),

   der durch Einführung eines Modells, das als "virtueller Sensor" dient, ohne interne Schichtung auskommt,

3. die *state hierarchy* (2.1.4.4),

   die eine umfangreiche Schichtendifferenzierung entsprechend der Art der temporalen Zusammenhänge und der internen Datenrepräsentation aufweist,

in dieser Folge vorgestellt. Als Spezialfall des *state hierarchy*-Konzepts treten die 3T-Architekturen auf.

2.1.4.2 Der *managerial style*

Diese hybride Architektur ist eine vergleichsweise einfache Art der Integration von expliziter Planung auf einer unverändert reaktiven Basis. Die Bezeichnung *managerial style* rührt von der Hierarchie zwischen übergeordneten, überwachenden und nur allgemeine Direktiven ausgebenden Instanzen und untergeordneten, über Details lokal entscheidenden Instanzen her. Die Beziehung zwischen deliberativer und reaktiver Schicht – wie auch zwischen den Elementen der deliberativen Schicht, sofern sie feiner untergliedert ist – die Idee des hierarchischen Organisationsprinzips wider, wie sie bereits im Zusammenhang mit bestimmten deliberativen Systeme auftrat (vgl. 2.1.2.3).

Schema. Der Aufbau von *managerial style*-Architekturen ist vergleichsweise einfach und entspricht in den unteren Schichten recht unmittelbar einer reaktiven *behavior-based*-Architektur, die durch einen übergeordneten Planer ergänzt wurde (Abb. 2.13). Diese deliberative Ebene erhält die der Planung zugrundeliegenden Daten unmittelbar von den Sensoren, die in weiterer Abgrenzung zu den reaktiven Architekturen nun also nicht mehr lokal zu speziellen *behaviors* erscheinen. Über die Ausgabe des Planers werden nicht etwa Kommandos an die Effektoren übermittelt, sondern lediglich die einzelnen *behaviors* gemäß dem Weltmodell parametrisiert, damit diese wiederum in der Lage sind, eine situations- und planangemessene Effektoraktion auszulösen (oder ggf. eine Zustandstransition in einer *behavior*-lokalen AFSM zu bewirken).

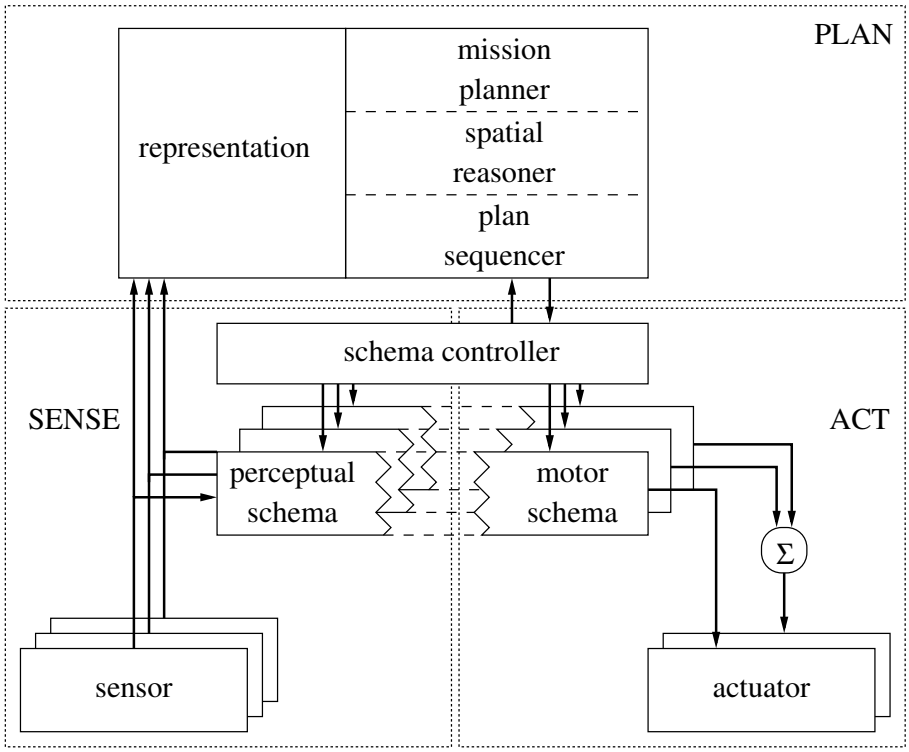


Abb. 2.13: Hybrider *managerial style*: ARKINs AuRA. Gut erkennbar ist die Erweiterung der reaktiven *behavior based*-Architektur (vgl. Abb. 2.11) um die deliberative Komponente, hier in hierarchischer Manier (vgl. Abb. 2.5).

Eine Aktion des Gesamtsystems kann also auf zwei grundsätzlich verschiedene Ursachen zurückzuführen sein:

1. ein Sensorereignis kann gemäß der aktuellen Konfiguration der Perzeptions- und Motorschemata zu einer *reaktiv* (und mit geringer Latenz) ermittelten Aktion führen,

2. ein in Umsetzung begriffener Planungsschritt kann als *virtuelles* Sensorereignis ein Motorschema auslösen. Diese virtuellen Sensorereignisse können sowohl zustandsgesteuert (z.B. nach einer bestimmten Verzögerung) als auch sensorgesteuert ausgelöst werden, da die Planungskomponente ebenfalls Zugriff auf die Sensoren hat.

Praktisch wird das reaktive System sowohl vom Planer zum jeweiligen Modellstand passend konfiguriert und parametrisiert, um eine rasche und angemessene passive Systemreaktion zu gestatten, als auch derart gesteuert, daß aktiv eine bestimmte deliberative Handlung vorgenommen wird.

Motivation. Das System läßt bestehende *behavior-based*-Lösungen weitgehend intakt und faßt lediglich die oftmals ohnehin schon eingeführten lokalen Zustände strukturell zusammen. Ein *managerial style*-System mit inaktivem Planer zeigt also das gleiche Verhalten wie das entsprechende *behavioral based*-System, und ist demnach ohne weitere Anpassung gleich befähigt – und weist insbesondere in Bezug auf die Verarbeitungsgeschwindigkeit die gleiche Leistung auf. Neben der strukturerhaltenden Einbettung einer bekannten Architektur, für die viele wertvolle Arbeiten geleistet wurden, fallen alle durch die Lokalität der Zustände bedingten Einschränkungen weg; Über ein angemessenes Systemverhalten – das neue Parameterset des reaktiven *behaviors* – muß nicht mehr lokal und unter Unkenntnis weiterer Sensorinformation entschieden werden. Durch die zentrale Kontrolle wird auch die Abarbeitung von Aktionssequenzen (wie der Fußfolge für einen Roboter mit Beinen, vgl. erneut [Bro89]) erheblich vereinfacht.

Randbedingungen. Erweist sich bei Umsetzung der Lösung eines Anwendungsproblems ein Bedarf für explizite Planung, kann ein vorhandenes *behavior based*-System verhältnismäßig leicht konsistent erweitert werden. Die Frage nach einem konstruktiven Vorgehen bei der Entwicklung einer angemessenen reaktiven Schicht (was sollte Sensor-, was Motorschema sein) bleibt allerdings ebenso offen wie die nach der Identifikation von für eine Kontrolle sinnvollen Parametern.

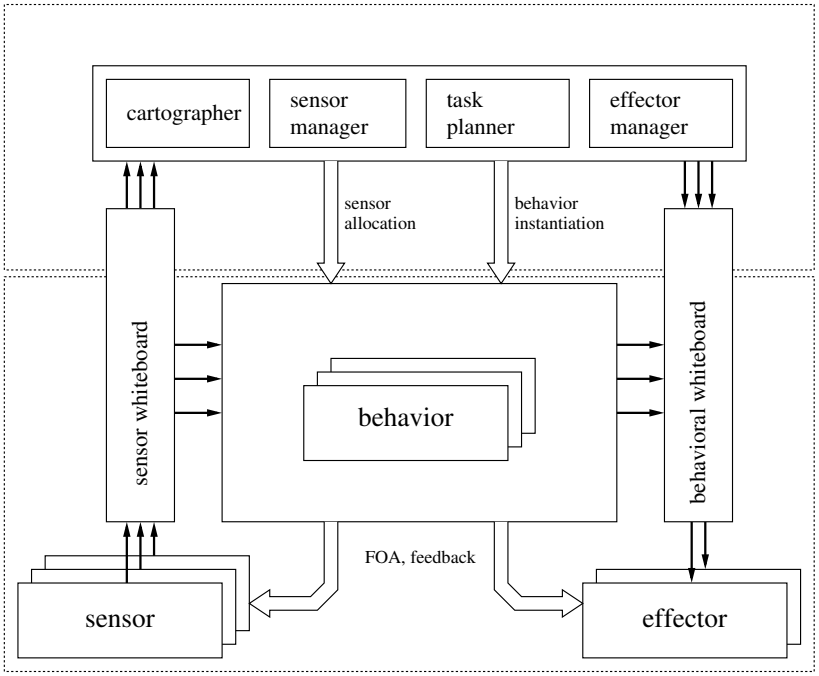


Abb. 2.14: Hybrider *managerial style* in SFX nach [MA92] und [Mur00].

Da die deliberative Schicht in aller Regel eine deutlich geringere Verarbeitungsgeschwindigkeit als die reaktive Basis aufweisen wird, wird eine dem aktuellen Perzept angemessene Parameterkonfiguration unter Umständen erst mit deutlicher Verzögerung wirksam. Systeme im *managerial style* reagieren entsprechend der durch Parameter eingestellten Direktiven schnell, stellen jedoch bei veränderten Gegebenheiten ihr Verhalten relativ langsam um – und könnten in der Verzögerungszeit noch Aktionen durchführen, die unangemessen sind: Im Vergleich würde ein rein deliberatives System das Problem langsamer, doch korrekt lösen.

Fazit. Die Erweiterung reaktiver Systeme um ein die einzelnen *behaviors* parametrisierendes Teilsystem, dem echte Deliberation gestattet ist, stellt einen Lösungsansatz für die in 2.1.3.5 aufgeführten Probleme dar. Das erste als hybrid zu bezeichnende Modell AuRA wurde bereits in [ARH87] beschrieben und wird fortgesetzt weiter evaluiert [AB97].

Im Gegensatz zu reaktiven Systemen, die eine lokale Bindung der Sensoren an *behavior*s vorgesehen haben, besteht zudem erstmalig die Option zur Zusammenfassung der Daten mehrerer Sensoren. Einen frühen Entwurf unter

Berücksichtigung speziell der Probleme der Sensorfusion stellt das System SFX (*sensor fusion effects*) von MURPHY und ARKIN dar (Abb. 2.14). Problematisch bleibt an dieser Stelle, daß eine große Abhängigkeit von Anwendungsaufgabe *und* konkreter Gerätekonfiguration besteht:

> *[The sensing plan] is also the structure that has to be constructed by the designer for each new task or sensor configuration.* – [MA92]

Eine geringere Verhaftung der *behavior*s mit der Art und Zahl der auf der Plattform verfügbaren Sensoren verspricht eine hybride Variante, die nicht die Sensoreingabe selbst, sondern ein abstrakteres Weltmodell als Eingabe für die reaktiven Elemente einsetzt.

2.1.4.3 Der *model-oriented style*

Das erstmalig im Zusammenhang mit dem *managerial style* auftretende Konzept, auch Planungsdaten in Form eines virtuellen Sensors den reaktiven Einheiten zur Verfügung zu stellen, löst das *coupling problem* insofern, als daß keine wechselsetige Abstimmung (*arbitration*) zwischen einem deliberativen und einem reaktiven Teil geschehen muß. Der *model-oriented style* führt diesen Ansatz noch weiter.

Schema. Im *model-oriented style* werden sämtliche Sensorinformationen zunächst einem globalen Weltmodell hinzugefügt. Dies impliziert insbesondere, daß kein Sensorereignis unmittelbar reaktive Aktionen auslösen kann, wie das beim *managerial style* mit der parametrisierten reaktiven Schicht noch der Fall war. Das globale Weltmodell kann zum Beispiel in Form eines *local perceptual space* vorliegen (LPS in Abb. 2.15), in dem sowohl unverarbeitete Sensordaten als auch extrahierte Umgebungsmerkmale unter Verwendung robozentrischer Koordinaten abgelegt werden. Dieses Modell dient wiederum einer Reihe von reaktiven *behaviors* als "virtueller Sensor" – die reaktive Schicht besitzt also keinen direkten Sensorzugang. Effektorseitig wird das *coupling problem* beispielsweise bei Saphira dadurch gelöst, daß jedes *behavior* neben der Steuerausgabe einen unter Kenntnis des LPS ermittelten Wert einer "*desirability function*" berechnet, der in einer gewichteten Mittelung (sog. reaktives *context-dependent blending*) Verwendung findet.

Motivation. Die Eliminierung des direkten Sensorzugangs durch Schaffung einer Indirektion ermöglicht idealerweise, auch völlig andere Sensoren

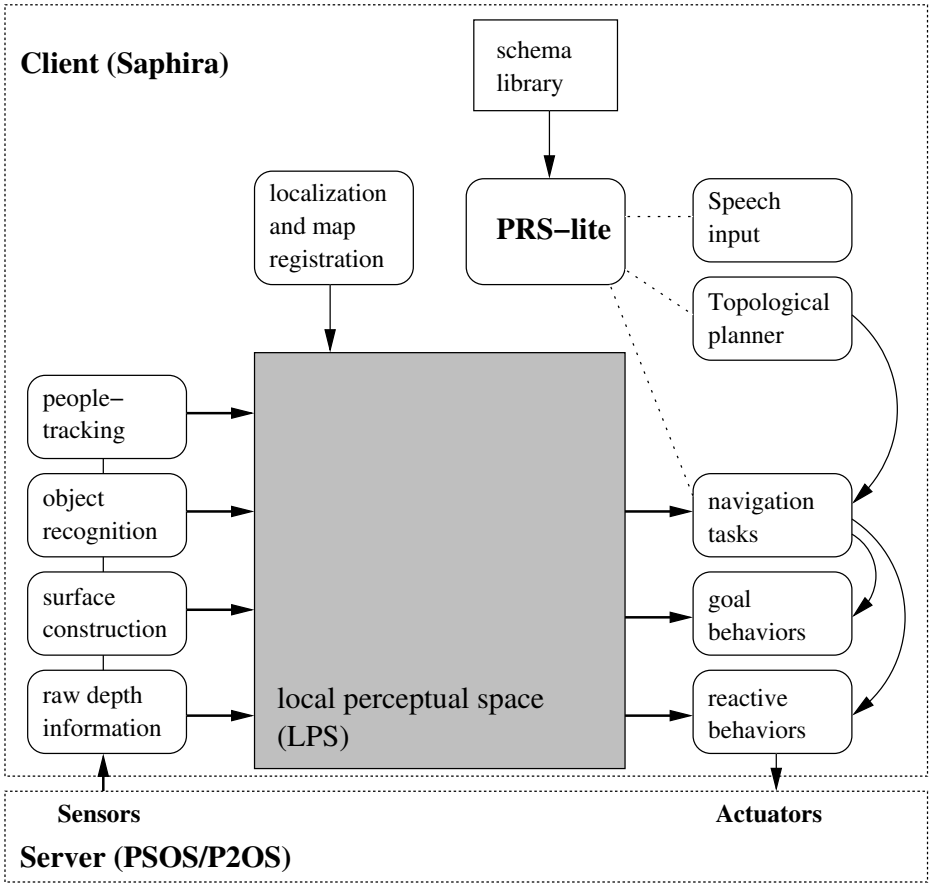


Abb. 2.15: Saphira-Architektur nach [KM99, S. 220] (ergänzt). Kern des Client-Systems bildet der *local perceptual space*, ein durch Fusion entstandenes Modell der Umwelt. Es dient allen *behaviors* als virtueller Sensor.

(z.B. einen LRF statt des Sonars) nachträglich ohne Architekturanpassungen zu integrieren, da die Daten im LPS weder die Sensormodalität noch den Sensor des Ursprungs ausweisen: sowohl die Planung als auch die reaktiven *behaviors* arbeiten auf einer uniformen Planungsgrundlage, die aus der Sensorfusion hervorgeht.

Auch das Verzögerungsproblem des *managerial style* bezüglich der *behaviors*, die wegen des langsamen Planers möglicherweise noch nicht korrekt parametrisiert werden konnten, läßt sich im *model-oriented style* vermeiden, wenn die reaktiven Komponenten nicht unverarbeitete Sensordaten, sondern (deliberativ) zu Merkmalen zusammengefaßte Metainformation als virtuelle sensorische Eingabe verwenden: bei entsprechendem Bedarf lassen sich also deliberative Verarbeitung und Reaktion sogar sequentiell anordnen.

Randbedingungen. Durch die Eintragung *aller* Sensorinformation in einem globalen Weltmodell entstehen grundsätzliche neue Probleme. Der Ausfall eines Sensors hat im Gegensatz zu einem ähnlichen Ereignis in reaktiven Architekturen keine lokalen Konsequenzen, sondern beeinträchtigt das Gesamtsystem. Es läßt sich allerdings argumentieren, daß die Auswirkungen gering ausfallen, sofern redundante Sensoren das gemeinsame Modell versorgen. Eine gegenüber reaktiven Varianten zusätzliche Zeitverzögerung entsteht durch die erforderliche Eintragungen von Sensorereignissen im global verfügbaren Modell. Dies führt in Fällen, in denen überhaupt nur ein einziges reaktives *behavior* auf diese Daten angewiesen wäre, im Vergleich mit beispielsweise dem *managerial style* zu einer etwas schlechteren Performanz. Das größte Problem dürfte sein, daß die Verwendung eines globalen Weltmodells erfordert, daß sich die Sensordaten tatsächlich fusionieren oder wenigstens in normierter Form im Modell hinterlegen lassen. Das ist nicht allgemein gegeben: so sind beispielsweise die Daten im Saphira-LPS auf das Format "Punkte in $\mathbb{R}^2$" beschränkt. Daten anderer Sensoren (z.B. von Kameras) lassen sich praktisch nicht geeignet in diesem LPS fusionieren; entsprechende Versuche zeigen diese Einschränkung rasch auf [BHJ+99].

Fazit. Das modellorientierte hybride Paradigma vermeidet ein potentielles Problem der *managerial style*-Architekturen, indem es den reaktiven Komponenten des Systems keine autonome Handlungsfreiheit mehr zugesteht, erreicht wegen der grundsätzlich erforderlichen Vorverarbeitung aller Sensordaten aber auch nicht ganz deren Performanz. Die Spezifikation ei-

nes angemessenen Weltmodells stellt noch höhere Anforderungen in Bezug auf Universalität und Performanz als bei den deliberativen Architekturen, da die Modelldaten nicht nur weiterhin Planungszwecken dienen, sondern auch die Rolle des Sensors für sämtliche reaktiven Einheiten übernehmen müssen.

2.1.4.4 Schichtenarchitekturen

Als Erkenntnisse aus der Entwicklung deliberativer und reaktiver Roboterarchitekturen sind (stark vereinfacht) festzuhalten, daß einerseits Planung bei der Durchführung komplizierter Aufgaben sehr hilfreich ist, andererseits vergleichsweise unkomplizierte reaktive Systeme für ein "Überleben" des Systems sorgen können. Die Herstellung einer Verbindung zweier Teilsysteme dieser Arten zu einem hybriden System stellt eine Lösung von Problemen dar, die beide Ansätze einzeln entweder nicht leisten können, oder aber eine unzweckmäßige und schlecht an veränderte Anwendungsprobleme anzupassende Form annehmen. Grundsätzlich besteht die Frage nach der *Art einer sinnvollen Kopplung*, die die völlig unterschiedlichen Arbeitsgeschwindigkeiten beider Ebenen berücksichtigt: der *managerial style* weist den Nachteil einer zu schnellen reaktiven Schicht auf, die unter Umständen noch gemäß veralteten und inzwischen unangemessenen Parametervorgaben agiert, der *model-oriented style* verlangsamt sämtliche reaktiven *behaviors* dadurch, daß ihnen allein das globale Weltmodell als virtueller Sensor zur Verfügung steht.

Um eine Lösung für das Problem der Kopplung von Implementierung reaktiven Verhaltens mit konventionellen symbolisch operierenden Planern zu erreichen, wurden in [Gat92, Gat97] und [BFG+97] sogenannte "drei-Schichten-Architekturen" (*three-layer architectures*) vorgeschlagen. Ohne Bezug auf die Zahl der Schichten wird auch die Bezeichnung *state hierarchy* verwendet. Kennzeichnend für Architekturen dieses Typs sind die Ideen, die Planung und das Reagieren asynchron zueinander auszuführen und die Aktionen des Roboters durch den Plan leiten, aber nicht im Detail kontrollieren zu lassen [AC89].

Schema. Die hybriden Architekturen zeichnet aus, daß eine reaktive Basisschicht und eine vergleichsweise langsame deliberative Planungsschicht in hierarchischer Beziehung (vgl. 2.1.2.3) zueinander stehen. Wegen des Unterschieds in der Verarbeitungsgeschwindigkeit beider Schichten ist zur Anpassung zusätzlich mindestens eine Zwischenschicht vorhanden, die nicht

Modus	Atlantis	3T
deliberativ	*deliberator*	*planning layer*
koordinierend	*sequencer*	*sequencing layer*
reaktiv	*controller*	*skill layer*

Abb. 2.16: Drei-Schichten-Architekturen: Gegenüberstellung der Terminologie in Atlantis [Gat92] bzw. 3T (BONASSO und KORTENKAMP).

für das niedrigfrequente deliberative Generieren von Plänen, sondern für das sequentielle Ausführen von angemessenen Teilschritten durch Steuerung und Überwachung ("*reactive planning*") der reaktiven Basisschicht zuständig ist. In der Bezeichnung dieser Zwischenschicht als *sequencer* besteht weitgehende Einheitlichkeit in der Terminologie (Abb. 2.16).

Abbildung 2.17 zeigt das Schema der "3T"-Architektur (*three-tiered*) nach [BFG+97]. Die deliberative Kontrollinstanz, die reaktive Untereinheiten, sogenannte RAP (*reactive action packages*) aktiviert, weist die Architektur nach [Mur00] bzw. [PHTO+00] als *managerial style* bzw. deliberative Reaktion aus.

Motivation. Die Einführung einer Zwischenschicht soll zusammen mit der explizit asynchronen Ausführung reaktiver und deliberativer Komponenten eine erheblich einfachere Gesamtintegration gestatten. Aus der Zuordnung von Problemaspekten zu bestimmten Umsetzungsweisen ergibt sich eine weitere Strukturierung des Systems. So wird die Aufteilung in genau drei Schichten im Spezialfall der TLAs von GAT auf eine weitere Art ("*by time scope*") begründet. Gemäß [Gat97] besitzt die Planungskomponente einen internen Zustand, der *zukünftige Zustände* als Schritte der Planausführung extrapoliert, während der *sequencer* nach Maßgabe des aktiven Plans und seiner aus Plan- und Sensordaten bestehenden *Vorgeschichte* geeignete reaktive Verhaltensmuster selektiert und aktiviert. Der *controller* letztlich, in dem einfache reaktive Fähigkeiten (*skills*) ohne internen Zustandsdaten implementiert sind, bezieht sich allein auf *gegenwärtige* Daten (Abb. 2.18).

Vertreter und Randbedingungen. Die erste hybride TLA dürfte GATs Atlantis (*A three-layer architecture for navigating through intricate situations*) sein, die 1992 entstand. Die Planung erfolgt symbolisch und deliberativ in einer *deliberator*-Schicht, ein *sequencer* aktiviert entsprechend der Planung *behavior*s, die auf der untersten Schicht asynchron ablaufen – und ihrerseits re-

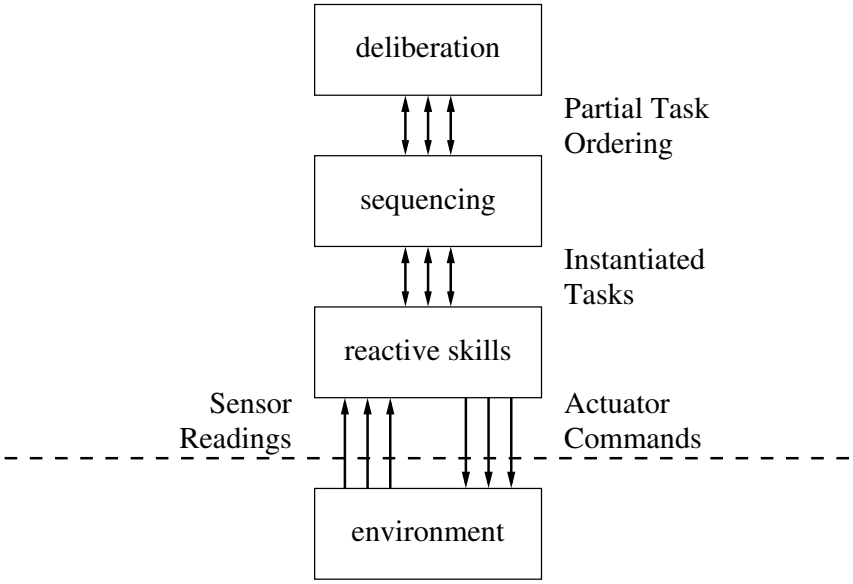


Abb. 2.17: Hybride Systemarchitektur ("3T") nach BONASSO et al. [BFG+97]. Der *sequencing layer* vermittelt zwischen deliberativen und reaktiven Schichten.

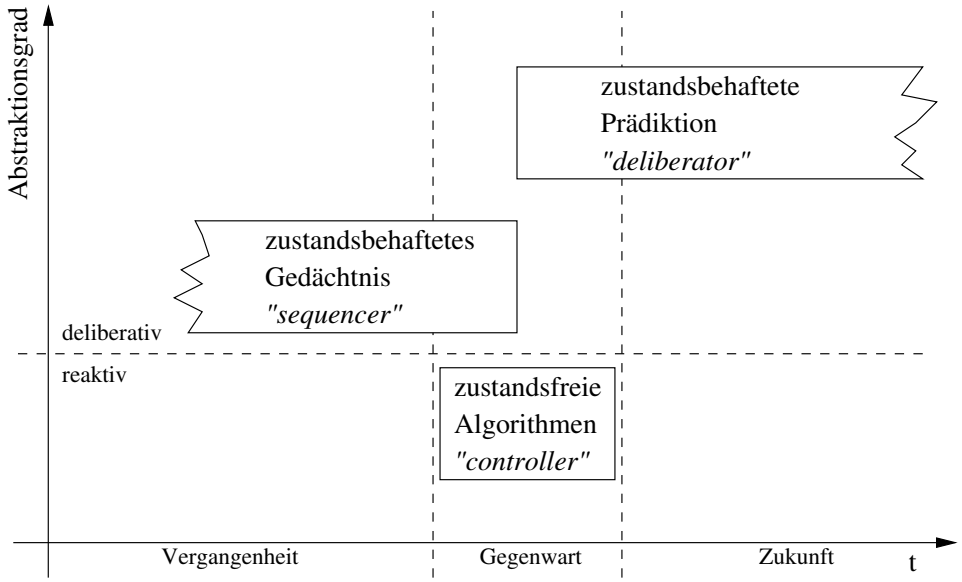


Abb. 2.18: GATs Vorschlag aus [Gat99] für ein Ordnungskriterium zur Partitionierung der Algorithmen in TLAs nach temporalem Datenbezug.

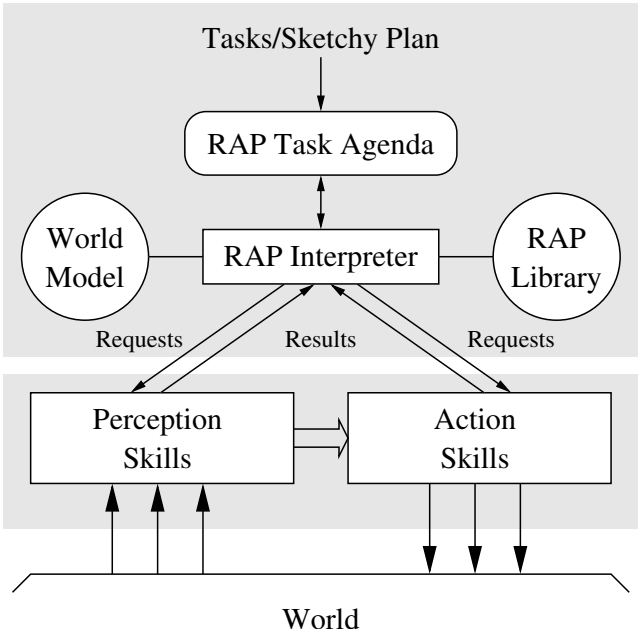


Abb. 2.19: *Animate agent*-Architektur nach [FPS99, S. 247] (ergänzt).

aktiv sein sollten. Im Gegensatz zu subsumptionsartigen Architekturen ist hier explizite Parallelität der Ausführung dergestalt integriert, daß die Verhalten nicht aufgerufen werden, sondern kontinuierlich laufen (*continous action model*) – die deliberative Schicht dient als "advising system" zur Parametrisierung. Ebenfalls 1992 schlägt CONNELL mit SSS eine sehr ähnliche Architektur mit drei Schichten vor, bei der allerdings auch die vermittelnde Schicht noch deutlich reaktive Züge trägt [Con92]. Im gleichen Jahr entwickeln LYONS und HENDRIKS den *planner/reactor* [LH92], bei denen ebenfalls ein reaktives ausführendes Systems kontinuierlich parametrisiert wird (*adaptation system*); beide Komponenten laufen unabhängig und asynchron – allerdings liegt keine ausgesprochene Aufteilung in Schichten vor, und die Problemlösung findet durch adaptive Parameteränderung, nicht auf symbolischer Ebene statt. In der Arbeit [Fer92], die thematisch auf Agentensysteme beschränkt ist, werden ähnliche Strukturen vorgelegt.

Auch die bereits in [Fir89] von FIRBY vorgeschlagenen RAPs wurden durch Integration der deliberativen Planungsschicht zu einer *animate agent*-Architektur weiterentwickelt [Fir94, FPS99], die dem TLA-Konzept entspricht (Abb. 2.19). In den *skills* wird die reaktive Kontrolle kontinuierlicher Prozesse abgewickelt, während durch den RAP-Interpreter (*"task execution level"*) die Kontrolle mehrschrittiger deliberativer Pläne jeweils mit einem *suc-*

cess test als Abbruchbedingung erfolgt. Als Nachfolger des Atlantis-Projekts (GAT 1992) wurde 1996 von BONASSO und KORTENKAMP 3T vorgestellt, das das Modell der Schichtung praktisch unverändert übernimmt, die oberste (deliberative) Schicht jedoch als *adversarial planner* (AP) ausführt und um weiterreichende Planung unter Berücksichtigung dynamischer Umwelt erweitert. Die *behaviors* der untersten Schicht der Atlantis-Architektur weichen den RAPs. Die letzten Entwicklungen stellen PIRJANIANs CAMPOUT für Multirobotszenarien [PHTO+00], die nur mehr zweischichtige Architektur CLARAty (*Coupled Layer Architecture for Robotic Autonomy*) von VOLPE als objektorientiertes Redesign [VNE+00, VNE+01, MPN+02] und ALLIANCE von PARKER, in dem mit *motivational drives* als Handlungsimpetus Versuche unternommen werden [Par98, SA01].

Über Beschränkungen durch architekturbestimmte Randbedingungen des erfolgreichen Einsatzes existieren wegen des erheblichen Gestaltungsspielraums praktisch noch keine Untersuchungen. Allgemeines Erfordernis ist jedoch, daß eine Zuordnung von Aspekten einer Anwendungsproblemlösung zu einer der hierarchischen Schichten erforderlich wird, über die in der Praxis je nach erforderlichen Verarbeitungsgeschwindigkeit entschieden wird.

Fazit. Die Kopplung von in Schichten angeordneter Reaktion und Deliberation steht im Schwerpunkt der *state hierarchy*-Architekturen. Durch die asynchrone Ausführung wird die Leistung der jeweiligen Schichten verbessert, bringt aber als zusätzliche Komplikationsmöglichkeit mit sich, daß die Kopplung entsprechend komplizierter werden kann. Sowohl Ansätze mit direktem Bezug zwischen den Schichten als auch solche mit einer adaptierenden Zwischenschicht (*sequencer*) werden verwendet. Die technischen Details der zweckmäßigen Konstruktion dieser Zwischenschicht bleiben in der Regel offen.

Konzeptionell ist nach wie vor eine Verortung von bestimmten Problemlösungsaspekten in einer der verwendeten Hauptschichten erforderlich. Dies ist nur genau dann kein besonderes Problem, wenn das Verfahren auf das deliberativ oder reaktiv zu lösenden Problemfeld zugeschnitten ist. Die zu bearbeitende Anwendungsaufgabe wird üblicherweise im deliberativen Teil explizit formuliert, während im reaktiven Teil die zur Lösung erforderlichen Fähigkeiten implementiert werden. Es wäre zu prüfen, ob eine Auflösung der Separation in Schichten nicht eine weitere Leistungssteigerung mit sich bringen würde.

Die Zuordnung von Verfahren zu Schichten gemäß der temporalen Zuord-

nung der von ihnen verarbeiteten Daten ist interessant, aber mit gewisser Vorsicht zu betrachten. Wenn nach GAT allein der *sequencer* über Daten aus der Vergangenheit verfügt, wäre diese mittlere Schicht beispielsweise bei der Navigation zu einem bekannten Ziel das aktive Element, und müßte sich zur Erfüllung seiner Aufgabe ausgerechnet der Dienste der übergeordneten Planungsschicht bedienen.

2.1.4.5 Zusammenfassung und Vergleich

Die angeführten hybriden Modelle bieten eine Integration des deliberativen Schemas in eine reaktive Basisarchitektur und unterscheiden sich im Wesentlichen darin,

- wie die reaktiven Komponenten von einer symbolischen Planung gesteuert werden können,

  der *managerial style* sieht eine explizite Parametrisierung vor, die anderen Varianten gehen indirekt vor,

- wie auf Sensordaten zugegriffen werden kann,

  der *model-oriented style* trifft besondere Maßnahmen zur Vereinheitlichung durch Einführung eines globalen virtuellen Sensors, der aktuelle und historische Daten aller Sensoren in einer Weltrepräsentation zusammenfaßt, und

- wie die interne Organisation der Systembestandteile und ihre interne Kommunikation geschehen soll,

  was bei den Schichtenarchitekturen durch asynchrone Ausführung mit ggf. einer Zwischenschicht gelöst wird.

Hybride Systeme können in allen genannten Fällen von der Partitionierung zwischen prozeduralem (planbaren und in mehreren Schritten durchführbaren) und reflexhaftem Vorgehen (unter Beibehaltung temporaler Lokalität) profitieren.

Ungeklärt bleibt das Problem der Zuordnung von Aufgabenbestandteilen: Bereits im Systementwurf müssen die Teilaspekte der jeweiligen Problemlösung einer der beiden genannten Vorgehensweisen zugeordnet und in der jeweiligen Komponente implementiert werden. Dies geschieht in der Praxis

auf Basis von Erfahrungswerten unter Berücksichtigung sowohl der zeitlichen Rahmenbedingungen (eine Kollisionsvermeidung soll schnell reagieren und wird vermutlich rein reaktiv gelöst werden) als auch der Aufgabenkomplexität (eine mehrschrittige Aufgabe wird in der hierarchischen Komponente eines Planers realisiert werden). Insgesamt stellt dies allerdings noch keine Lösung im Sinne eines orthogonalen Systementwurfs dar.

2.2 Schnittstelle zur Welt

Über die in 2.1 behandelte interne Kontrolle hinaus ist eine zweite Klasse von Komponenten für Roboter wesentlich. Sie ist weitaus augenfälliger, denn sie besteht im Wesentlichen aus Hardware. Sensoren und Effektoren stellen für die steuernde Software eine Schnittstelle zur Außenwelt dar, die bereits für sich einige wichtige Besonderheiten im Vergleich zu dem aufweist, was in der Informatik sonst als "Schnittstelle" bezeichnet wird.

In den beiden folgenden Abschnitten sollen die allgemeinen Eigenschaften einiger wesentlicher Schnittstellenelemente skizziert werden, soweit sie bei der Integration in den in Kapitel 4 vorgestellten Entwurf von Bedeutung sind.

2.2.1 Sensoren

Sensoren dienen Robotersystemen der Gewinnung von Daten über den Zustand der Außenwelt. Die Art ihres Einsatzes erfolgt nicht nur unter Berücksichtigung von Annahmen über eine statische oder dynamische Umwelt, sondern auch der konkreten Umweltgegebenheiten, z.B. der Sichtbarkeit oder Verdeckung von Teilen der Umwelt.

Ein weiterer, erheblicher Einfluß auf die Art des Sensoreinsatzes wird bereits implizit durch die Art der Kontrollarchitektur ausgeübt, wie der folgende Unterabschnitt 2.2.1.1 kurz erläutern soll. Da im Kontext der hybriden modellorientierten Architekturvariante in 2.1.4.3 bereits ohne nähere Erläuterung von "virtuellen Sensoren" die Rede war, soll in 2.2.1.2 zur konzeptionellen Klärung eine abstrakte Klassifikation der Menge der Sensoren, die offensichtlich über die der Geräte hinausreicht, nachgeliefert werden.

Dieser Abschnitt soll den Bezug zwischen Sensorik und Kontrollarchitektur aufweisen und in 2.2.1.3 einige für diese Arbeit wesentliche Eigenschaften

bestimmter Sensoren skizzieren, auf die in Kapitel 4 zurückgegriffen werden wird. Er ist in keiner Weise als Einführung in die Sensorik und Sensordatenverarbeitung angesetzt. Für ein entsprechendes Information sei auf [Eve95, BEF96] verwiesen.

2.2.1.1 Arten des Sensoreinsatzes

In jedem Fall liefert ein konkreter Sensor Daten einer ganz bestimmten Art – unabhängig von der Architektur, in die er eingegliedert ist. Zumindest bei internen Sensoren, die im Fall mobiler autonomer Systeme vermutlich die Mehrheit stellen, liegen diese Daten zudem in Bezug auf ein lokales Sensorkoordinatensystem vor: Pixelzeile und -spalte in Bezug auf das optische Zentrum einer Kamera, Echolaufzeit in Bezug auf Position und Ausrichtung eines Sonaremitters. Möglicherweise werden diese Daten unter Nutzung von Wissen über die (feste) Montageposition des Sensors einer Koordinatentransformation unterzogen, um Werte in Bezug auf ein gemeinsames *robozentrisches* System zu erhalten.

Trotz der invarianten Modalität der Sensordaten selbst wird in Abhängigkeit des jeweiligen Architekturparadigmas auf unterschiedliche Weise von diesen Daten Gebrauch gemacht. In den Fällen reaktiver Architekturen und der reaktiven Basiskomponenten hybrider Architekturen findet eine unmittelbare Abbildung von sensorischer Information auf die Effektorsteuerung statt. Dies impliziert, daß von dieser Information grundsätzlich ein *subsymbolischer*, modellfreier Gebrauch gemacht wird. Dies läßt besonders solche Sensoren für den Einsatz in reaktiven Komponenten geeignet erscheinen, die Daten liefern, die zur Laufzeit keiner besonderen Interpretation bedürfen – das implizite Modell ist gewissermaßen bereits in der Konstruktion verborgen, und steuert beispielsweise im Zusammenhang mit den Potentialfeldverfahren (2.1.3.2), welche Wahrnehmung eine attraktive bzw. repulsive Wirkung auf das System entfalten soll.

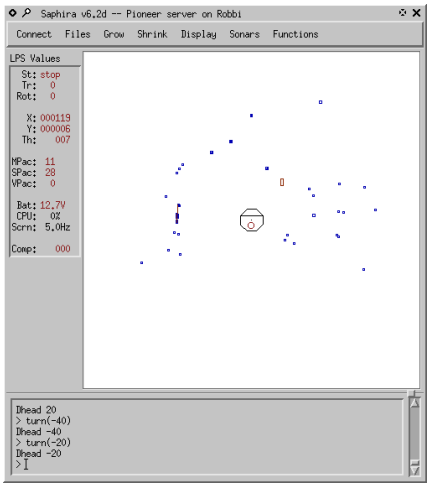
Der Einsatz von expliziten Planungsverfahren in Systemen mit deliberativer Komponente korrespondiert hingegen mit einer interpretierend-modellhaften Verarbeitung *symbolischer* Information: das Planungssystem wird nicht das vielgestaltige Perzept, sondern nur seine abstrakte symbolische Bedeutung verarbeiten. Um zu dieser abstrakteren "Bedeutung" zu gelangen, ist eine Erkennung der mit Symbolen korrespondierenden Dinge, mindestens jedoch eine (möglicherweise mehrstufige) Extraktion von Merkmalen (*features*) erforderlich. Die dazu eingesetzen Verfahren bedienen sich eines explizit konstruierten oder unter Nutzung der Persistenz der Zustandsinfor-

mation dynamisch erlernten Modells[3] – für ein solches Modell kommen beispielsweise im einfachsten Fall Merkmalsparameter in Frage.

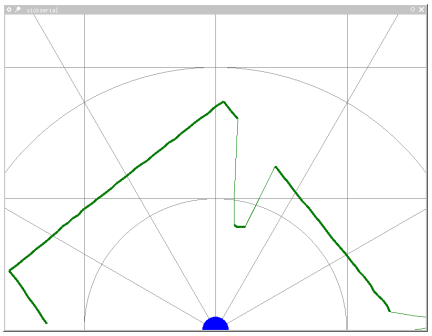
Entsprechende Unterschiede weist ggf. eine Fusionierung der Daten mehrerer Sensoren auf. Im Fall des Einsatzes auf subsymbolischer Ebene kann eine Fusion allein durch Kombination der Wirkung der einzelnen mit den Sensoren verbundenen *behaviors* stattfinden, während bei Verwendung symbolischer Daten eine (z.B. spatiale) Korrespondenz der Daten verschiedener Sensoren unter geeigneten Umständen die Bildung von "höherwertigen" symbolischen Merkmalen gestattet. Aus dem gleichen Grund deliberative Systeme, die symbolische Daten mehrerer Quellen verarbeiten, durch unvorhergesehene Ausfälle einzelner Sensoren weit härter betroffen als vergleichbare reaktive Systeme, bei denen immerhin eine gewisse Teilfunktionalität erhalten bleibt.

Zur Illustration des verschieden ausfallenden Interpretationsbedarfs – und der in der Praxis üblicherweise erfolgenden Zuordnung bestimmter Sensoren zu subsymbolisch/reaktiven bzw. symbolisch/deliberativen Systemkomponenten – zeigt Abb. 2.20 drei Sätze von Daten der gleichen Situation, die jedoch von verschiedenen Sensoren aufgezeichnet wurden. In 2.20(a) sind Sonarechos dargestellt, die über einige Sekunden akkumuliert wurden (der Roboter ist in der Mitte des Feldes dargestellt und nach oben orientiert). Zwar unterstützt die verwendete Software Saphira auch die Extraktion einfacher (Linien-)Merkmale, doch konnte hier keine erfolgreiche Rekonstruktion vorgenommen werden. Dennoch wären diese Daten uninterpretiert als Zentren repulsiver Potentialfelder für die Bewegung der Plattform zu benutzen. In 2.20(b) ist ein zwei Quadranten umfassender Scan des LRF abgebildet. Die vorgenommene ein einfaches Modell (Plattformgröße, Sensorauflösung) benutzende *Interpretation* zeigt die Echos durch stärkere Liniensegmente verbunden, die als zusammengehörend betrachtet werden. Für eine Navigationsplanung kann von den Einzelpunkten so abstrahiert werden, doch wird nicht deutlich, was durch das Hindernis voraus verdeckt wird, oder um was für ein Hindernis es sich handeln könnte. 2.20(c) zeigt schließlich das korrespondierende Kamerabild: die Darstellung offenbart einige Schwächen des Sonars (verzögerte Echos aus der Heizungsverkleidung liefern fehlerhafte Daten) und des LRF (der linke Kegel liegt unter der Abtastebene des Sensors). Für sich genommen ist das Kamerabild allerdings wenigstens durch einen Roboter kaum hinreichend interpretierbar, da

[3] Das zur Extraktion/Erkennung benutzte Modell ist nicht notwendig mit dem durch die symbolische interne Repräsentation des Weltzustands erzeugten Planungsmodell in deliberativen Architekturkomponenten identisch.



(a) Sonardaten



(b) LRF-Daten

(c) Kamerabild

Abb. 2.20: Beispiel zur Ausprägung und Interpretation von Sensordaten – alle "Sichten" zeigen die gleiche Situation. Zur Interpretation s. Text.

ihm ein entsprechend komplexes Weltmodell und die Fähigkeit zur Bildung angemessener Hypothesen fehlt.

2.2.1.2 Sensorklassifikation

Wie im vorigen Unterabschnitt dargelegt bilden die physikalischen Sensoren, die als Hardware implementiert sind, lediglich eine (im reaktiven Fall allerdings die einzige) Schicht in einem mehrschichtigen Aufbau, der der Gewinnung von Umgebungsinformation dient. Auf den Schichten lassen sich physikalische, virtuelle und abstrakte Sensoren unterscheiden.

Auf der Ebene der physikalische Sensoren werden Messungen von Umweltgrößen durchgeführt; Kameras, Sonar, Berührungssensoren und viele mehr sind Vertreter dieser Klasse. Die Qualität der Sensordaten ist abhängig von der Güte des Sensors, jedoch grundsätzlich als Messung im physikalischen Sinn mit systematischen und stochastischen Fehlern behaftet.

Die virtuellen Sensoren werden von vorverarbeiteten Daten physikalischer Sensoren gespeist und stellen eine Abstraktion von Geräten und Betriebsmodalitäten dar. Mit genau diesem Ziel wurden sie im Kontext der modellbasierten Architekturparadigmen (vgl. 2.1.4.3) eingeführt und sogar als Grundlage auch für reaktive *behaviors* eingesetzt – dies ist allerdings nur deshalb möglich, da die Vorverarbeitung der betreffenden Sensordaten kein umfangreiches Weltwissen voraussetzt. Ein weiterer Nutzen virtueller Sensoren ist die Schaffung einer einfachen Option zur Fusionierung von Daten mehrerer Sensoren. Diese Möglichkeit steht allerdings im Normalfall reaktiven Systemkomponenten sowohl wegen der in die Fusion eingehenden Modelldaten als auch des wenigstens partiell symbolischen Resultats nicht zur Verfügung. Beispielhaft könnte eine Stereokamera als ein virtueller Sensor betrachtet werden: als Modell gehen die Parameter der Kamerageometrie ein, und die gewonnenen Daten könnten sich – um eine vergleichsweise unverarbeitete, recht idealisierte Form zu wählen – als Menge von Paaren von Grauwerten und robozentrischen Koordinaten in $\mathbb{R}^3$ darstellen.

Abstrakte Sensoren stellen letztlich eine Verallgemeinerung des Sensorkonzepts dar. Die zu einer abstrakten Weltrepräsentation zusammengefaßten Daten können der Orthogonalität des Systementwurfs halber als abstrakte Sensoren aufgefaßt werden, die auf eine gleichartige Weise Informationen – eigentlich: Hypothesen unter der Annahme einer vorhersagbaren Außenwelt – liefern sollen. Als ein Beispiel für einen abstrakten Sensor könnte eine metrische Karte angeführt werden, die von einem separaten Aktualisierungsprozeß mit wiederum aus den interpretierten Daten eines virtuel-

len Sensors gewonnenem Inhalt gefüllt wird. Diese Karte würde von einem in der Navigation verwendeten Planungsverfahren exklusiv als abstrakter Sensor benutzt. Das Konzept der abstrakten Sensoren führt zu einer verbesserten Generizität der Verfahren, die Sensordaten oder abstrakte Planungsdaten nutzen: auf diese Weise ist beispielsweise der Austausch von physikalischen Sensoren durch ähnliche Instanzen möglich. Die Erweiterung der aus aktuellen Sensordaten bestehenden Datenbasis des virtuellen Sensors um persistente Komponenten (z.B. registrierte Landmarken) ermöglicht zugleich eine einfache Art der Sensorfusion: nämlich die Annotation von verarbeiteten Meßwerten verschiedener Sensoren beispielsweise in einer gemeinsamen Karte.

2.2.1.3 Eigenschaften spezieller Sensoren

Virtuelle wie abstrakte Sensoren spiegeln keine unmittelbar gemessenen Werte, sondern bereits interpretierte Messungen wider, und weisen damit Eigenschaften auf, die vorwiegend von der konkreten Dateninterpretation abhängig sind. Ihre spezifischen Eigenschaften sind nicht allgemeingültig, sondern konstruktionsbedingt; entsprechend werden sie im Zusammenhang des jeweiligen Entwurfs behandelt (Abschnitt 4.3).

Physikalische Sensoren hingegen lassen sich zweckmäßig nach dem Ort der gemessenen Größe in interne bzw. externe Sensoren aufteilen. Wie bereits einleitend bemerkt stellen die mit der Sensorik verbundenen Fehlerquellen und der Bedarf nach Interpretation der Meßwerte eine der Schwierigkeiten bei der Konzeption von Roboterarchitekturen dar.

Interne Sensoren: Odometrie. Größen, die intern gemessen werden, beziehen sich beispielsweise auf die Lage der Plattform, den Betriebszustand des Antriebs oder die Energieversorgung. Von diesen Sensoren ist vor allem die durch Ablesung von Radencodern realisierte Odometrie von Bedeutung für das vorzustellende System. Da eine Odometrie in keinem Fall eine robozentrische Information liefert, sondern ein an einem externen Bezugspunkt verankertes Koordinatensystem voraussetzt, kommt dieser Sensor für den Einsatz in reaktiven Systemkomponenten grundsätzlich nicht in Frage.[4]

Die von diesem Sensor gelieferten Daten bedürfen nicht eigentlich der Interpretation. Die Schwierigkeiten rühren aus mehreren Quellen [BF95, BF96],

[4]Sofern der Sensor über einen entsprechenden Modus verfügt, wäre allenfalls die Verwendung differentieller Werte denkbar.

die interne und externe Ursachen haben. Zu den wichtigsten internen gehören die (durch die Auflösung des Radencoders bestimmte) Quantisierung der Meßwerte, diverse systematische Fehler wie eine durch ungleichmäßige Beladung hervorgerufene effektive Radiusdifferenz der Reifen, und die unscharfe Bestimmung des Kontaktpunktes zwischen Reifen und Untergrund, die sich bei Kurvenfahrt bemerkbar macht. Die externen Ursachen werden durch Probleme bei der Kraftübertragung von Rad auf Untergrund hervorgerufen – [GRS99] nennt einen Translationsfehler von bis zu 25%, und selbst eine ideale Kopplung wird bei einer ungleichmäßigen Struktur des Untergrunds dazu führen, daß beispielsweise Unebenheiten als längere Strecke detektiert werden.

Einige der Quellen systematischer Fehler können eliminiert werden (4.2.3.2), andere können erst durch Hinzunahme von Information anderer Sensoren kompensiert werden (4.3.3.4).

Externe passive Sensoren: Kamera. Eine Unterscheidung der außerhalb des Roboters liegende Größen messenden Sensoren müßte zweckmäßig unter Beachtung des Meßprinzips geschehen: passive Sensoren werden nicht in unbeabsichtigte Interaktion mit beispielsweise gleichartigen Sensoren weiterer Roboter treten, während aktive durchaus dieses Potential haben. Sofern keine weiteren Roboter eingesetzt werden, kann dieses mögliche Problem allerdings vernachlässigt werden.

Die Kamera ist ein solcher passiver Sensor. Einmal abgesehen von dem vergleichsweise geringen Öffnungswinkel des Blickfelds und der beim Einsatz von Weitwinkelobjektiven auftretenden erheblichen Bildverzerrung durch das optische System (für die praktischen Konsequenzen und deren Berücksichtigung im Systementwurf vgl. 4.2.3.3) ist hier die Interpretation der so gewonnenen Daten für Zwecke der Robotik besonders aufwendig. Wenn auch optische Sensoren tatsächlich zur reaktiven Steuerung eingesetzt werden können (*visual servoing*), so sind doch alle Merkmalsextraktion, Objekterkennung und höhere Bilddeutung erfordernden Probleme meist recht unmittelbar mit symbolischer, d.h. deliberativer Verarbeitung verknüpft. Alle Ansätze, die sich um eine Verringerung des Deutungsaufwand bemühen, arbeiten üblicherweise mit weiteren Modellannahmen (so bei der monokularen invers-perspektivischer Transformation, s. [KWS+98b, BM00]), oder greifen aktiv in die Umwelt ein (so unter Verwendung eines Streifenprojektors oder allgemein von strukturierter Beleuchtung, vgl. [ED99, KW99]). Methoden der sog. *active vision* wie probabilistische Blicksteuerung [DFDN99] werden wegen ihres unklaren Nut-

zens bei der algorithmischen Lösung exakt spezifizierter Aufgaben hingegen eher selten eingesetzt (vgl. [WKH98]).

Externe aktive Sensoren: Sonar und LRF. Zwei prinzipiell einander ähnliche aktive Sensoren, die eine Fernwirkung auf die Umwelt ausüben können, sind das Sonar und der Laser Range Finder (LRF). Beide gehören zur Klasse der *time of flight*-Sensoren (ToF), die eine Energieform – sei sie mechanisch (Schall, Ultraschall) oder elektromagnetisch (Radar, Laser) – emittieren und aus der bekannten Ausbreitungsgeschwindigkeit und der gemessenen Laufzeit des reflektierten Signals, eventuell auch der Absorption, Rückschlüsse über die Umgebung ziehen.

Bei allen aktiven Sensoren kann *crosstalk* zwischen mehreren Geräten, durch Mehrfachreflektion oder durch "späte" Echos auftreten. Einige dieser Teilprobleme können durch Trennung in den Domänen Raum, Zeit oder Frequenz erreicht werden; entsprechende Ansätze sehen vor, daß sich z.B. die Scan(halb)ebenen mehrerer Geräte nicht schneiden oder die Geräte synchronisiert werden, oder mit codierten oder Pseudozufallssequenzen gearbeitet wird [AHT98, BBGG99, BJP99].

Die Interpretation von Daten der Sensoren des ToF-Typs ist hingegen vergleichsweise einfach. Um überhaupt eine nützliche Dateninterpretation vornehmen zu können, muß eine zusätzliche Hypothese der lokalen Kontinuität der Umwelt eingeführt werden, die besagt, daß die unmittelbare Umgebung des gemessenen Punktes ähnliche Eigenschaften aufweist:

$$\forall x, y : dx \approx 0 \wedge dy \approx 0 \implies (p(\mathrm{Occ}(x, y)) \approx p(\mathrm{Occ}(x + dx, y + dy))) \tag{2.10}$$

Was an dieser Stelle "klein" (≈ 0) ist, hängt natürlich von der Qualität der Meßdaten (σ_d und σ_ϕ), der zu erwartenden Granularität der Umgebungsstruktur und der Entfernung ab, aus der die Messungen vorgenommen werden; einen Anhaltspunkt für einen wenigstens im Zusammenhang mit Navigation sinnvollen Wert liefert der Radius der Plattform. Erst unter Annahme einer solchen lokalen Kontinuität und bei Wahl entsprechender Parameter läßt sich der Versuch rechtfertigen, aus den Einzelpunkten eines Scans Strukturmerkmale extrahieren zu wollen.

Ideale Belegtheitinformation. An dieser Stelle wird deutlich, was in Abbildung 2.20(b) bereits an nicht näher ausgewiesener modellbasierter (und parameterbehafteter) Interpretationsleistung eingegangen ist. Ein einzelnes Echo bei den Polarkoordinaten (d, α) besagt über die Belegtheit (*occupancy*)

für einen Ausschnitt der Umwelt eigentlich, daß

$$\mathrm{Occ}_{\mathrm{ideal}}(\mathbf{p}) = \begin{cases} \text{frei} & \text{wenn } |\mathbf{p}| < d \wedge \measuredangle(\mathbf{p}, \mathbf{e}_x) = \alpha \\ \text{belegt} & \text{wenn } |\mathbf{p}| = d \wedge \measuredangle(\mathbf{p}, \mathbf{e}_x) = \alpha \\ \text{unbekannt} & \text{sonst} \end{cases} \tag{2.11}$$

Nichts in den Meßdaten rechtfertigt also, ohne weitere Annahmen eine Verbindung der einzelnen Meßpunkte zu unterstellen. Tatsächlich zeigt Abb. 2.20 bereits mit dem "Schattenwurf" des Kegels und den radial vom Betrachterstandpunkt fliehenden Linien eine typische Situation, in denen diese einfache Hypothese zu offensichtlich fehlerhafter Interpretationen führen würde. Aus diesem Grunde ist bereits diese einfache Merkmalsextraktion allein deliberativen Systemen zugänglich.

Reale Belegtheitinformation aus ToF-Scans. Die bereits benannte Qualität der Messung hat einen Einfluß auf die reale Belegtheitsinformation, die sich aus einer Messung gewinnen läßt. Neben dem Fehler der Distanzmessung (Parameter σ_l einer Normalverteilung) tritt als weiterer Parameter die Standardabweichung σ_ϕ der Winkelmessung in Erscheinung. Im Fall eines LRF stehen systematische Fehler des Meßinstruments im Vordergrund; im Fall der Sonarentfernungsmessung ist eher die Signalausbreitung mit unscharfer Richtcharakteristik wesentlich (bei den handelsüblichen 50 mm-Polaroid-Sensoren entspricht der Meßbereich grob einem Kegel mit einem Öffnungswinkel von etwa 30°).

Unter Annahme einer Normalverteilung für Entfernungs- und Winkelabweichung (dazu mehr in 4.2.3.5 für ein konkretes Instrument) könnte eine Belegtheitsfunktion

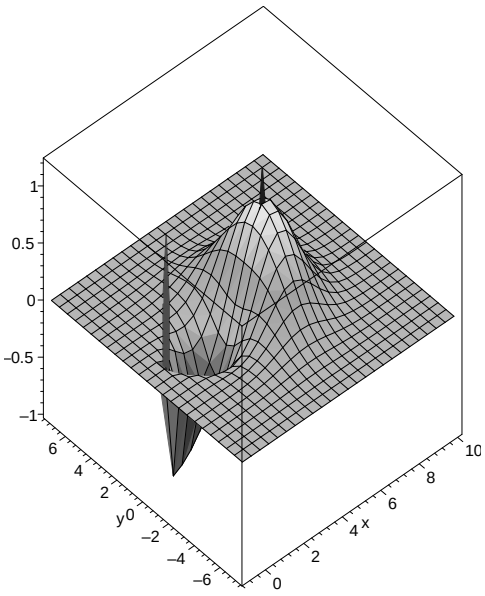
$$\mathrm{Occ} : (x, y) \rightarrow [-1, 1] \tag{2.12}$$

zur Interpretation eines einzelnen Meßereignis bei $(d, 0)$ mit

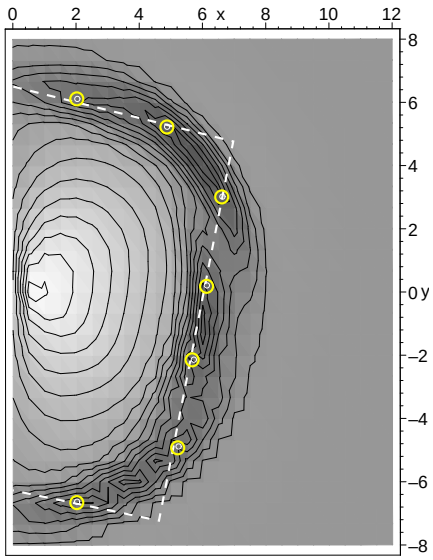
$$N_{\mu,\sigma}(t) := e^{-\frac{(t-\mu)^2}{\sigma^2}} \tag{2.13}$$

– d.h., der *nicht* normierten Normalverteilung, die ihr Maximum bei μ mit $N_{\mu,\sigma}(\mu) = 1$ annimmt – beispielsweise so modelliert werden:

$$\mathrm{Occ}(x, y) = \underbrace{N_{0,\sigma_\phi}\left((x, y)\measuredangle(1, 0)\right)}_{\text{Winkelterm}} \cdot \left(\underbrace{N_{d,\sigma_l}\left(||(x, y)||\right)}_{\text{Belegtheits-Term}} - \underbrace{N_{0,\frac{d}{2}}\left(||(x, y)||\right)}_{\text{Freiflächen-Term}} \right) \tag{2.14}$$



(a) Belegtheitsinformation des Sensormodells – zusätzlich markiert sind die Position des Roboters bei $(0, 0)$ und der rekonstruierte Ort eines Reflektors bei $(5, 0)$.



(b) Überlappende Belegtheitsinformation im Mehrfachscan (überblendet die reale Struktur und die Meßpunkte).

Abb. 2.21: Die Werte der Funktion (2.14) werden zur Beschreibung des Zustands eines Punktes der Scanebene verwendet (hier mit $\sigma_l = 1.2$, $\sigma_\phi = 0.4$). Die extrahierbare Struktur im Mehrfachscan nähert sich dem Streckenzug durch die gemessenen Punkte.

Erst die Auswertung der Belegtheitsinformation unter Berücksichtigung des Sensormodells (vgl. Abb. 2.21) gestattet den Schluß, daß tatsächlich eine brauchbare Möglichkeit zur Identifikation bestimmter Merkmale (z.B. Wände, Türöffnungen) besteht.

2.2.2 Effektoren

Effektoren dienen der Manipulation der sensorisch erfaßten Umwelt gemäß einer reaktiven Antwort auf einen subsymbolischen Stimulus oder einer deliberativen Entscheidung auf Basis extrahierter symbolischer Information. Bei Berücksichtigung mobiler Roboter ist speziell deren Lokomotionsapparat als Effektor zu berücksichtigen.

Im Umgang mit Effektoren sind generell Leistungsgrenzen zu beachten. Diese ergeben sich beispielsweise aus der maximalen erzielbaren Beschleunigung, einer höchstens erreichbaren Wiederholgenauigkeit oder grundsätzlichen kinematischen Einschränkungen wie den Winkel- und Translationsbereichen für Gelenke und den Abmessungen des freien Arbeitsbereichs (*workspace*). Im Fall der in Kapitel 4 vorwiegend verwendeten Plattform sind die Effektoren auf das Fahrwerk, den Greifer und die bewegliche Kameramontage beschränkt. Die spezifischen Eigenschaften dieser Effektoren werden im Rahmen der jeweiligen Subsystemintegration behandelt. Weiter bleiben die bereits genannten sensorseitigen Unsicherheiten zu nennen, die in die Grundlage der Berechnung oder Planung von Effektoraktionen eingehen.

Zusammenfassung

Nach Betrachtung der aus Sensoren und Effektoren bestehenden Schnittstelle zur Außenwelt läßt sich die Behauptung aufstellen, daß der Kontrolleinheit des Robotersystems grundsätzlich nur interpretationsbedürftige und aus verschiedenen Gründen auch mit Unsicherheit behaftete Daten über die Umwelt zur Verfügung stehen. Ebenso geschieht die Umsetzung aller Aktion unter anderem wegen verschiedener mechanischer Problemen nur unvollkommen. Dieser Aspekt läßt – beispielsweise – den Einsatz deliberativer Planer unter einem anderen Licht erscheinen, da sowohl bei der sensorbasierten Pflege des Weltmodells als auch bei der Ausführung von darauf basierenden Plänen schwerwiegende Fehler auftreten können, die nicht (zumindest nicht vorab) durch eine Planungskomponente berücksichtigt werden können.

Eine der wesentlichen Herausforderungen der Systemkonstruktion ist, mit Standardmethoden der Informatik eine Integration der fehlerbehafteten Komponenten an der "Weltschnittstelle" so vorzunehmen, daß das Gesamtsystem dennoch zu einer Leistung befähigt wird. Ein in diesem Sinne verwendbarer Gestaltungsspielraum existiert allerdings praktisch allein im Bereich der Systemarchitektur.

Kapitel 3

Bestandsaufnahme und Motivation der Fragestellung

Dieses Kapitel soll in Abschnitt 3.1 eine Bestandsaufnahme des Ertrags des vorangehenden Kapitels in Bezug auf das bearbeitete Thema vornehmen. Dies dient der in 3.2 vorgenommenen Motivation der diese Arbeit bestimmenden Fragestellung. In Folge soll in Abschnitt 3.3 ein kurzer Ausblick auf die Struktur des weiteren Vorgehens gegeben werden. Da die behandelten Architekturentwürfe dazu neigen, sich sinnvollen quantitativen Vergleichen zu entziehen, werden ergänzend qualitative Testkriterien als Ansatz zur Bewertung von möglichen Lösungen der behandelten Fragestellung formuliert, auf die in Kapitel 5 zurückzukommen sein wird.

3.1 Bestandsaufnahme

Das Unternehmen, unter Betrachtung der allgemeinen Komponenten von Robotersystemen festzustellen, wo für den Systementwurf ein wesentlicher Freiraum besteht, führt zu der Erkenntnis, daß dieser praktisch nur in der System- und Kontrollarchitektur zu finden sein kann. Dennoch ist auch auf den darunterliegenden Ebenen Arbeit zu leisten, um den *bottom-up*-Einfluß einer konkreten Ausstattung mit Sensoren und Effektoren auf die Wahl der Architektur zu reduzieren. Zwar sind keine architekturtypischen Elemente der Schnittstelle zur Außenwelt feststellbar, doch erscheint eine wesentliche Differenz in ihrer spezifischen Verwendung. Dies gilt insbesondere für die Art der Interpretation der Sensordaten.

Schlüssel zur Einordnung. Als Schlüssel für eine Einordnung der in der Literatur beschriebenen vielfältigen Roboterarchitekturen wurde die Anordnung und das Zusammenwirken der drei Hauptblöcke *sense*, *plan* und *act* verwendet. Zwei dieser Elemente befassen sich mit der Anbindung der genannten Schnittstelle zur Außenwelt und dienen im konkreten Architekturaufbau als Platz für die spezifischen paradigmenspezifischen Methoden – das Vorhandensein der "Weltschnittstelle" hat gegenüber der Informatik zusätzliche Probleme eingeführt, die auf mitunter sehr unterschiedliche Weise adressiert werden.

Die wesentliche Differenz der bei Verwendung dieses Ordnungsschlüssel entstehenden Klassen von Roboterarchitekturen liegt jedoch in der Binnenstruktur – der Anordnung und dem Zusammenwirken der Komponenten (vgl. Schemata der Basisarchitekturen in Abb. 2.1.1, S. 25).

Gründe der Vielfalt. Die als Architekturmodelle vorgeschlagenen Lösungen unterscheiden sich wesentlich in zahlreichen qualitativen Merkmalen. Die wichtigsten sind die bereits angeführte strukturelle Gliederung, die Präferenz des Lokalitätsprinzips oder alternativ der Fusionierung von Sensordaten, die Abstimmung des Zugriffs auf die Effektoren, die Existenz und ggf. Funktion eines zur Laufzeit aktiven Planers, die eng damit verbundenen Funktion einer Repräsentation der Außenwelt samt der Nutzung von internen Zuständen, und letztlich der Grad und die Granularität der internen Parallelisierung. Diese qualitativen Merkmale variieren zwischen Vertretern verschiedener Architekturklassen erheblich, innerhalb einer Klasse jedoch nur mäßig.

Die Vielfalt an Vorschlägen ist zu einem gewissen Teil historisch durch die fortschreitende Entwicklung der Disziplin der Robotik bedingt. Praktisch allen der in 2.1 umrissenen Schemata kommt eine Berechtigung bereits durch den Beitrag zu, den sie mit der erfolgreichen – und oftmals erstmaligen – Lösung bestimmter Anwendungsprobleme leisten. Die Vielfalt der Ansätze ist somit ein zuverlässiger Indikator für die Vielgestaltigkeit der relevanten Anwendungsprobleme.

Alle wesentlichen quantitativen Merkmale von Roboterarchitekturen (die wichtigsten sind Takt und Reaktionszeit) sind hingegen emergente Resultate, die aus Funktionsprinzip, konkreter Implementierung und den Eigenschaften der technischen Plattform hervorgehen. Spätestens sobald quantitative Merkmale einbezogen werden, muß festgestellt werden, daß viele der vorgeschlagenen Architekturen die Probleme jeweils einer bestimmten Pro-

blemklasse nahezu exklusiv überzeugend – d.h. mit akzeptablem Aufwand sowohl in der Entwicklung als auch im Einsatz – lösen.

Paradigmenspezifische Anwendungsprobleme. Enge Grenzen sind der Austauschbarkeit von erfolgreichen Problemlösungen gegen solche gesetzt, die unter Zugrundelegung eines jeweils anderen Paradigma entstanden. Für viele Probleme existieren ausreichende und elegant zu formulierende reaktive Lösungen (wie die Potentialfeldverfahren in der lokalen Navigation, s. 2.1.3.2, oder die Erzielung eines robusten emergenten Verhaltens, ohne daß eine explizite prozedurale Notation vorgenommen werden muß, s. 2.1.3.4), während andere Probleme schon von der Art der an Symbolen orientierten Problemstellung her eine deliberative Lösung nahelegen (wie im Fall der hierarchischen Aufgabenteilung bei der Navigation im Rahmen eines mehrschrittigen Plans, s. 2.1.2.3).

Diese Spezialisierung geht im Fall der deliberativen und reaktiven Systeme so weit, daß umgekehrt diese Architekturklassen jeweils mit bestimmten Problemklassen in Verbindung gebracht werden, und sich mitunter alle Methodologie darin zu erschöpfen scheint, den zum Anwendungsproblem "passenden" Architekturtypus zu empfehlen. Da komplexe Anwendungsszenarien Teilprobleme beider Art enthalten können, konnte dies nicht die endgültige Lösung sein.

Probleme hybrider Ansätze. Da keiner der entwickelten deliberativen oder reaktiven Architekturentwürfe eine hinreichend effiziente und allgemeine Lösung darstellt, wurde als Folge die Entwicklung hybrider Architekturen unternommen, die Entwurfsmerkmale beider Vorgängertypen in sich vereinen sollen. Durch die erfolgte Einbettung eines deliberativen Planers und eines reaktiven Basissystems ist zumindest schlüssig, daß diese Systeme wenigstens prinzipiell über die erforderlichen Fähigkeiten beider zuvor genannter Systementwürfe verfügen sollten. Trotz der erheblichen Verdienste der hybriden Ansätze ist festzustellen, daß mit ihnen je ein strukturelles und ein (zumindest teilweise davon abhängiges) konzeptionelles Problem verbunden ist.

Das strukturelle Problem resultiert aus der nun erforderlichen Kopplung der reaktiven und deliberativen Komponenten, die neben der erweiterten internen Kommunikation die Hauptbestandteile hybrider Systeme ausmachen. Beide Teilsysteme weisen eine grundsätzlich voneinander abweichende Arbeitsweise auf und operieren zudem auf Daten gänzlich verschiedener Abstraktionsstufen. Im Rahmen der aktuellen Entwürfe wird der Ver-

such unternommen, durch Einfügen einer *sequencer*-Zwischenschicht eine Adaption von Planung und Ausführung vorzunehmen. Derzeit ist allerdings noch nicht vollständig geklärt, welche Form diese Wechselwirkung zweckmäßigerweise anzunehmen hat (vgl. [Gat99]).

Die sich auch teilweise aus diesem Sachverhalt ergebende konzeptionelle Schwierigkeit ist, daß kein geeignetes konstruktives Verfahren zum Systemaufbau besteht. Ein Verfahren zur Abbildung einer Dekomposition der Lösungen ganz verschiedenartiger Anwendungsprobleme auf die jeweils vorgeschlagene *eine* feste und unveränderliche Struktur einer hybriden Systemarchitektur erscheint daher nicht in Reichweite. Ungünstig macht sich weiter bemerkbar, daß die Architekturen mit zahlreichen Annahmen über Funktionseinheiten der Plattform (so der in 2.1.4.3 behandelte modellorientierte Ansatz, der die Integration von Sensoren anderer Modalität nicht ermöglicht) oder unter erheblichem Einfluß eines konkreten Anwendungsproblems spezifiziert wurden.

Bei der Systemkonstruktion besteht die grundsätzliche Notwendigkeit, die Lösungen aller Teilaspekte des Anwendungsproblems innerhalb der Gesamtarchitektur zu verorten, d.h. jede einzelne spezifisch reaktive bzw. deliberative Lösung in die entsprechende Schicht zu integrieren. Dies impliziert, daß durch diesen Prozeß unverändert eine anwendungsproblemspezifische – und nicht etwa eine allgemeine – Architektur entsteht: Auch hybride Architekturen entstehen also grundsätzlich anwendungsproblembezogen. Ihr Entwurf ist aufwendig und wegen der höheren Komplexität schlechter beherrschbar als der Entwurf konventioneller Systeme. Da dennoch eine erheblich umfangreichere Klasse von Anwendungsproblemen mit diesen Systemen erfolgreich bearbeitbar ist, ist die Beschäftigung mit hybriden Systemen unbedingt lohnend. Den Entwurf betreffend besteht allerdings ein erheblicher Systematisierungsbedarf.

Zusammenfassung. Die praktische Entscheidung zugunsten bestimmter Entwurfsparadigmen und letztlich der verwendeten Architektur erscheint unter den Hintergrund dieser Ausführungen klar anwendungsproblembezogen und basiert vorwiegend auf Heuristiken und Designtraditionen. Es bestehen keine klaren Designregeln. Robotikprojekte sind zum gegenwärtigen Standpunkt des Erkenntnisprozesses nicht etwa deshalb anspruchsvoll, weil benötigte Systemelemente – wie schnelle Rückkopplung noch im Fall der deliberativen Architekturen, oder die Option auf ein globales Gedächtnis bei den reaktiven Systemen – *fehlen*, sondern weil kein klarer und hinreichend allgemeiner Weg existiert, die verschiedenen Aspekte des Anwen-

dungsproblems auf eine vorab gegebene feste Struktur selbst einer hybriden Systemarchitektur *angemessen abzubilden*. Insoweit ist jede vorgeschlagene Architektur auf bestimmte Anwendungsprobleme, unter Umständen sogar auf nur eine einzige Gerätekonfiguration bezogen. Eine weitergehende Abstraktion von festgefügten Architekturen ist erforderlich, um zur Entwicklung allgemeiner oder zumindest zwecks Anpassung an verschiedenste Anwendungsprobleme leicht rekonfigurierbarer Systeme beizutragen.

3.2 Fragestellung

Ein wesentliches Ziel ist, die Grundlage zur Konstruktion von Robotersystemen zu schaffen, die möglichst einfach einer Vielzahl von wechselnden Anwendungsaufgaben angepaßt werden kann.

In Bezug auf einzelne (zumeist reaktive) Entwürfe wird für eine verwandte Eigenschaft der Begriff der *targetability* oder des *ecological niche fitting* verwendet: Wie muß ein Roboter beschaffen sein, um potentiell eine möglichst breite Nische auszufüllen? Diese Fragestellung beschäftigt sich allerdings nur mit dem Design eines einzelnen Systems. Sie soll hier in einem abweichenden Kontext angewandt werden: *Wie muß eine Grundlage zur Konstruktion einer Roboterarchitektur zweckmäßigerweise beschaffen sein, um leicht Systeme zur Lösung möglichst vielfältiger Probleme konstruieren zu können?*

Es steht nicht zu erwarten, daß eine spezifische Roboterarchitektur entwickelt wird, die ohne mehr oder minder umfassende Rekonfiguration ihrer Struktur eine elegante oder auch nur akzeptable Lösung aller denkbaren Anwendungsprobleme darstellen kann. Grundsätzlich werden dem Problem angepaßtere Architekturen eine einfachere und effizientere Lösung stellen können.

Wenn als Kernaufgabe – unabhängig aller Paradigmen – betrachtet wird, eine angemessene Struktur zur Bearbeitung von Anwendungsproblemen zu formulieren, die einer möglichst umfassenden Zahl von nicht vorab festgelegten Problemszenarien gerecht wird, wird die Antwort also nicht in Form einer statischen Architekturspezifikation erfolgen können, sondern in der Form eines Systems gegeben werden müssen, das eine Rekonfiguration von Roboterarchitekturen gestattet.

Vorhandene Ansätze. Die vorhandenen und bereits exemplarisch in Abschnitt 2.1 vorgestellten Architekturen sind nicht allgemein und bringen jeweils verschiedene Einschränkungen mit sich.

Zunächst ist unbedingt anzumerken, daß dies keineswegs einen Mangel der beschriebenen Systemarchitekturen darstellt. Diese sind zum Teil "gewachsene Systeme", die durch inkrementelle Entwicklung an einen bestimmten Einsatzkontext herangeführt wurden. Andere wiederum wurden als Demonstratoren zur Lösung von bestimmten bis zu ihrer Veröffentlichung nicht oder nicht angemessen lösbaren Problemen vorgestellt. Entsprechend dieser Zielsetzung ist festzuhalten, daß diese Leistung voll erbracht wird.

Bei der Entwicklung dieser Systeme wurde allerdings kein besonderer Wert auf die Adaptierbarkeit des Entwurfs (bis hin zur Metamorphose zu einer *anderen* Architektur) oder auch nur die Wiederverwendbarkeit von Teillösungen (speziell nicht im Rahmen womöglich völlig anders gearteter Architekturen) gelegt. Dies ist durchaus ein Mangel in Bezug auf das am Anfang dieses Abschnitts genannte Ziel, eine einfache Anpassung an wechselnde Anwendungsaufgaben vornehmen zu wollen.

Ziele dieser Arbeit. Ein Grunderfordernis ist also, eine Basisstruktur zu entwickeln, die zweckmäßig für die Konstruktion von Roboterarchitekturen unter den anfangs benannten Randbedingungen verwendet werden kann. Die dazu erforderlichen Eigenschaften sind im Detail

- die Abstraktion von feststehender Architektur und von konkreten Anwendungsproblemen, um eine möglichst weitgehende Allgemeinheit des Entwurfs zu gewährleisten,
- eine Trennung einerseits der Lösungskomponenten der Anwendungsprobleme von andererseits den Strukturkomponenten der Architektur, was eine leichtere strukturelle Rekonfigurierbarkeit ermöglichen soll,
- eine weitgehende und zweckmäßige Unterstützung von Modularität der Komponenten einer Problemlösung, die erst eine Option auf separate Testbarkeit, Benchmarks und Wiederverwendbarkeit der Systemkomponenten schafft,
- eine Einfachheit, die durch die Möglichkeit der einfachen Einbindung auch fremder Systeme – ohne ihnen für Rahmenwerke typische Randbedingungen aufzuerlegen – die Interoperabilität fördern soll.

Es geht also darum, adaptierbare (später vielleicht: strukturell adaptive) Systeme fertigen zu können. Um dies erreichen und mit Hilfe eines technischen Demonstratorsystems verifizieren zu können, werden weitere Zwi-

schenschritte erforderlich werden, die ab der Hardwareabstraktion beginnend untersucht werden.

Prüfbarkeit der Resultate. Da die wesentlichen Architektureigenschaften wie in 3.1 dargelegt ausschließlich qualitativer Natur sind, kann auch der Grad des Erreichens der Designziele einer "Metaarchitektur" nur qualitativ bestimmt werden. Anhaltspunkte sollen dabei sein, ob es überhaupt gelingt, auch die klassischen Architekturen unter Nutzung allein der Mittel der konzipierten generischen Trägerstruktur umzusetzen, welcher zusätzliche Aufwand dabei getrieben werden muß, und ob tatsächlich eine Entkoppelung, Herauslösung und Wiederverwendung von Systembestandteilen möglich ist. Auch das Potential solchermaßen konstruierter Architekturen für nachträgliche gezielte Änderungen ohne Eingriff in bestehende Systemkomponenten soll untersucht werden. Darüber hinaus sind die weiteren Eigenschaften der Metaarchitektur zu untersuchen und ihr Nutzen zu analysieren.

Zur Bedeutung der Fragestellung. Diese Arbeit soll keine weitere (wie dargelegt zwangsläufig problem- und systemgebundene) Architektur vorlegen, sondern einen Beitrag zur weiteren Systematisierung des Architekturentwurfs leisten. Dies ist sinnvoll, da sich auch in absehbarer Zukunft wohl keine Architektur als Innenstruktur von Robotersystemen als endgültig und universell erweisen wird: Zum einen werden auch nach über 50 Jahren der Entwicklung laufend neue Vorschläge in die wissenschaftliche Diskussion eingebracht und alte variiert, zum anderen entsteht mit einer sich stetig vergrößernden Basis eingesetzter Systeme verstärkt die Notwendigkeit der Anpassung alter Systeme an veränderte und oft komplexere Gegebenheiten.

In allen Fällen erscheint der Ansatz unzweckmäßig, standardmäßig ein vollständiges Neudesign vorzunehmen. Für das, was die Softwaretechnik an Entwurfssystematisierung für die Informatik beigetragen hat, gibt es in der Robotik noch keine Entsprechung: Die "Robotertechnik" ist ein gänzlich anderes Gebiet. Die Untersuchung der vorgelegten Fragestellung dieser Arbeit soll einen Schritt in diese Richtung darstellen.

3.3 Skizze des Lösungsansatzes

Um die systematische Entwicklung einer tragfähigen Struktur vornehmen zu können, werden verschiedene dabei auftretende Teilprobleme auf unterschiedlichen Schichten des Ansatzes bearbeitet.

Da die Betrachtung des Begriffs der Roboterarchitektur stets mit der Verwendung von Hardware verbunden ist, werden sowohl die Hardwareebene als auch die der als Software vorliegenden Gerätetreiber eine gewisse Rolle spielen. Da auf diesen Ebenen der Designfreiraum sich auf dieser Ebene auf die Wahl der Komponenten bzw. die Konstruktion neuartiger Komponenten beschränkt, soll hier keine vertiefende Betrachtung dieser Details erfolgen.

Aus der Entscheidung für eine Gerätekonfiguration resultiert oftmals eine durch technische Randbedingungen wie der Verfügbarkeit von Schnittstellen geprägte Struktur, die nach Möglichkeit jedoch keinen *bottom up*-Einfluß auf das System ausüben sollte. Auf der untersten hier zu untersuchenden Schicht wird also exemplarisch für ein Prototypensystem, einen mit zusätzlichen Aggregaten ausgestatteten Pioneer 2, eine Lösung für das Problem der nichtorthogonalen internen Kommunikation erarbeitet, um einen Zugangspunkt zu allen Systemdiensten zu schaffen und einen geordneten Zugriff auf eine im Detail den oberen Schichten unbekannte Hardware zu gestatten.

Die Verfügbarkeit einer orthogonalen Geräteschnittstelle leistet eine Abstraktion von den Gegebenheiten des Hardwareaufbaus, doch noch nicht von den eigentlichen Geräten selbst und den jeweils spezifischen Protokollen. Eine weitere Softwareschicht befaßt sich mit der Kapselung von Datenformaten und Protokollautomaten, um eine einfache Nutzung der Komponenten durch eine konventionelle Aufrufschnittstelle aus einer Standardprogrammiersprache zu gestatten. Diese Schnittstelle liegt in Form einer Klassenbibliothek vor. Auf dieser Ebene findet die Hardwareabstraktion statt, die einen Austausch einzelner Hardwarekomponenten oder selbst der gesamten Plattform durch funktionsverwandte Systeme ermöglicht, ohne daß die darauf aufbauenden Ebenen davon betroffen sein müssen.

Die Verwendung mehrerer Geräte (Sensoren, Effektoren) ist ein Spezifikum eines Verfahrens, das auf die Hardwarekonfiguration eines bestimmten Roboter zugeschnitten ist. Ähnliches gilt für die Verwendung eines jeden Geräts, dessen physische Montageposition in Relation zu der Roboterplattform Bedeutung hat. Da eine Berücksichtigung solcher Verhältnisse in einer generischen Bibliothek nicht sinnvoll erscheint, werden derartige Verfahren

– zunächst separat – in Form unabhängiger Subsysteme spezifiziert. Für die Ausstattung der Demonstratorplattform werden einige solcher Subsysteme vorgestellt. Subsysteme kapseln die durch Verfahren beschriebenen Fähigkeiten einer Roboterplattform.

Die sinnvolle Verbindung ausgewählter Subsysteme zu einem funktionierenden Gesamtsystem ist die Aufgabe der letzten Schicht: Auf ihr wird die Architekturkonfiguration unternommen. Es ist erstrebenswert, daß beim Entwurf von Subsystemen keine besondere Rücksicht auf implizite Randbedingungen genommen werden braucht, die sich aus der Präsenz oder Abwesenheit weiterer Subsysteme ergeben. Aus diesem Grunde muß eine spezielle Methode der Kommunikation verfügbar sein, die den ansonsten unabhängig voneinander ausgeführten Subsystemen eine Interaktion gestattet. Aus der speziellen Randbedingung, daß auch die Abwesenheit oder temporäre Nichtverfügbarkeit eines Kommunikationspartners kein Blockieren eines möglicherweise mit eigenen Pflichten versehenen Subsystems auslösen darf, ergibt sich die Struktur der indirekten Kopplung über Kanäle mit besonderen Eigenschaften. Zur Erleichterung des Zusammenfügens von Subsystemen zu einem vollständigen System wird weiterhin eine Instanz benötigt, die unter Berücksichtigung von Subsystemanforderungen und -leistungen angemessene Verknüpfungen erstellt. Diese Verknüpfungen müssen zudem dynamisch aktualisiert werden, sobald die Konfiguration durch Zu- oder Entladen von Subsystemen zur Laufzeit verändert wird, damit die Leistung des Systems entsprechend erweitert wird bzw. eine Restfunktionalität erhalten bleibt. Eine weitere Variante der strukturellen Änderung wird durch das Überladen bereits existierender Kanäle ermöglicht. Das auf diese Weise mögliche nachträgliche Einbringen von weiteren Subsystemen in Verarbeitungsketten gestattet die transparente Anpassung und Erweiterung auch von Systemen, die nicht unter Berücksichtigung späterer Modifikation konzipiert wurden.

Um abschließend zu demonstrieren, daß die gewählten Strukturmittel für eine Abbildung der üblichen Systemformen ausreichend sind, wird für ein bestimmtes Anwendungsproblem je eine Referenzimplementierung in reaktiver, deliberativer und hybrider Manier vorgestellt. Bei dieser Gelegenheit wird zudem deutlich, daß bei Verwendung des vorgestellten Verfahrens kein besonderer Mehraufwand erforderlich wird, und daß in hohem Maße eine Wiederverwendung von Subsystemen, die nicht für einen Einsatz in anderen Architekturen spezifiziert waren, geschehen kann.

Kapitel 4

Systemkonzeption, Realisierung und Bewertung

Wissenschaft treiben ist auch immer Konstruieren.

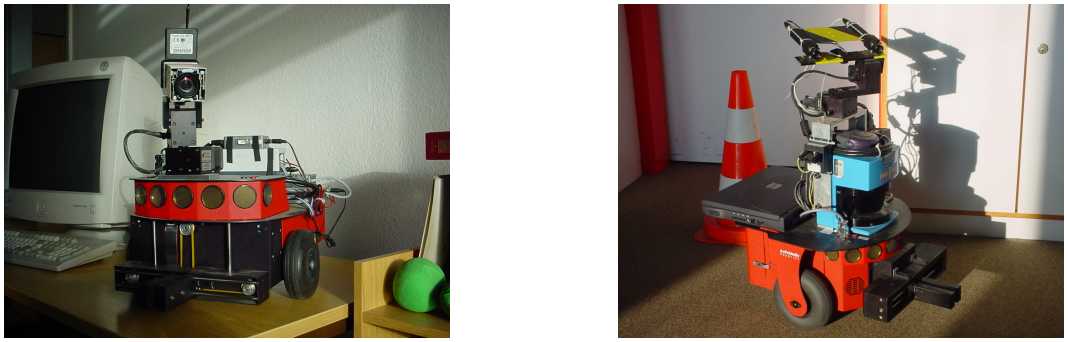
Cassirer

Die abstrakte Aufgabenstellung, ein System zu konzipieren, das wiederum die Konstruktion von Roboterarchitekturen unter den in 3.2 genannten Randbedingungen des Entwurfs gestattet, bedarf zunächst einer weiteren Konkretisierung. Im Fall der vorliegenden Arbeit wurden zwei Experimentierplattformen (Abbildung 4.1) und verschiedene Simulatorsysteme (vgl. s. [BBBM01]) als Zielsysteme eingesetzt, um eine Verifikation unter den Randbedingungen verschiedenartiger Geräteanbindungen vornehmen zu können.

Unter Verwendung konkreter Hardware – also exemplarischer Vertreter der in 2 vorgestellten Systemkomponenten mit gerätespezifischen Eigenschaften – ist Arbeit auf mehrere Integrationsstufen zu leisten, um das für den Architekturentwurf erforderliche Abstraktionsniveau zu erreichen. Die im Anschluß vorgestellte Gliederung des *bottom up*-Entwurfs mit anschließender Bewertung der Resultate soll als Struktur dieses Kapitels dienen.

Ebenen des Entwurfs. Während der Konstruktion eines vollständigen Robotersystems werden Designentscheidungen auf den im Folgenden beschriebenen Ebenen anstehen.

1. Auf der *Hardwareebene* findet neben der mechanischen Montage die elektrische Verkabelung von Hardwarekomponenten statt.



(a) Pioneer I

(b) Pioneer II

Abb. 4.1: Verwendete Experimentierplattformen.

Im gegebenen Fall standen zum freien Aufbau eine ActivMedia Pioneer 2-Basis (mit frontalem Sonar), ein 2-DoF-Greifer, ein SICK Laser Range Finder des Typs LMS 200, eine ActivePerception Pan/Tilt-Unit, zwei Kameras SONY XC-999, ein Bordrechner (Notebook P III 600 MHz), ein HT Framegrabber, zwei WLAN-Karten (IEEE 802.11b) und verschiedene externe Steuerrechner zur Verfügung. Einen Eindruck vom mechanischen Systemaufbau der mobilen Komponente kann Abb. 4.1(b) vermitteln. Der in 4.1(a) abgebildete Pioneer 1 wurde im Rahmen eines früheren Projekts zu einer Fallstudie für *active vision*-Systeme [BHJ+99] genutzt und verfügt bereits über eine technische Systemanbindung.

2. Auf der gerätespezifischen *Treiberebene* werden im Fall des Framegrabbers und der WLAN-Anbindung Gerätetreiber der jeweiligen Herstellers eingesetzt; die anderen Hardwarekomponenten können über standardisierte Schnittstellen angesprochen werden.

   Hardware- und Treiberebene werden in dieser Arbeit *nicht* behandelt, da an dieser Stelle keine architekturrelevanten Entscheidungen zu treffen waren.

3. Auf der hardwarestrukturspezifischen *technischen Kommunikationsebene* wird ein softwaremäßiger Zugang zu den Geräteschnittstellen geschaffen, da nicht alle Geräte unmittelbar mit dem Steuerrechner verbunden werden können.

   Auf dieser Ebene entstehen *bottom up*-Einflüsse, die auf konkrete Ei-

genschaften der Komponenten und ihre technische Kommunikationsstruktur zurückzuführen sind.

Da explizit nicht erwünscht ist, daß diese Strukturspezifika einen Einfluß auf die höheren Ebenen ausüben, sind bereits an dieser Stelle Entwurfsentscheidungen zu treffen, die in 4.1 erörtert werden.

4. Auf der Ebene der sensor- und effektorspezifischen *Protokolle* steht bereits eine Kommunikationsverbindung zum Gerät zur Verfügung, das somit über ein geräte- und herstellerspezifische Kommunikationsprotokoll angesprochen werden kann. Das Ziel ist hier, auf erhöhtem Abstraktionsniveau einen hardware- und protokollunabhängigen Zugang zu schaffen. Dies ermöglicht, aus einer höheren objektorientierten Programmiersprache elementare Operationen auf Komponenten der Hauptblöcke *sense* und *act* vorzunehmen.

   Die Maßnahmen zur Schaffung eines möglichst generischen Zugangs zu Roboterhardware werden in 4.2 behandelt.

5. Auf der Ebene der *Subsysteme* werden – ggf. mit lokaler Hierarchie – die eigentlichen interagierenden Architekturkomponenten implementiert.

   Eine unmittelbare Programmierung auf Ebene der elementaren Aktionen ist denkbar, würde aber zum Entstehen eines monolithischen Systems führen. In Subsystemen werden also zweckmäßige Einheiten der Verarbeitung gekapselt. Dies könnten beispielsweise Verfahren zur Sensordatenverarbeitung (bei Systemen mit reaktivem Anteil: Anwendung einer Funktion zur Berechnung von Steuerinformation, bei Systemen mit deliberativem Anteil: Sensorfusion, Merkmalsextraktion, virtuelle Sensoren), ggf. der Planung (Planer, globaler Speicher) oder der effektorseitigen Steuerung (z.B. *arbiter*) sein.

   In Abschnitt 4.3 werden die auch im Rahmen der Testszenarien benötigten Subsysteme vorgestellt.

6. Auf der Ebene der *Architekturkonfiguration* werden die Interaktion der Subsysteme spezifiziert und die entsprechenden Kommunikationswege implementiert. Dabei sind verstärkt die in 3.2 vorgestellten Randbedingungen zu beachten, die Modularisierung, Testbarkeit und Rekonfigurierbarkeit sicherstellen sollen.

   Abschnitt 4.4 beschreibt die gewählte Lösung, demonstriert die Leistung des gewählten Ansatzes und vergleicht und bewertet ihn im Vergleich mit den in Kapitel 2 beschriebenen Ansätzen.

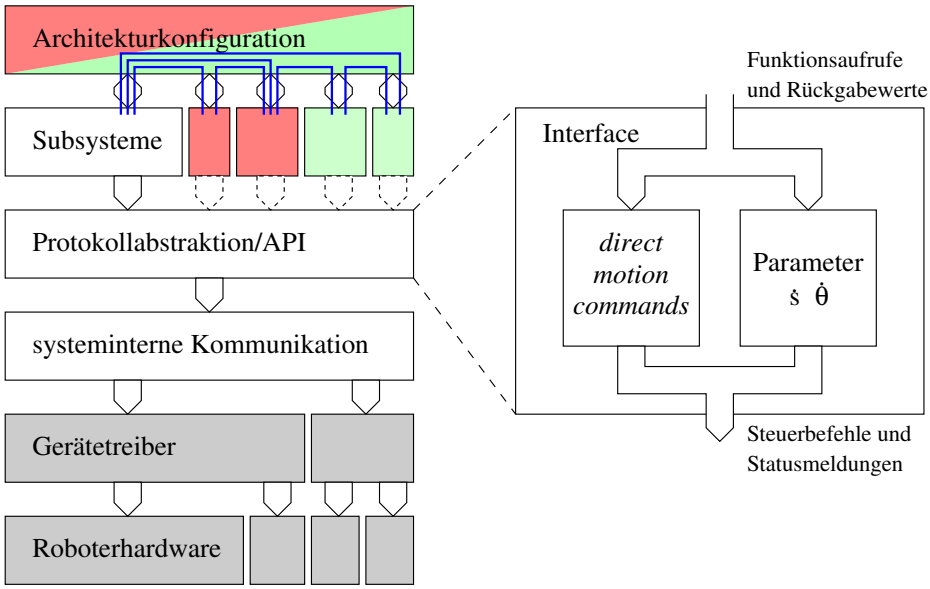


Abb. 4.2: Verwendetes Schichtenmodell. Subsysteme werden durch die Architekturkonfiguration miteinander vernetzt und können sich der Protokollabstraktionsschicht bedienen, um mit der Systemhardware zu interagieren. Auf dieser Schicht treten u.U. Dienstegruppen auf, deren gleichzeitige Verwendung ausgeschlossen ist (rechts). Die systeminterne Kommunikation dient der transparenten Anbindung der verteilten Hardwareinstanzen (die darunterliegenden Schichten werden in dieser Arbeit nicht behandelt).

Im grundsätzlichen Aufbau wird also ein Robotersystem als aus verschiedenen Schichten zusammengesetzt betrachtet werden. Diese Sicht entspricht der eines Rechners, der sich (aus Sicht des Softwareentwicklers) aus physisch faßbarer Hardware, Treibersoftware zur Ansteuerung spezieller Hardware und dem höhere und kanonisierte Dienste erbringenden Betriebssystem zusammensetzt. Abbildung 4.2 zeigt ein Schema der zu diesem Zweck eingeführten Schichten.

In allen unteren Ebenen liegt der Entwurfsschwerpunkt darauf, jeglichen *bottom up*-Einfluß der Zufälligkeiten technischer Gegebenheiten zu vermeiden und entsprechend möglichst keine Strukturvorgaben zu implizieren, die allgemein die Instantiierung einer vom Entwickler zu spezifizierenden Architektur unmöglich machen könnten.

Erst ab der Ebene der Subsysteme besteht überhaupt eine Affinität jeweils bestimmter Subsysteme zu bestimmten Architekturparadigmen, was wie in 2.2.1.1 erläutert auf die spezifische Art der Verarbeitung zurückzuführen und damit unvermeidbar ist. Bei Subsystemen handelt es sich also um Komponenten, die zur Konstruktion von Roboterarchitekturen eines bestimmten Typs verwendet werden. Da bei Nutzung des darunterliegenden Interface, das eine Interaktion mit den Hardwarekomponenten des Robotersystems gestattet, mitunter Gruppen von Diensten existieren, die zueinander inkompatibel sind (in Abb. 4.2 ist als Beispiel die Steuerung der Roboterbewegung über die Kontrolle der Translations- und Winkelbeschleunigung vs. der sog. *direct motion commands* angeführt), bietet es sich an, ohne Vorwegnahme von Architekturentscheidungen zur Vermeidung von Konflikten ein designiertes Subsystem als Schnittstelle zum Roboter zu verwenden.

Die Ebene der Architekturkonfiguration gestattet schließlich, unter freier Zusammenstellung der verfügbaren Subsysteme eine dynamische Wahl der eigentlichen Architekturentscheidung zu treffen. – Die in 4.3 eingeführten Subsysteme sind auch in diesem Sinne keineswegs als funktional vollständige Menge von Bauelementen für Architekturen jedes beliebigen Typs zu verstehen, sondern lediglich als genau die Bausteine, die zur Erzeugung der Beispielarchitekturen benötigt werden.

In den folgenden Abschnitten 4.1 bis 4.4 werden die Details der vier hier relevanten Schichten behandelt, während Abschnitt 4.5 sich mit der Zusammenfassung der Verifikationsergebnisse für den Gesamtentwurf befaßt.

4.1 Hardwarestruktur: Ausgangspunkt der Konstruktion

Auf dieser Ebene ist eine Komponentenintegration der technischen Komponenten – der Sensoren, der Effektoren und der Plattform – dergestalt vorzunehmen, daß eine Kommunikation mit jedem einzelnen Gerät von einem Punkt aus möglich wird. Dies ist eine notwendige Vorbedingung für die Nutzung der verfügbaren Hardware.

4.1.1 Problem

Da mit dem hauptsächlich eingesetzten Pioneer 2 eine kommerzielle Trägerplattform verwendet wurde, sind zur Herstellung einer Basisfunktionalität keine besonderen Arbeiten auf dieser Ebene erforderlich. Die Erweiterung um die zusätzlichen Sensoren (LRF, Kameras) und Effektoren (PTU) erfordert hingegen zusätzlichen Aufwand, da diese technischen Subsysteme weder von der den Roboter steuernden Software noch von der Bordelektrik explizit vorgesehen sind. Im Falle der verwendeten Plattform werden zwar prinzipiell sowohl LRF des eingesetzten Typs als auch Kameras unterstützt, doch nur in Verbindung mit einem proprietären (und in diesem Fall nicht verfügbaren) Bordrechner. Im Fall einer über die Systemsoftware Saphira anzubindenden und so in das System zu integrierenden Kamera geht die Unterstützung zudem über eine einfache Farb-*blob*-Erkennung nicht hinaus [Kon97].

Eine erfolgreiche Integration der zusätzlichen Sensoren in die mit der Roboterplattform gelieferte Betriebssoftware ist nicht möglich, da die bereits in 2.1.4.3 dargelegten Beschränkungen des in dieser Software verwendeten modellorientierten hybriden Stils nur die Eingliederung von Sensordaten in das Weltmodell zulassen, in denen allen Merkmalen zweidimensionale Koordinaten zukommen. Zudem verfügt auch der vorgesehene Bordrechner nicht über hinreichende Leistung (600 MHz P-III), um die geplante Verarbeitung durchführen zu können; eine vollständige Treiberunterstützung besteht zudem nur für eine Betriebssystemoption des Bordrechners, die nicht als Entwicklungsplattform in Frage kommt.

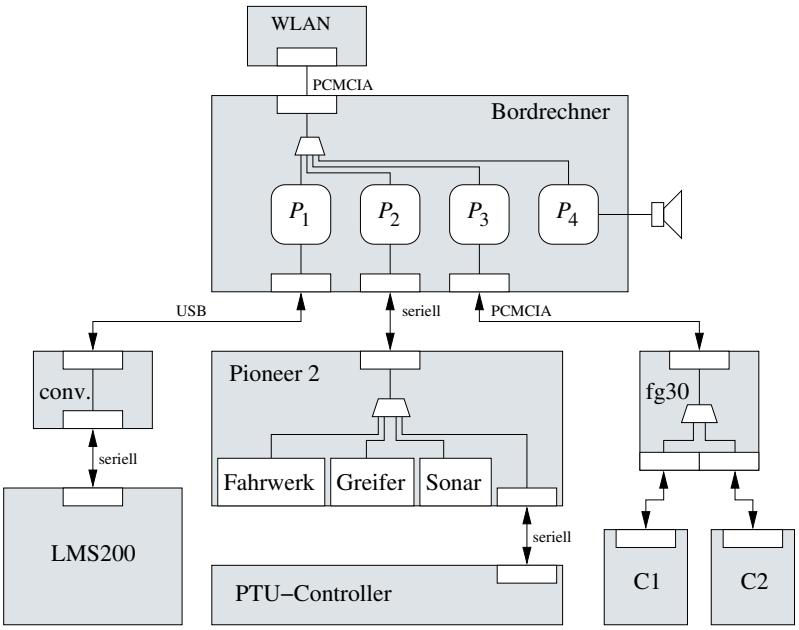


Abb. 4.3: Kommunikationsstruktur in Versuchsaufbau Pioneer-2: die gewählten Verbindungen sind von Schnittstellenverfügbarkeit diktiert und aus Sicht des Architekturentwurfs akzidentell. Die Serverstruktur (s. Text) ermöglicht dennoch den von einem Punkt ausgehenden Zugriff auf die Hardwarekomponenten, allerdings nur unter Beachtung des spezifischen Geräteprotokolls.

4.1.2 Lösung und Ergebnisse

Um einen zentralen Zugriff auf die irregulär verteilt anfallenden Sensor- und Effektorschnittstellen zu ermöglichen, werden auf dem Bordrechner vier separate Prozesse ($P_1 \ldots P_4$ in Abb. 4.3; mit Ausnahme von P_2 sämtlich im Rahmen dieser Arbeit entwickelt) eingesetzt, um über drahtlose Netzwerkverbindungen die Dienstleistungen der Roboterhardware in Anspruch nehmen zu können. Im einzelnen dienen die Prozesse den folgenden Zwecken:

P_1 dient der Kommunikation mit dem LRF. Der Mangel an seriellen Schnittstellen des Bordrechners wird über eine transparente Umsetzung zwischen USB und seriellem Port des LRF behoben.

P_2 ist ein über den Hersteller des Roboters verfügbares Programm mit der Bezeichnung `ipthru`, das eine P_1 vergleichbare Aufgabe für den Roboter betreffende Datenpakete erfüllt und eine Anbindung der Roboterbasis und der über diese erreichbaren Sonarsensoren ermöglicht.

P_3 dient der Anbindung der Kameras unter Nutzung des Framegrabbers und der Treiber des Herstellers. Im Fall dieser Sensoren wird eine lokale Vorverarbeitung vorgenommen, da zum einen keine direkte Abbildung des Geräteprotokolls mit direktem Zugriff auf I/O-Ports auf ein Netzwerkprotokoll sinnvoll ist, und zum anderen eine Datenreduktion durch Umsetzung eines ggf. angeforderten Subsamplings besteht. – Eine andere Arbeit befaßt sich mit der Anwendung von Codierungs- und Kompressionsverfahren, um eine weitere Reduktion des zu übertragenden Datenvolumens zu erreichen [Hol03].

P_4 ermöglicht über ein eigenes Protokoll den Transfer von Dateien (ähnlich FTP) und das Starten von Prozessen (ähnlich RPC) auf dem wegen des eingesetzten Betriebssystems ansonsten zu diesen Diensten nicht befähigten Rechner. Dieser Prozeß wird unter anderem zur Ausgabe von Klangdateien aus der Sprachsynthese verwendet.

Auf die beschriebene Weise besteht via Netzwerk ein praktisch vollständiger softwaremäßiger Zugang zu sämtlichen Geräten.

Die Auslagerung der eigentlichen Verarbeitung erlaubt zudem, die unzureichende *on board*-Rechenkapazität zu ergänzen und eine angemessenere Betriebssystemumgebung für die Entwicklung einzusetzen, führt allerdings auch eine zusätzliche Latenz und Verzögerung durch die limitierte Datenrate der Netzwerkverbindung (max. 11 MBit/s) ein.

4.2 Geräte- und Protokollabstraktion

Die Hardware- und Protokollabstraktionsschicht bedient sich der Möglichkeiten der systeminternen Kommunikation, um unabhängig vom Ort der jeweiligen Hardware und der tatsächlichen Schnittstellenverbindung Zugriffe vorzunehmen.

Um den Prozessen, die diese Geräte einsetzen zu gestatten, auch von weiteren Gegebenheiten wie dem geräte- und herstellerspezifischen Protokoll zu abstrahieren, sind weitere Maßnahmen erforderlich. Eine möglichst weitgehende Abstraktion von diesen akzidentellen Eigenschaften bringt den Nutzen einer Austauschbarkeit von Geräten gegen ähnliche Typen, die Option zum Einsatz der Softwaresysteme auf anderen Hardwareplattformen und die Möglichkeit zum transparenten Austausch der Experimentierplattform gegen eine Simulationsumgebung. Dies stellt für Tests und Qualitätsbewertungen eine Grundlage wiederholbarer Experimente dar.

4.2.1 Problem

Die Protokolle, die die Hersteller der Baugruppen vorgesehen haben, fallen beispielsweise durch zeitliche Randbedingungen der Kommunikation oder die Wahl verschiedener Koordinatensysteme und Einheiten extrem uneinheitlich aus. Die Wahl von verschiedenartigen Komponenten mit abweichenden Schnittstellen[1] liefert keine orthogonale API, sondern eine Sammlung unterschiedlichster Schnittstellen mit stark variierenden Anforderungen. In vielen Fällen kann eine Kommunikation mit dem jeweiligen Gerät zudem nicht zustandsfrei erfolgen. Der in solchen Fällen erforderliche Protokollautomat würde eine Nutzung aus rein reaktiven Architekturen ohne weitere Maßnahmen verbieten.

Die Aufgabe dieser Schicht sieht zudem vor, eine weitgehende Generizität zu erreichen und eine Austauschbarkeit der Plattform (für *cross-platform*-Entwicklung) oder ihrer Komponenten zu gewährleisten. Im Fall dieser Arbeit war eine Anforderung, zusätzlich zur vorgestellten Hardware weitere Experimentalplattformen zu steuern, beispielsweise den in Abb. 4.1(a) gezeigten Pioneer I[2] und ein Framework zur Simulation virtueller Roboter [BB01, BBBM01].

[1] Insbesondere abweichenden Schnittstellen*konzepten*; die kannibalistische Natur der Robotik führt mitunter zur Nutzbarmachung von nicht explizit für die Robotik entwickelten Subsystemen [Mur00].

[2] Das Gerät wurde im Rahmen von [Ros98] in Betrieb genommen. Das Gerät ver-

Um diese Probleme auflösen zu können, muß nicht nur die bereits erfolgte Trennung von der Hardwarestruktur, sondern auch von den Spezifika des technischen Systems erreicht werden. Das Ziel ist also die softwaretechnische Schaffung einer angemessenen und allgemeinen Programmierschnittstelle.

4.2.2 Vorgehen

Im Gegensatz zu den unteren Schichten tut sich hier bereits ein größerer Spielraum für die Art der Lösung auf. In 4.2.2.1 wird dieser Raum erkundet, während 4.2.2.2 den gewählten Ansatz – den einer Klassenbibliothek – näher beschreibt. Mit der Umsetzung in Bezug auf die Gerätekomponenten der Experimentierplattform und den Maßnahmen, die der Kompensierung bestimmter unerwünschter Geräteeigenschaften dienen, beschäftigt sich der anschließende Abschnitt 4.2.3.

4.2.2.1 Ansätze

Es existieren bereits verschiedene konzeptionelle Ansätze, die alle dem Zweck der Schaffung einer angemessenen Programmierschnittstelle dienen sollen; insbesondere lagen zu drei der Alternativen bereits innerhalb der Arbeitsgruppe entwickelte Arbeiten vor. Im einzelnen sind dies der Versuch einer geeigneten Erweiterung von Saphira [Ros98], die Verwendung von Khoros [BHJ+99], der im Kontext der Simulationsumgebung verwendete Rahmenwerksansatz [BBBM01] und darüber hinaus die Spezifikation einer problembezogenen Sprache wie der objektorientierte Klassenbibliotheksansatz.

Mit Saphira liegt ein System vor, das eine integrierte Programmierumgebung bietet [Kon97], mit der aus [Ros98] Erfahrungen vorliegen. Die alternative Verwendung einer Simulationsumgebung wird ebenfalls unterstützt. Zwei Gründe sprechen gegen die Verwendung dieses Systems. Erstens erfordert die Integration von Sensoren, alle Sensordaten in das feststehende Modell der hybriden modellorientierten Architektur eingliedern zu müssen. Zwar müssen unter Saphira laufende Prozesse nicht die Kommunikationspfade der vorgegebenen Systemarchitektur benutzen, sodaß sich auch andere Architekturmodelle umsetzen ließen, doch führt am globalen Modell als Ort der Aufbewahrung von allen Perzepten kein Weg vorbei. Ei-

fügt nicht über einen eigenen Bordrechner, sondern ist über mehrere analoge Datenfunkstrecken und eine Bildfunkstrecke für den Kameraaufbau angebunden [Sch99], [Bit00].

ne Integration von Sensoren mit einer vom Entwickler nicht vorgesehenen Modalität ist wegen der fixierten Datenstruktur jedoch nicht möglich. Der zweite Grund resultiert aus einer impliziten Randbedingungen des Rahmenwerks: in einem aktivierten Prozeß muß nach spätestens 100 ms die Kontrolle an das Rahmenwerk zurückgegeben werden, damit das ebenfalls in Zeitscheiben abzuwickelnde Protokoll mit der Plattform nicht gestört wird. Dies ist weder mit aufwendigeren Bildverarbeitungsoperationen noch überhaupt mit Verfahren vereinbar, zu denen keine Zusicherungen über eine maximale Ausführungszeit gemacht werden können.

Im Rahmen der Entwicklung von Komponenten eines sog. aktiven Sehsystems [BHJ+99] wurde der Einsatz von Khoros bei der Konstruktion einer Systemarchitektur untersucht. Der Robotikanteil dieses Systems erforderte die Steuerung der in 4.1(a) abgebildeten Pioneer I-Experimentierplattform. Die Software Khoros ist ein Rahmenwerk, das primär der datenflußgesteuerten Modellierung von Bildverarbeitungsoperationen dient. Da innerhalb der in Verarbeitungssequenz aktivierten Teilprogramme keine persistente Zustandsinformation gehalten werden kann, wurde es im angeführten Fall erforderlich, die Resultate der Bildverarbeitungsoperationen über ein Netzwerkprotokoll zwischenzeitlich im globalen Modell des ebenfalls eingebundenen Saphira zwischenzuspeichern [BHJ+99, S. 154]. Ohne derartige Zusätze wäre Khoros offensichtlich nur zur Umsetzung zustandsfreier, d.h. rein reaktiver Roboterarchitekturen verwendbar; und auch die Notwendigkeit des fortwährenden Transports sämtlicher Zustandsdaten im Fall der Beschäftigung mit nicht rein reaktiven Systementwürfen scheint unzweckmäßig.

Mit [BBBM01] wurde ein Rahmenwerksansatz für eine einfache, leistungsfähige und von der konkreten Ausprägung der technischen Kommunikation unabhängige Schnittstelle zu den Versuchsaufbauten vorgelegt. Ein klarer Vorzug eines solchen Rahmenwerks ist, daß die Einhaltung semantischer und technischer Randbedingungen (z.B. in der zugelassenen Aufrufsequenz von Akzessorfunktionen) leicht sicherzustellen ist, da die Ablaufsteuerung überwacht werden kann. Auf den Stil von Software, die eine Robotersteuerung übernehmen soll, wird allerdings aus mehreren Gründen ein massiver Einfluß ausgeübt. Dies resultiert unter anderem aus verpflichtenden Namenskonventionen, der Forderung der Einhaltung von Echtzeitbedingungen (vgl. der 100 ms-Takt in Saphira), der Vorgabe einer *event loop*-Struktur oder von bestimmten *callback*-Schnittstellen. In Konsequenz ist die Integration einer rahmenwerkbasierten Robotersteuerung in andere Systeme wegen der komplexen Gesamtanforderungen an die Ausführungsumgebung praktisch unmöglich.

Als weitere Option existieren eine Reihe von speziellen Sprachen zur Steuerung von Robotern; dieser Ansatz ist überwiegend bei stationären Industrierobotern gebräuchlich [BJ83], wurde aber auch zur Spezifikation reaktiver Systeme mit Erfolg eingesetzt [Bro90a]. Auch das Saphira-System verfügt über einen integrierten Interpreter für die Sprache Colbert. Da allerdings nicht nur ein Betreiben des selbständigen, d.h. von anderen Erfordernissen losgelösten Roboters vorgesehen ist, sondern die Plattform auch im Kontext anderer Experimente und Untersuchungen verwendet werden soll, scheidet auch die Alternative der Spezifikation einer speziellen Programmiersprache wegen der fehlenden Integrationsoptionen aus.

Die Lösung durch eine Klassenbibliothek hat gegenüber den genannten Ansätzen die Vorzüge, daß eine Nutzung aus praktisch jeder Software heraus möglich wäre: die "Nachrüstung" nahezu beliebiger Software um Kommunikation mit einem Roboter wird leicht möglich. Stil und struktureller Aufbau von Client-Programmen, in denen der Roboter mit einer Auswahl seiner Systemkomponenten angesteuert werden soll, werden nicht durch eine vorgegebene Softwarearchitektur beeinflußt. Zudem ist der Aufwand für einfache Aufrufe geringer als der für die Einhaltung eines *peer to peer*-Protokolls. Da die Verantwortung für die korrekte Ablaufsteuerung allerdings allein bei dem die Bibliothek verwendenden Programm liegt, tritt als neue Schwierigkeit bei Verwendung dieses Ansatzes auf, die semantischen Randbedingungen der Nutzung von technischen Komponenten sicherzustellen. Einige Aggregate besitzen einen komplexen internen Zustand, und zu jedem Zeitpunkt ist statusabhängig nur eine bestimmte Untermenge der möglichen Bedienaktionen zulässig. Ein Lösungsweg kann sein, die konforme Nutzung der Hardware durch Abbildung auf geeignete Konstrukte objektorientierter Sprachen abzubilden. Da der Klassenbibliotheksansatz die geringsten *bottom up*-Einflüsse auf die übergeordneten Schichten vermittelt und dem weiteren Systementwurf die wenigsten Randbedingungen auferlegt, wird er weiter verfolgt.

4.2.2.2 Objektorientierte Hardwareabstraktion

Abstraktion von gerätespezifischen Protokollen. Der Einsatz einer objektorientierten Sprache ermöglicht eine vollständige Kapselung des Geräteprotokolls. Durch das Verbergen der Kommunikation mit den u.U. entfernten Geräten, zu denen durch Dienste der Kommunikationsschicht eine Verbindung besteht, kann eine Präsentation als Hardware-*proxy*-Klassen stattfinden. Im Fall von Geräten, die nicht über ein einfaches *request/reply*-Schema angesteuert werden, wird der Protokollautomat ebenfalls in den

entsprechenden Klassen untergebracht, um eine Nutzung auch durch zustandsfreie Verfahren zu ermöglichen. Dabei unternehmen deren Konstruktoren den initialen Verbindungsaufbau und hinterlegen die Verbindungsinformation (dies können je nach Kommunikationspartner Datei- oder Portdeskriptoren oder Referenzen auf *shared memory*-Abschnitte sein) in einem internen Zustand, während die Destruktoren für den geordneten Abbau der ggf. zustandegekommenen Verbindung zuständig sind.

Die Nutzung der Komponenten geschieht über eine Reihe von gerätetyp-, aber nicht protokollbezogenen

1. *configure*-Methoden zum Festlegen von Betriebsparametern,

   diese Funktionen setzen z.B. eine maximale Translationsgeschwindigkeit für den weiteren Betrieb der Plattform fest oder stellen die Auflösung ein, mit der folgende Sensorabfragen durchgeführt werden,

2. *query*-Methoden zur Abfrage von Betriebszuständen und Sensordaten,

   Funktionen dieses Typs geben Werte eines speziellen Datentyps zurück, so z.B. im Fall des LRF eine (im Fehlerfall leere) Liste von Scanpunkten (`list<point2>`), im Fall einer Kamera einen Bildtyp (`pic2d<unsigned char>`) etc., und

3. *command*-Methoden zur Übermittlung von Steuerungsinformation,

   diese Funktionen steuern beispielsweise Effektoren und geben typischerweise einen Resultatwert des Typs `bool` (Aktion erfolgreich oder fehlgeschlagen) zurück.

Technisch wird in dieser Weise objektorientiert gelöst, was in ähnlichen Lösungen durch Sensorvariablen oder in strukturierter Form durch einen sog. *state reflector* [Act00] vorgenommen wird. Ein erster Vorzug einer expliziten Abfrage gegenüber einer selbstaktualisierenden Sensorvariablen oder einer Struktur aus solchen Variablen wird spätestens bei der Verwendung von Sensoren mit hohem Datenaufkommen bei gleichzeitig begrenzter Kommunikationsbandbreite deutlich. Zusätzlich unterstützen viele Geräte einen Wechsel zwischen Betriebsarten, so z.B. die Kamera zwischen Farb- und Graustufenbildern bei zudem wechselnder Auflösung. Eine der Übermittlung durch *configure*-Methoden vergleichbar einfache Möglichkeit besteht bei Sensorvariablen nicht. – Mit Einführung einer Klassenbibliothek stellt sich die Situation nun wie in Abb. 4.4 dar: es besteht ein einheitlicher Zugangspunkt für Gerätedienste.

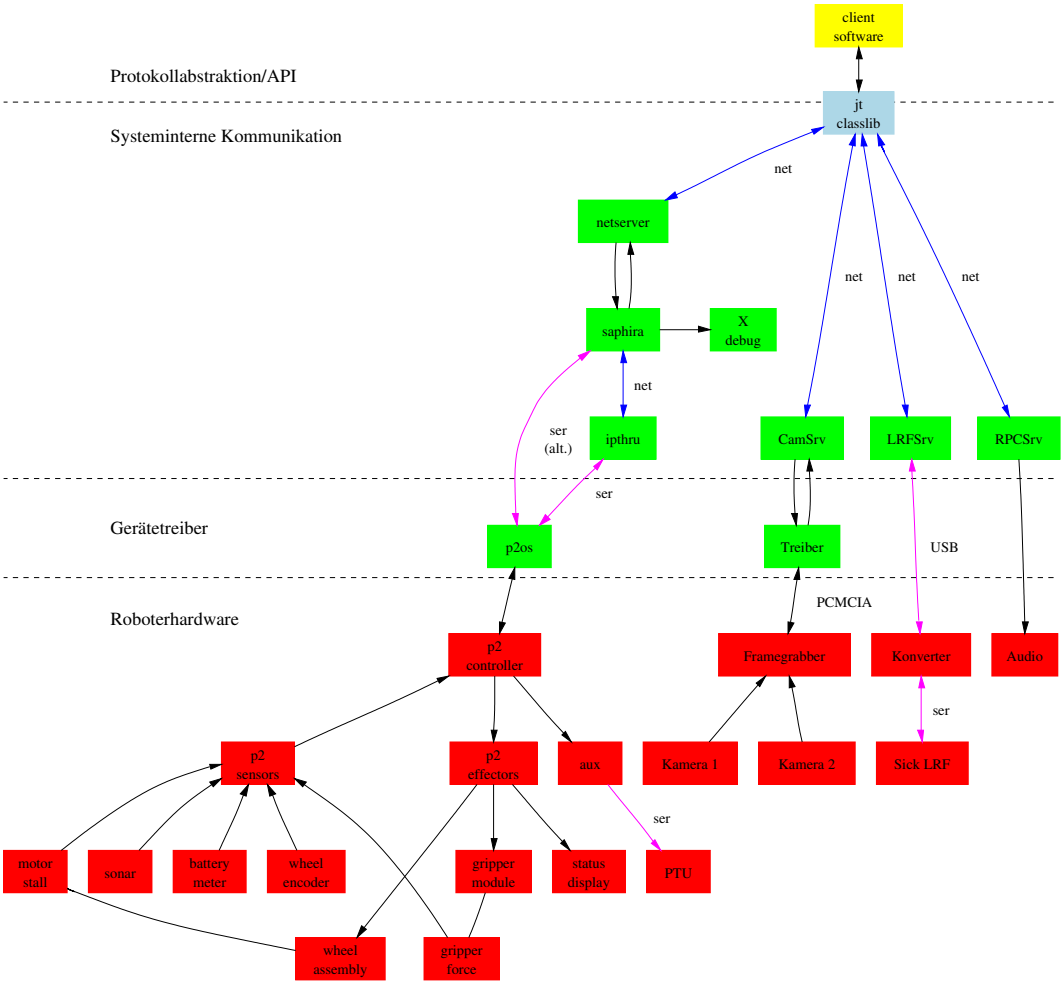


Abb. 4.4: Kommunikationspfade und Komponentenzugang über die Dienste einer Klassenbibliothek (vgl. Abb. 4.2). Unbeschriftete Kanten symbolisieren eine direkte elektrische Kopplung bzw. eine Aufrufschnittstelle.

Gruppen ähnlicher Geräte. Die Zusammenfassung von Geräten unter einer gemeinsamen Schnittstelle, denen nicht notwendig ähnliche Funktionsprinzipien unterliegen, doch die qualitativ vergleichbare Daten erzeugen oder konsumieren, erleichtert die Hardwareabstraktion. So ist beispielsweise die Testplattform mit zwei verschiedenen Typen von ToF-Sensoren ausgestattet: werksseitig mit Sonarsensoren, die zwei Quadranten der bevorzugten Bewegungsrichtung abdecken, und mit einem nachträglich montierten LRF mit gleichem Erfassungsbereich. Die Gruppierung beider Geräte unter einer abstrakten Oberklasse `tof_sensor` gestattet, Algorithmen wechselseitig mit beiden Datenquellen zu betreiben, soweit dies bei den Unterschieden in Qualität und Zahl der Messungen sinnvoll ist. Ähnliche Schnittstellen finden sich effektorseitig beispielsweise bei den *direct motion commands*, die eine Bewegung (Translation oder Rotation) der Plattform ungeachtet ihrer Fortbewegungsaggregate auszulösen vermögen.

Eine baumartige Strukturierung (vgl. Abbildung 4.5) mit extensiver Vererbung von Objekteigenschaften und Zugriffsmethoden sorgt für die gewünschte Einheitlichkeit der Schnittstellen.[3] Die Nutzung von Vererbung ermöglicht für solche Programme, die ausschließlich bestimmte Basiseigenschaften der Roboterplattformen nutzen, bereits einen gewissen Grad an Generizität. Durch Verwendung der Basisklassen (`autonomous` für die Plattform mit dem *differential drive*-Antrieb, `camera` und `tof_sensor` für die Sensoren) wird auf Basis der Klassenbibliothek geschriebene Software auf insgesamt fünf Plattformen (Pioneer I, Pioneer 2 und mehreren Simulatorplattformen) lauffähig.

Eine Eigenschaft, die allen Hardware-*proxy*-Klassen gemeinsam ist, hat damit zu tun, daß Geräte nicht wie Instanzen einer Klasse nach Belieben dynamisch instantiiert werden können. Um einen entsprechenden Mißbrauch auszuschließen, sind weitere Maßnahmen angezeigt.

Komponentensemantik von Hardware-*proxy*-Instanzen. Die Instantiierung von Objekten ist mit der Aufnahme eines Protokolls zur entsprechenden physischen Entität verknüpft und schlägt genau dann fehl, wenn keine korrespondierende kommunikationsbereite Hardwarekomponente existiert. Jede zu einem späteren Zeitpunkt stattfindende Replizierung eines erfolgreich instantiierten Objekts durch einen explizit formulierten oder sy-

[3] Die in einem komplexen Projekt erforderliche Dokumentation und die Generierung der Diagramme wurde mit Hilfe des Werkzeugs “Doxygen” vorgenommen [van02]. Eine graphische Benutzungsoberfläche erfordernde Komponenten wurden unter Verwendung der Bibliothek Qt [Tro00] erstellt.

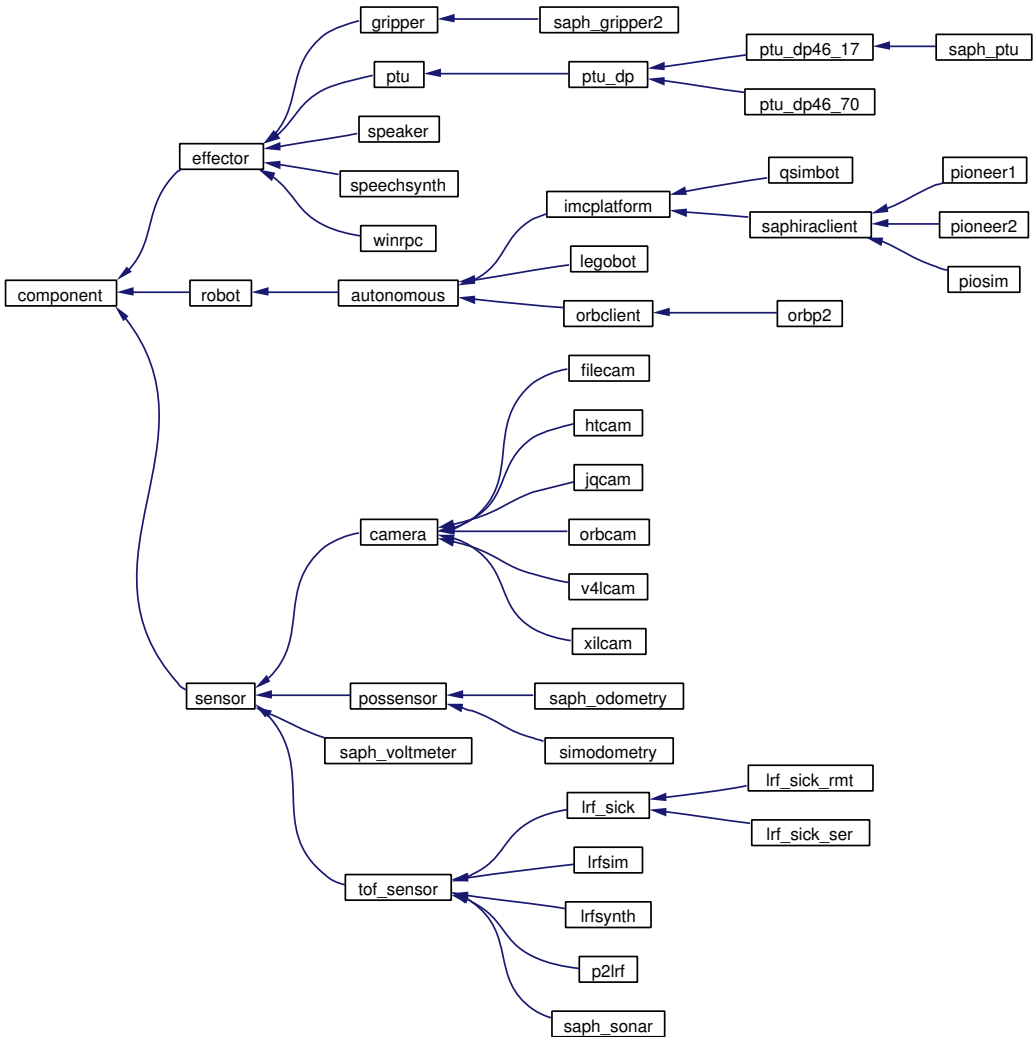


Abb. 4.5: Vererbungshierarchie in der Klassenbibliothek für Hardware-*proxy*-Klassen (Ausschnitt). Die Schnittstellenbeschreibung in gewöhnlichen oder rein virtuellen Oberklassen sichert die Einheitlichkeit der Schnittstelle und die Einhaltung der Benutzungssemantik zur Laufzeit.

stemintern bereitgestellten sog. *copy*-Konstruktor ist semantisch nicht sinnvoll, weil offensichtlich kein physisches Korrelat der Konstruktion existiert. Diese problematische Form von Wertsemantik tritt explizit beim Kopieren von Instanzen wie auch implizit bei ungeeigneter Form der Parameterübergabe auf. Im Umgang mit Instanzen von Klassen, die reale physische Objekte repräsentieren und dem Anwenderprogramm nur mehr als Kommunikationsschnittstelle dienen, ist also die Verwendung von *Wertsemantik* zu unterbinden: Die hier angemessene *Komponentensemantik* soll das Dereferenzieren (und ggf. noch die verwandte Bildung von Zeigern) zwar zulassen, doch die Bildung von getrennten Kopien grundsätzlich verhindern.

Der naheliegende Ansatz der Ableitung aller Klassen zur Komponentenrepräsentation von einer *singleton*-Klasse, die Mehrfachinstantiierung verhindert (vgl. [GHJV95]), löst zwar das Problem, ist allerdings als nur unvollkommen anzusehen, da durchaus mehrere Instanzen eines Typs (z.B. der Kameras) in einer Softwareeinheit bestehen können und sollen. Neben dem erwünschten Verbot der Duplizierung *einer* individuellen Instanz wäre generell auch das Konstruieren *verschiedener* Instanzen einer Klasse nicht mehr möglich. Im Fall der hier verwendeten Programmiersprache ist es möglich, das gewünschte Verhalten zu erreichen und ein Kopieren von Instanzen bestimmter Klassen grundsätzlich zu verhindern, indem ein *copy*-Konstruktor mit beliebigem, hier sogar leeren Rumpf spezifiziert wird, der als `private`-Methode gekennzeichnet wird: Diese Maßnahme führt zum Überladen des öffentlichen Standard-*copy*-Konstruktors, der eine elementweise binäre und damit zwangsläufig funktionsunfähige Kopie herstellt. Durch die Deklaration als `private` werden Kopierversuche bereits zur Übersetzungszeit entdeckt; auf diese Weise kann die gewünschte Komponentensemantik durchgesetzt werden. Da sämtliche Klassen, die Hardwarekomponenten repräsentieren, von einer gemeinsamen Klasse erben, ist es ausreichend, diese Maßnahme an einer einzigen Stelle zu treffen, nämlich in `component`.

Notation einer Gerätehierarchie. Einmal abgesehen von der *is a*-Hierarchie der Klassen einer Bibliothek tritt bei der Abbildung auf reale Systeme eine *has a*-Verknüpfung zwischen Geräten und ihren (mechanischen) Trägern auf. Diese ist mitunter strukturell stark von dem Schema der elektrischen Verbindung abweichend, das keinen *bottom up*-Einfluß über die Kommunikationsschicht hinaus ausübt. Die Komposition einzelner Instanzen entsprechend der physikalischen Kombination der vorhandenen Hardware erlaubt, die technische Konstruktion zu reflektieren und ermöglicht die Konstruktion einer parallelen Schöpfung als im Programm verfügbares und

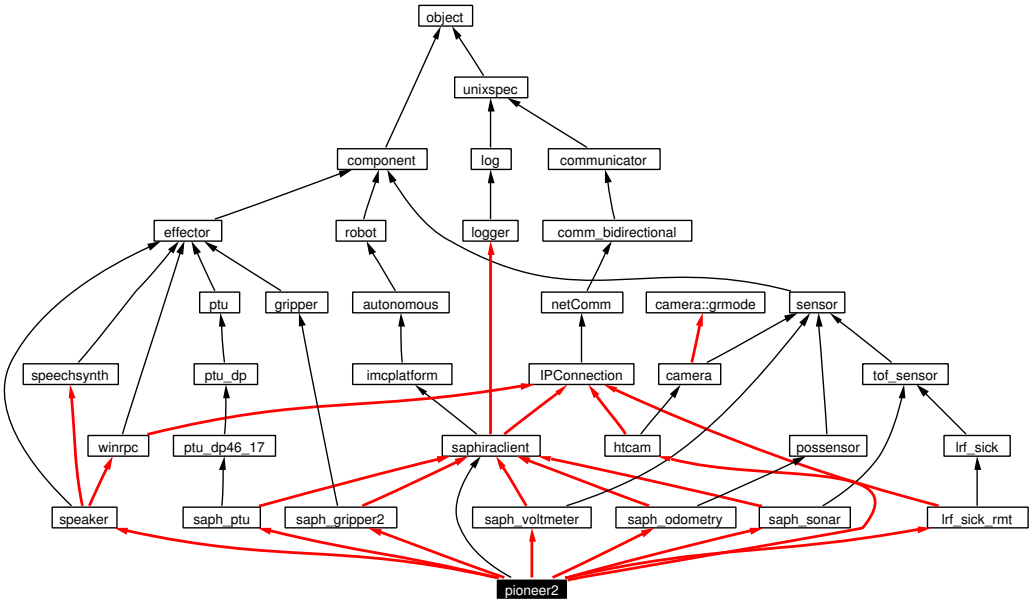


Abb. 4.6: Kollaborationsdiagramm für eine Plattform: das Diagramm zeigt die *is a*-Eigenschaft der Klassen (vgl. Abb. 4.5) und die *has a*-Verknüpfung der Instanzen.

manipulierbares Objekt (Abbildung 4.6 zeigt das zur gegenwärtigen Hardwarekonfiguration des Roboters gehörende Kollaborationsdiagramm).

Ob tatsächlich eine tief gestaffelte Anordnung der Geräte, z.B. von Kinematikketten, oder nur eine flache Zuordnung aller Geräte zur Roboterbasis vorgenommen werden soll, liegt im Belieben des Entwicklers, der die abgeleiteten Klassen spezifiziert. Für Abb. 4.6 wurde die zweite Möglichkeit gewählt. Geräte werden in beiden Fällen in *dotted notation* über ihre logischen Besitzer angesprochen.

4.2.3 Ergebnisse

4.2.3.1 Anbindung der Roboterbasis

Die Basisplattform ermöglicht im Wesentlichen nur die Fortbewegung und das Abfragen der Sonarsensoren. Sie ist mit einem *on board*-Mikrocontroller (Siemens 88C166, 20 MHz) ausgestattet. Ein serielles paketorientiertes Protokoll (RS-232, 19200 Baud) dient der laufenden Übermittlung von Informationen über Stellung der Radencoder, Messungen der Sonarsensoren und

einige andere Betriebsparameter. Durch eine besondere Software auf dem Bordrechner (P_2 in Abb. 4.3) wird dieses Protokoll über WLAN zum jeweils aktiven Steuerrechner weitergeleitet. Auf dem Steuerrechner dient eine Komponente der Software Saphira dazu, diese Protokollinformation zu decodieren bzw. Bewegungskommandos in Informationen zur Motorsteuerung umzurechnen.

Sobald eine Verbindung mit dem Roboter hergestellt wurde, beginnt dieser unaufgefordert, alle 100 ms-Zyklen Informationsblöcke zu liefern (SIP, *server information packets*). Dabei werden nicht nur geänderte Daten, sondern stets die vollständigen Information übertragen. Bleiben für jeweils mehr als 20 dieser Zyklen Steuerinformationen an die Roboterbasis aus, wird die Plattform aus Sicherheitsgründen angehalten und deaktiviert.

Um eine Trennung der Anwendungssoftware höherer Schichten von diesen zeitlichen Bedingungen der Kommunikation und weiteren Spezifika der Plattform zu ermöglichen, wurde an dieser Stelle ein zustandsbehaftetes System "`netserver`" integriert, das für die Kommunikation mit dem Roboter auf dieser Ebene zuständig ist. Das mit der Klassenbibliothek zur Steuerung des Roboters verbundene Anwendungsprogramm kommuniziert (über eine *socket*-Verbindung, um eine Verteilung auf mehrere Rechner zu ermöglichen) schließlich mit dem `netserver`, der die plattformspezifische Kommunikation vornimmt und die Einhaltung der Echtzeitbedingungen überwacht (vgl. Abb. 4.7). Dies setzt voraus, daß eine geeignete Abbildung der mechanischen und sensorischen Fähigkeiten der Roboterplattform auf Funktionsaufrufe stattfindet (das `netserver`-Protokoll umfaßt ca. 30 Kommandos; etliche werden allein für die Steuerung des Greifers benötigt).

Neben Kommandos zur Steuerung des `netserver` selbst (Austausch von Versionsinformation, Vermittlung zu verschiedenen technischen Plattformen, Festlegen von zu überwachenden Parametern wie der max. Geschwindigkeit etc.) muß konzeptionell zwischen synchron und asynchron auszuführenden Anweisungen unterschieden werden.

Die größere Gruppe der synchron ausführbaren Client-Kommandos erfordert die Übermittlung genau eines Antwortpakets, die nach vollständigem Abschluß oder festgestelltem Fehlschlag der ausgelösten Aktion erfolgt. Diese Variante wird beispielsweise beim Zugriff auf die internen Sensoren (Radencoder, Batteriespannung, Sonar) verwendet; unter anderem sind hier auch die sog. *direct motion commands* gelöst. Unternommene Ausführung synchroner Anweisungen gelingen entweder oder schlagen unmittelbar fehl.

Für einige Aktionen ist absehbar, daß die Ausführung einige Zeit (im Be-

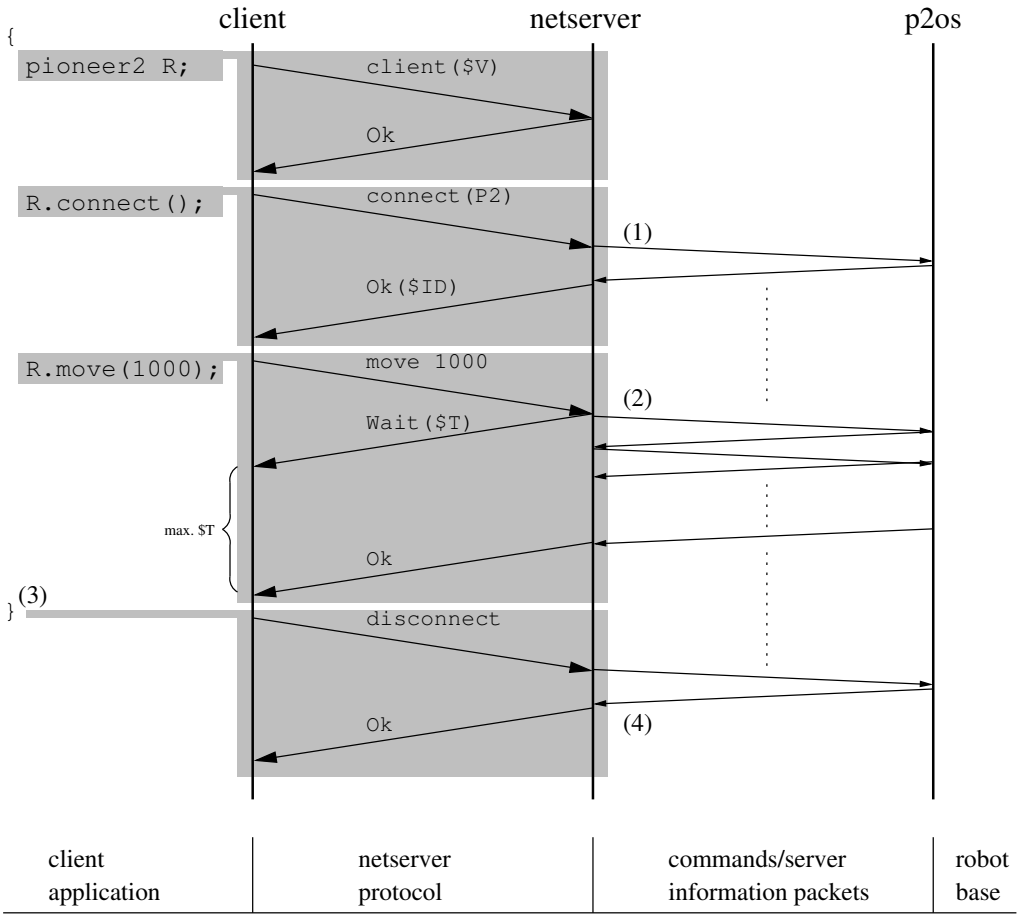


Abb. 4.7: `netserver`-Ort/Zeit-Diagramm. Nach Austausch der Versionsinformation `$V` (implizit im Code des Konstruktors) wird ein Verbindungsaufbau mit der selektierten Plattform (`P2`) durchgeführt; die Funktionsausführung endet erst mit dem Eintreffen der Rückmeldung. Ab (1) findet ein kontinuierlicher Datenaustausch zwischen `netserver` und Roboterbasis im 10 Hz-Takt statt; dargestellt sind nur die relevante Steuerinformation tragenden Pakete. Im Fall von Kommandos mit voraussichtlich längerer Ausführungszeit findet asynchrone Ausführung mit Übermittlung einer *timeout*-Schätzung `$T` statt (2). Auch durch einen impliziten Destruktoraufruf (3) wird ein geordneter Verbindungsabbau (4) herbeigeführt.

reich mehrerer Sekunden) in Anspruch nehmen wird. Generell gilt dies für sämtliche Kommandos zur Plattformbewegung. Die Ausführung solcher Kommandos erfolgt asynchron, da während der Bewegung weitere Aktionen des steuernden Programms z.B. zur sensorischen Überwachung des Fahrvorgangs möglich sein sollen. Aus diesem Grunde werden nur Beginn und Abschluß und ggf. der Abbruch der Bearbeitung solcher Kommandos signalisiert (vgl. Abb. 4.7, ab Zeitpunkt (2)). Für solche asynchron auszuführenden Anweisungen ist eine einfache zustandsbasierte Kontrolle innerhalb des `netserver` erforderlich. Als Abbruchbedingungen zur Entdeckung der Komplettierung oder des Fehlschlags von Aktionen sind die Unterschreitung einer Zieltoleranz, die Überschreitung des vorab übermittelten *timeout*-Wertes und der Empfang neuer, abweichender Steuerinformation vorgesehen. Dabei sind die ersten beiden Bedingungen plattform- und anwendungsspezifisch festzulegen; hier spielen beispielsweise die von der Plattform erreichbare und die vom Anwender noch akzeptierte Positionierungenauigkeit (eng verbunden mit der maximal für Nachregelung gestatteten Zeit) eine Rolle. Das steuernde Programm, das zwischenzeitlich beispielsweise weitergehende sensorische Information abgerufen und verarbeitet hat, ist durch Übermittlung neuer Steuerbefehle in der Lage, auch nicht komplettierte asynchrone Aktionen abzubrechen.

Eine besondere Form asynchroner Anweisungen dient der Einstellung der Parameter zur aktuellen Bewegungs- und Rotationsgeschwindigkeit. In diesem Fall ist tatsächlich nur der Empfang von neuer Steuerinformation als Abbruchkriterium zugelassen, da kein Ziel erreicht wird, das allein unter Betrachtung dieser Parameter feststellbar wäre.

Aus Gründen der Betriebssicherheit wird ein Anhalten der Plattform ausgelöst, sobald ein Verbindungsabbau zwischen Steuerprogramm und dem Prozeß `netserver` stattfindet; der geordnete Abbau dieser Verbindung ist wiederum durch den implizit stattfindenden Destruktoraufruf der *proxy*-Klasse gesichert. Wird andererseits die Kommunikation zwischen diesem Prozeß und der Roboterbasis unterbrochen, so wird der eingangs beschriebene zwei-Sekunden-*timeout* wirksam und führt zu einem Anhalten und der Deaktivierung der Plattform.

Ergebnis. Durch die Spezifikation entsprechender Hardware-*proxy*-Klassen ist es sinnvoll möglich, die Grundfunktionen auch verschiedener Roboterplattformen zu nutzen. Dabei muß das den Roboter einsetzende Programm nicht in Verbindung mit dem plattformspezifischen Protokoll und seinen zeitlichen Randbedingungen der Ausführung gebracht werden.

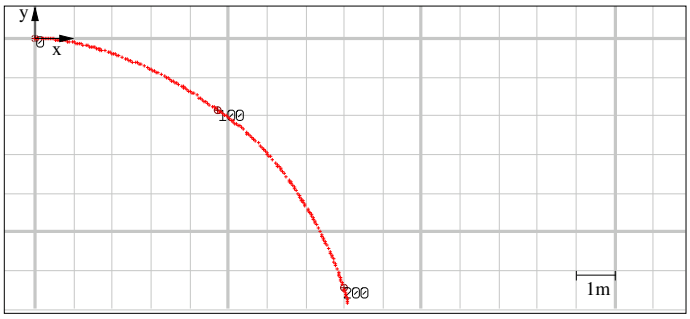


Abb. 4.8: Gemessene Odometriedaten bei tatsächlicher Geradeausfahrt mit $v = 0.1\,\mathrm{m} \cdot \mathrm{s}^{-1}$ in **x**-Richtung durch extern induzierte Bewegung.

4.2.3.2 Sensor: Odometrie

Die Odometrie ist als interner Sensor der Pioneer-Plattform über die Bibliotheken von Saphira zugänglich. Die Messung der Raddrehung wird durch einen optischen Sensor vorgenommen, der auf eine an der jeweiligen Radachse fixierte Scheibe mit einem Strichmuster gerichtet ist (5600 $ticks \cdot \mathrm{U}^{-1}$ im Fall des Pioneer 2). Unter Kenntnis von Raddurchmesser und Radstand kann indirekt ein grober Anhaltspunkt über die Plattformbewegung erhalten werden.

Fehlerquellen. Entsprechend der in 2.2.1.3 beschriebenen allgemeinen Eigenschaften dieses Sensors zeigt ein Test unter kontrollierten Bedingungen deutliche Abweichungen zwischen tatsächlicher und gemessener Bewegung auf. Auch hier sollen systematische Fehler bereits auf der Ebene der Sensoranbindung kompensiert werden, um entsprechend korrigierte Meßwerte zur Verfügung stellen zu können.

Die Beobachtung des Systems zeigt, daß die Vorwärtsfahrt auf einer allein nach Odometrieinformation gesteuerten Geraden z.B. durch Nutzung des in Saphira integrierten Automatismus des `move`-Kommandos einen annähernd gleichmäßigen Kreisbogen ergibt (linksdrehend, $r \approx 10\,\mathrm{m}$). Wird hingegen die Plattform antriebslos auf einem geradlinigen Kurs bewegt, so wird eine rechtsdrehende Kreisbahn mit gleichem Radius sensorisch wahrgenommen (Abb. 4.8). Nach Ausschaltung von ungleichmäßiger Beladung als Fehlerquelle und Kalibrierung der Sensoren (sog. *turn calibration* nach [Kon96]) und mechanischer Stabilisierung der Achslager trat der Effekt unverändert auf. Technisch wäre ein Parametersatz mit Skalarfaktoren für den linken bzw. rechten Radencoder ausreichend, um das Problem zu lösen,

doch wird dies vom Controller der Roboterbasis nicht unterstützt. Im Kontext von [Küp03] wurde daher der letztlich erfolgreiche Versuch unternommen, durch eine leichte asymmetrische Veränderung der Radradien eine Verbesserung zu erreichen.

Als größter nichtsystematischer und daher auf dieser Ebene nicht kompensierbarer Fehler verbleibt eine starke Abhängigkeit vom Bodenbelag. Ohne weitere Optimierung beispielsweise durch Fusion mit Daten anderer Sensoren sollte der Odometriesensor daher nur als Indikator für die relative Bewegung über kurze Zeitspannen dienen.

Softwarezugang. Die jeweils aktuelle Position und Orientierung (x, y, θ) kann durch eine einfache Memberfunktion der Klasse `saph_odometry` abgerufen werden, die das entsprechende Interface von `possensor` erbt. Da dieser Sensor roboterseitig über die Herstellerbibliotheken angesprochen werden kann, die auch der Anbindung der Basisplattform dienen (vgl. 4.2.3.1), sind somit keine weiteren Maßnahmen erforderlich.

4.2.3.3 Sensor: Kameras

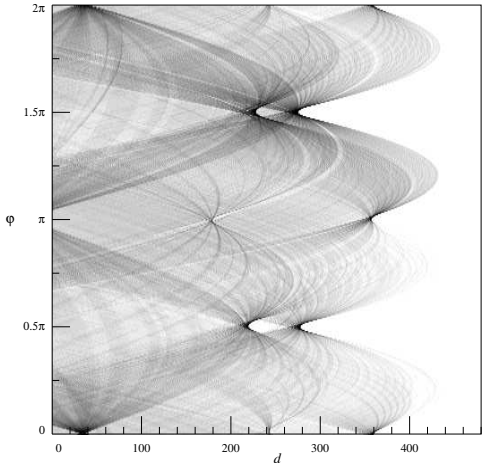
Die verwendeten PAL-Kameras (Sony XC999) sind über einen PCMCIA-Framegrabber (HaSoTec FG30) angebunden. In der praktischen Handhabung kann nach Auslösung eines von einem mitgelieferten Treibers bearbeiteten Softwareinterrupts $\frac{1}{2} \cdot w \cdot h$-mal die Bytesequenz Y, U, Y, V ausgelesen werden, um die Bildinformation zeilenweise zu erhalten. Erst zwei benachbarte Bildpixel ergeben die vollständige Farbinformation.

Ein Serverprozeß auf dem Bordrechner (in Abb. 4.3 mit P_3 bezeichnet) dient der Hardware-*proxy*-Klasse als Protokollpartner zum Abruf von Kamerabildern aus Anwendungsprogrammen. Da die Codierung $YUYV$ für je zwei Pixel für den Transport platzsparender als ein Bytetriplett RGB pro Bildpixel ist, wird die erforderliche Farbraumkonversion [Bou95, Poy98]

$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{bmatrix} 1 & -0{,}0009267 & 1{,}4016868 \\ 1 & -0{,}3436954 & -0{,}7141690 \\ 1 & 1{,}7721604 & 0{,}0009902 \end{bmatrix} \cdot \begin{pmatrix} Y \\ U - 128 \\ V - 128 \end{pmatrix} \tag{4.1}$$

unter Beachtung der Regeln der Sättigungsarithmetik ($R = \text{byte}(r \cdot 255)$ etc.) erst auf dem jeweiligen Steuerrechner durchgeführt. Im Protokoll zum Serverprozeß besteht die Option, bedarfsweise nur den Luminanzanteil Y

(a) Eingabebild mit Kissenverzerrung



(b) Hough-transformiertes Bild eines Kantenbildes von 4.9(a) für $k = 0$ (Grauwerthistogramm angepaßt, $\gamma = 0.2$). Durch die Bildverzerrung entstehen unscharf begrenzte Maxima.

Abb. 4.9: Ermittlung der radialen Verzeichnung

übertragen zu lassen oder Bilder in verschiedenen ganzzahligen Subsamplingstufen anzufordern.

Auch dieser Sensor weist systematische Fehler auf, die auf dieser Ebene korrigiert werden können.

Abbildungsfehler. Die Kamera verwendet ein Sensorelement mit bereits quadratischen Pixeln; eine Skalierung ist also nicht erforderlich. Die verwendeten Weitwinkelobjektive mit $f = 6\,\text{mm}$ bedingen jedoch eine störende bereits mit bloßem Auge erkennbare Verzerrung des Bildes (Abb. 4.9(a)). Aus bauartbedingten Gründen wird die Verzerrung als radialsymmetrisch mit dem Hauptpunkt in Bildmitte angenommen. Der wahre Ort $\mathbf{x}'$ eines bei $\mathbf{x}$ erscheinenden Bildpunktes läßt sich damit nach [Jäh97] zu

$$\mathbf{x}' = \frac{\mathbf{x}}{1 + k \cdot \mid \mathbf{x} \mid^2} \tag{4.2}$$

berechnen.

Zur Ermittlung des Parameters k wird diese Transformation mit verschiedenen k_a auf Kameraaufnahmen angewandt. Das dazu eingesetzte Verfahren ähnelt dem in [Ioc99] vorgeschlagenen, kommt dabei aber ohne einen

speziellen Testkörper aus. Das Bild soll möglichst im Randbereich des Bildes – hier tritt nach (4.2) die maximale Verzerrung auf, während Strecken durch den Hauptpunkt keinen Beitrag für das Verfahren leisten können – geradlinige Kanten in beliebiger geometrischer Konstellation (Entfernung, Orientierung im Raum) zeigen. Diese Kanten werden entsprechend der unbekannten Kamerageometrie verzerrt abgebildet. Das Resultat der Durchführung einer Kantendetektion auf dem transformierten Bild wird nun durch eine Hough-Transformation in einen diskreten (φ, l)-Parameterraum H_a überführt. Jedes Kantensegment wird genau ein lokales Maximum im Parameterraum erzeugen.

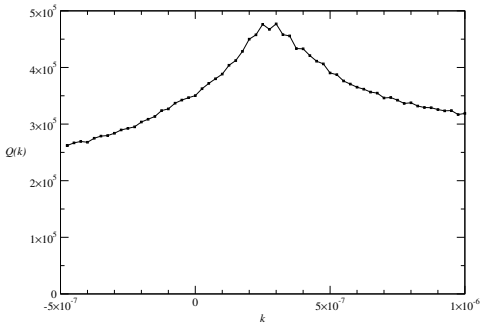
Die Varianz in der lokalen δ-Umgebung der Maxima (Abb. 4.9(b)) kann als Maß für die Qualität des geschätzten k_a dienen, da zu erwarten ist, daß bei verringerter Krümmung der Kantensegmente schärfer begrenzte Maxima im Parameterraum auftreten. Dieses von IOCCHI beschriebene Verfahren weist zwei Nachteile auf. Zum einen ist eine Ableitung von δ aus der Struktur des Kamerabildes notwendig, um ausschließen zu können, daß sich benachbarte Maxima wechselseitig beeinflussen – in [Ioc99] wird aus diesem Grund ein Testkörper verwendet, dessen Struktur sicherstellt, daß die Parameterdarstellungen der Maxima in wenigstens einem Parameter hinreichend unterschiedlich sind; in "natürlichen" Szenen (Abb. 4.9(a)) ist diese Eigenschaft wegen der dicht benachbarten parallelen Kanten nicht gegeben. Zum anderen ist es erforderlich, Maxima tatsächlich in jedem Parameterraum aufzufinden und ein Korrespondenzproblem zu lösen, da die Maxima unter verschiedenen k_a nur ungefähr an gleiche Orte abgebildet werden.

Um diese Einschränkung aufzuheben, wird der gesamte Parameterraum in die Berechnung einbezogen (vgl. [BM02]).

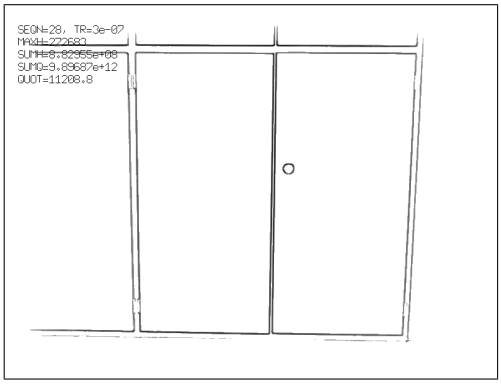
Da die Kompensation der radialen Verzerrung bereits durch die dabei stattfindende Interpolation die Summe über alle Pixelwerte verändert, werden die (vorzeichenlosen) Kantenbilder E_a so normiert, daß sie eine gleiche Summe c

$$c = \sum_{i=1}^{w(E_a)} \sum_{j=1}^{h(E_a)} E_a[i][j] \tag{4.3}$$

aller Pixelintensitäten aufweisen. Auf diese Weise ist sichergestellt, daß im nächsten Schritt betragsmäßig der gleiche Gesamteintrag in die (φ, l)-Parameterräume H_a vorgenommen wird. Um nun die Qualität der Approximation von k zu bewerten, wird der Grad ermittelt, mit dem Bildpunkte im Parameterraum aufeinander abgebildet werden: im Fall nur einer Kante mit den Parametern φ, l und eines korrekt geschätzten k wird $H_a(\varphi, l) = c$



(a) Serie der Q_a bei Variation von k: bei Wahl des optimalen von k_a wird ein absolutes Maximum angenommen.



(b) Korrigiertes Kantenbild für $k = 3 \cdot 10^{-7}$

Abb. 4.10: Korrektur der radialen Verzeichnung

gelten. Als Maß für diese Qualität wird

$$Q_a = \sum_{\substack{0 \leq i < w(H_a) \\ 0 \leq j < h(H_a)}} f\left(H_a[i][j]\right) \tag{4.4}$$

berechnet, wobei für f praktisch jede mit geringem Aufwand berechenbare Funktion mit den Eigenschaften

$$\forall x > 0 : f'(x) > 0 \wedge f''(x) > 0 \tag{4.5}$$

in Frage kommt. Eine Serie der Q_a für Variation von k für das Eingabebild aus Abb. 4.9(a) zeigt Abbildung 4.10(a). Trotz der Diskretisierung sowohl des Kantenbildes als auch des Parameterraums kann dort ein eindeutiges Maximum bestimmt werden. Durch Wahl des k_a mit dem größten Wert für Q_a wird die beste Schätzung für k selektiert.

Die Anwendung von Gl. 4.2 mit diesem Parameter auf das ursprüngliche Kantenbild demonstriert die Auswirkungen (Abb. 4.10(b)). Der so bestimmte Wert ist spezifisch für den Kameraaufbau und das verwendete Objektiv; eine wiederholte Bestimmung zur Laufzeit ist nicht erforderlich. Die Kompensierung der radialen Verzeichung kann als fester Bestandteil der Vorverarbeitung integriert werden.

4.2.3.4 Sensor: Sonar

Das Sonar des Roboters ist über die Roboterbasis angebunden; es ist keine weitere Integrationsleistung erforderlich.

Acht einzelne Emitter/Sensoren liefern Informationen über die Echolaufzeit, die unter Annahme von Standardumweltbedingungen in Entfernungsangaben umgerechnet werden.

Fehlerquellen. Die Frequenz, mit der vollständige Scans zur Verfügung stehen, ist abhängig von der Anzahl der aktivierten Emitter. Die Weiterschaltung in der Feuerfolge der Sensoren findet jeweils nach 0.04 s statt; Echos in größerer Entfernung als 6.8 m können bei einer Schallausbreitung in Luft mit $v = 343\,\mathrm{m} \cdot \mathrm{s}^{-1}$ grundsätzlich nicht korrekt gemessen werden (vgl. [KA98]). Besonders in langen Fluren kann es dazu kommen, daß auch nach Ablauf des Meßintervalls ausreichend starke Echos eintreffen, die Messung eines der nächsten aktiven Sensoren beeinflussen. Teilweise ist es möglich, durch Berechnungsverfahren diesen Einfluß zu verringern (EERUF, s. [BK95]).

Die Messung der Echolaufzeit wird nicht (wie in Saphira modelliert) auf einem Strahl, sondern in einem Kegel vorgenommen: die Messung einer Distanz d ist nicht so zu interpretieren, daß in exakter Ausrichtung des Sonaremitters bei d tatsächlich ein Hindernis vorhanden ist; der Meßwert zeigt nur an, daß im Bereich des Öffnungskegels bis zur angegebenen Entfernung mutmaßlich kein Hindernis vorhanden ist. Dies ist im Sensormodell (vgl. 2.2.1.3) entsprechend zu berücksichtigen.

Wiederholgenauigkeit. Abbildung 4.11 zeigt die Verteilung des Meßfehlers im Dauerversuch bei statischem Aufbau ($n = 8000$). Durch Umgebungsgeräusche, *crosstalk* und andere Effekte kam es in immerhin 4.6% aller Messungen zu innerhalb der gewählten Skala nicht mehr darstellbaren Ausreißern. Erst nach näherer Analyse der Meßdaten und der Feststellung, daß stark variierende Werte ausschließlich von den Sensoren mit den Ausrichtungen $-90°$ und $+30°$ rührten, ergibt sich nach entsprechender Bereinigung eine Normalverteilung mit $\sigma \approx 0.99$ ($\chi^2 = 0.07$).

Als Ergebnis läßt sich allein festhalten, daß ein Sonarecho möglicherweise als Indikator für die Präsenz eines Hindernisses interpretiert werden darf und die Abstandsbestimmung im Erfolgsfall tatsächlich recht genau vorgenommen wird, aber von der Abwesenheit eines Sonarechos keinesfalls

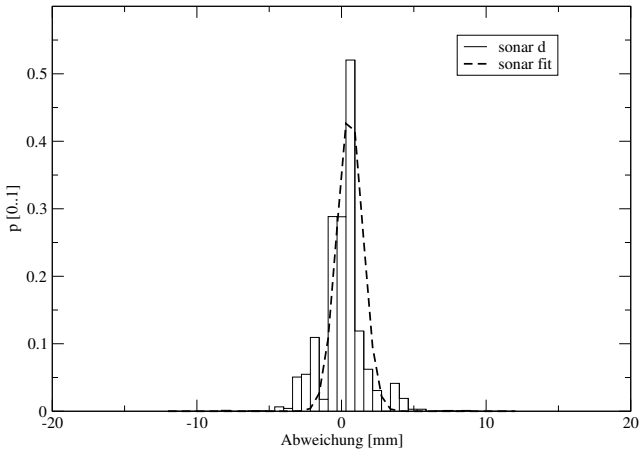


Abb. 4.11: Verteilung des Sonar-Meßfehlers ($n = 8000$)

sicher auf freien Raum geschlossen werden sollte. Ein systematischer auf dieser Ebene zu kompensierender Fehler wurde nicht entdeckt.

Softwarezugang. Da der im folgenden Unterabschnitt behandelte LRF eine strukturell vergleichbare Information mit erheblich höherer Auflösung liefert, wird das Sonar lediglich als Option zur Kollisionsvermeidung im Rahmen eines überwachten `move`-Kommandos des `netserver`-Prozesses (s. 4.2.3.1) eingesetzt. Für den programmiertechnischen Zugang ist eine Klasse `saph_sonar` vorgesehen, die den wesentlichen Teil der Schnittstelle, den Abruf aktueller Meßwerte, von der Oberklasse `tof_sensor` erbt: eine Memberfunktion mit der Signatur `list<point2> points() const` liefert die in auf das lokale System des Roboters bezogenen kartesischen Koordinaten zurück.

4.2.3.5 Sensor: LRF

Der LRF liefert für zwei Quadranten Entfernungsangaben zu Hindernissen durch Messung der Laufzeit und Phasenverschiebung von reflektiertem Laserlicht. Die Genauigkeit liegt nach Herstellerangaben [SIC00] “im Einzelschuß” zwischen 2 cm und 5 cm. Zur Verringerung des Meßfehlers werden in den meisten Betriebsarten mehrere Samples aufgenommen und unter impliziter Annahme eines stationären Systems gemittelt. Die Entfernungsmessung ist ab Werk kalibriert und im Gegensatz zum Sonar sehr stabil gegen wechselnde Umwelteinflüsse.

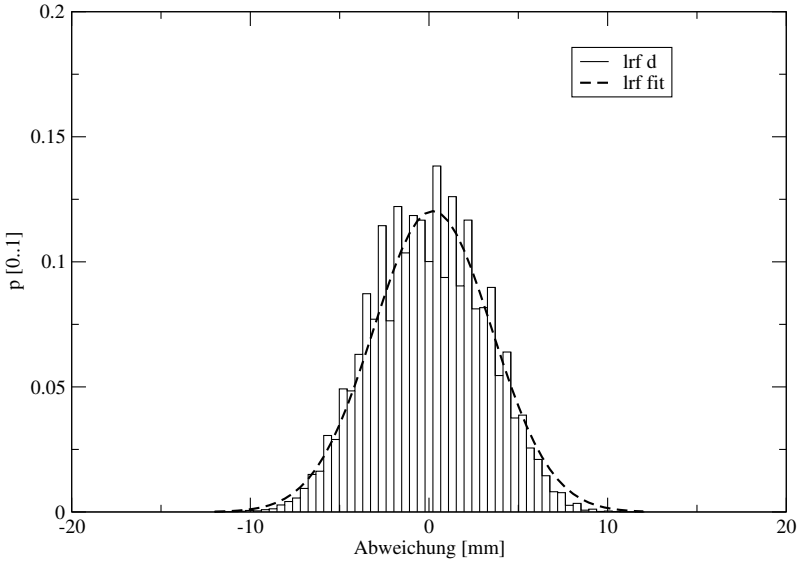


Abb. 4.12: Verteilung des Meßfehlers (LRF) im Dauerversuch bei statischem Aufbau ($n = 180000$). Die Meßwertquantisierung führt zur unregelmäßigen Feinstruktur; die Abweichung entspricht auch ohne weitere Aufbereitung gut einer Normalverteilung mit $\sigma = 3.31$ (bei $\chi^2 = 0.0039$).

Fehlerquellen. Bei einer Untersuchung der Meßgenauigkeit wurde festgestellt, daß das Meßrauschen mit guter Näherung als normalverteilt mit $\sigma_l = 2.2 \cdot 10^{-3}$ m angenommen werden kann; ein nennenswerter systematischer Fehler wurde nicht entdeckt (vgl. Abb. 4.12).

Zur korrekten Einschätzung für weitere Verarbeitung, z.B. für die Detektion von Features, wird der Näherungswert der Standardabweichung erforderlich.

Softwareanbindung. Da die Roboterplattform nicht mit dem vom Hersteller vorgesehenen proprietären Bordrechner ausgestattet ist, fehlt jede Integration in Saphira. Prinzipiell wären diese Sensordaten als Punktartefakte in das Weltmodell der Saphira-Architektur zu integrieren, da sie den Sonardaten strukturell sehr ähnlich sind.

Im entwickelten System dient ein eigenständiger, über einen Netzwerk-*socket* erreichbarer Serverprozeß – in Abb. 4.3, S. 101 bezeichnet als P_1 – auf dem Bordrechner als Kommunikationspartner für die Klassenbibliothek. Das vollständige serielle übermittelte Protokoll [SIC00] zwischen Bordrech-

ner und Sensorcontroller kann so praktisch unverändert über das Netzwerk übertragen werden. Zur Vereinfachung der Kommunikation werden jeweils nur vollständige Datenpakete mit zusätzlich vorangestelltem Längenheader im Block übertragen.[4]

Der synchrone Abruf von Daten in der bereits vom Sonar bekannten Form `list<point2>` wird generisch in `tof_sensor`, das Sensorprotokoll in der davon abgeleiteten Klasse `lrf_sick` implementiert (s. Abb. 4.5). Spezialisierungen dieser Klasse dienen der Kommunikation über entweder eine lokale serielle Schnittstelle oder den o.g. Serverprozeß.

4.2.3.6 Effektoren

Fahrwerk und Greifer. Die beiden wichtigsten Effektoren des Roboters, das Fahrwerk und der Greifer, sind über Standardfunktionen der Saphira-Bibliothek bedienbar; die mit den Geräten korrespondierenden Proxy-Klassen greifen dementsprechend auf den in 4.2.3.1 beschriebenen `netserver`-Prozeß zu.

Zur Berücksichtigung der Ordnung der *has a*-Gerätehierarchie existiert eine separate Klasse `saph_gripper2`, von der eine Instanz als *member*-Variable der Klasse `pioneer` auftritt. Für das Fahrwerk und Antriebseinheit ist hingegen keine separate Klasse vorgesehen, da die Fortbewegung durch Kombination von Translation und Rotation als durch die Basisklasse `autonomous` vermittelt betrachtet wird.

PTU. Die *pan/tilt-unit* besitzt einen externen Controller, der wiederum über eine serielle Schnittstelle über ein einfaches Protokoll gesteuert wird [Dir00]. Wegen des geringen Kommunikationsumfangs mit diesem Effektor – Kommandos und Rückmeldungen sind nur wenige Bytes lang – war es möglich, eine praktisch undokumentierte Schnittstelle (`SERAUX`) der Pioneer 2-Controllerplatine zu nutzen. Ein bestimmtes Steuerkommando der Saphira-Bibliothek ermöglicht, kurze Zeichenketten auf diese Schnittstelle zum PTU-Controller auszugeben, Antworten des Controllers einer vorab bekannten Länge auszulesen und als Paket eines speziellen Typs `SERAUXpac` zu erhalten. Durch Installation eines entsprechenden *packet processors* (vgl. [Kon00, S. 37]) können diese aus der normalen Kommunikation herausgefiltert und ausgewertet werden.

[4] Das längste vorkommende Paket – von STX bis CRC – ist ein vollständiger Scan bei maximaler Winkelauflösung (361 Werte) mit 732 Bytes.

Da die Anbindung dieser Komponente mittels der Saphira-Bibliothek geschieht, findet alle Kommunikation zwischen Bordrechner und Controller auf dem bereits in 4.2.3.1 beschriebenen `netserver` statt. In `ptu_dp` wird die generische Schnittstelle für Schwenk-/Neigeeinheiten dieser allgemeinen Bauweise, in `ptu_dp46_17` das gerätetypspezifische Protokoll und in `saph_ptu` zusätzlich der Kommunikationsweg über die Netzwerk-`socket`-Schnittstelle implementiert.

Sprachausgabe. Das Audiosystem des Bordrechners kann von einem entfernten Steuerrechner zum Abspielen von allgemeinen Klangdateien verwendet werden. – Da das Betriebssystem des Bordrechners weder gängige Dienste zur Übertragung von Dateien zwischen den Dateisystemen (ftp, rcp, scp) noch Serverdienste die Möglichkeit zum von anderen Rechnern ausgelösten Starten von Programmen (wie durch telnet, rsh, ssh möglich) vorsieht, implementiert ein weiterer, in Abb. 4.3 mit P_4 bezeichneter Serverprozess diese Funktionen. – Beispielsweise bedient sich die Funktion `speaker.play()` intern dieser Dienste, um eine wiederzugebende Audiodatei auf den Bordrechner zu transferieren und mittels dort lokal vorhandener Software abzuspielen. Zur Ausgabe von natürlicher Sprache wird eine Aufbereitung von Textstrings zu Audiodateien mittels txt2pho[5] zur Umwandlung in Phonemketten und anschließender Sprachsynthese durch Mbrola vorgenommen.[6] Der gesamte Aufbereitungsprozeß ist wiederum in der Klasse `speechsynth` gekapselt. Durch einen synchronen Aufruf der Art `speaker.play(speechsynth.wave("Hallo"));` würde die Sprachausgabe angesprochen.

4.2.4 Bewertung

Strukturelle Neuordnung. Die Ergebnisse erlauben unter Verwendung eines angemessen strukturierten Interface die Nutzung sämtlicher Komponenten praktisch im vollen Funktionsumfang. Die noch aus der akzidentellen hardware-technischen Struktur hervorgehenden Bedingungen sind auf dieser Ebene weitgehend aufgehoben: es verbleiben als Randbedingungen lediglich durch die Kommunikationsstruktur vermittelte Einschränkungen; so könnte z.B. die PTU nicht betrieben werden, ohne daß eine aktive Verbindung zur für die Kommunikation zu dieser Komponenten erforderlichen Roboterplattform besteht. Diese nicht auf Klassenhierarchie und eine

[5] txt2pho geht aus dem Projekt "hadifix" [PSW90] hervor und wurde am Institut für Kommunikationsforschung und Phonetik, Universität Bonn entwickelt.

[6] Mbrola wurde an der Faculte Polytechnique de Mons entwickelt [DPPFB96].

containment-Struktur abzubildenden Einschränkungen erfordern immerhin nur mehr ein minimales Wissen über die Struktur der unterliegenden Ebenen.

Die strukturelle Neuordnung der Komponenten nach Schaffung von Hardwarekomponenten repräsentierenden *proxy*-Klassen ermöglicht, auf dem Zwischenresultat eine Architektur zu errichten, die von den Mängeln einer spezifischen Hardwarestruktur weitgehend frei sein sollte. Dies sollte sowohl eine Übertragbarkeit auf dieser Basis erstellter Softwaresysteme auf vergleichbare Systeme als auch den Austausch von Komponenten zulassen, ohne daß zwangsläufig ein Einfluß auf die höheren Ebenen entsteht. Die problemlose Verwendung auch anderer Plattformen erhärtet diese Behauptung.

Einfluß auf die Architekturwahl. Die Frage der zu wählenden Roboterarchitektur bleibt auf dieser Ebene wie beabsichtigt noch vollständig offen: weder sind verschiedene Sensoren miteinander fusioniert worden noch bestehen verzichtbare lokale Zustände.

Zwei Ausnahmen wurden aus pragmatischen Gründen zugestanden. Erstens wurden in den Protokollautomaten z.B. des `netserver` Zustände zugelassen, um die Durchführung bestimmter atomarer Aktionen mit zeitlicher Ausdehnung überwachen zu können (so z.B. der Funktion `move(`d`)` mit der Distanz als Parameter). Zweitens wurde ein Mechanismus zum Anhalten der Plattform bei unmittelbarer durch Sonar festgestellter Kollisionsgefahr integriert. Dieser im `netserver` realisierte Code nach reaktivem Schema auf Basis der Sonardaten spricht unter üblichen Betriebsbedingungen nicht an und gehört nicht zum steuerbaren Systemverhalten, sondern soll lediglich eine Sicherung der Roboterhardware darstellen. Auch die Bereitstellung von *direct motion commands* nach deliberativem Schema impliziert keine Architekturwahl, sondern stellt nur für den Fall einer solchen übliche atomare Operationen zur Verfügung.

Grenzen der Funktionsprimitive. Der orthogonale Zugriff auf generische, doch nach wie vor (einzel-)komponentenbezogene "Dienste" des Roboters ist eine Erleichterung für alle weitere Entwicklung, ermöglicht aber noch nicht die Gestaltung vollständiger Systeme auf dem gewünschten Abstraktionsgrad. Insbesondere die mitunter recht aufwendige Vorverarbeitung (besonders im Fall von Kamerabildern, vgl. 4.2.3.3) ließe sich bedarfsorientiert auf funktionaler Ebene noch deutlich weiter treiben (für eine

Übersicht für Leistungen, die mitunter als "Funktionsprimitive" von Kamerasystemen gelten, s. [RRN02]).

Viele höherwertige Dienste werden allerdings nicht nur auf einzelne Systemkomponenten bezogen sein: diese Art von nicht-lokaler Sensordateninterpretation (ggf. als Sensorfusion) ist mit gutem Grund Gegenstand der folgenden Entwurfsebene. Jede Bemühung, eine höhere Systemleistung beispielsweise durch Kombination von Sensordaten verschiedener Quellen und ggf. den Einsatz von Weltmodellen zu erzielen, würde Einschränkungen in Bezug auf die *konkrete* Plattform und ihre technische Ausstattung einführen. Diese sollten wenigstens auf dieser Ebene noch unbedingt vermieden werden sollen.

4.3 Ebene der Subsysteme

Die in 4.2 beschriebene Schicht erlaubt den strukturierten Zugang zur sensorischen und motorischen Ausstattung einer Hardwareplattform, ohne dabei auf bestimmte Spezifika der Geräte (Protokoll, Timinganforderungen, intern verwendete Maßeinheiten) Rücksicht nehmen zu müssen. In der Praxis kann diese Ebene mit Vorteil als geeignete Grundlage der Konstruktion von Systemen in fester Struktur nach einem der in 2.1 beschriebenen Paradigmen verwendet werden, und wurde auch entsprechend in begleitenden Projekten erfolgreich eingesetzt [FS01, HKWW01, Küp01, BSK02, Gei02, GS03, Hol03, KM03, Küp03, San03, SB03, Wil03]. Auch die Abstraktion von der konkret verwendeten Hardware konnte z.B. durch den Einsatz zur Plattform alternativer Simulatoren wie im Fall von [BB01, BBBM01] gezeigt werden.

Zur Schöpfung größerer Strukturen und schon zur Konstruktion komplexer Systemfähigkeiten bietet aus den in 4.3.1 angeführten Gründen jedoch ein anderer Weg größere Vorteile in Hinblick auf Testbarkeit und Wiederverwendbarkeit von Softwaresystemkomponenten und Änderbarkeit der Gesamtarchitektur. Eine Modularisierung und interne Strukturierung ist dem direkten Entwurf monolithischer Systeme hier vorzuziehen. Die Erkenntnis der Notwendigkeit einer solchen Strukturierung ist nicht neu, sondern wurde explizit bereits vielfältig geäußert, so z.B. von BROOKS:

> *"The behaviors are the building blocks, and the functionality is emergent. This differs from the traditional approach in which the functional modules are the building blocks and the behaviors are emergent."*
> – [Bro91b]

Diese Aussage ist besonders darum bemerkenswert, weil sie sowohl die Sicht aus Perspektive der Entwickler reaktiver wie der deliberativer ("traditioneller") Systeme behandelt. Angesichts der Knappheit der Aussage sind beide Positionen recht vollständig wiedergegeben.

In 4.3.2 wird als Antwort zu der Frage nach der Wahl konkreter und für alle Architekturtypen gemeinsamer Strukturmittel ein Subsystemkonzept zur Gestaltung von Roboterarchitekturen vorgestellt. Die für das Modellsystem entwickelten spezifischen Subsysteme, die sich durch Bezug auf mehrere Aggregate einer Roboterplattform oder die Kapselung von architekturspezifischen Eigenheiten auszeichnen, folgen in 4.3.3. Abschnitt 4.3.4 bewertet den Ansatz und benennt die sich aus diesem ergebenden offenen Punkte.

4.3.1 Problem

Die Leistung der vorgeordneten Schicht des Entwurfs läßt folgende Konstruktionsprobleme offen:

1. Die Klassen spiegeln nur die Möglichkeiten einzelner Baugruppen wieder, nicht jedoch die Optionen, die durch gleichzeitige Nutzung mehrerer Baugruppen entstehen.[7]

2. Das Entstehen abgeschlossener Verarbeitungseinheiten jeweils für bestimmte Teilprobleme ist keine notwendige Folge des Einsatzes des in 4.2 umrissenen Komponenteninterface. Die geforderte Modularität kann auf diese Weise noch nicht erreicht werden.

Die Designentscheidung, die die Einschränkung zu 1. herbeiführt, wurde getroffen, um die Allgemeinheit des Ansatzes zu erhalten: Die Kombination bespielsweise zu einer "LRF-und-Kamera"-Instanz mit zusätzlicher aus der Sensorfusion rührender Funktionalität wäre in hohem Maße spezifisch zum Aufbau eines speziellen Roboters und erschien auf dieser Ebene noch nicht sinnvoll. In Konsequenz tritt die Schwierigkeit auf, daß jede Weiterverarbeitung von Sensordaten in symbolische Repräsentation, jede Zusammenführung der Daten mehrerer Sensoren im Rahmen der Sensorfusion und jede Umsetzung von Sensordaten oder symbolischen Daten in Kommandos zur

[7] Eine Ausnahme scheinen auf den ersten Blick die *member*-Instanzen des Roboterobjekts in *has-a*-Semantik zu bilden. Faktisch handelt es sich jedoch nur um eine strukturelle Gruppierung der Klassen ensprechend der physischen Montage der Hardwarekomponenten; es entsteht kein Mehrwert im Sinne einer funktionalen Erweiterung.

Effektorsteuerung im Anwendungsprogramm auf der Ebene der verwendeten Programmiersprache fest codiert werden muß. Dies führt unmittelbar zum unter 2. genannten Problem.

4.3.2 Ansatz

Zur Lösung beider Probleme wird der Ansatz verfolgt, *Subsysteme* zu konstruieren, die sowohl die Möglichkeit zur Interaktion untereinander als auch mit ggf. mehreren Instanzen technischer Baugruppen haben.

Der Ansatz, eine funktionale Dekomposition eines Gesamtsystems in "Module" zur Strukturierung zu betreiben, ist nicht neu. ALAMI et al. verwenden einen ähnliche Ansatz auf einer der Ebenen einer TLA, allerdings zu einem anderen Zweck: hier werden lediglich auf der untersten (funktionalen) Ebene der Architektur Sensor- und Effektorfunktionalität ("*built-in robot action and perception capacities*") gekapselt [ACF+98], während der hier vorgestellte Ansatz das auch für die höheren Verarbeitungsstufen vorschlägt. Im Unterschied zu der Arbeit von ALAMI soll zudem noch keine Festlegung auf einen bestimmten Architekturtypus getroffen werden, um die Allgemeinheit der Lösung zu wahren.

Als Leitlinie zur Partitionierung eines Gesamtsystems in Subsysteme dienen übliche Kriterien wie Abgeschlossenheit und Einheit der Funktion, Orthogonalität und Umfang der Schnittstelle und die jeweils – z.B. für Testbarkeit und sinnvolle Austauschbarkeit – gewünschte resultierende Strukturgranularität. Das Problem der Wahl einer geeigneten Aufteilung wird mit sämtlichen anderen Entwurfsparadigmen geteilt: je nach Betrachtungsdistanz existieren unterschiedliche "natürliche" Bausteine eines Systems. Die Wahl der Subsystemgrenzen ist damit eine durch den jeweiligen Entwickler zu treffende Entscheidung, bei der vor allem der Aspekt der Modellierung der *Kommunikation* zwischen den Einheiten eine wesentliche Rolle spielt; es erfolgt keine automatische Partitionierung.

Im Rahmen der strukturelle Erweiterung werden auf dieser Ebene behandelt:

1. die (reaktive) Sensor/Effektor-Kopplung,
2. die Nutzung mehrerer Sensoren zur (deliberativ/hybriden) Sensorfusion,
3. die explizite Implementierung von Systemzuständen und

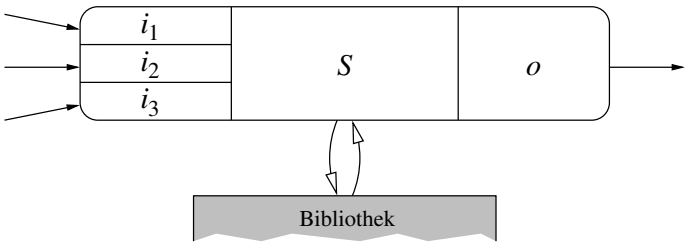


Abb. 4.13: Einzelnes Subsystem in schematischer Darstellung. Das Subsystem S besitzt eine asynchrone Schnittstelle zu anderen Subsystemen (hier die Eingänge $i_1 \ldots i_3$ und den Ausgang o), und ist durch synchrone Bibliotheksaufrufe mit der Roboterhardware verbunden.

4. die Implementierung (hybrider) virtueller Sensoren, die höherwertige symbolische Sensorinformation liefern.

Subsysteme dieser Ebene wären immerhin insoweit plattformspezifisch, als daß die adressierten Baugruppen auf nur einer gemeinsamen Plattform vorhanden sein müßten. Die eigentliche Spezialisierung liegt in der zu erfüllenden Aufgabe, sofern eine parametrisierte Verarbeitung – wie die Extraktion von symbolischen Merkmalen aus Sensordaten – vorgenommen werden muß. Bestimmte Subsysteme wären zudem mit bestimmten Architekturparadigmen durch ihre für diese typische Funktion verknüpft, wie die obige Aufzählung bereits andeutet.

Eigenschaften der Subsysteme. Ein Subsystem wird in diesem Kontext definiert als Einheit, die eine abgeschlossene "Funktion" erfüllt oder Lösung einer Aufgabe im Rahmen einer Gesamtarchitektur implementiert. Es verfügt zu diesem Zweck über eine Auswahl folgender Komponenten (vgl. auch Abb. 4.13):

1. lokale Kontrolle,

   technisch umgesetzt als *thread*,

2. lokalen Speicher,

   technisch durch den lokalen *heap* des *thread*-Adressraums gelöst,

3. Hardwarezugriff,

   technisch gelöst durch Zugriff auf die in 4.2 beschriebene Bibliothek durch synchrone Aufrufe,

4. Kommunikationsschnittstellen,

   die sich mit anderen Subsystemen verbinden lassen, um beispielsweise im hybriden Fall einen virtuellen Sensor oder ein virtuelle Sensordaten konsumierendes Subsystem, im reaktiven Fall ein Summationsglied für verteilt entstehende Steuerungskommandos oder einen *arbiter* implementieren zu können.

Ein aus nur einem einzelnen Subsystem bestehendes Gesamtsystem unterscheidet sich also nicht von einem monolithischen System; diese Lösung stellt also zunächst nur eine prinzipielle Partitionierbarkeit bereit.

Lokalität der Kontrolle. Wie bei Einführung der *behavior based*-Architektur in 2.1.3.3 und der reaktiven Subsumption in 2.1.3.4 dargelegt ist die Option zur parallelen Codeausführung notwendiger Bestandteil der Modelle. Auch alle hybriden Architekturen verwenden mindestens in ihrem reaktiven Teil diese Option. Es existiert außerdem ein weiteren Grund für die Einführung einer lokalen Kontrolle in Kombination mit paralleler Ausführung von Programmstrukturen.

Im praktischen Einsatz der Experimentalplattform treten bei der Kommunikation mit Sensorservern Verzögerungen auf, die durch Protokollabwicklung, Zeitbedarf für Durchführung der Messung und Datenrate der Übertragung der Resultate bedingt sind. Die diesen Ablauf kapselnden Bibliotheksfunktionen der mit den Sensoren korrespondierenden *proxy*-Klassen werden synchron ausgeführt, d.h. sie blockieren die Programmausführung bis zum Abschluß. Ähnliches gilt für bestimmte Effektoraktionen, wie bereits Abb. 4.7 andeutet. Aus Gründen der Effizienz und Sicherheit beispielsweise im Kontext von Plattformbewegung bei gleichzeitig aktiver Kollisionsvermeidung ist es sinnvoll, grundsätzlich nur einen lokalen Teil des Gesamtsystems vollständig anzuhalten, nämlich den, der die Sensordaten angefordert hat.

Die Lösung beider Probleme geschieht über eine Abbildung der in 2.1.3 und 2.1.4 als parallel identifizierten Einheiten, die in allen bekannten Fällen auch tatsächlich die notwendigen Kriterien in Bezug auf Geschlossenheit der Funktion erfüllen, auf die hier eingeführten Subsysteme, und die technische Umsetzung der Subsysteme als separate *threads*. Die Lokalität der Kontrolle bringt neben den o.g. Eigenschaften mit sich, daß auch ein ausgefallenes Subsystem das Gesamtsystem nicht notwendig anhält. Als zusätzliches Merkmal tritt eine Option auf Datenpersistenz hinzu: ein geeignet implementiertes reaktives oder hybrides Subsystem kann ebenfalls auf

Basis einer lokalen Verwaltung vor Abschaltung nach Belieben Zustandsinformation wie Kalibrierungsdaten oder die Außenwelt betreffende Modellinformation (wie Parameter zur Merkmalsextraktion bzw. Kartendaten) externalisierten, um nach einem Neustart diesen Stand wiederaufzunehmen. Der wesentliche Nachteil des Verfahrens bestehet darin, daß eine explizite Kommunikation zwischen den separat ausgeführten *threads* vorgesehen werden muß. Dieser Aspekt wird in 4.4 behandelt werden.

4.3.3 Spezifische Subsysteme

Dieser Abschnitt befaßt sich mit spezifischen Subsystemen, die durch Verwendung mehrerer Hardwarekomponenten der Plattform eine Sensorfusion durchführen oder durch Berechnung von Merkmalen einen evtl. so aufgaben- und plattformspezifischen virtuellen Sensor zur Verfügung stellen. Dies macht die Berücksichtigung eines Sensormodells und von Strukturhypothesen über die Umwelt erforderlich. Subsysteme sind also aufgaben- und kontextspezifisch, nehmen aber noch immer keine Architekturentscheidung vorweg. Diese soll separat erst auf der höchsten Entwurfsebene getroffen werden.

4.3.3.1 Sensorkinematik und visuelle Messung

Da Sensoren teilweise in Relation zur Plattformbasis beweglich montiert sind, kann eine angemessene Reaktion des Gesamtsystems auf Sensordaten nur erfolgen, wenn eine Kombination von Effektorsteuerung (Position, Ausrichtung) und Sensorabruf erfolgt. Die verwendete Plattform verfügt über zwei auf einer PTU mit zwei Freiheitsgraden beweglich montierte Kameras (Abb. 4.14). Um sie beispielsweise zur visuellen Vermessung der Umwelt gebrauchen zu können, ist ein Subsystem erforderlich, daß über eine Verbindung zu beiden Komponenten verfügt.

Kinematik des Effektors. Die Position und Orientierung der Sensoren in Relation zur Roboterbasis wird allein durch die Ausrichtung der einzelnen Effektorglieder bestimmt. Im Folgenden sei mit **0** im zum Roboter lokalen rechtsdrehenden Referenzkoordinatensystem der Punkt vereinbart, der auf der Fahrebene $z = 0$ liegt und um den die Basis bei einer reinen Rotation um die Hochachse dreht. Die **x**-Achse entspricht dabei der allgemeinen Fahrtrichtung.

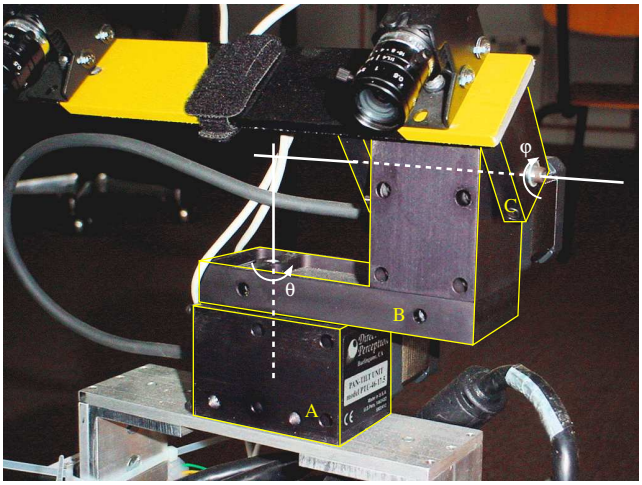


Abb. 4.14: Kameramontage und PTU: Die Komponente A ist fest montiert, Teil B kann um θ in einem Bereich $-160°..160°$ geschwenkt werden. C und die damit verbundenen Teile des Aufbaus können zusätzlich im Bereich von etwa $-45°..40°$ nach oben und unten geneigt werden.

Die Basistransformation für die komplette Kinematikkette in homogenen Koordinaten lautet unter Bezug auf einfache Translations- und Rotationsmatrizen

$$\mathbf{T} = {}^{0}\mathbf{T}_1 \cdot {}^{1}\mathbf{T}_2 \cdot {}^{2}\mathbf{T}_3 \cdot {}^{3}\mathbf{T}_4 \cdot {}^{4}\mathbf{T}_5 \tag{4.6}$$

mit

$$ {}^{0}\mathbf{T}_1 = \mathrm{Trans}_z\,(0.55\,\mathrm{m}) \tag{4.7}$$

$$ {}^{1}\mathbf{T}_2 = \mathrm{Rot}_z\,(\theta) \tag{4.8}$$

$$ {}^{2}\mathbf{T}_3 = \mathrm{Trans}_y\,(0.082\,\mathrm{m}) \cdot \mathrm{Trans}_z\,(0.059\,\mathrm{m}) \tag{4.9}$$

$$ {}^{3}\mathbf{T}_4 = \mathrm{Rot}_y\,(-\varphi) \tag{4.10}$$

$$ {}^{4}\mathbf{T}_5 = \mathrm{Trans}_y\,(c \cdot b) \quad \text{mit } c \in \{0, 1\} \tag{4.11}$$

entsprechend der Montage der PTU auf dem Roboter (4.7), der bauartbedingten Dimensionen der PTU (4.8–4.10) und der Position der Kameras in Bezug zum Endeffektor (4.11). Die Breite der Basis zwischen linker und rechter Kamera schließlich wurde zu $b = 0.14\,\mathrm{m}$ bestimmt. Die Position P_0 der linken Kamera ($c = 0$ in Gl. 4.11) kann somit unter Zuhilfenahme von Gl. 4.6 zu

$$P_0 = \mathbf{T} \cdot [0, 0, 0, 1]^{\top} \tag{4.12}$$

bestimmt werden. Für $\theta = \varphi = 0$ ist vereinbart, daß die Kamera in Fahrtrichtung ausgerichtet ist; die optische Achse verläuft parallel zur Fahrebene. Das Vorzeichen in Gl. 4.10 ist abweichend vom üblichen Drehsinn der Rotationsachse so gewählt, daß negative Drehwinkel die Kamera zur Fahrebene neigen.

Monokulare Triangulation. Bereits ohne Kenntnis der weiteren Sensorparameter kann Gl. 4.12 unter Zuhilfenahme der Tatsache, daß

$$\mathbf{v} = \mathbf{T} \cdot [1,0,0,0]^\top \tag{4.13}$$

den Vektor der Blickrichtung (durch den Hauptpunkt in Bildmitte) ausdrückt, zur Bestimmung der Koordinaten $\mathbf{p}$ von Objekten verwendet werden, deren Höhe über der Fahrebene z bekannt und hinreichend von P_{0z} verschieden ist, da

$$\begin{gathered} \mathbf{p} = P_0 + \lambda \mathbf{v} \quad \text{mit } \lambda > 0 \\ \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \mathbf{T} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} + \lambda \cdot \mathbf{T} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \end{gathered} \tag{4.14}$$

mit den nur drei Unbekannten x, y, λ eindeutig bestimmt ist. Diese Methode ist trotz der geringen Bauhöhe des Roboters durch die Breite der entstehenden virtuellen Stereobasis sehr effektiv und wurde erfolgreich in [BM00] zur Vermessung der Position von Objekten mit bekannter Höhe eingesetzt; Ergebnisse der Untersuchungen von [OWH01] deuten darauf hin, daß eine Entsprechung des Verfahrens auch beim menschlichen Sehen eine große Rolle spielt.

Kameramodell. Die Berücksichtigung des Kameramodells ermöglicht bei bekannter Brennweite und gegebener Dimension des Sensorarrays ($w \times h$ Bildpunkte der Größe $d = 8.6\,\mu$m), eine Gl. 4.14 entsprechende Berechnung für beliebige Bildpunkte anzustellen. Es sei das übliche Bildkoordinatensystem (Ursprung links oben) angenommen. Im einfachsten Fall – bei einem Lochkameramodell – werden zu $\mathbf{v}$ in Gl. 4.14 zwei vektorielle Komponenten in Richtung der Achsen der Bildebene addiert, um auch die Pixelkoordinaten p_x, p_y zu berücksichtigen.

$$\mathbf{v}(p_x, p_y) = f \cdot \frac{\mathbf{v}}{||\mathbf{v}||} - d \cdot \left(p_x - \frac{w}{2}\right) \cdot \mathbf{T} \cdot [0,1,0,0]^\top - d \cdot \left(p_y - \frac{h}{2}\right) \cdot \mathbf{T} \cdot [0,0,1,0]^\top \tag{4.15}$$

In der Praxis ist die durch Gl. 4.15 erhaltene Beziehung unter der Randbedingung des Auftretens von Abbildungsverzerrungen nur für Objekte einsetzbar, die nahe des Hauptpunktes abgebildet werden. Wird dieser Abbildungsfehler bereits auf einer tieferen Schicht (s. 4.2.3.3) kompensiert, fällt diese Einschränkung weg: Unter Kenntnis der Schwenk- und Neigewinkel der PTU kann die Position visuell detektierter Objekte in Relation zur Roboterbasis ermittelt werden, für deren Koordinaten eine weitere Randbedingung (z.B. die Höhe über der Fahrebene) bekannt ist.

Stereoverfahren. Auch Stereoverfahren stellen einen Sonderfall der erläuterten Triangulation dar; die Basisbreite ist allerdings konstant ($b = 0.14\,\mathrm{m}$) und nicht vom Schwenkwinkel der Kamera abhängig. Trotz der stabileren Stereogeometrie hat die geringe Basisbreite einen erheblichen Einfluß auf die auch unter optimalen Umständen erzielbare Ortsauflösung.

Die Untersuchung von effizienten Stereoverfahren mit der hier vorgestellten Plattform wird in [Fis03] diskutiert.

4.3.3.2 Sensorabgleich und Datenfusion

Unter der Annahme einer *flat world* kann eine invers-perspektivische Transformation gemäß Gl. 4.15 auf monokulare Kamerabilder angewandt werden (Abbildung 4.15), die eine sensorische Untersuchung der näheren Umgebung wegen der Darstellung von Objekten in entfernungsinvarianter Größe erheblich vereinfachen kann [Har89]. Die Kombination dieses Verfahrens mit den Daten eines LRF ermöglicht nach einem Abgleich beider Sensoren, genau die Bereiche der Fahrebene zu ermitteln, für die die Modellannahme *nicht* gilt. Dies ist eine wesentliche Vorbedingung für Kartierung und kollisionsfreie Navigation.

Die in 4.3.3.1 getroffenen Annahmen über die Kameraausrichtung anhand der gesteuerten Effektorkinematik stellen ein Ideal dar, das im praktischen Einsatz selten gegeben ist. Das Problem macht sich trotz kompensierter Kamerageometrie (4.2.3.3) noch auf verschiedene Weisen bemerkbar. Es kann zu einer Dejustierung kommen, da nicht nur bei physischem Kontakt mit Hindernissen, sondern auch bereits bei Bewegung von PTU oder Plattform Kräfte auf den Kameraaufbau wirken. Die Glieder der Kinematikkette, die Basis der Roboterplattform und Sensor miteinander verbinden, sind selbst hinreichend versteift, um Daten einer mechanischen Vermessung dauerhaft als Berechnungsgrundlage verwenden zu können. Allerdings verfügt der

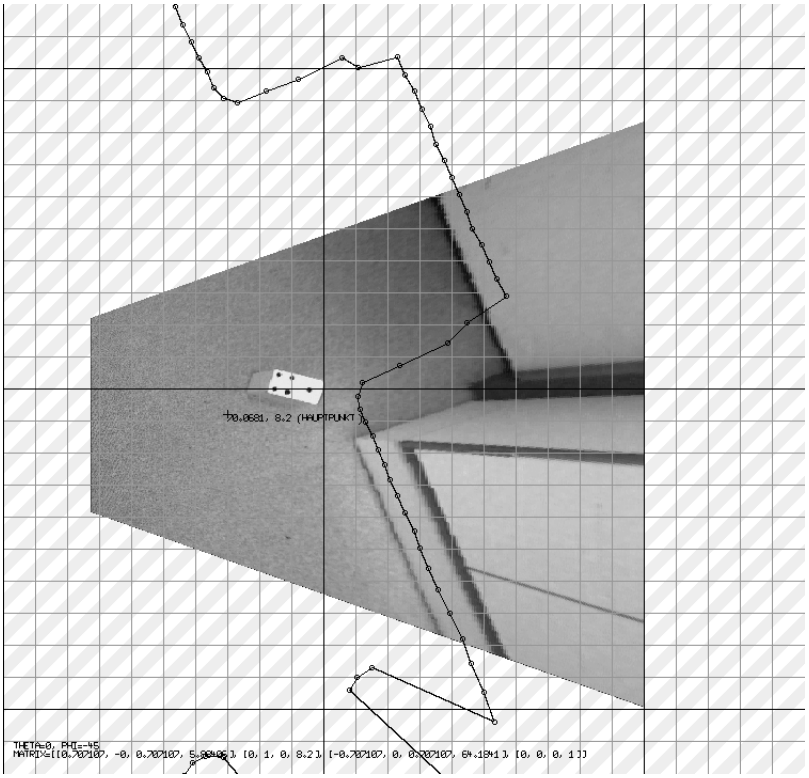


Abb. 4.15: LRF-Scan und rückprojiziertes Kamerabild (ohne Abgleich der Sensoren).

Roboter nicht über eine Lageregelung (oder -messung). Eine durch Eigenschaften des Fahrwerks bedingte variante Neigung der Plattform führt zu Fehlern der Vermessung.

In Konsequenz ist es nicht möglich, ohne weitere Maßnahmen eine Fusion der Kameradaten mit denen des in Relation zur Basis fest montierten LRF vorzunehmen. Als zusätzliche Schwierigkeit taucht auf, daß der LRF zwar fest, doch bauartbedingt mit unbekanntem Versatz auf der Plattform montiert ist. In Abb. 4.15 ist insbesondere der Versatz des optischen Zentrums des LRF gegenüber der Kamerageometrie erkennbar, die über die Kinematikkette des beweglichen Aufbaus wie in 4.3.3.1 beschrieben berechnet wurde.

Für die Kompensierung dieser technisch und mechanisch nicht mit vertretbarem Aufwand zu verhindernden Effekte bieten sich grundsätzlich als Lösungsalternativen die Durchführung einer statischen Kalibrierung oder eines dynamischen Sensorabgleichs an. Verfahren der statischen Kalibrierung

werden mit großen Erfolg für stationäre Kamerasysteme eingesetzt [Bab00]. Die Nachteile des Einsatzes vergleichbarer Verfahrens bei Verwendung mobiler Systeme sind allerdings, daß ein – mechanisch empfindlicher – Testkörper mitgeführt werden muß, und daß durch einen solchen konventionellen Testkörper allein die Parameter der optischen Systeme in Relation zu nur einer Lage und Position der Plattform ermittelt werden können. Da die Parameterschätzung wegen der ohnehin vergleichsweise unpräzisen Bewegung der Plattform nicht mit vergleichbarer Genauigkeit erfolgen muß, aber die Möglichkeit sowohl des Abgleichs verschiedener Sensoren als auch der Plattformbewegung selbst in die Analyse einbezogen werden soll, wird an dieser Stelle ein Verfahren des dynamischen Sensorabgleichs gewählt.

Verschiebung des LRF. Das optische Zentrum des LRF ist bauart- und montagebedingt gegenüber der dem Koordinatensystem der Trägerplattform verschoben. Dieser Effekt ist klar in Abb. 4.15 erkennbar, der eine nicht kompensierte Fusion eines LRF-Scans und eines gemäß der Transformationsmatrix der Kinematikkette (mit den Parametern $\theta = 0$, $\varphi = -45°$) rückprojizierten Kamerabildes zeigt. Die Bestimmung der Parameter dieser unbekannten Translation $(x, y, 0)$ ist erforderlich, um Entfernungsdaten des LRF relativ zur Basis des Roboters transformieren zu können. Die Bestimmung der Translationsparameter durch direkte Messung der Parameter einer Vorwärtskinematik ist praktisch nicht möglich, da die Position des optischen Zentrums des LRF verborgen bleibt. Bei der Verschiebung des LRF gegenüber der Plattform handelt es sich im vorliegenden Fall um eine bauartbedingte Konstante, die streng genommen nur ein einziges Mal ermittelt werden müßte; das vorzustellende Verfahren integriert die Bestimmung dieser Parameter allerdings ohne zusätzlichen Aufwand.

Plattformneigung. Als statisch stabile Plattform mit zwei angetriebenen seitlichen Rädern und einer Stützrolle am Heck dominiert bei gleichmäßiger Beladung die Inklination zur x-Achse den Fehler. Die abweichenden Elastizitätseigenschaften von Rädern und Stützrolle begünstigen diesen Effekt. Die Plattform ist – je nach Gewicht der transportierten Nutzlast und der Beschaffenheit des befahrenen Untergrunds – ggf. um einige Grad geneigt. Dies entspricht in guter Näherung einer Rotation um die y-Achse des Basiskoordinatensystems, die durch die Kontaktpunkte der Räder mit der Fahrebene verläuft.

Verfahren, die von einer *flat world assumption* ausgehen, sind von diesem Inklinationsfehler erheblich betroffen. Eine unvorhergesehene Abweichung

der Plattformneigung e_{inc} führt zu einem Fehler e_m der visuellen Lokalisierung, der von der Distanz d des zu lokalisierenden Objekts und der Höhe h des Sensors abhängig ist:

$$e_m(d,h) = h \tan\left(e_{inc} + \tan^{-1}\frac{d}{h}\right) - d \tag{4.16}$$

Bei einer $h \approx 0.6\,\mathrm{m}$ über dem Niveau der Fahrebene und einem in einer Entfernung von 1.5 m auf dem Boden liegenden Objekt tritt bei einem nicht ungewöhnlichen Fehler von $e_{inc} = 2^\circ$ bereits ein Fehler der Entfernungsmessung von mehr als 10% der gesuchten Strecke auf. Dieser Effekt macht auch die Fusion mit Daten anderer Sensoren praktisch unmöglich. Die Ermittlung des nicht durch Sensorik unmittelbar erfaßten Neigungswinkels e_{inc} ist also zur Verbesserung der Meßgenauigkeit erforderlich, wenn Entfernungsbestimmungen von visuell detektierten Objekten durch Triangulation vorgenommen werden sollen.

Kamerarotation. Eine weitere Fehlerquelle stellt die Montage der Kameras auf der PTU dar. Die Befestigungsklammern verfügen über einen unerwünschten rotatorischen Freiheitsgrad (der allerdings bei Kollisionen mit Objekten eine Zerstörung des Sensors verhindern soll). Auch ohne unerwünschten Kontakt zu Objekten der Umgebung neigen derart montierte Kameras während längeren Gebrauchs dazu, sich zu dejustieren (vgl. auch [Fey00]). Bereits eine visuell noch nicht bemerkbare und mechanisch schlecht kompensierbaren Dejustierung ($|\theta_\delta| \leq 2^\circ$) bleibt nicht ohne gravierende Auswirkungen auf die Auswertung der Sensordaten.

Ansatz. Die mit dem mechanischen Aufbau des Roboters korrespondierende Abbildung 4.16 faßt die Fehlerquellen noch einmal zusammen.[8]

Eine Lösung des Konsistenzproblems über mehrere Sensoren kann durch das Auffinden und Auswerten von Landmarken geschehen, die für beide beteiligten Sensorarten leicht detektierbar sind [BC86]. Vorzugsweise sollte es sich dabei um "natürliche" Umgebungsstrukturen handeln [Fau88]. Beliebige konvexe Ecken von Wänden, Möbeln und anderen für die Dauer des Abgleichs stationären Objekten, die sowohl von LRF und Kamera erfaßt werden, können an Stelle der Testkörper verwendet werden. Hier werden

[8] Die gegenüber der ursprünglichen Veröffentlichung [BM02] abweichende Darstellung ist dadurch bedingt, daß die Korrektur der Abbildungsgeometrie ohne Verwendung eines zusätzlichen Sensors "lokal" und folglich bereits auf einer unteren Designebene gelöst werden kann; vgl. 4.2.3.3.

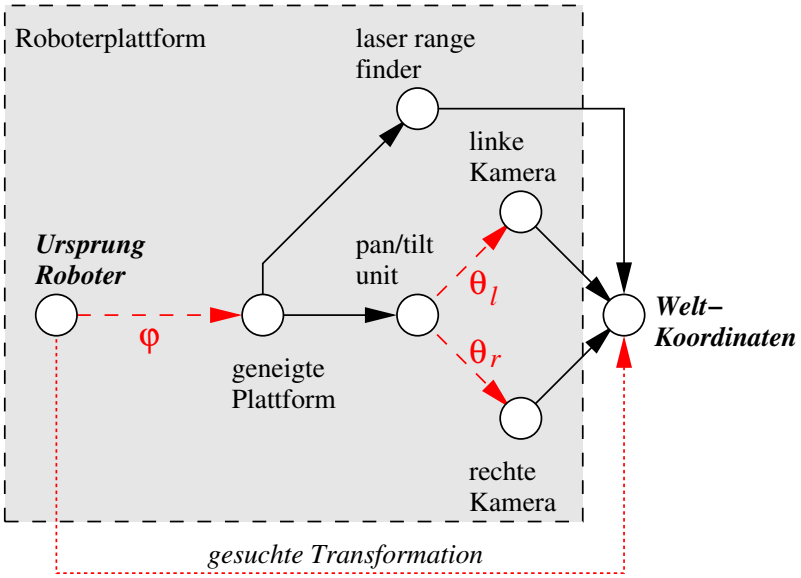


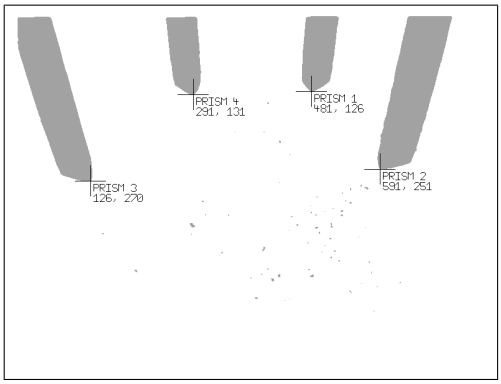
Abb. 4.16: Fehlerquellen in der Effektorkinematik, nach [BM02]: Plattformneigung φ und rotatorischer Freiheitsgrad θ der Kameraaufhängung.

aus gewöhnlichem Papier gefaltete Prismen mit dreieckigem Querschnitt verwendet.

Detektion. Eine einfache Binarisierung unter Zuhilfenahme einiger Modellannahmen mit anschließendem *labelling* dient dem Auffinden der Landmarken im Kamerabild. Die Koordinaten des jeweils "nächsten" Punktes (markiert in Abb. 4.17(b)) werden einer invers-perspektivischen Transformation unterzogen, um Koordinaten in Bezug auf das Roboterkoordinatensystem zu erhalten.

Die Detektion bekannter geometrischer Struktur (hier der Kantenlänge und des Winkels zwischen Kanten) mit einem LRF ist einfach. Abbildung 4.18 zeigt das Resultat für die in Abb. 4.17 entsprechende Versuchssituation. Wegen der relativen Größe des Testkörpers und der dichten Abtastung (Winkelauflösung 0.5°) wurde kein zusätzlicher Versuch unternommen, den genauen Ort der dem Sensor zugewandten Ecke des Primas zu rekonstruieren, was die Genauigkeit der Lokalisierung möglicherweise noch verbessert hätte.

(a) Kamerabild ($\varphi = -45°$)



(b) Binarisiertes Bild mit Objektkoordinaten (Bildkoordinaten)

Abb. 4.17: Sensordatenabgleich. Aus den sowohl von der Kamera als auch vom LRF gleichzeitig wahrnehmbaren Objekten lassen sich einige wichtige Sensorparameter ermitteln. Voraussetzung ist, daß die Objekte sicher detektiert und anschließend zuverlässig Korrespondenzen hergestellt werden können.

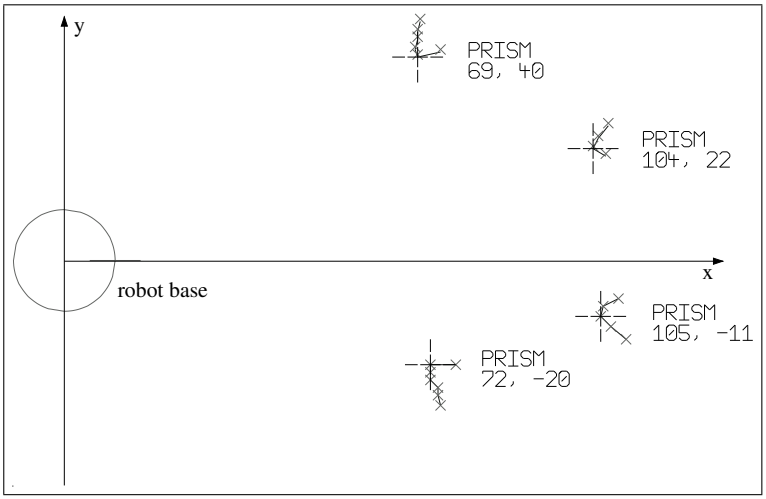


Abb. 4.18: Sensordatenabgleich. Diese Abbildung zeigt die zu der in Abb. 4.17 abgebildeten Situation entsprechenden Detektionsresultate bei Einsatz des LRF.

Verwertbarkeit von Korrespondenzen. Eine neben der Detektierbarkeit wesentliche Eigenschaft der Landmarken ist die geometrische Relation zwischen den von beiden Sensoren gemessenen Punkten: die Testkörper müssen hinreichend aufrecht stehen, sofern die Scanebene des LRF nicht mit der Projektionsfläche übereinstimmt (faktisch ist der LRF so montiert, daß er $\approx 20\,\mathrm{cm}$ oberhalb der Fahrebene Messungen vornimmt). Eine Abschätzung der maximalen theoretischen Schieflage der Testkörper gibt Gewißheit über die Einhaltung dieser Randbedingung und ermöglicht die Bestimmung der Distanz von visuellem Meßergebnis und senkrecht auf die Fahrebene projiziertem LRF-Echo. Um eine stabile Schieflage zu besitzen, muß das senkrecht nach unten projizierte Bild S' des Prismenschwerpunkts S innerhalb der Grundfläche des Testkörpers liegen. Bei gleichseitig dreieckiger Grundfläche und Kantenlänge l kann der Schwerpunkt also maximal den Umkreisradius

$$r = \frac{l}{2\sin(\pi/6)} = l$$

ausgelenkt sein. Bei einer gegebenen Höhe h des Prismas beträgt die maximale Winkelabweichung von der Senkrechten also

$$\delta = \tan^{-1}\left(\frac{2r}{h}\right) = \tan^{-1}\left(\frac{2l}{h}\right)$$

d.h. bei gegebenem maximalen δ, l muß h zu mindestens

$$h \geq \frac{2l}{\tan\delta}$$

gewählt werden.

Da h im Experiment zu 420 mm (der Höhe eines A3-Blattes) und l durch die Faltung auf Viertel festgesetzt ist, kann das Prisma eine maximale Schieflage $\sigma \approx 2.1^\circ$ aufweisen. Bei einer Höhe der Abtastebene des LRF über Grund von $b = 0.2\,\mathrm{m}$ beträgt der maximale theoretisch auftretende systematische Fehler des Verfahrens durch Fehler der eingesetzten Landmarken $e = b\tan\sigma$ hier 6.9 mm. Da diese obere Schranke deutlich unterhalb der Positioniergenauigkeit des Roboters und damit der angestrebten Genauigkeit liegt, kann von einer weiteren Verfeinerung des Verfahrens durch offensichtliche Schritte – wie der Verkleinerung von l – abgesehen werden.

Anzahl und Anordnung der Landmarken. Um überhaupt eine Beziehung zwischen den kartesischen LRF-Koordinaten und den aus der Projektion

der Kamerabilder gewonnenen Punkten herstellen zu können, sind wenigstens drei nicht kollineare Punkte erforderlich; zur Verbesserung der Lösung und Bewertung des Fehlermaßes sind mindestens vier Landmarken erforderlich. Bei dem letztlich angestrebten Verzicht auf synthetische Testobjekte kann eine geeignete Menge von Messungen in der gewünschten geometrischen Konfiguration unter Umständen auch erzeugt werden, indem nur eine Landmarke (z.B. ein Türrahmen) mehreren Messungen mit zwischenzeitlich veränderter Roboterposition und -orientierung unterzogen wird; hierbei muß allerdings auf das vergleichsweise kleine Sichtfeld der Kamera entsprechend Rücksicht genommen werden. Die Nachteile dieses Verfahrens sind, daß die zu bestimmende Plattformneigung bei unregelmäßigem Untergrund leicht variieren kann, und daß eine eventuelle Schieflage des mehrfach als Landmarke verwendeten Objekts die weitere Rechnung mehr beeinflußt als die bei mehreren Landmarken zu erwartende zufällige Verteilung.

Bestimmung und Auswertung von Korrespondenzen. Die Korrespondenzen zwischen den Resultaten der Kamera- und LRF-Detektion lassen sich nach Vorsortierung nach dem Winkel in Polarkoordinatenrepräsentation unmittelbar herstellen. Die durch Steuerung des Effektors erfolgte Transformation kann überdies zur Vereinfachung als initiale Schätzung eingehen. Bei mehr als drei Meßpunkten (wie im vorliegenden Beispiel) sind die Vektoren in $\mathbb{R}^2$ nicht mehr linear unabhängig; das Gleichungssystem wird überbestimmt. Formal wird aus

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{A}^\top\mathbf{A}\mathbf{x} &= \mathbf{A}^\top\mathbf{b} \\ \mathbf{x} &= (\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top\mathbf{b} \end{aligned} \tag{4.17}$$

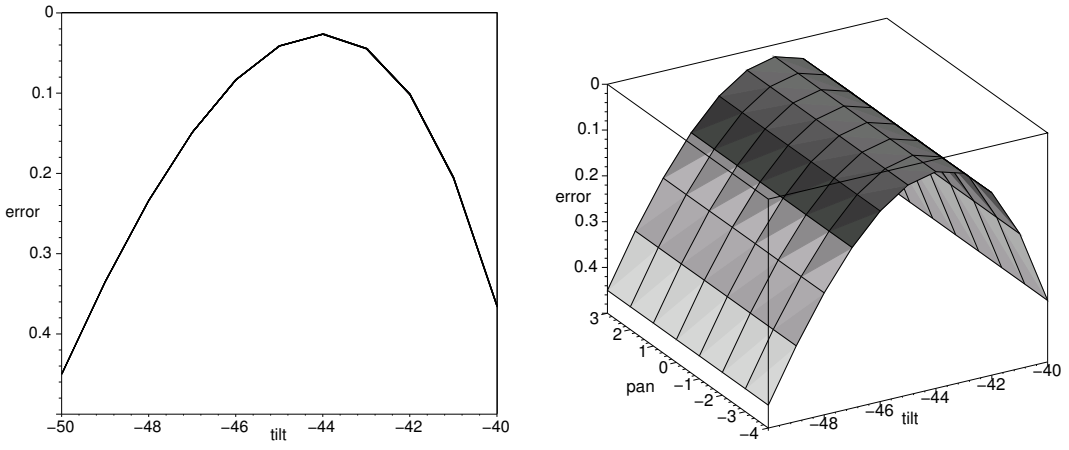
die Lösung mit dem kleinsten Residuum

$$(\mathbf{A}\mathbf{x}-\mathbf{b})^\top(\mathbf{A}\mathbf{x}-\mathbf{b}) \rightarrow \min \tag{4.18}$$

bzw. im Sinne kleinster Quadrate der l_2-Norm

$$||\mathbf{A}\mathbf{x}-\mathbf{b}||_2 \rightarrow \min \tag{4.19}$$

bestimmt. Die Ermittlung des quadratischen Fehlers erlaubt nicht nur eine Abschätzung der Güte der Korrespondenzen (und ggf. eine Erkennung und Zurückweisung falscher Korrespondenzzuordnungen), sondern ermöglicht



(a) Fehler gegen φ (für $\theta = 0°$)

(b) Fehler gegen θ (pan), φ (tilt)

Abb. 4.19: Fehler für die Lösung des Abgleichproblems nach (4.17) bei Variation von θ und φ.

über die Minimierung dieses Fehlers eine iterative Verbesserung der Annahmen über die Sensorlage der Kameras, die bis dahin ausschließlich über die Vorwärtskinematik gewonnen wurde.

Abb. 4.19(a) zeigt, daß nicht der über den Controller der PTU eingestellte und in der Kinematikkette angenommene Neigewert von $-45°$ einen minimalen Fehler liefert, sondern die tatsächliche Neigung der Kamera offensichtlich flacher verläuft. Der tatsächliche *tilt*-Winkel kann also durch Bestimmung des Transformationsparameters angenähert werden, der den minimalen Fehler der Abbildung liefert. Im Fall der Beispieldaten wurde eine Plattformneigung von etwa $-1°$ ermittelt. Die Hypothese für die Kameraorientierung kann also durch das Einbeziehen des LRF-Scans deutlich verbessert werden.

Eine vergleichbare Korrektur sollte theoretisch auch für den Schwenkwinkel der Kamera möglich sein. Wie Abb. 4.19(b) zeigt führt ein θ-Fehler allein jedoch nicht zu einer signifikanten geometrischen Verzerrung der Rückprojektion, da die *pan*-Achse die Scanebene offenbar (wie konstruktiv auch vorgesehen) rechtwinklig durchstößt; jeder Fehler der *pan*-Achse wird bereits durch die ermittelte optimale Transformation kompensiert. – Für Stereobildverarbeitung ist die Ermittlung der θ-Abweichung der Kameras untereinander hingegen wieder von Bedeutung.

Ergebnisse. Die vorgestellte Methode ermöglicht die Schätzung der eingangs beschriebenen Fehler: der Verschiebung des LRF gegenüber der Plattform, der Plattforminklination und einer eventuellen Dejustierung der einzelnen Kameras in θ. Die Ermittlung dieser Parameter gestattet eine Fusion der Daten beider Sensoren.

Da für eine Berechnung der Parameter der Kontakt zu bis zu vier Komponenten des Roboters, nämlich einer Kamera, dem LRF, der PTU und ggf. der beweglichen Basis selbst (im Fall des Verzichts auf künstliche Landmarken) erforderlich ist, zudem eine lokale modellhafte Interpretation der Sensordaten vorgenommen wird, wird das Verfahren nicht im Rahmen der Klassenbibliothek, sondern auf Subsystemebene implementiert. Als auf die Schicht der Protokollabstraktion rückwirkender *top down*-Einfluß erscheint die Notwendigkeit der Einführung von zusätzlichen Parametern der *proxy*-Klassen, die eine Übernahme der so zu ermittelten Abgleichwerte gestatten. Auf diese Weise kann wiederum die transparente erweiterte Funktion der Geräte-*proxy*-Klassen bewirkt werden, von der alle verwendenden Instanzen profitieren.

4.3.3.3 Gewinnung von Umgebungsmerkmalen

Die Verwendung von fehlerbereinigten Sensordaten, die als einfache Zustandsvariablen interpretiert werden können, ist für reaktive Subsysteme hinreichend. Für deliberative Verfahren ist hingegen die Extraktion von Merkmalen der Umgebung eine Vorbedingung. Dies dient der Reduktion des zu verarbeitenden Datenvolumens, der Verringerung des Einflusses von Sensorrauschen, der Schaffung einer Option auf eine angemessene persistente Repräsentation. Diese Form der Weiterverarbeitung ist weitgehend aufgabenspezifisch – abhängig davon, welche Merkmale zur Erfüllung einer Aufgabe erkannt werden müssen – und nicht allgemein zu lösen. Zwar kann der Fall des in 4.2.3.3 benutzten Merkmals (Kontrast-)"Kanten" in Kamerabildern noch als generisch angesehen werden (und ist daher zwar nicht als Funktion des Kamera-Objekts, aber doch als Operator auf dem Bildtyp implementiert), doch wäre die Suche nach den in 4.3.3.2 als Merkmal verwendeten Paaren von einander berührenden Segmenten bekannter Länge in LRF-Scans, die einen bestimmten Winkel einschließen, nur schwer generisch zu formulieren – einmal abgesehen von der Tatsache, daß die Segmente ihrerseits nicht parameterfrei aus den vom Sensor gelieferten Punktmengen hervorgehen. Aus diesen Gründen sind Methoden, die aus beliebigen Sensordaten unter oft nur implizitem Rückgriff auf Strukturannahmen über

die Umwelt eine symbolische Repräsentation generieren oder eine Datenreduktion vornehmen, auf der Subsystemebene zu implementieren.

Architektureinflüsse auf die Betrachtung der Sensoren. Gerade Systeme, die nach den Regeln deliberativer Architekturen entworfen wurden, neigen wegen der für die höheren Ebenen erforderlichen Datenreduktion (s. 2.1.2.5) zur Verwendung von abstrakten Umgebungsmerkmalen – nicht zuletzt, damit der symbolisch operierende Planer geeignete Eingangsdaten erhält. Das hierbei erforderliche *symbol grounding* steckt dabei in den Modellannahmen, die auch den eigentlichen Merkmalsextraktionsmechanismus leiten. Faktisch greift eine Planungsinstanz in deliberativen Schemata nur mittelbar auf die Sensordaten zu, und bedient sich virtueller Sensoren, die symbolische Daten liefern.

Beim Übergang zu hybriden Systemen wird diese Einschränkung nur partiell aufgehoben: zwar können Rohdaten geeigneter Sensoren auch einen unmittelbaren Einfluß auf die Effektorsteuerung nehmen, doch sind zum einen nicht sämtliche verfügbaren Sensoren überhaupt geeignet (eine Kamera ist es in der Regel nicht), und besteht zum anderen auch hier die Notwendigkeit der Versorgung eines Planers mit symbolischen, abstrakten Eingangsdaten. Im Fall des in 2.1.4.3 behandelten hybrid-modellorientierten Ansatzes müssen sogar *sämtliche* Sensordaten auf nur ein gemeinsames Organisationsschema des globalen Weltmodells abgebildet werden, was ebenfalls die Abstraktion von Sensordaten zu symbolischen Merkmalen voraussetzt.

Virtuelle Sensoren. Das Problem der konzeptionellen Integration der ggf. mehrstufigen Weiterverarbeitung zu letztlich symbolischer Information läßt sich elegant durch Einführung virtueller Sensoren lösen. Begrifflich soll als virtueller Sensor jede Instanz bezeichnet werden, die nicht unmittelbar einem physikalischen Sensor entspricht und uninterpretierte Sensordaten liefert, sondern Verarbeitung unter Nutzung von Annahmen über die Außenwelt inkorporiert und Metadaten liefert, die sich auf höherer Ebene als sensorische Eingabe nutzen lassen. Dieses zusätzliches Modellwissen kann unter anderem sein

1. im Fall der Fusion zwischen gleichartigen Sensoren das Wissen über die Anordnung der Sensoren zueinander,

2. im Fall der Fusion zwischen verschiedenen Sensoren die Annahmen über die unterschiedliche Erscheinungsmodalität identischer Objekte,

3. im Fall der Generierung von Metadaten mit nur einem Sensor das Wissen über Auflösung des Sensors oder Annahmen über typische Strukturmerkmale der Umwelt,
4. im Fall der temporalen Fusion mittels einer Akkumulatorstruktur die Schätzung der Eigenbewegung der Plattform oder die Bewegungsmuster von Objekten in dynamischer Umgebung.

Für die Fälle zu 1 und 2 wurden bereits in 4.3.3.2 die Grundlagen gelegt; hier soll der Fall 3 untersucht werden.

Merkmale in LRF-Scans. Exemplarisch soll der LRF als Datenquelle verwendet werden, um die mehrstufige Extraktion von Merkmalen zu erläutern. Bei der Interpretation von LRF-Messungen ist erforderlich, Annahmen zu treffen, die bereits in 4.2.3.5 angeführt wurden; insbesondere ist eine Kenntnis der Standardabweichung der Entfernungsmessung σ_{LRF} erforderlich (die Winkelabweichung wird wegen der robusten Geometrie dieses Sensors vernachlässigt). Als mögliche Merkmalsklassen in hierarchischer Anordnung werden verwendet: F_1 für um protokoll- und gerätespezifische ausgezeichnete Werte (*out of range*, keine Reflektion) bereinigte Rohdaten, F_2 als Merkmale zweiter Ordnung für rekonstruierte geometrische Strukturen, die ein physisches Korrelat in der Außenwelt besitzen, und F_3 für rekonstruierte Metastrukturen, die keine physische Entsprechung besitzen müssen. Beispiele für Merkmale zweiter Ordnung sind Strukturen, die Wänden, Türen oder anderen Hindernissen entsprechen; als Merkmale dritter Ordnung sind Metastrukturen wie die "Mitte" eines Raumes (als Punkt) oder eines Flurs (als Geradensegment) zu nennen. Im folgenden sollen die einzelnen Merkmale dieser Klassen mit ihren Eigenschaften der Methode ihrer Berechnung kurz vorgestellt werden.

Punkte. Bei Verwendung im Vergleich mit beispielsweise der Positioniergenauigkeit des Roboters sehr genauer Sensoren wie einem LRF können die Meßdaten direkt weiterverarbeitet werden. Um die Liste der Punktmerkmale F_1^P zu erhalten, werden lediglich besondere Werte aus der Liste der Rohdaten entfernt, die beispielsweise die Überschreitung der maximalen Meßentfernung signalisieren oder auf bestimmte Fehlfunktionen des Sensors hinweisen. Die Meßpunkte (x_i, y_i) eines Scans D_t sind wegen der Winkelquantisierung nicht unmittelbar mit denen eines Folgescans D_{t+1} in Bezug zu bringen, sofern irgendeine Bewegung der Plattform oder eines Objekts im Erfassungsbereich des Sensors stattgefunden hat, da eine erneute Messung in der Regel nicht am gleichen Satz von Orten stattfindet. Da

	Art	Quelle	Eigenschaften
F_1^P	Punkte	Sensor	nicht unmittelbar zwischen verschiedenen Scans vergleichbar
F_2^S	Geradensegmente	F_1^P	mit Orientierung
F_2^C	Kreisbögen	F_1^P	durch charakteristisches Entfernungsprofil segmentierbar. Problematisch: maximal der dem Sensor zugewandte Kreisteil ist abtastbar.
F_3^P	Schnittpunkte	F_2^S	Kreuzungspunkte rekonstruieren z.B. die Ecken eines Raumes

Tabelle 4.1: Klassen von Umgebungsmerkmalen in LRF-Scans

zudem der Abstand benachbarter Meßpunkte mit wachsender Entfernung vom Sensor stark zunimmt (nur die Winkeldifferenz bleibt bauartbedingt konstant), ist es nicht unbedingt empfehlenswert, F_1^P direkt als Merkmal für das *matching* von Scans heranzuziehen (vgl. [LM94]). Die meisten Ansätze verwenden ein höheres Merkmal (vgl. [GWN99]).

Um höhere Merkmale zu gewinnen muß eine merkmalsabhängige Gruppierung in Teilmengen S_i durchgeführt werden. In Folge wird die Schätzung der jeweiligen Merkmalsparameter durch Minimierung einer ebenfalls merkmalsspezifischen Fehlerfunktion E für jede Teilmenge S_i durchgeführt.

Gerade. Ein sehr einfaches Merkmal ist die Gerade. Obgleich keine physischen Korrelate dieses Konzepts in der Umwelt vorkommen, hat das Merkmal Gerade die u.U. wertvolle Eigenschaft, Meßpunkte auf abschnittweise vorhandenen unterbrochenen Strecken – beispielsweise eines Flurs, dessen Wände Türöffnungen aufweisen – zu sammeln. Um diese Eigenschaft zu erhalten muß bei der Suche nach Geraden über sämtliche Punkte der Liste der Meßwerte iteriert werden. Die Hypothese einer Geraden durch eine Menge von Punkten (x_i, y_i) wird dargestellt durch eine Ausgleichsgerade, die durch $(\bar{x}, \bar{y})$ führt, und die unter Verwendung der Momente

$$m_{j,k} = \sum_{i=1}^{N} x_i^j \cdot y_i^k \tag{4.20}$$

durch Steigung

$$A = \frac{n \cdot m_{1,1} - m_{1,0} m_{0,1}}{n \cdot m_{2,0} - m_{1,0}^2} \tag{4.21}$$

und y-Achsabschnitt

$$C = \frac{1}{n}(m_{0,1} - A \cdot m_{1,0}) \tag{4.22}$$

leicht berechenbar ist. Nach Umformung der Gleichung $Ax + By + C = 0$ mit $B = -1$ durch Multiplikation mit $\mu = \frac{-\operatorname{sgn} C}{A^2+1}$ ergibt sich mit $\mu A = \cos\theta$ und $-\mu C = l$ Orientierung bzw. Länge des Normalenvektors der einfacher transformierbaren HESSE-Normalform. Die so ermittelten Parameter lassen sich zur Aufstellung der Fehlerfunktion

$$E_L = \sum_{i=1}^{N} (x_i \cos\theta + y_i \sin\theta - l)^2 \tag{4.23}$$

verwenden, die die Quadrate der kleinsten Abstände eines Meßpunktes (x_i, y_i) zum durch die Gleichung in Normalform $x\cos\theta + y\sin\theta - l = 0$ beschriebenen Merkmal bestimmt. Durch die Geometrie des Sensors ist $l > 0$ sichergestellt.

Ein Ansatz wie der in [LM94] beschriebene setzt voraus, das bekannt ist, welche Gruppe von Meßpunkten zur zu gewinnenden Struktur beitragen sollen. In der Praxis hat es sich bewährt, die Menge der dem Merkmal zuzuschlagenden Punkte zu schrittweise zu auszudehnen, soweit

1. der hinzukommende Punkt einen kleinen senkrechten Abstand d (z.B. $d \ll 2\sigma_{LRF}$) von der aus der aktuellen Hypothese resultierenden Geraden hat, und

2. die unter Berücksichtigung des hinzugekommenen Punktes verbesserte Hypothese (θ, l) einen mit Gl. 4.23 zu ermittelnden, auf die Anzahl der beteiligten Punkte normierten Fehler unterhalb einer akzeptablen Grenze besitzt.

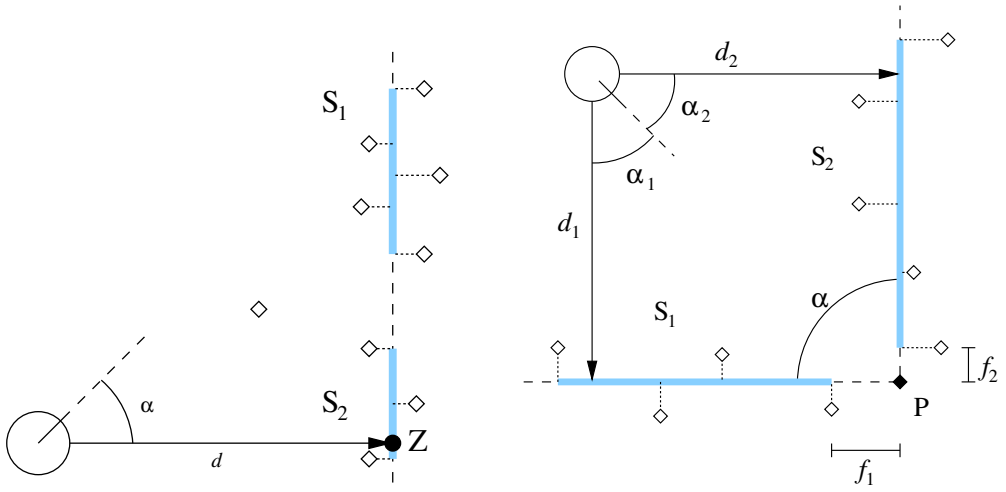
Eine Vernachlässigung der zweiten Bedingung könnte dazu führen, daß Punkte auf Kreisbögen großer Radien, bei denen jeder weitere Punkt nur gering von der Ausgleichsgeraden ausgelenkt ist, abschnittweise durch Geraden modelliert werden, was ausdrücklich nicht gewünscht ist.

Segmente. Zur Gewinnung von Segmenten ist es zweckmäßiger, mit einer nach Winkel der Polarkoordinatendarstellung sortierten Liste der Meßpunkte und einer leeren Liste für die jeweils zu einem Segment gruppierten Punkte zu beginnen. Die Liste der zu einem Segment gehörenden Punkte wird solange erweitert, wie genau der in Winkelsortierung folgende Punkt die im letzten Unterabschnitt genannten Bedingungen einhält; mit dem ersten nicht mehr passenden Punkt wird eine neue Segmenthypothese eröffnet. Dieser *threshold* von $2 \cdot \sigma_{\mathrm{LRF}}$ gestattet die Merkmalsbildung trotz Sensorrauschen und geringen Unebenheiten in der beobachteten Umgebungsstruktur. Kandidaten für Segmente mit einer nicht hinreichenden Zahl von Scanpunkten oder einer zu geringen Elongation des Segments werden verworfen. Segmente werden durch Anfangs- und Endpunkt definiert, die auf der Ausgleichsgeraden liegen; diese Punkte können erhalten werden, indem die dem Segment zuzuschlagenden Punkte senkrecht auf diese Ausgleichsgerade projiziert werden (Abb. 4.20(a)).

Durch partielle Verdeckung und die beispielsweise aus Bewegung der Plattform resultierende Bedeckungsveränderung sind besondere Maßnahmen beim Versuch des Segment-*matching* als Bewertung des Grades an Koinzidenz erforderlich, die in 4.3.3.4 angesprochen werden.

Schnittpunkte und Ecken. Die Schnittpunkte von Geraden können mit vorhandenen Strukturen (z.B. Ecken des Raums) wie mit rein virtuellen Punkten korrespondieren, die nicht mit Umgebungsstrukturen übereinstimmen. In der Menge der extrahierte Segmente hingegen sind grundsätzlich keine Schnittpunkte von je zwei Merkmalen zu erwarten. Zur Rekonstruktion von diesen nicht direkt meßbaren Metamerkmalen hat es sich bewährt, in einer nach Winkel des Mittelpunktes sortierten Liste (in der benachbarte Strukturen auf benachbarte Indizes in der Liste abgebildet werden) von orientierten Segmenten den Endpunkt von S_i und den Anfangspunkt von S_{i+1} zu betrachten. Die Konfidenz der Existenz einer "Ecke" steigt mit wachsender Ähnlichkeit des Winkels zwischen den Segmenten zu 90°und fallender Distanz zwischen beiden Punkten. Ausschlußkriterium für eine Ecke (nicht jedoch einen virtuellen Schnittpunkt) ist allerdings, daß der rekonstruierte Punkt außerhalb der Segmentgrenzen liegen muß (Abb. 4.20(b)).

Für das *matching* sind Ecken wertvoll, da zum einen in die Berechnung ihrer Koordinaten viele Einzelmessungen eingehen und so störende Einflüsse minimiert werden, zum anderen das Merkmal gegenüber wechselnden Verdeckungen recht unempfindlich ist – im äußersten Fall zeigen die in zwei



(a) Segmente und Geraden

(b) Schnittpunktrekonstruktion

Abb. 4.20: Merkmale in LRF-Scans. Geraden und Segmente werden in (4.20(a)) durch Orientierung und Länge des Normalenvektors (α, d) bestimmt; für Segmente erfolgt eine Angabe des abgedeckten Intervalls mit Z als Nullpunkt. In der Schnittpunktrekonstruktion (4.20(b)) wächst die Konfidenz des rekonstruierten Punkts P mit $|\alpha - \frac{\pi}{2}| \rightarrow \min$ und $f_1, f_2 \rightarrow \min$. Dabei ist $f_1 \cdot f_2 > 0$ notwendige Bedingung für eine Ecke.

konsekutiven Scans extrahierten Segmente keinerlei Überlappungen und erlauben dennoch die Rekonstruktion der selben Ecken.

Kreisbögen. Das Feature F_2^C wird ebenfalls direkt aus den Sensordaten gewonnen; es liegt nahe, das Verfahren auf die nach erfolgter Extraktion von F_2^S verbleibenden Restdaten anzuwenden. Punkte auf Abschnitten von Kreisbögen – bei denen maximal der dem Sensor zugewandte Kreisbogen sichtbar ist – sind als charakteristisches, unabhängig von der Distanz stets symmetrisches Entfernungsprofil in den Scandaten gut erkennbar und so leicht zu segmentieren. Für die Konstruktion der Parameter eines Kreises $((x, y), r)$ aus einer Liste von $n \geq 3$ Randpunkten (x_i, y_i) existiert eine Reihe von Verfahren, die im Wesentlichen eine zu (4.23) analoge Fehlernorm wie

$$E_C = \sum_{i=1}^{N} \left(r - \sqrt{(x - x_i)^2 + (y - y_i)^2} \right)^2 \tag{4.24}$$

minimieren. Auch andere Fehlernormen, die den Einfluß sehr dicht beieinanderliegender Punkte in radialer Anordnung reduzieren sollen, sind gebräuchlich (vgl. [UJ00]). Aufgrund der gegebenen Sensorgeometrie ist dieser Fall hier allerdings nicht gegeben. Algorithmisch kann das Zentrum (x, y) als der Punkt bestimmt werden, der als *least squares*-Lösung für den Schnittpunkt der $\binom{N}{2}$ möglichen Mittelsenkrechten entsteht, die aus den approximierten Kreissehnen aller Kombinationen von je zwei Punkten hervorgehen. Der Radius ist im nächsten Schritt als Mittelwert der Distanz zu den Randpunkten zu bestimmen.

Merkmale dieser Art werden durch einen Punkt und einen Radius modelliert. Ein *matching* zwischen Scans kann über einen Vergleich der Translationsdistanz der Zentren vorgenommen werden, wobei die Radien der Korrespondenzfindung zusätzliche Sicherheit geben.

Subsystemeinbettung. Wie an einigen Stellen explizit erwähnt sind die der Merkmalsextraktion dienenden Verfahren in der Regel nicht parameterfrei (wenigstens σ_{LRF} wurde in allen Fällen benötigt). Weiterhin besitzen diese Verfahren vergleichsweise komplexe Signaturen, da Listen von Tupeln aus Merkmalsart, Konfidenz der beschreibenden Parameter (und ggf. der Rohdaten, aus denen die Merkmale hervorgehen) erzeugt werden. Aus diesem Grunde hätte eine statische Integration in eine allgemeine Bibliothek nur begrenzten Nutzen. Für die Implementierung als Subsystem spricht zudem, daß eine Kommunikation zwischen verschiedenen Merkmalsextraktoren den Nutzen bringt, daß die von einem Merkmalsdetektor verwendeten

Meßpunkte aus der Liste der von den anderen Verfahren zu verarbeitenden Punkte wechselseitig entfernt werden.

4.3.3.4 Lokale Selbstlokalisierung

Bedarf. Die Fortführung einer über zahlreiche Bewegungsschritte hinweg konsistenten Positionshypothese ist für viele nicht rein reaktiven Anwendungsprobleme wesentlich. Die Durchführung von Navigation, also koordinierter und zielgerichteter Fortbewegung, ist essentiell für eine mobile Plattform, die sonst über keine besonders große Effektorenreichweite verfügen würde. Für die Überwachung der Navigationsschritte selbst wie auch für praktisch alle weitergehenden Aufgaben wie Exploration und Kartierung ist eine möglichst genaue Kenntnis der eigenen Position in Relation zu einem festen Bezugssystem erforderlich. Dieses Bezugssystem kann dabei willkürlich gewählt sein, bleibt aber mindestens für die Dauer der Durchführung einer Teilaufgabe fest. Die Angabe von Koordinaten in Relation zu einem externen Bezugssystem bleibt – im Gegensatz zur Verwendung robozentrischer Koordinaten – deliberativen Systemen vorbehalten.

Deterministisch vs. probabilistisch. Es existieren mehrere Wege, um eine Lokalisierung zu erreichen. Generell kann das Feld der Methoden in deterministische und probabilistische Varianten aufgeteilt werden; letztere führen nicht nur eine, sondern zahlreiche Positionshypothesen, die mit ihrer Wahrscheinlichkeit annotiert werden (z.B. MCL, s. [DFBT99], oder Lokalisierungsversuche unter Nutzung einer Referenzdatenbank, s. u.a. [Wol01]). Sofern es nicht um initiale Positionsbestimmung anhand einer vorhandenen Karte (des sog. *captured robot problem*) geht, wird für jegliche Verarbeitung auf symbolischer Ebene typischerweise genau eine Positionshypothese bearbeitet. Aus diesem Grunde stehen die deterministischen Lösungsoptionen hier im Vordergrund.

Verfahrenskategorien. In Reihenfolge steigenden Aufwands, aber auch steigender Genauigkeit werden unter der Voraussetzung einer "flachen Welt" (*flat world assumption*) folgende Strategien zur Lokalisierung verwendet:

1. Integration der *gesteuerten* Bewegung,

   dieses sog. *de[a]d reckoning* berücksichtigt allein die gesteuerten Bewegungskommandos, deren ideale Umsetzung mangels sensorischer Prüfbarkeit angenommen wird,

2. Integration der lokal *gemessenen* Bewegung,

   dieses Verfahren verwendet lokale Sensoren (wie Radencoder),

3. Auswertung der Veränderung räumlichen Relation zu Umgebungsmerkmalen unbekannter Position,

   in diesem Verfahren werden korrespondierende Merkmale identischer Strukturen vor bzw. nach einem Bewegungsschritt in Beziehung gesetzt,

4. Bestimmung der räumlichen Relation zu Landmarken *bekannter* Position,

   hier wird durch Bestimmung der relativen Koordinaten der Plattform zu den Landmarken eine globale Positionshypothese berechnet.

In den Fällen 1–3 werden die ermittelten relativen Positions- und Orientierungsänderungen zur schrittweisen Fortschreibung einer initialen und lokalen Positionshypothese (meist willkürlich $(0,0),0^\circ$) verwendet; Fall 4 gestattet hingegen die direkte Ermittlung einer Positionshypothese in Bezug auf das globale System ohne Bezug auf vorherige Messungen und kann somit allein die sonst sich akkumulierende Unsicherheit der Positionshypothese vermeiden.

Problembeschreibung. Das zu 1 genannte Verfahren ist kaum auf längere Sicht mit Erfolg verwendbar, da die effektorseitige Schnittstelle zur Außenwelt von unvorhersehbaren und u.U. zeitlich varianten mechanischen Unzulänglichkeiten erheblich beeinträchtigt werden kann. Die im Fall 2 genannte Hinzunahme von Sensoren ist wiederum nicht in der Lage, Fehlerquellen auch nur zu detektieren, die ihre Ursache beispielsweise in der Kraftkopplung von Antrieb und Untergrund haben. Erst die Berücksichtigung vorzugsweise immobiler externer Bezugspunkte (Fälle 3 und 4) gestattet, auch diese Fehlerquellen in der Fortführung einer konsistenten Positionshypothese zu berücksichtigen.

Lokale vs. globale Selbstlokalisierung. Aufgabe der globalen Selbstlokalisierung ist, die Position der Plattform ohne initiale Positionshypothese durch Auswertung der Sensordaten zu ermitteln. Dies kann auf verschiedene Weisen gelingen: durch Verwendung identifizierbarer Landmarken bekannter Position und bei Vergleich der Sensordaten mit einer Karte und Ermittlung der besten Hypothese geschehen. In der Regel werden weder Landmarken noch eine vollständige Karte vorhanden sein. Aus diesem

Grund wird lokale Selbstlokalisierung erforderlich. Begrifflich wird von einer *lokalen* Selbstlokalisierung gesprochen, wenn das Verfahren der Ermittlung inkrementeller Updates der Positions- und Orientierungshypothese während einer ebenfalls inkrementellen Bewegung der Plattform dient. Der einzige Weg, dies mit maximaler Genauigkeit zu tun, ist, auf externe Umgebungsmerkmale Bezug zu nehmen. Zu diesem Zweck werden zwei Sensorscans von jeweils der letzten und der aktuellen Position der Plattform verwendet. Das Problem besteht darin, die differentielle Änderung in den Freiheitsgraden der Plattformbewegung – $(\mathbf{x}, \mathbf{y}, \theta)$ – zu bestimmen. Verfahren dieser Art benötigen also einen Zustand, der die Sensorwahrnehmung zum Zeitpunkt $t-1$ vor der zu ermittelnden Bewegung geeignet faßt.

Mehrere Gründe können zu einem Fehlschlagen der lokalen Selbstlokalisierung führen. Die wichtigsten sind ein Mangel verwertbarer Merkmale oder sich aus der gefundenen Merkmalskorrespondenz ergebende Widersprüche. Im ersten Fall kann dies an einer Abwesenheit von Merkmalsträgern oder ihrer geometrisch ungünstigen Konstellation liegen, der zweite Fall tritt mitunter dann ein, wenn sich bewegende Objekte als zur Orientierung verwendete Bezugspunkte eingesetzt werden. In diesen Fällen kann vorübergehend ein *failover* zur Integration der gemessenen Bewegung bei gleichzeitiger Vergrößerung der Positionsunsicherheit erfolgen. Diese Positionsunsicherheit wäre zu einem späteren Zeitpunkt nur noch durch Durchführung einer globalen Selbstlokalisierung wieder zu beheben.

Allgemeines Verfahren. Bei dem vorgestellen Verfahren (s. [BKM03b]) zur lokalen Selbstlokalisierung handelt es sich um einen Prozeß, während der Bewegung der Plattform aktiviert sein muß, d.h. um ein Subsystem, das parallel zu allen stattfindenden Bewegungen aktiv wird, die Positionshypothesen der Odometrie erhält und Sensorscans durchführt, um schließlich den Odometriedaten signaturverwandte Daten als verbesserte Positionshypothese zu generieren. Auf welche Weise diese optimierte virtuelle Odometrie auch anderen Subsystemen transparent zur Verfügung gestellt werden kann, wird in Abschnitt 4.4 behandelt.

Zum Zeitpunkt der Initialisierung des Subsystems müssen eine Positionshypothese $H \leftarrow ((0,0),0^\circ)$ – auch eine beliebige andere, sofern globale Koordinaten bekannt sind – und eine Menge S von sensorisch erfaßbaren Umgebungsmerkmalen festgehalten werden: $S \leftarrow features(scan())$. Das Subsystem weist also notwendig einen Zustand auf. Nach jeder Positionsveränderung des Roboters durch entweder motorischen Einsatz oder nur qualitativ detektierten Umwelteinfluß wird eine neue Menge von Umge-

bungsmerkmalen $S' \leftarrow features(scan())$ bestimmt. Die neue Positionshypothese wird zu $H' \leftarrow H + match(S, S')$ durch Summation des relativen, durch Vergleich der Merkmale berechneten Versatzes errechnet. Zur Erleichterung des mit dem Merkmalsvergleich verbundenen Korrespondenzproblems können evtl. vorhandene konventionelle Odometrieinformationen herangezogen werden. Im letzten Schritt werden Positionshypothesen $H \leftarrow H'$ und zugehörige Merkmalsmenge $S \leftarrow S'$ als gültig übernommen, und das Verfahren am der Initialisierung folgenden Schritt fortgesetzt.

Extraktion von Merkmalen. Die Wahl geeigneter Umgebungsmerkmale ist von der Struktur der Umwelt und der Leistung des verwendeten Sensors abhängig. Üblicherweise sind die primären – in der Regel punktförmigen – Merkmale eines LRF-Scans wegen der begrenzten Winkelauflösung des Sensors nicht auf die eines von einem anderen Ort gemachten Sensorscans abbildbar, obgleich durchaus solche Ansätze unternommen werden [LM94]. Zudem wird die Dichte der Meßpunkte mit steigendem Abstand des mit dem Merkmal korrespondierenden Objekts zum Sensor rasch geringer – dieser Effekt kann durch entsprechende Filterung (*density filter* zur Löschung eines Teils der nahen Scanpunkte, [GS96]) kompensiert werden, doch nur unter Inkaufnahme eines zusätzlichen Datenverlustes. Auch die Verwendung abgeleiteter punktförmiger Merkmale wie in [WWP94, PFPB99] ist anfällig gegen partielle Verdeckungen. Aus diesen Gründen scheint es ratsam, höhere Merkmale wie die in 4.3.3.3 beschriebenen zu verwenden, die unabhängig von der Anordnung von Sensor und Objekt die für die Aufgabe der Lokalisierung relevante Essenz des Objekteindrucks fassen.

Modellannahmen. Der Übergang zu höheren Merkmalen ist durch zusätzliche implizite Modellannahmen insoweit gerechtfertigt, als daß durch diesen Schritt keine Änderung des Verhaltens der Plattform angenommen werden muß. Sind zwei Meßpunkte weniger als beispielsweise 5 cm voneinander entfernt, würde eine Substitution durch eine die Punkte verbindende Sehne bei einem Robotradius von ca. 40 cm wenigstens für die Zwecke der Navigation keinen Unterschied machen. Im vorliegenden Fall werden zunächst Geradensegmente F_2^S (s. 4.3.3.3), wie sie in Wänden, Türen und Büromöblierung auftreten, als relevante und, vor allem: statische Strukturen ausgewählt (vgl. Abb. 4.21). In Abhängigkeit vom Einsatzkontext müßten unter Umständen auch andere Strukturen herangezogen werden.

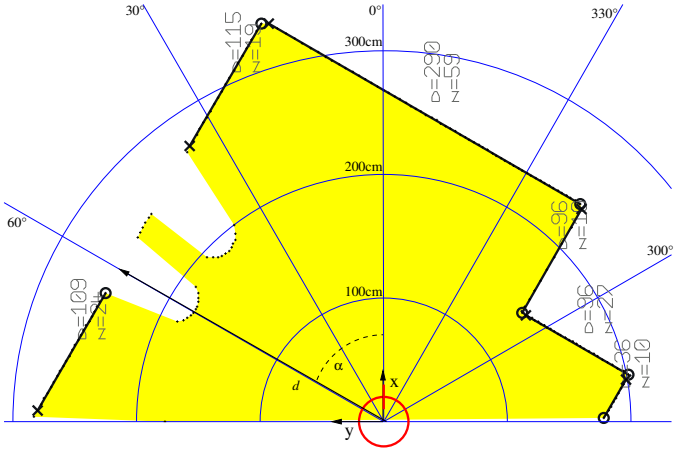


Abb. 4.21: Merkmalsextraktion für Selbstlokalisierung. Beispielsituation: sechs Segmente werden detektiert. Ihre Normalenvektoren werden in roboterbezogenen Polarkoordinaten $(d[\text{cm}], \alpha[°])$ durch $((250, 60), (250, 60), (130, 330), (235, 330), (54, 240), (150, 240)$ beschrieben. Die vollständige Beschreibung der Segmente enthält zusätzlich die Abschnitte auf den so beschriebenen Geraden (vgl. Abb. 4.20(a)).

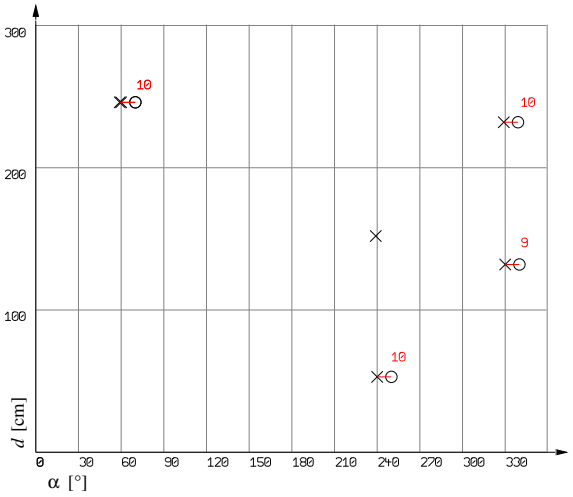


Abb. 4.22: Nutzung von Merkmalskorrespondenzen zur Selbstlokalisierung (s. Text). Die ✕-Eintragungen im Parameterraum entsprechen der in Abb. 4.21 dargestellten Beispielsituation, die ◯-Eintragungen einer veränderten Situation nach unbekannter Plattformbewegung.

Korrespondenzen im Merkmalsraum. Nach Vorverarbeitung der aufeinanderfolgenden Sensorscans D_i und D_{i+1} und Extraktion der Liste der Segmente werden die Mengen der die Ausgleichsgeraden beschreibenden Punkte in ein (α, d)-Polarkoordinatensystem mit den Achsen für Orientierung und Länge des Normalenortsvektors transformiert. Als Ursprung dient in beiden Fällen der Nullpunkt des Roboterkoordinatensystems mit der gegenwärtigen Ausrichtung der Plattform. Sofern keine Plattformbewegung stattgefunden hat, keine sich bewegenden Objekte präsent waren und vom Meßrauschen abgesehen werden kann, werden die Punkte beider Mengen auf jeweils gleiche Koordinaten abgebildet werden.

Um Korrespondenzen in den Merkmalsmengen aufzufinden, kann unter Annahme einer nur kleinen Änderung von Position und Orientierung eine einfache Suche nach dem *nearest neighbor* der jeweils anderen Merkmalsmenge erfolgen. Zur Verwendung einer sinnvollen Distanzfunktion zwischen den Polarkoordinaten bei der Suche nach Korrespondenzen ist eine Skalierung der Achsen α, d erforderlich. Zu diesem Zweck wurden die Achsen durch Wahl von s_1, s_2 so in Abhängigkeit von der Plattform skaliert, daß die Bedingung

$$\sigma_d^2 s_1 \cdot (\sigma_\alpha^2 s_2)^{-1} = 1 \tag{4.25}$$

erfüllt ist. Dies entspricht der Erwartung, daß ein beobachteter Fehler der Distanzmessung σ_d^2 als genau so wahrscheinlich wie ein Fehler der Winkelmessung von σ_α^2 angenommen werden soll.

Abbildung 4.22 zeigt den Merkmalsraum für die in Abb. 4.21 abgebildete Situation; die Merkmalsparameter sind durch das Symbol "$\times$" bezeichnet. Merkmale aus dem Folgescan ("$\bigcirc$") konnten weitgehend zugeordnet werden (Verbindungslinien), wenn auch ein Merkmal ($\alpha = 240°, d = 1.5\,\mathrm{m}$) aus der neuen Situation nicht mehr detektiert wurde. Aus dem Vergleich der Parameter wird erkennbar, daß die zu bestimmende Transformation der Plattform einer Rotation um ca. 10° im Uhrzeigersinn entspricht.

Da die typische Einsatzumgebung im Gebäudeinneren zu einem gewissen Grad Selbstähnlichkeiten aufweisen wird (z.B. Flure mit Türen in regelmäßigen Abständen), macht es Sinn, eine möglicherweise aus der aktiven Steuerung des Roboters bekannte Hypothese über die vorgenommene Transformation in die Berechnung einfließen zu lassen. Zu diesem Zweck werden die Merkmale aus dem *vor* der Bewegung aufgenommenen Sensorscan der geschätzten Transformation unterworfen. Dies erhöht die Ähnlichkeit der Merkmalsmengen und verringert die Wahrscheinlichkeit, daß falsche Korrespondenzzuordnungen getroffen werden.

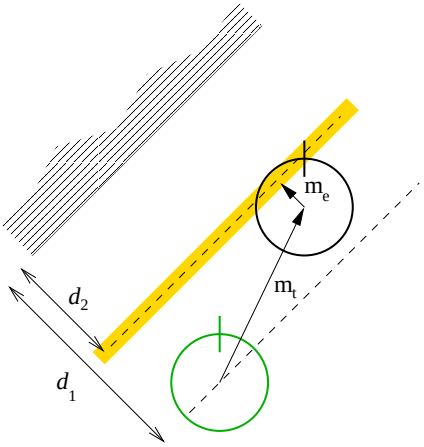
Verifikation der Korrespondenzen. Jede rotatorische Bewegung der Plattform ohne Translationsanteil kann durch Beobachtung einer entsprechenden Verschiebung der Merkmalseinträge des Scans D_{i+1} gegenüber D_i allein entlang der α-Achse festgestellt werden. Dies gilt auch für alle gemischten Bewegungsformen, nur daß in diesen Fällen zusätzlich Änderungen der d-Werte auftreten werden. In allen denkbaren Fällen jedoch sollte die α-Verschiebung für sämtliche Korrespondenzen einen im Rahmen der Meßgenauigkeit übereinstimmenden Wert ergeben. Diese Eigenschaft erlaubt, bei der Korrespondenzfindung möglicherweise auftretende fehlerhafte Paare zu finden und zu eliminieren. Ein zusätzlicher Test der mutmaßlich korrespondierenden Segmente nach Anwendung der geschätzten Transformation auf die Merkmale des ursprünglichen Scans ermöglicht eine weitere Bereinigung, wenn unterbrochene kollineare Strukturen vorhanden sind. Zum Ausdruck eines Ähnlichkeitsmaßes wird

$$\mathrm{Sim}((\alpha_1, d_1), (\alpha_2, d_2)) := \sqrt{(d_1 - d_2)^2 + \Delta(\alpha_1, \alpha_2)^2} \cdot \left(1 - \frac{2|R_1 \cap R_2|}{|R_1| + |R_2|}\right) \tag{4.26}$$

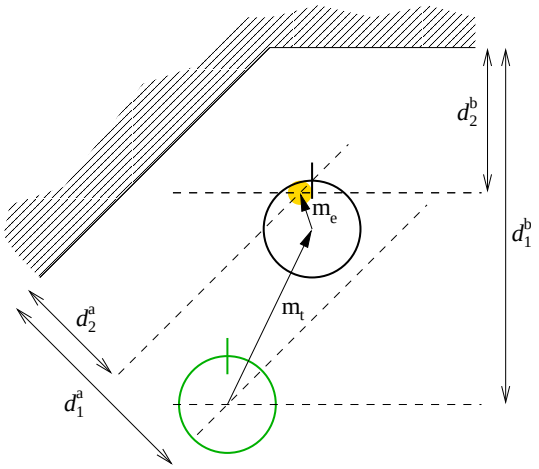
herangezogen, wobei der linke Term die Ähnlichkeit der gemäß Gl. 4.25 skalierten Parameter und der rechte Term die Fraktion der Überlappung des Achsabschnitts R_i der Segmente angibt.

Korrektur des Odometriefehlers. Sofern die Umgebung des Roboters statisch ist, werden alle Merkmale ein im Rahmen der Meßgenauigkeit gleiches gemeinsames $\Delta\alpha$ für den Rotationsanteil der gesuchten Transformation (bzw. des Rotations*fehlers*, wenn Odometriedaten als Schätzung eingeflossen sind) liefern (vgl. Abb. 4.22). In den Fällen fehlerhafter Korrespondenzbildung oder des Auftretens bewegter Strukturen – beispielsweise einer sich öffnenden Tür – wird, eine genügende Zahl korrekt zugeordneter statischer Merkmale vorausgesetzt, dieser Effekt klar in der Verteilung der Differenzwinkel erkennbar sein, was erlaubt, die fehlerhaften Korrespondenzen zu eliminieren.

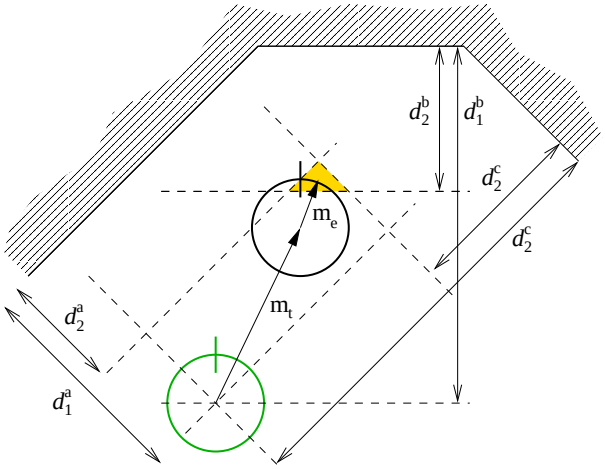
Nach der Wahl einer mit allen verbleibenden Merkmalen verträglichen Winkelhypothese wird eine entsprechende Rotation mit entgegengesetztem Vorzeichen auf die Merkmale des aktuellen Scans angewendet. Das Δd der Korrespondenzpaare gibt jeweils an, um welche Distanz in Richtung der (nun praktisch gleich ausgerichteten) Normalenvektoren die Plattform verschoben wurde. In Abhängigkeit von der Zahl der Merkmalspaare ergeben sich verschiedene Vorgehensweisen:



(a) Unterbestimmter Fall (eine Korrespondenz)



(b) Exakt bestimmter Fall (zwei Korrespondenzen)



(c) Überbestimmter Fall (drei oder mehr Korrespondenzen)

Abb. 4.23: Nutzung von Featurekorrespondenzen zur Selbstlokalisierung

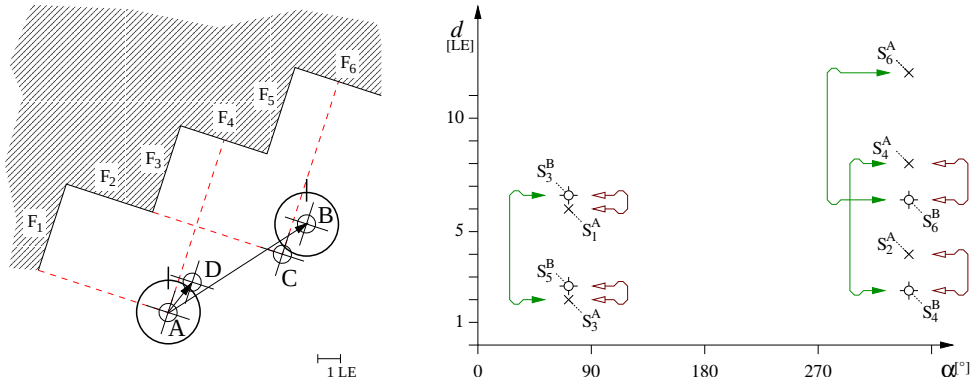
1. Existiert keine Korrespondenz, so ist keine Korrektur von Bewegungsschätzungen möglich.

2. Wenn genau eine Korrespondenz ermittelt werden konnte, ist das zugehörige Gleichungssystem unterbestimmt: obgleich der Rotationsanteil bestimmt werden kann, sind alle Orte in der gleichen Distanz d_2 kompatibel mit Schätzung und Messung. In diesem in der Praxis seltenen Fall wird die Positionshypothese so angepaßt, daß mit minimaler Änderung ein mit der Schätzung konsistenter Zustand erreicht wird (Abb. 4.23(a)). Diese Heuristik könnte nur durch weiteres Wissen über die Art des typischen Bewegungsfehlers verbessert werden.

3. Wenn nur genau zwei Merkmalspaare mit verschiedenen Winkeln detektiert wurden, wird die Positionshypothese so verändert, daß sie beiden Randbedingungen genügt (Abb. 4.23(b)). Die Abschätzung eines Fehlers ist nicht möglich.

4. Sobald mehr als zwei Korrespondenzen verfügbar sind, besteht ein überbestimmtes Gleichungssystem für den neuen Aufenthaltsort der Plattform. Das Gleichungssystem kann durch die Pseudoinverse direkt gelöst werden, um die Koordinaten zu erhalten, die einen minimalen quadratischen Fehler (l_2-Norm) in Bezug zu allen durch die Merkmale festgelegten Geraden des Aufenthalts aufweisen (vgl. Abbildung 4.23(c)).

Schwierigkeiten selbstähnlicher Umgebungen. Das vorgestellte Verfahren zeigt gute Leistungen, solange korrekte Korrespondenzpaare angenommen werden. Trotz der Möglichkeit, fehlerhafte Paare durch die Unvereinbarkeit der Winkelhypothese zu erkennen und eliminieren, kann eine selbstähnliche Struktur der Umgebung zu konsistenten, aber fehlerhaften Korrespondenzannahmen führen. Abbildung 4.24(a) zeigt einen solchen Fall. Wird eine Verschiebung des Roboters $A \rightarrow B$ ohne zwischenzeitliche Aktualisierung der Selbstlokalisierung vorgenommen, wird nicht die Bewegung $A \rightarrow B$ detektiert. Offensichtlich liegt die Ursache darin, daß $A \cong C$, und entsprechend nur die verbleibende Differenz $A \rightarrow D$ wahrgenommen wird. Eine Bewegung $A \rightarrow C$ würde überhaupt nicht detektiert werden können. Im Merkmalsraum (Abb. 4.24(b)) wird der Effekt deutlicher: die Verwechslung der von A erkannten Merkmale S_1^A etc. mit denen des zweiten Scans S_3^B usw. führt zu zahlreicheren und im Sinne des Distanzmaßes besseren Zuordnungen, die wegen der Selbstähnlichkeit auch noch miteinander konsistent sind.



(a) Selbstähnliche Umgebung

(b) Merkmalsraum für eine Bewegung $A \rightarrow B$

Abb. 4.24: Selbstähnliche Umgebungen: Eine Fahrt $A \rightarrow B$ (links) ohne Odometrie kann zu einer Fehleinschätzung der tatsächlich gefahrenen Strecke führen, da die Features der Orte A und C ähnlich sind (rechts). Die angemessene Merkmalszuordnung (Pfeile li.) kann nicht korrekt vorgenommen werden, stattdessen wird durch *nearest neighbor* eine konsistente, aber unpassende Zuordnung (Pfeile re.) vorgenommen.

Durch die Nutzung von Odometrieinformation kann der Fehler nicht generell vermieden werden. Existiert jedoch eine ungefähre Schätzung der zurückgelegten Strecke und der Veränderung der Orientierung, können durch deren Berücksichtigung tatsächlich korrespondierende Merkmale im Merkmalsraum "ähnlicher" dargestellt werden, und kann zudem die Korrespondenzsuche auf Merkmale beschränkt werden, die überhaupt auch von beiden Orten sichtbar gewesen wären. Ist keine initiale Schätzung der erfolgten Bewegung verfügbar oder die Struktur der Umgebung sehr feingranular, bleiben allein die Möglichkeiten, die Zeitintervalle zwischen aufeinanderfolgenden Scans zu verringern, oder die Plattformgeschwindigkeit entsprechend zu verringern. Der maximal anwendbare Quotient aus Geschwindigkeit und Scanrate ist dabei allein von der Umgebungsstruktur abhängig und läßt sich im Merkmalsraum solchermaßen bestimmen, als daß die zurückgelegte Strecke oder die Orientierungsänderung die Hälfte der minimalen Merkmalsdistanz (Δd bzw. $\Delta\alpha$) nicht überschreiten darf, da sonst die *nearest neighbor*-Korrespondenzsuche falsche Paare bildet (vgl. erneut Abb. 4.24(b)). Diese Schranke kann allerdings nicht *a priori* bestimmt werden, da sie sich aus der unter Umständen erst während der Fahrt erschlossenen Merkmalsstruktur ergibt.

Anwendbarkeit auf verschiedene Sensoren. Jeder Sensor, der eine Lokalisierung von Merkmalen in roboterbezogenen Koordinaten zuläßt, ist prinzipiell zur Anwendung des beschriebenen Verfahrens verwendbar. Unterschiede bestehen allein in der allgemeinen Verarbeitungsgeschwindigkeit, die weitgehend durch den Vorgang der Merkmalsextraktion bestimmt wird, der Meßgenauigkeit des Sensors und des Sichtfeldes (*field of view*, FoV) des Sensors. Unter praktischen Gesichtspunkten führen Festlegungen auf eine bestimmte Sensorausprägung zusätzliche Randbedingungen des Verfahrens ein. Auch im Fall der im Folgenden demonstrierten inkrementellen monokular-visuellen Selbstlokalisierung resultiert dies aus den prinzipiellen Anforderungen: der Fähigkeit zur genügend genauen Lokalisation von Landmarken in Bezug auf das Koordinatensystem des Roboters, und der Detektion von mindestens zwei nicht-parallelen Merkmalen.

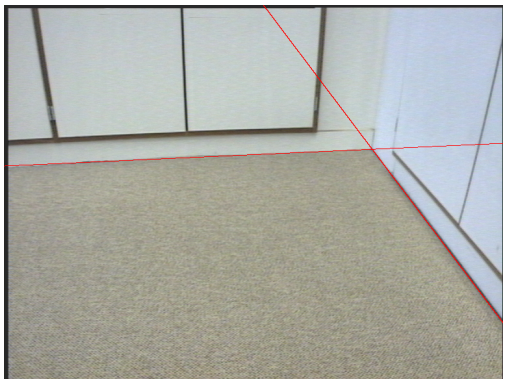
Die erste Bedingung ist eine Einschränkung allein in Bezug auf die Möglichkeit zur Detektion entsprechender Features. Wird – wie im Folgenden – der Übergang von Wand zu Bodenfläche verwendet, besteht eine Verwandtschaft zu dem Featuretyp F_2^S (nach der Klassifikation in Tab. 4.1). Dies impliziert Einschränkungen in Bezug auf Farbe/Textur der Flächen und die

Struktur der Umgebung (keine Verdeckung durch Möbel etc.).[9] Die zweite Bedingung ist nur bei omnidirektionalen Kameras dauerhaft erfüllt; bei Verwendung konventioneller Kameras muß der Blick zum Beispiel in eine Ecke des Raumes gerichtet sein. Um die selben Merkmale erneut detektieren zu können, ist wegen des vergleichsweise kleinen Sichtfelds in der Regel während der Bewegung eine Nachführung der Kamera entsprechend der Bewegungshypothese erforderlich.

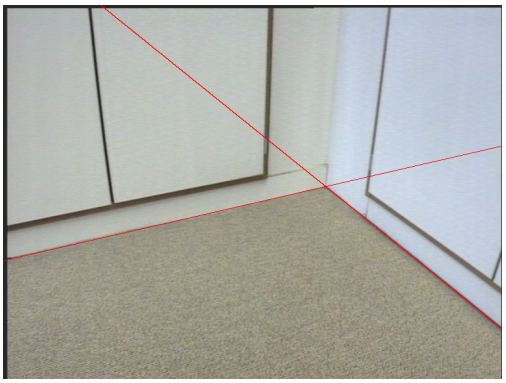
Beispiel: Kamera. Da der Roboter mit Weitwinkelkameras ausgestattet ist, ist bei entsprechender Beschränkung des Rotationsanteils auch eine visuelle Bestimmung der gesuchten Transformation möglich. Sofern dem Subsystem zusätzlich eine aus der Odometrie gewonnene Bewegungshypothese als initiale Schätzung zugestanden wird, kann durch die bewegliche Kameramontage ein größeres virtuelles Sichtfeld genutzt werden. Als Merkmale dienen die Kanten, die die Fahrebene von Wand oder sonstigen Objekten trennen. Die Fähigkeit zur Extraktion der Merkmale wird vorausgesetzt; die Vermessung kann wegen der bekannten Höhenhypothese $z = 0$ monokular erfolgen (vgl. 4.3.3.1).

Abbildung 4.25 zeigt ein solches Experiment – zwischen den Aufnahmen wurde die Plattform einer aus Translation und Rotation zusammengesetzten Transformation unterzogen. Ist es möglich, in den Kamerabildern solche Merkmale zuverlässig zu bestimmen, können diese unter Berücksichtigung der Roboter- und Kamerageometrie in Geraden bzw. Geradensegmente auf der Fahrebene (Annahme $z = 0$ für den Übergang zwischen Wand und Fußboden) rückprojiziert werden. Der in der Abbildung trapezförmig umrissene Bereich in der Rückprojektion deutet den einsehbaren Bereich an und zeigt die Notwendigkeit des Nachrichtens der Kamera, um mit höherer Wahrscheinlichkeit korrespondierende Merkmale auffassen zu können. Aus der Merkmalszuordnung kann die gesuchte Positionstransformation der Plattform abgeleitet werden; im vorliegenden Fall wurde entsprechend eine Translation von (345 mm, 8 mm) in Richtung x- bzw. y und eine Rotation um 32° (linksdrehend) ermittelt. Die Grenzen des Verfahrens mit diesem Sensor liegen in dem geringen Sichtfeld des Sensors und der langsamen,

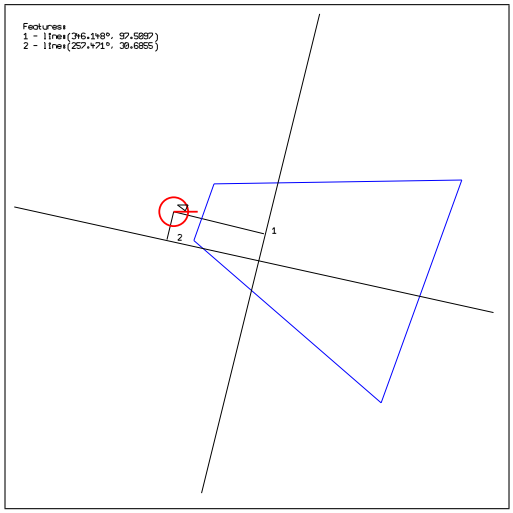
[9] Ebenfalls denkbar wäre eine Nutzung des Übergangs von Wand zur Decke, da so in einer geringeren Zahl von Fällen mit Verdeckungen durch andere Objekte zu rechnen ist. Dieses Merkmal wird – für ein anderes Verfahren – in [Jen99, Jen01] in Kombination mit einem LRF tatsächlich verwendet. Gegen den Einsatz bei visuellen Sensoren sprechen zum einen der in der Regel geringe Kontrastunterschied zwischen den Flächen, und zum anderen die Notwendigkeit der Einführung einer Hypothese über die Deckenhöhe (die vom LRF direkt gemessen wird).



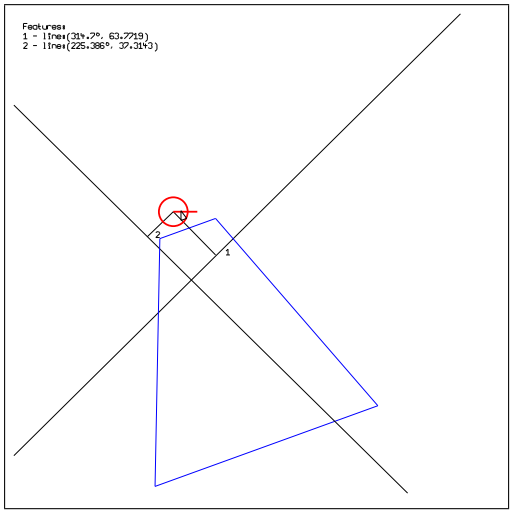
(a) Kamerabild zu t_0: $\theta = -20°$, $\phi = -40°$



(b) Kamerabild zu t_1: $\theta = -80°$, $\phi = -40°$



(c) Rekonstruktion für t_0



(d) Rekonstruktion für t_1

Abb. 4.25: Versuch zur inkrementellen visuellen Selbstlokalisation. Im einsehbaren Bereich (in der Projektion auf $z = 0$ trapezförmig umrissen) werden je zwei Merkmale erkannt. Zwischen t_0 und t_1 fand eine zu ermittelnde Bewegung der Plattform statt. Nach Herstellung der Merkmalskorrespondenz können Rotation und Translation der Plattform aus den Normalformparametern der Merkmale direkt bestimmt werden.

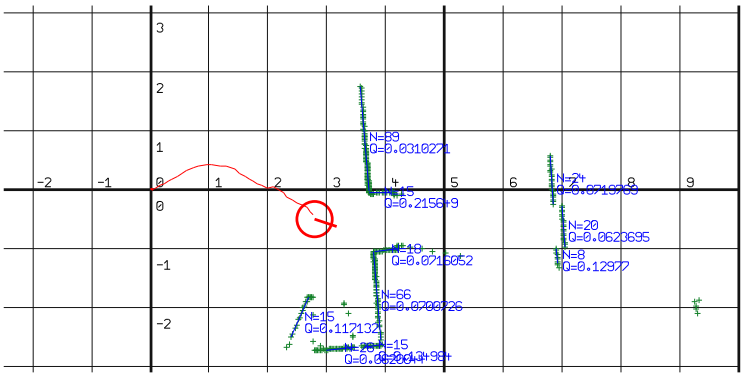


Abb. 4.26: Tracking der Plattformbewegung durch LRF-merkmalsbasierte Selbstlokalisierung

zudem nicht hinreichend robusten Merkmalsgewinnung, die ein hohes Datentransfervolumen in der Kommunikation mit dem Sensor mit sich bringt und entsprechend hohe Rechenzeit für die Extraktion der Merkmale in Anspruch nimmt.

Beispiel: LRF. Das vergleichsweise große Sichtfeld des LRF von 180°und die hohe Meßgenauigkeit (4.2.3.5), die schnelle Anbindung des Meßsystems und die stark vereinfachte Merkmalsextraktion (4.3.3.3) lassen diesen Sensor zum Einsatz im hier vorgestellten Selbstlokalisierungsverfahren besonders geeignet erscheinen. Experimente zeigen (Abb. 4.26), daß ein Tracking der Eigenbewegung ohne Verwendung von Odometriedaten möglich ist. Dabei erscheinen in Bezug auf ein globales Koordinatensystem registrierte Merkmale in der nur temporären "Karte" statisch. Faktisch wird auf diese Weise die bereits in 4.2.3.2 festgestellte sonst wachsende Orientierungsunsicherheit der reinen Odometrie behoben, solange zwischen allen Teilschritten der Bewegung eine inkrementelle Positionsbestimmung möglich ist.

Da keine unabhängige Möglichkeit zur Bestimmung der tatsächlichen Plattformbewegung besteht, wurde ein Test der Lokalisierung im Rahmen einer die Sensoreigenschaften berücksichtigenden Simulation durchgeführt (die auch in [Küp03] bei der Bewertung der dort vorgestellten Verfahren zum Einsatz kam). Nach 80 simulierten Bewegungsschritten – jeweils eine Bewegung um 0.1 m in x-Richtung der Plattform und eine Rotation um 10°– wurde in einem einfachen Szenario ohne jede zusätzliche Odometrieinformation ein Fehler der akkumulierten Positionshypothese von ((0.0027 m, 0.02815 m), 0.41°) bestimmt. Der Fehler rührt hauptsächlich aus der mo-

dellierten Ungenauigkeit der Meßwertgewinnung und fällt angesichts der acht Meter Vorwärtsfahrt und der Rotation um insgesamt 400°akzeptabel aus. Für eine Analyse der Entwicklung des sich akkumulierenden Fehlers sei erneut auf [Küp03] verwiesen.

Einbindung. Die vorgestellte Systemkomponente nimmt den Status eines virtuellen Sensors an, da die von ihr produzierten Daten nicht unmittelbar auf einen der Sensoren der Plattform zurückgehen. Das Verfahren ist zustandsbehaftet, wenngleich nur ein relativ schmales Zeitfenster existiert, in dem Merkmalsdaten gespeichert werden müssen, und nimmt bei Verfügbarkeit von Odometriedaten eine Form von Sensorfusion vor, da diese Information ebenfalls vorteilhaft in die Ausgabedaten eingeht. Wegen dieser Verknüpfung und des notwendig parallelen Ablaufs zu anderen Instanzen, die aktiv eine Plattformbewegung vornehmen, wurde eine Integration in Form eines Subsystems vorgenommen. Die Ausgabedaten dieses Subsystems sind dergestalt, daß sie von allen Konsumenten der ursprünglichen ungenauen Systemodometrie an ihrer Stelle verwendet werden können; dies würde zu einer verbesserten Systemleistung führen. Die Form, in der dies geschehen kann, ist Gegenstand der Architekturkonfiguration (Abschnitt 4.4) und soll hier noch nicht behandelt werden.

4.3.4 Auswertung

In den vorangegangenen Abschnitten wurde demonstriert, wie bestimmte Teilprobleme innerhalb eines lokalen Kontexts gelöst werden können. Bei der Lösung tritt insbesondere keine Bindung an die spezifische Roboterplattform auf, da eine Abstraktionsschicht die Verfahren auf Subsystemebene von der eingesetzten Hardware trennt. Eine bleibende Voraussetzung für die Funktion bestimmter Subsysteme ist hingegen das Vorhandensein einer bestimmten Substruktur der Hardwareplattform, auf die die gekapselten Verfahren aufsetzen: so wurden beispielsweise in 4.3.3.1 Kamera *und* PTU benötigt.

Wie in 4.2.2.2 erläutert, können die Hardware-*proxy*-Klassen nur einen Ausschluß der Existenz mehrerer Instanzen zu einer Hardwarekomponenten sicherstellen. Eine Konfliktbeseitigung war auf dieser Ebene nicht vorgesehen. Durch Instantiierung mehrerer Subsysteme kann es geschehen, daß diese in Konkurrenz untereinander um Sensoren und Effektoren treten. Der Fall des gemeinsamen Effektorzugriffs ist offensichtlich problematisch,

doch auch Sensorzugriffe können den Wechsel des Betriebsmodus der Sensorkomponente bewirken und so eine gemeinsame Nutzung unmöglich machen. Die Lösung auf dieser Ebene liegt in der Einführung eines weiteren Subsystems, das exklusiv mit der Roboterhardware kommuniziert. Wie die Konfliktvermeidung auf dieser Ebene geschieht, ist ebenfalls Gegenstand des Folgeabschnitts 4.4.

Generell können auf der Ein- und Ausgabeseite von Subsystemen unverarbeitete Sensorinformation, verarbeitete Information von virtuellen Sensoren und plattformbezogene Zustandsinformation auftreten. Die vorgenommene Erweiterung zur Kommunikation mit einem den Roboter steuernden Subsystem läßt auch Kommandos zur Effektorsteuerung zu.

Trennbarkeit der Teilsysteme. Aus der Darstellung der exemplarisch vorgestellten Subsysteme ist ersichtlich, daß kein Teil des Gesamtsystems über ein weiteres Wissen verfügen muß oder einen zusätzlichen Zugriff auf andere technische Systeme der Roboterplattform benötigt, um seine Aufgabe vollständig erfüllen zu können. Bereits aus softwaretechnisch-methodischer Sicht ist eine Kapselung der aufeinander bezogenen, aber in ihrer Funktion getrennten Teile in funktional unabhängige Subsysteme naheliegend.

Sollte es möglich sein, eine angemessene und allgemeine Kommunikationsschnittstelle zwischen den Teilsystemen zur Verfügung zu stellen, wäre der Gewinn eine Kapselung der einzelnen Funktionsmechanismen unter Achtung des Geheimnisprinzips mit den Folgen der Entwurfsvereinfachung, separaten Testbarkeit und späteren Austauschbarkeit.

Lokalität von Architektur. Bei näherer Analyse der vorgestellten Teilsysteme ist festzustellen, daß einige nur mehr eine reine funktionale Datentransformation vornehmen, während andere über einen persistenten teilsystemlokalen Zustand verfügen, der in einem deliberativen Prozeß in die Verarbeitung eingeht. Ebenso sind Teilsysteme möglich, die eine durch lokale Zustandsinformation parametrisierte funktionale Kopplung implementieren.

Insgesamt ist also festzuhalten, daß bereits einzelne Subsysteme genau die Charakteristika aufweisen, die in Kapitel 2 vollständigen Architekturen zugeschrieben wurden: Ein Subsystem kann “reaktiv”, “deliberativ” oder “hybrid” sein. Im Grunde besteht tatsächlich kein qualitativer Unterschied zwischen den in den Abschnitten 2.1.2–2.1.4 vorgestellten Architekturen und einem allgemeinen Teilsystem nach der Diktion von 4.3.2, das sämtliche

Funktionalität erbringt (d.h. sich mit allen Sensoren, Effektoren und der wie auch immer gearteten Kontrollimplementierung befaßt).

Die Annahme, daß eine komplexere Architektur allein aus Teilsystemen eines bestimmten Typs aufgebaut zu sein habe (wobei die Kommunikationswege zwischen den Teilsystemen noch zu ergänzen wären), geht am Kern des Problems vorbei. Aus pragmatischen Gründen macht es mehr Sinn, *lokal* reaktiv, deliberativ oder hybrid implementierte und von den Verdiensten der jeweiligen Kontrollparadigmen profitierende *Teilsysteme* in eine Gesamtarchitektur zu integrieren, um ihre spezifische Funktion nutzbar zu machen. Damit ginge eine wesentliche vormals den Architekturen zugeschriebene Eigenschaft auf ihre Komponenten über.

Um diese Integrationsleistung zu erbringen, ist die *Kopplung* der Teilsysteme erforderlich. Mit der Frage, wie dies zweckmäßigerweise zu geschehen hat, befaßt sich der folgende Abschnitt.

4.4 Architekturkonfiguration

Die im vorangegangenen Abschnitt 4.3 vorgestellte Methode ist zwar der Partitionierung von Gesamtsystemen in Subsysteme dienlich, geht aber noch nicht auf die Interaktion mehrerer Subsysteme ein, die letztlich ein System konstituieren.

Abschnitt 4.4.1 geht auf die Probleme ein, die sich sowohl aus dem Zusammenfügen separater Module wie der eigentlichen Kommunikationsschnittstelle ergeben. In 4.4.2 wird der gewählte Ansatz mit der seinen konstruktiven wie emergenten Eigenschaften und schließlich der technischen Umsetzung diskutiert, während 4.4.4 auf die Realisierung von entsprechenden Prototypen zu Testzwecken eingeht. Abschließend wird in 4.4.5 eine zusammenfassende Bewertung des Systemmodells vorgenommen.

4.4.1 Problem

Im letzten Abschnitt wurde die Schaffung einer Möglichkeit zur Bereitstellung allgemeiner, nicht zweckgebundener und von der Hardwareplattform weitgehend abstrahierter Systemfähigkeiten vorgenommen, die als Subsysteme auftreten. Diese Subsysteme weisen jeweils die vollständigen Charakteristika der Architekturen reaktiver, deliberativer bzw. hybrider Systeme auf.

Dies ist darauf zurückzuführen, daß spezifische Problemlösungen mit bestimmten *Architekturschemata* eng korrespondieren: die Verarbeitung geschieht jeweils lokal in einfachen (parameterfreien) Fällen reaktiv, deliberativ (unter Nutzung lokaler Zustandsinformation, was sich technisch in subsystemlokalen Variablen niederschlägt) oder hybrid, wenn sowohl Zustandsinformation als auch eine schnelle Reaktion gefordert sind.

Zur abschließenden Beantwortung der Fragestellung, wie eine systematische Konstruktion möglichst allgemeiner Robotersysteme zu geschehen habe, ist nun eine Lösung des Problems zu bestimmen, ob und wie nicht nur nicht näher bestimmte "Softwaremodule", sondern Einheiten, die alle praktischen Erscheinungsmerkmale von "Architekturen" aufweisen, miteinander zu einem System zusammengefügt werden können.

4.4.1.1 Wege der Organisation von Untereinheiten

Für die Organisation von Untereinheiten werden üblicherweise zwei zueinander komplementäre Ansätze verwendet, namentlich die Spezifikation von (Klassen-)Bibliotheken und die Konstruktion von Rahmenwerken.

Bei Verwendung einer Bibliothek oder ihrer objektorientierten Variante, der Klassenbibliothek, findet grundsätzlich eine explizite Übergabe der Kontrolle an die verarbeitende Einheit statt. Dies wird zwei Randbedingungen nicht gerecht: zum einen handelt es sich bei Komponenten der Roboterhardware um logisch unabhängige Komponenten (wenigstens Sensoren können in der Regel ohne jede logische Interferenz von mehreren Subsystemen benutzt werden), zum anderen sind auch die Subsysteme bis auf die bei Parameterübergabe und Resultatrückgabe auftretende Synchronisierung voneinander logisch unabhängig. Dieses Potential für Nebenläufigkeit und die somit unausgesetzte wenigstens partielle Funktion – auch im Fall eines Versagens von Systemteilen – ist mit einem einfachen Bibliotheksansatz nicht auszuschöpfen.

Unter Vorgabe eines Rahmenwerks kann diese Einschränkung umgangen werden, wenn eine Nebenläufigkeit von Kontrollinstanz und allen weiteren Teilen des Systems vorgesehen wird. Hier ist aber ein Grad an kommunikativer Verschränkung zwischen den Systemteilen erforderlich, der in Robotikanwendungen aus verschiedenen Gründen nur schwer erreicht werden kann. Der Rahmenwerksansatz beispielsweise in [KM99] erlegt den Teilsystemen durch Einsatz eines nicht-präemptiven Multitasking eine regelmäßige Rückgabe der Kontrolle auf und scheitert mit nicht vorhersagbarem Resultat, sobald auch nur ein Teilnehmer den Zeittakt nicht einhalten

kann. Die Verarbeitungsgeschwindigkeit von Datenempfängern entspricht zudem in der Regel nicht der Rate der Generierung der Daten in (nebenläufigen) Systemteilen, und eine Verarbeitung von *events* kann zudem datenabhängig verschiedene Verzögerungen hervorrufen. Langsame nebenläufige Systemteile könnten so zu einem monotonen Wachstum der *event queue* führen, was für ein simuliertes, aber nicht für ein reales System akzeptabel ist.

Beide Lösungen sind für das Problem der Organisation von Subsystemen in Robotikanwendungen demnach unangemessen und kommen hier nicht in Frage. Es ist also zu prüfen, ob eine den speziellen Anforderungen der Robotik angemessene allgemeine Grundlage für die Umsetzung von Architektur konstruiert werden kann, die diese Einschränkungen nicht aufweist.

4.4.1.2 Randbedingungen des Problems

Die Randbedingungen für eine solche Systemgrundlage erwachsen aus der Notwendigkeit der Organisation von Subsystemen zu einem Gesamtsystem, aus der zur Laufzeit erfolgenden Kommunikation zwischen den Subsystemen und aus den Eigenschaften der Subsysteme selbst. Bei allen Teilaspekten ist zu berücksichtigen, daß die Subsysteme unter der Annahme einer selbständigen Ausführung in eigenen *threads* formuliert wurden, eine verfahrensbedingt unterschiedliche und zeitlich variable Rate der Kommunikation aufweisen, und von Funktionsausfällen von sowohl der verwendeten Hardware wie auch des Verfahrens selbst betroffen sein können.

Organisation. Im einfachsten Fall könnte ein einzelnes, wenn auch komplexes Subsystem über die Schnittstelle der in Abschnitt 4.2.2.2 vorgestellten Klassenbibliothek Zugriff auf Sensordaten vornehmen, eine wie auch immer geartete Verarbeitung durchführen und die Effektoren entsprechend ansteuern. Solch ein monolithisch konzipiertes System zieht allerdings keinen Profit aus der Gesamtstruktur. In der Regel wird eine Zusammenarbeit von Subsystemen erfolgen, die einen spezifischen Aufgabenaspekt lösen. Dies bedarf der vermittelnden Organisation.

Zu den Aufgaben der Kontrolle zählt das Subsystemmanagement mit den Teilaufgaben der Aktivierung und Deaktivierung von Subsystemen, die Schaffung der Möglichkeit des Austausches solcher Systemteile und natürlich die Bereitstellung einer Infrastruktur zur Kommunikation zwischen den Subsystemen. Es ist zu berücksichtigen, daß durch das Rahmenwerk nicht genau eine Architektur des Systems bestimmt sein soll. Wie bereits

im Abschnitt 4.4.1 einleitend dargelegt kann möglicherweise die Auswahl der in einer Gesamtkonfiguration verwendeten Subsysteme das determinieren, was als "Roboterarchitektur" bezeichnet wird. Hingegen soll die allgemeine Struktur, die Subsysteme wie auch immer organisiert, dies nicht tun, um den Freiraum des Designs nicht im Vorwege unnötig einzuschränken. Aus dem gleichen Grund ist die *späte Festlegung der Systemkonfiguration* zur Übersetzungs-, besser noch: Laufzeit gewünscht.

Kommunikation. Das Instantiieren von unabhängigen Subsystemen, die verschiedene Aspekte der Verarbeitung unter Verwendung einer lokalen Bindung zu je eigenen Sensoren und Effektoren durchführen, ergibt jedoch bestenfalls ein reaktives Gesamtsystem nach dem *behavior based*-Modell (2.1.3.3), ist aber bereits für einen Subsumptionsstil (2.1.3.4) nicht mehr ausreichend. Als minimale Anforderung ist also zusätzlich eine Möglichkeit zur Signalisierung zwischen den Subsystemen vorzusehen.

Die sich aus dem Kommunikationsaspekt ergebenden Randbedingungen sind also, daß echte Nebenläufigkeit möglich sein soll, soweit logische Unabhängigkeit herrscht; dies impliziert insbesondere die Forderung einerseits nach einem Synchronisationsmechanismus, andererseits nach der Vermeidung des partiellen oder totalen Blockierens im Fall nicht kommunikationsbereiter (d.h. langsamer oder gar versagender) Subsysteme. Da die von einem Subsystem generierte Information eine bestimmte syntaktische Beschaffenheit hat, handelt es sich bei der zu kommunizierenden um typisierte Information. Über eine Verbindung von Subsystem zu Subsystem soll allerdings erst frühestens zur Übersetzungszeit, besser erst zur Laufzeit entschieden werden, um ein Maximum an Flexibilität zu erreichen.

Die zu diesem Zweck erforderlichen minimalen Fähigkeiten des Rahmenwerks in Bezug auf ein einzelnes Subsystem sind also die Entgegennahme der Arbeitsvoraussetzungen (sofern eine Eingabe von anderen Subsystemen geschehen soll), die Aktivierung (spätestens dann, wenn eine zu verarbeitende Eingabe vorliegt) oder Benachrichtigung, das vorläufige Entgegennehmen von Ausgaben und die Weiterleitung von solchermaßen produzierter Ausgabe an einen Kreis von Empfängern.

Eigenschaften der Subsysteme. Die Subsysteme wurden in 4.3.3 unter dem Aspekt ihrer internen Verarbeitung und Kopplung mit den Aggregaten des Roboters über die hierfür als Grundlage dienende (in 4.2.2.2 vorgestellte) Klassenbibliothek dargestellt. Dies schloß weder die Einbettung der Subsysteme in eine "Gesamtarchitektur" noch die Berücksichtigung von

Schnittstellen zwischen verschiedenen Subsystemen ein. Auch aus diesen Aspekten resultieren Anforderungen an das Rahmenwerk, um keine weiteren Auflagen an die Entwicklung von Subsystemen knüpfen zu müssen.

Die minimalen Anforderungen an ein Subsystem, das im Zusammenhang des beschriebenen Rahmenwerks funktionieren soll, sind die Fähigkeiten, geladen und entladen zu werden, zu "laufen", auf eine Eingabe von einem Eingang oder mehreren Eingängen zu warten und die Eingangsdaten zu synchronisieren, einen Zustand zu besitzen (im Fall integrierter Deliberation) und eine Ausgabe durch Berechnung als Funktion von Zustand und Eingabe zu produzieren und sie dem Rahmenwerk zu übergeben, und schließlich, sich zu deaktivieren. Ob ein Subsystem tatsächlich über einen eigenen Zustand oder einen lokalen Kontrollfluß verfügen soll, ist sehr von dessen Aufgabe abhängig; wie bereits eingangs festgestellt, entsprechen Subsysteme in ihrem lokalen Geltungsbereich praktisch dem Schema einer vollständigen Architektur. Grundsätzlich ist davon auszugehen, daß ein Subsystem weitgehend asynchron zu allen anderen Softwarekomponenten ausgeführt werden sollte – denn in diesem Fall wären dem Subsystem keine weiteren Beschränkungen in Bezug auf Rechenzeit, Verfügbarkeit, Antwortverhalten und Kooperation beim Multitasking auferlegt. Als optional ist die Forderung anzusehen, Systemteilen mit deliberativer Komponente eine Datenpersistenz über mehrere Sitzungen zu gestatten. Da einige Subsysteme von dieser Eigenschaft in erheblichem Maße profitieren (wie z.B. der Speicherung einer Karte und der globalen Plattformposition in ihr vor dem Terminieren), ist explizit ein "Entladen" bzw. "Laden" des Subsystems vorgesehen. Während dieser Vorgänge soll noch unter Kontrolle des Subsystems die Speicherung bzw. die Wiederaufnahme eines komplexen persistenten Subsystemzustands erfolgen können.

Wegen der Allgemeinheit der Anforderungen ist es sinnvoll, die Mechanismen ihrer Erfüllung vom einzelnen Subsystem auf das Rahmenwerk zu verlagern.

4.4.1.3 Zur Frage der Testbarkeit

Über die Frage, wie eine Integration von vorhandenen und zukünftig zu entwickelnden Systemfähigkeiten – über die prinzipielle Funktionsfähigkeit hinaus – *sinnvoll* geschehen kann, stellt sich selbstverständlich auch die Frage nach der Methode des Testens eines solchen Entwurfs. Wenigstens dies ist relativ einfach zu beantworten: sollte ein solcher Ansatz jeweils die Leistungen der Grundarchitekturschemata subsummieren können, mag er

als mindestens gleichwertig bezeichnet werden. In 4.4.4 werden nach Vorstellung des Ansatzes entsprechende Untersuchungen vorgenommen.

4.4.2 Ansatz

Nicht nur in der Wahl der Robotersystemkomponenten, sondern vor allem in ihrer Anordnung und Strukturierung zu einem Gesamtsystem besteht erheblicher Freiraum (s. Kap. 2). Die Gestalt, in der Robotersysteme auftreten können, ist vielfältig. Nicht in allen Fällen lassen sich durch ihre verschiedene Funktionalität getrennte Komponenten eines Systems erkennen (vgl. die in 2.1.3.2 eingeführten Potentialfeldverfahren). Solche Systeme erscheinen praktisch monolithisch. In den meisten Fällen, so auch in allen anderen in 2.1.2–2.1.4 vorgestellten Ansätzen, existiert sehr wohl zumindest eine konzeptionelle Partitionierung in Funktionseinheiten, die typischerweise über eine Schnittstelle mit nur wenigen Schnittstellenelementen miteinander verbunden sind. Diese Gliederung wird insbesondere durch die Verwendung der von den jeweiligen Entwicklern übernommenen – mitunter aus Gründen der vereinheitlichten Präsentation leicht angepaßten – Darstellung als Blockdiagramm betont. In den meisten Fällen ist unklar, ob bzw. wie diese *konzeptionelle* Partitionierung innerhalb der *technischen* Systemrealisierung ein Gegenstück findet. Sofern überhaupt explizit von einer Kommunikation zwischen Systemkomponenten die Rede ist, werden Standardtechniken (definierte Aufruf-Schnittstellen der verwendeten Programmiersprache, synchronisierte Sensorvariablen, Datenaustausch über Netzwerk-*sockets* etc.) genannt. Auf diesem stark vergröberten Abstraktionsniveau lassen sich drei Klassen von Systementwürfen unterscheiden:

1. die monolithischen Systeme, die unter Verzicht auf explizite blockexterne Kommunikation entworfen werden,

2. der Objektansatz mit definierten Aufrufschnittstellen und temporärer Übergabe der Ausführungskontrolle, und

3. der Kommunikationsansatz (“*message passing*”), der nebenläufig auszuführende Systembestandteile vorsieht und einen Weg der Kommunikation (Sensorvariablen, *sockets*) vorsieht.

Da zu allen Ansätze funktionierende Prototypen aufzufinden sind, steht die prinzipielle Eignung jeweils außer Frage.

4.4.2.1 Anforderungen an eine Architektur

Die einzig notwendige Anforderung an Architekturen scheint zu sein, daß die Kommunikation zwischen Systemteilen ermöglicht wird, sofern überhaupt vom Entwickler eine solche Partitionierung in Teile vorgenommen wird. Neben dieser notwendigen Eigenschaft der Unterstützung von interner Kommunikation sind bereits aus softwaretechnischen Gründen weitere Eigenschaften *erwünscht*, die aus *nicht* mit der Funktionalität des Systems in Beziehung stehenden Metaanforderungen wie

- der Abstraktion von konkreten Gegebenheiten wie denen der Hardwareplattform,
- der Fokussierung jeweils auf Teile einer Gesamtproblemlösung,
- der Testbarkeit der Teile einzeln im Testbett wie gemeinsam im System,
- der Austauschbarkeit von Komponenten z.B. zum Zweck der Bewertung,
- der Wiederverwendung von Komponenten,
- der Möglichkeit zur problemangemessenen Systemgliederung,
- der Beachtung von üblichen Entwurfsrichtlinien, z.B. des Geheimnisprinzips

erwachsen.

Um diese Anforderungen erfüllen zu können und eine angemessene Entkopplung der Subsysteme zu erreichen, werden im folgenden Abschnitt 4.4.2.2 die Kommunikationskanäle zwischen den Subsystemen behandelt. Da neben einzelnen Kanälen natürlich auch eine Gesamtstruktur existieren soll, wird in 4.4.2.3 die Metaarchitektur als Verwaltungseinheit diskutiert, durch die erst aus kommunikationsbereiten Subsystemen eine spezifische Konfiguration entsteht, die letztlich die Roboterarchitektur ausmacht. Um einen höheren Grad an Abstraktion und Allgemeinheit bei der Systemkonstruktion zu erlangen, ist eine Koppelbarkeit der Subsysteme unter Wegfall der Notwendigkeit expliziter manueller Vernetzung erwünscht. Um dies zu erreichen und eine dynamische Systemrekonfiguration zur Laufzeit zu ermöglichen, werden weitere Anforderungen an den Rahmen der Subsysteme und die Interaktion der Subsysteme mit der Metaarchitektur gestellt,

die in 4.4.2.4 behandelt werden. Die resultierenden Eigenschaften des Gesamtsystems werden in 4.4.2.5 behandelt. Mit den technischen Details der Umsetzung befaßt sich Abschnitt 4.4.3.

4.4.2.2 Eigenschaften der Kommunikationskanäle

Kanäle verbinden Subsysteme durch Vermittlung von Kommunikation und stellen so eine Trennstelle zwischen den Subsystemen dar. Der Begriff des Kommunikationskanals ist weit gefaßt und soll hier unter den besonderen Randbedingungen der Kopplung von Teilsystemen einer Roboterplattform inhaltlich verfeinert werden.

Bei der Kommunikation von Subsystemen, die zur unabhängigen, asynchronen Ausführung ohne besondere Berücksichtigung der Erfordernisse anderer Subsysteme formuliert wurden (wie das für ein weitgehend allgemeines Design sinnvoll ist), muß die äußere Form der zu kommunizierenden Daten festgelegt sein: die Kommunikation erfolgt durch Übermittlung typisierter Daten. Nicht unbedingt festgelegt sind hingegen die Zahl der sendenden bzw. empfangenden Teilnehmer eines solchen Datenaustausches, die Rate, mit der Datenpakete generiert bzw. empfangen und verarbeitet werden können. Als allgemeine Voraussetzung ist zudem eine geeignete Adressierung der Kommunikationskanäle erforderlich, um an einem Austausch von Daten in der Rolle als Sender oder Empfänger überhaupt teilnehmen zu können. Die hier beschriebenen Eigenschaften befassen sich allein mit den Eigenschaften der Kanäle und der Art, wie durch sie eine manuelle Konfiguration von Subsystemen zu einer Systemarchitektur vorgenommen werden kann. Auf den Beitrag des Metaarchitektursystems wird im folgenden Abschnitt eingegangen.

Art und Typisierung der Kommunikation. Als Ansätze der Kommunikation zwischen einer variablen Zahl von einander nicht bekannten Kommunikationspartnern sind das *blackboard*-Kommunikationsmodell und die Kommunikation per *broadcast* denkbar. Die erste Methode bedarf einer zentralen Instanz, die als Fehlerquelle (*single point of failure*) bedeutsam ist, sondern auch ein ausgesprochen hohes Kommunikationsvolumen verwalten müßte. Bei *broadcast*-Kommunikation hingegen wäre an jedes Subsystem die Aufgabe delegiert, den relevanten Teil der Kommunikation aus einem gemeinsamen Datenstrom herauszufiltern; dies würde insgesamt zu einer erheblich erhöhten Systemlast führen. Daher soll hier jede Kommunikation innerhalb einer Architektur “themengebunden” durchgeführt werden und

allein Subsysteminstanzen tangieren, die zu dem jeweiligen Inhalt einen Bezug als Generator oder Konsument besitzen.

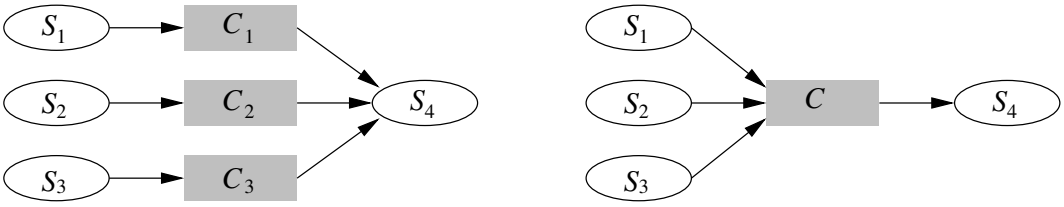
Zur Unterscheidung der themenbezogenen Kanäle werden Signaturen vergeben. Alle Kommunikation zu einem Thema *S* ist wenigstens prinzipiell öffentlich; ein entsprechender Kanal ist allen Subsystemen zugänglich, die von der verwendeten Signatur Kenntnis besitzen. Eine "private" ungesicherte Kommunikation zwischen zwei Subsystemen ist genau dann möglich, wenn ein teilnehmender Partner einen Kommunikationskanal dynamisch mit einer eindeutigen Signatur generiert und diese nur dem Kommunikationspartner mitteilt. Die Anzahl der Signaturen ist konzeptionell nicht limitiert.

Die Teilnehmer einer Kommunikation werden in Bezug auf einen einzelnen Kanal in Produzenten (*server*) und Abnehmer (*client*s) unterschieden: die Kommunikationskanäle sind asymmetrisch und gerichtet. Zwei Teilnehmer (Subsysteme) können in Dialog miteinander treten, sofern zwei signaturverschiedene Kanäle bestehen, in denen die betreffenden Subsysteme jeweils wechselseitig als Server und Client auftreten.

Anzahl der Teilnehmer. Eine unbestimmte Zahl verschiedener Subsysteme kann einen Kommunikationskanal versorgen, d.h. als Server für Daten eines bestimmten Typs auftreten. Es ist dabei möglich, daß ein Kommunikationskanal keine Zulieferer hat, sofern die entsprechenden Subsysteme nicht geladen oder inaktiv sind. Die Zahl der Abnehmer eines Kanals ist ebenfalls unbestimmt – es kann keinen, einen oder mehrere Abnehmer für die Information geben. Es ist vorgesehen, daß ein Kanalserver ungeachtet möglicher anderer Server des selben Kanals Daten ohne Verzug in den Kanal übermitteln kann. Dies setzt voraus, daß die Implementierung des Kanals eine Zwischenspeicherung der entgegenzunehmenden Daten durchführt, damit kein Blockieren der Server vorgenommen wird. Obwohl die Generierung von Information, für die es derzeit keinen Abnehmer gibt, überflüssig und unzweckmäßig erscheint, wird dennoch aus verschiedenen Gründen der Sonderfall einer solchen Kanalfunktion zugelassen. Zum einen soll bereits die Frage der bloßen Existenz eines oder mehrerer Klienten des Kanals für den Server wegen des Geheimnisprinzips – Subsysteme sollen Strukturwissen weder benötigen noch erhalten – keine Rolle spielen, zum anderen soll der evtl. nur temporäre und kurzfristige Ausfall eines Clients nicht notwendig dazu führen, daß andere Systembestandteile ihren Arbeitsmodus wechseln.

Rate der Kommunikation. Die Rate, mit der eventuell vorhandene aktive Zulieferprozesse neue Daten im Kanal zur Verfügung stellen werden, ist durch die Eigenschaften des als Server auftretenden Subsystems oder mehrerer solcher Subsysteme bestimmt und grundsätzlich damit unbekannt. Alle Abnehmer eines Kanals benötigen daher eine Möglichkeit, feststellen zu können, ob überhaupt neue Informationen vorhanden sind. Sowohl die Form des *busy waiting*, die weitere Aktivität gestattet, wie auch die des Anhaltens des wartenden Subsystems, bis ein Kanalereignis eingetreten ist, werden unterstützt. Da die Ereignisrate im Kanal die Verarbeitungskapazitäten eines abnehmenden Subsystems erheblich überschreiten kann (so ist der Abruf von Kamerabildern, obgleich wegen des hohen Datenaufkommens vergleichsweise langsam, doch schneller durchführbar als viele Bildverarbeitungsfunktionen) darf explizit keine *event*-Benachrichtigung z.B. als *callback* in das Subsystem erfolgen. Auch der Betrieb einer eingangsseitigen Nachrichtenwarteschlange ist unangemessen, da typischerweise nur die aktuellen Daten der jeweils letzten Übermittlung benötigt werden. In der Beziehung der Kopplung erinnern die hier vorgeschlagenen Kanäle an die *latches*, die AMIR et al. in [AM99] einführen; ein wesentlicher Unterschied ist jedoch, daß die *latches* als Endpunkte der Kommunikation zwischen genau zwei Schichten (einer vierschichtigen Subsumptionsarchitektur nach BROOKS) dienen und lediglich den Aspekt der unterschiedlichen Verarbeitungsgeschwindigkeiten adressieren sollen. Im Fall des hier vorgestellten Systems haben die Kanäle weitere Funktion bei der Strukturbildung und sind nicht nur zwischen genau zwei Kommunikationspartnern einzusetzen.

Strategische Kommunikationskonfiguration. Die zu einem Problem gewählte strategische Konfiguration der Kommunikation obliegt dem Systementwickler. Grundsätzlich ist es möglich, jede gewünschte Kommunikationsstruktur allein unter Nutzung unmittelbarer Verbindungen herzustellen, die zwischen jeweils genau zwei Partnern hergestellt werden. Dieser Zustand tritt beispielsweise ein, wenn jede Signatur jeweils textuell durch Konkatenation der Bezeichnungen des Sender- und des Empfängersubsystems hergestellt wird (sofern kein Subsystemname einen Präfix eines anderen solchen Namens bildet). Unter Umständen ist die Trennung selbst von inhalts*gleichen* Kanälen durch Verwendung explizit unterschiedlicher Signaturen dann tatsächlich sinnvoll, wenn das empfangende Subsystem (S_4 in Abb. 4.27) darauf angewiesen ist, zwischen verschiedenen Produzenten unterscheiden zu können. Diese Anforderung tritt beispielsweise auf, wenn ein *arbiter*-Subsystem genau eine Antwort eines der Quellsysteme selektieren soll, oder ein Summations-Subsystem einen verträglichen Wert aus



(a) Konfiguration für Summation oder einen *arbiter*: unterschiedliche Kanalsignaturen sind erforderlich.

(b) Konfiguration mit mehreren Quellen: S_1, S_2 und S_3 verwenden die gleiche Kanalsignatur. S_4 referenziert die Quellen nicht.

Abb. 4.27: Strategische Kommunikationskonfiguration: müssen Datenquellen unterschieden werden, ist die Verwendung signaturverschiedener Kanäle C_i zweckmäßig (links). Produzieren Quellen grundsätzlich gleichartige Information, z.B. ein Stopsignal bei Kollisionsgefahr, ist nur ein Kanal C ausreichend (rechts). Vorzug der letztgenannten Konfiguration ist die leichte Erweiterbarkeit, ohne den Code von S_4 anpassen zu müssen.

allen verfügbaren Quelldaten berechnen soll. Eine Unterscheidung des Datenursprungs in S_4 ist erforderlich, da $S_1 \ldots S_3$ Ausgabedaten mit erheblich verschiedenen Raten produzieren könnten.

Die Verwendung unterschiedlicher Kanalsignaturen gestattet, aus Sicht des Empfängersubsystems S_4 auch das Versagen eines Produzentensubsystems zu diagnostizieren (sofern z.B. ein *timeout*-Wert überschritten wird). Problematisch ist allein, daß es nicht möglich ist, einen zusätzlichen Produzenten aufzunehmen, ohne den Code des Empfängersubsystems zu verändern: alle liefernden Subsysteme werden über die Signaturen der von ihnen versorgten Kanäle referenziert. Die Lösung, mit der dennoch eine dynamische Erweiterbarkeit geschaffen werden kann, sieht einen Übergang zu einer einkanaligen Kommunikation mit entsprechender Erweiterung des Protokolls vor. Zusätzlich zu den übertragenen Nutzdaten muß der Produzent durch einen eindeutigen Identifikator ausgewiesen werden. Ein Summationssubsystem müßte eine Liste der Daten liefernden Erzeuger führen, um dabei neu erscheinende wie inaktiv gewordene Quellen erkennen. Erheblich einfacher stellt sich die Situation dar, sofern nicht zwischen den jeweiligen Produzenten der Daten unterschieden werden muß. In der Praxis der Architekturkonstruktion ist dies der normale Fall, wenn signaturidentische Daten auch tatsächlich funktionsäquivalent sind. In diesen Fällen sollte – wie in Abb. 4.27(b) für den Fall von mehreren Produzenten äquivalenter Daten dargestellt – ein einziger Kanal verwendet werden. Dies geschieht genau

dann, wenn alle produzierenden Subsysteme die gleiche Signatur für die Ausgabe wählen. Der Vorzug der Verwendung eines gemeinsamen Kanals ist, daß eine Erweiterung der Liste Daten beitragender Subsysteme möglich ist, ohne daß ein Konsument in irgendeiner Weise modifiziert werden muß. Auch Ausfälle einzelner oder mehrerer Produzenten führen lediglich zu einer schrittweisen Verschlechterung der Leistung des Gesamtsystems durch die in größeren Zeitabständen aktualisierten Daten, ohne daß allerdings die grundsätzliche Funktionsfähigkeit gefährdet wird.

4.4.2.3 Eigenschaften der Metaarchitektur

Eine Metaarchitektur verwaltet die in 4.4.2.2 vorgestellten Kanäle und ihre Signaturen. Der Präfix "Meta-" wird hier verwendet, da dieser feste Systembestandteil nicht mit einer bestimmten Architektur korrespondiert, sondern die Konstruktion verschiedener Architekturen gestattet.

Es sei vereinbart, daß zum Startzeitpunkt des Systems weder Kanäle existieren noch Subsysteme instantiiert sind.

Dynamik von Server- und Clientsystemen. Subsysteme nehmen unter Angabe ihrer jeweiligen kanalabhängigen Rolle als Server oder Client Bezug auf eine bestimmte Kanalsignatur. Wenn ein entsprechender Kanal noch nicht existiert, wird er zu diesem Zeitpunkt angelegt; er enthält in diesem Zustand allerdings noch keinen Wert, der ausgelesen werden könnte. Sofern ein durch Vergleich der Signaturen bestimmbarer geeigneter Kanal vorhanden ist, wird ein Server als zusätzliche Quelle entsprechender Daten mit dem Kanal verknüpft, bzw. ein Client auf den entsprechenden Kanal als Datenquelle verwiesen. Die Liste der Server bzw. Clients eines Kanals ist zeitlich variabel; bei Instantiierung eines weiteren Subsystems, das einen Kanal mit der passenden Signatur in welcher Zugangsart auch immer referenziert, wird die entsprechende Verbindung durch die Metaarchitektur selbständig hergestellt.

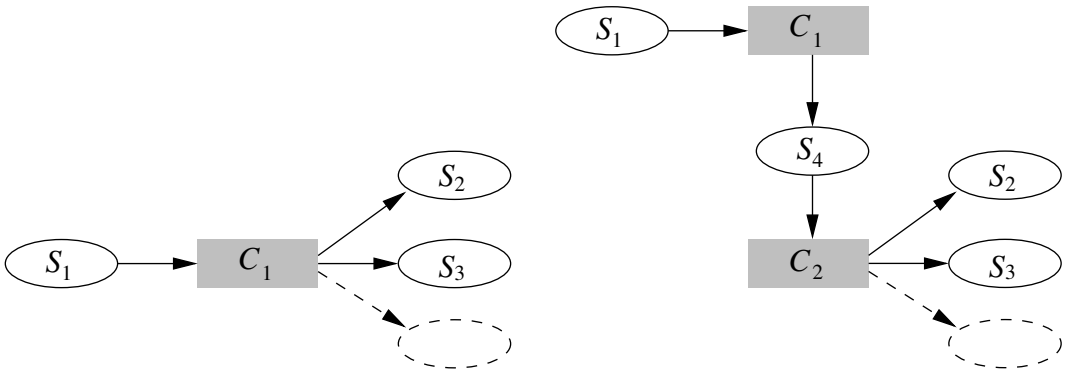
Subsysteme, die als Server auftreten und Daten produzieren, werden nicht angehalten, da der Kanal alle Ausgabe entgegennimmt, selbst wenn zu diesem Zeitpunkt noch kein Client präsent ist. Dies gilt auch im Fall mehrerer Server, die in den gleichen Kanal schreiben, wobei der Schreibzugriff allerdings serialisiert werden muß. Der Inhalt des Kanals spiegelt nach Schreibzugriffen die jeweils zeitlich letzte Ausgabe wider, die ein Server vorgenommen hat. Clients, die Daten von einem bestimmten Kanal lesen, werden

nur in dem Fall durch einen Synchronisationsmechanismus kurzfristig angehalten, wenn gerade ein als Server auftretendes Subsystem Daten in den selben Kanal schreibt und dabei der Inhalt des Kanal-lokalen Pufferspeichers aktualisiert wird. Wenn keine von diesem Client noch nicht gelesenen Daten abrufbar sind, kann dies vom Client anhand der separat abrufbaren Sequenznummer festgestellt werden; ein Lesevorgang für den Kanalinhalt liefert noch immer den letzten geschriebenen Wert.

Nichtmonotone Rekonfiguration. Die beschriebenen Kanaleigenschaften gestatten die feste strukturelle Vernetzung von Subsystemen und schaffen die Möglichkeit, eine solche Verbindungsstruktur auch bei einer unbekannten und auch während der Ausführungszeit variablen Zahl von Server- und Clientsubsystemen aufrechtzuerhalten. Auf diese Weise kann eine wesentliche Eigenschaft der Subsysteme bewahrt werden: die der Abstraktion von jeglichem Wissen über die Struktur, in der die Subsysteme eingesetzt werden. Diese Abstraktion ist vollständig bis auf die Benennung der benötigten und der belieferten Kanäle und das notwendige Wissen um Format und Inhalt der in den Kanälen übermittelten Information.

Häufig ist es sinnvoll, statt eines vollständigen Neudesigns einen nachträglichen Eingriff in eine funktionierende Systemstruktur vorzunehmen, d.h. eine bestimmte Form der Rekonfiguration durchzuführen. In diesen besonderen Fällen können Subsysteme partielles Wissen über einen bestimmten Aspekt der Systemstruktur nutzen: Wissen über existierende Kanäle, in denen eine suboptimale Informationsübermittlung stattfindet. Die Gründe für eine Bewertung der Information eines Kanals als suboptimal kann verschiedene Gründe haben; eine Möglichkeit könnte die Existenz eines verfeinerten, in irgendeiner Beziehung leistungsfähigeres Verfahrens sein. In solchen Fällen ist es nicht sinnvoll, Information *additiv* zu der bestehenden Kanalinformation hinzuzufügen, da sie der neu hinzugewonnenen u.U. sogar widerspricht.

Ein Subsystem kann daher nicht nur als Server eines *S*-Kanals auftreten, sondern auch versuchen, diesen zu *überladen*. Voraussetzung dafür ist, daß ein *S*-Kanal bereits existiert. Üblicherweise wird das Subsystem als Client des ursprünglich vorhandenen *S*-Kanals auftreten, um neue signaturgleiche, aber möglicherweise unter Berücksichtigung weiterer Information in einer bestimmten Weise optimierte *S*-Daten zur Verfügung zu stellen. Diese Betriebsart könnte beispielsweise verwendet werden, um das bereits vorgestellte Verfahren der Selbstlokalisierung (4.3.3.4) zur Verbesserung der Systemodometriedaten einzusetzen. Der Nutzen dieses Überschreibens liegt



(a) Ausgangssituation: S_1 produziert Daten mit der Signatur C. Alle hinzukommenden Abnehmer für C-Daten werden mit Kanal C_1 verbunden.

(b) S_4 überlädt C-Daten und wird exklusiv zum Abnehmer des bisher vorhandenen C-Kanals mit dem höchsten Index. Alle weiteren C_1-Empfänger werden auf C_2 umgelegt. Bei Destruktion von S_4 würden alle C_2-Empfänger automatisch wieder zu C_1-Empfängern.

Abb. 4.28: Überladen von Kanälen: die indirekte Kopplung der Kommunikationspartner ermöglicht, auch nachträglich zu Analyse- und Optimierungszwecken in feststehende Architekturstrukturen einzugreifen. Die strukturelle Änderung geschieht aus Sicht der Subsysteme transparent, da die Vernetzung nicht in den Subsystemen, sondern innerhalb der Metaarchitektur vorgenommen wird.

allgemein in der Option zum Ersatz fehlerhafter oder unvollständiger durch kompensierte bzw. ergänzte Daten, ohne daß irgendwelche weiteren Änderungen an der Konfiguration der Subsysteme (d.h. einer Architektur) vorgenommen werden müssen. Beispielsweise ließe sich so bedarfsweise ein Kanal "`Odo`", der die fehlerbehaftete Information der Systemodometrie übermittelt, durch einen Kanal mit der gleichen Signatur ersetzen, der (wie in 4.3.3.4 beschrieben) weitere sensorische Information nutzt (Abb. 4.29). Mit einem solchen Mechanismus kann also erreicht werden, daß auch Architekturen ohne jede Veränderung ihrer Subsystemkomponenten von einer Systemerweiterung profitieren können, die zum Zeitpunkt ihrer Konzeption weder vorgesehen noch absehbar war.

Da es nicht möglich ist, sämtliche zu ersetzende Information aus einem Kanal K_i zu entfernen, bevor sie von den anderen Subsystemen empfangen wird, und unklar wäre, was zu geschehen hätte, wenn gleich mehrere Subsysteme ein solches Überschreiben vornehmen wollten, wird in solchen Fällen stets ein weiterer neuer Kanal K_j mit gleicher Signatur erzeugt. Da die Kanäle geordnet entsprechend der Reihenfolge ihrer Erzeugung verwaltet werden ($i < j$), führt dies zunächst dazu, daß neu instantiierte Konsumenten mit dem "neuen" Kanal K_j verbunden werden und Benachrichtigung über eintreffende Daten allein aus ihm erhalten. In den Fällen bereits zu einem früheren Zeitpunkt entstandener Verbindung zu K_i ist die Situation komplizierter.

Kanalselektion und Überladen von Kanälen. Grundsätzlich wird der überladene *S*-Kanal von genau dem Subsystem mit Daten versorgt, das ein Überladen angemeldet hat. Sofern die Kanäle in Reihenfolge ihrer Erzeugung mit Indizes versehen werden, ist also stets der *S*-Kanal mit der höchsten Indexziffer der beste Datenlieferant für nach diesem Ereignis neu zu instantiierende Subsysteme. Bliebe es allerdings dabei, daß eine Kanalzuordnung nur bei der Instantiierung von Subsystemen vorgenommen würde, bestünde keine Möglichkeit, vorhandene Architekturen ohne Änderung der Ladereihenfolge – d.h. Änderung der Programmquellen – von neuen Verfahren wie der o.a. Odometriekorrektur profitieren zu lassen. Aus diesem Grund muß der Mechanismus erweitert werden. Das nachträgliche "Überladen" eines *S*-Kanals K_i durch einen signaturgleichen Kanal K_j ($i < j$) soll zur Folge haben, daß nicht nur neue, sondern auch alle bereits registrierten K_i-Clients zu K_j-Clients werden. Dieser Effekt soll als *upgrading* bezeichnet werden, da der überladene Kanal K_j in irgendeiner Weise höherwertigere Information – wie z.B. eine gegenüber der gewöhnlichen Odometrie unter

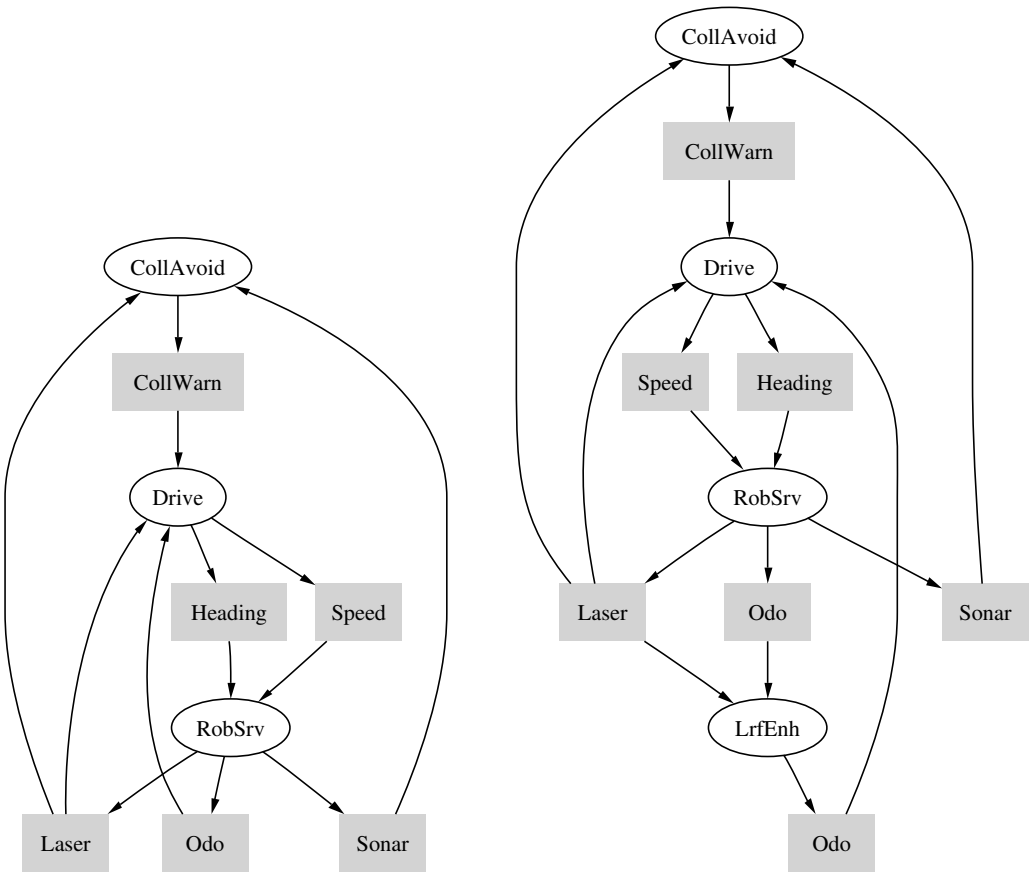
Berücksichtigung anderer Sensorinformation verbesserte Positionshypothese – führen soll.

In der Regel wird das überladende Subsystem ein K_i-Client sein. Dennoch darf es (allein) nicht Gegenstand des *upgrading* sein: es ist das einzige, das weiterhin die K_i-Daten des Typs S erhält. Um korrekte Funktion zu gewährleisten, ist natürlich erforderlich, daß auch der überladene Kanal K_j mit S-Daten gefüllt wird. Das nachträglich in die Kette der Verarbeitung eingefügte S-Daten Subsystem wäre einmal abgesehen von der internen Verzögerung funktional neutral, wenn es alle Daten aus K_i unverändert in K_j veröffentlichen würde.

Durch das Überladen von Kanälen ist es also möglich, ein neues Subsystem nachträglich und selbst zur Laufzeit des Systems transparent in eine bestehende kanalvermittelte Verarbeitungskette von Subsystemen einzugliedern, ohne den Code der bestehenden Subsysteme oder auch nur ihre Ladereihenfolge verändern zu müssen. Einzig erforderlich ist ein referentieller Zugang zur Metaarchitektur, bei der die Anmeldung des neuen Subsystems geschehen muß.

Wegfallen überladener Kanäle. Terminiert ein Subsystem, so treten Komplikationen auf, wenn dieses Subsystem ein Überladen eines Kanals vorgenommen hat. In diesem Fall müssen alle als Clients des Ausgabekanals K_j registrierten Subsysteme erneut auf den ursprünglich überladenen signaturgleichen Kanal K_i verwiesen werden, um die fortgesetzte Funktionalität der Gesamtarchitektur sicherzustellen. Dies gilt nicht nur für Subsysteme, die Gegenstand eines *upgrading*-Vorgangs wurden, sondern auch für solche, die überhaupt erst nach Instantiierung von K_j einem S-Kanal zugeordnet wurden. In der technischen Umsetzung entspricht das hier spezifizierte Verhalten einem *fallback* auf K_i-Daten – im Beispiel der durch LRF-Messungen und Selbstlokalisationssubsysteme optimierten Odometrie zur konventionellen rein odometrische Bestimmung des Plattformstandorts. Sobald nach einem Wegfall eines überladenen Kanals der ursprüngliche Kanal K_i erneut überladen wird, z.B. weil das Subsystem nach einer Fehlersituation neu anläuft, findet erneut der oben beschriebene Vorgang des *upgrading* statt.

Mehrstufiges Überladen. Technisch ist auch ein mehrstufiges *upgrading* ohne Probleme möglich, wobei allerdings jeweils nur der Kanal mit der höchsten Indexziffer überladen werden kann, nicht jedoch einer in einer Folge von überladenen Kanälen (an dieser Stelle endet die Flexibilität des



(a) Ausgangssituation

(b) Subsystem `LrfEnh` überlädt Kanal `Odo`.

Abb. 4.29: "Upgrading" von Kanalinformation durch Überladen von Kanälen. Die nachträgliche Instantiierung von `LrfEnh`, das den Kanal `Odo` überlädt, führt bei laufendem System transparent zu einer Leistungssteigerung.

Überladens ohne Eingriffe in die vorhandenen Subsysteme; durch Änderung von Kanalsignaturen kann natürlich dennoch ein Zugriff erreicht werden, da so die Namensverdeckung beseitigt werden kann). Bei mehrstufigem Überladen ist zudem die Ladereihenfolge der Subsysteme, die ein Überladen vornehmen, von gewisser Bedeutung: ein sich aus der Architektur ab- und wieder anmeldendes Subsystem erhielte automatisch für den neuen Kanal die höchste Indexziffer und würde so in der S-Verarbeitungskette an die letzte Position wechseln. In der Regel stellt dies kein Problem dar, sofern die S-Daten tatsächlich die gleiche semantische Bedeutung tragen. Sofern es explizite Gründe für eine bestimmte Anordnung von Subsystemen innerhalb einer Architektur geben sollte, ist diese Randbedingung nicht mehr erfüllt, und die Verwendung verschiedener Signaturen für die Daten ungleicher Funktion angeraten.

Anwendungs- und Ausschlußfälle für "upgrading". Ein Anwendungsfall für mehrstufiges *upgrading* könnte beispielsweise sein, aus einer Menge von Meßpunkten mehrstufig die Teilmengen von Punkten zu entfernen, die Merkmalen wie den in 4.3.3.3 beschriebenen entsprechen: sobald ein Meßpunkt als einem Merkmal angehörend zugeordnet wurde, soll er möglicherweise nicht bei der Berechnung von Vertretern einer anderen Merkmalsklasse herangezogen werden.

Die Eigenschaft der gleichen Funktion der unter einer Signatur übermittelten Daten – z.B. Mengen von Meßpunkten, aus denen nach einer Gruppierung Merkmalen zuzuschlagende Punkte entfernt werden sollen – ist allerdings nicht grundsätzlich gegeben. Wäre durch ein Subsystem beispielsweise eine Verarbeitung $f : L \rightarrow L$ definiert, das eine Bereinigung von LRF-Scans L um ungültige Messungen[10] vornimmt, und durch ein anderes mit $g : (L, \mathbf{v}) \rightarrow L$ die Verschiebung $\mathbf{v}$ der kartesischen Koordinaten so, daß sie relativ zur Roboterbasis und nicht zum Sensor angegeben werden, gilt $f(g(L, \mathbf{v})) \neq g(f(L), \mathbf{v})$, da der *magic*-Wert nach Applikation von g nicht mehr als solcher erkannt werden kann. Obwohl auf den ersten Blick die Annahme der Übermittlung qualitativ vergleichbarer Information stichhaltig schien, ist hier also die Verwendung verschiedener Signaturen angebracht, um die unter den Randbedingungen einzig sinnvolle Verarbeitungssequenz zu erzwingen.

[10] Koordinaten mit bestimmten *magic*-Werten werden geliefert, sobald z.B. die maximale Meßdistanz überschritten wird.

4.4.2.4 Eigenschaften der Subsysteme

Subsysteme fassen *clients*, Verarbeitung und *server* in einer Struktur zusammen. Der Verarbeitungsteil eines Subsystems kann dabei durch Nutzung der Hardware-*proxy*-Instanzen der Klassenbibliothek (4.2.2.2) Verbindung mit einer Hardwareplattform aufnehmen.[11] Die Subsysteme sind die Einheiten der Verarbeitung, die sich der Metaarchitektur und indirekt der durch sie vermittelten Kanäle bedienen. Zielsetzung war, ein minimales Interface zur Metaarchitektur zu definieren, um dem Entwickler von Subsystemen möglichst wenige Randbedingungen aufzuerlegen.

Normale Funktionsweise. Zur Integration eines Subsystems ist eine Anmeldung bei der Metaarchitektur erforderlich, während der vom Subsystem der Bedarf in der Rolle als Client und die Leistung in der Rolle als Server mitgeteilt werden. Diese Bindung an Kanäle bestimmter Signaturen – nicht an bestimmte Kanäle; diese Zuordnung nimmt die Metaarchitektur ohne Wissen des Subsystems vor – bleibt üblicherweise während der Lebenszeit einer das Subsystem repräsentierenden Objektinstanz unverändert und wird erst im Destruktor implizit aufgehoben. Unter normalen Umständen geschieht nach Anmeldung bei der Metaarchitektur der Eintritt in eine Funktion, deren Inhalt allein dem Entwickler des Subsystems obliegt; insbesondere müssen weder die Ein- und Ausgänge zu bestimmten Zeiten (oder überhaupt) bearbeitet noch die Kontrolle jemals abgegeben werden. Letztlich ist diese Funktion als asynchroner *thread* implementiert. In der Regel ist innerhalb der Funktion eine Endlosschleife anzutreffen, die auf Daten aus einem oder mehreren Kanälen wartet (`read`), auf dieser Grundlage und ggf. eines internen Zustands Berechnungen vornimmt, um die Resultate einem oder mehreren Ausgabekanälen (`write`) zu übermitteln.

Über nur eine geringere Verarbeitungsrate als die faktische Kanalrate verfügende Subsysteme werden nicht alle übermittelten Informationspakete erhalten, da der Kanal zu jeder Zeit nur genau ein solches Paket speichert: jedes vom gleichen oder einem anderen Kanalserver erhaltene weitere Paket überschreibt das gespeicherte. Aus diesem Grund sollten Pakete keine *relativen* oder inkrementellen Angaben enthalten, die bei einem Verlust von Teilen der Übertragung nicht mehr korrekt interpretierbar wären, sondern grundsätzlich absolute Daten enthalten. Müssen aus irgendeinem Grund

[11] Da die Serverprozesse der Plattform währenddessen keine Verbindung mit einem weiteren Client zulassen werden, erfolgt der Zugriff auf die Roboterhardware exklusiv. Dies begründet die Notwendigkeit eines speziellen "Robotserver"-Subsystems, sofern mehrere Subsysteme sich der selben Geräte bedienen.

alle einem Kanal anvertrauten Datenpakete verarbeitet werden, obgleich das aus Rechenzeiterwägungen oder anderen Gründen entweder vorübergehend oder grundsätzlich nicht möglich ist, steht es dem Entwickler des Subsystems frei, einen weiteren, nicht sonderlich rechenintensiven *thread* zu implementieren, der eine Queue der Datenpakete führt. Grundsätzlich sollte eine Kanalausgabe allerdings nur auf Basis der jeweils aktuellsten Eingabedaten erfolgen (d.h. bei leerer Queue), da der Zeitpunkt der Übergabe an den Kanal von Konsumenten auch als Zeitpunkt der Entstehung der Daten angenommen wird.

An das Subsystem werden keine besonderen Timinganforderungen gestellt, da Subsysteme in einem eigenen *thread* ablaufen. Eine synchronisierte Interaktion – verbunden mit der Gefahr, das Gesamtsystem anzuhalten – findet allein an den Stellen der Kommunikation statt. Da der kritische und durch Semaphore gesicherte Abschnitt sich nicht im Code des Subsystems, sondern der diese Eigenschaft vererbenden Oberklasse `subsys` befindet, ist davon auszugehen, daß die Kontrolle stets nach Vornahme einer einfachen Variablenzuweisung so rasch vorgenommen wird, daß der Ablauf anderer, auf den gleichen Kanal zugreifender Subsysteme praktisch nicht verzögert ist.

Terminieren von Subsystemen. Drei unterschiedliche Gründe für das Terminieren von Subsystemen können auftreten. Erstens kann eine planmäßige Destruktion der Subsysteminstanz geschehen, wie sie beim Terminieren des Gesamtsystems, dem Verlassen des Gültigkeitsbereichs, zu dem das Subsystem lokal instantiiert war oder bei einer expliziten Destruktion dynamisch auf dem *heap* des ausführenden Rahmenprogramms allozierter Subsysteme geschehen kann. In diesen Fällen hat das Subsystem die Möglichkeit, die Sicherung von Zustandsinformation auf einem externen Medium vorzunehmen, die bei einem Neustart wiederaufgenommen werden kann. Zweitens kann ein Terminieren des Subsystems bei Bedarf durch das Subsystem selbst ausgelöst werden. Diese Variante ist dann zweckmäßig, wenn beispielsweise eine Bedingung für die korrekte Funktion erkennbar dauerhaft nicht mehr gegeben ist, z.B. ein erforderlicher Sensor nicht mehr zur Verfügung steht. Diese Form der Terminierung kann nur durch Trennung aller Verbindungen zu den referenzierten Kanälen mit anschließender Inaktivierung geschehen, da die finale Destruktion der betreffenden Subsysteminstanz durch den Ort im Code und nicht durch die Gegebenheiten der Ausführung bestimmt werden kann. Drittens ist die Möglichkeit eines Versagens des Subsystems zu nennen, die sich in einer durch das Subsystem

unbehandelten Ausnahme (*exception*) oder einer anderen Art des Beenden der Ausführung niederschlagen könnte.

Jede der drei Formen der Terminierung eines Subsystems wird seine automatische Deregistrierung innerhalb der Architektur nach sich ziehen, auch wenn die durch den Quellcode des Gesamtsystems vorgesehene Lebensdauer (*extent*) der Instanz noch nicht unbedingt erreicht sein muß. Der vom Subsystem in Anspruch genommene Speicherplatz kann nicht zu einem früheren Zeitpunkt freigegeben werden. Die Deregistrierung bewirkt, daß alle nicht mehr referenzierten Kanäle entfernt werden, deren letzter *server* oder *client* das terminierte Subsystem war. Eine gesonderte Behandlung ist wiederum erforderlich, wenn das terminierte Subsystem einen überladenen Kanal mit Daten versorgt hat. Hier ist eine Fallunterscheidung nötig: Im üblichen Fall kann der überladene Kanal entfernt werden, wobei alle dort als *client* registrierten Subsysteme (transparent und für sie unbemerkt) zum Gegenstand eines *downgrade* werden und wieder eine Verbindung mit dem ursprünglich überladenen Kanal erhalten. Ist inzwischen jedoch ein weiteres Subsystem zu einem zusätzlichen *server* des überladenen Kanals geworden, so muß der vom terminierten Subsystem angelegte Kanal als solcher weiter bestehen.

Trotz dieser unter Umständen komplexen Vorgänge beim Terminieren eines Subsystems ist es zur Auslösung lediglich erforderlich, daß eine Deregistrierung bei dem verwaltenden Metaarchitekturobjekt geschieht.

4.4.2.5 Besondere Eigenschaften des Gesamtsystems

Auf den ersten Blick stellt das in 4.4.2.2 vorgestellte System ein weiteres Rahmenwerk dar. Es erfüllt selbst die in 4.4.2.1 genannten Anforderungen (Testbarkeit, Austauschbarkeit, Wiederverwendung, Geheimnisprinzip) bzw. unterstützt die Entwicklung von angemessenen Subsystemen (Problempartitionierung und Systemgliederung), wobei die letzte Verantwortung für die Einhaltung dieser Randbedingungen offensichtlich beim Entwickler liegt. Die Abstraktion von z.B. der gegebenen Hardware wird dabei auf unteren Schichten vorgenommen.

Über diese erwünschten Eigenschaften hinaus weist das System einige wesentliche zusätzliche, bislang noch nicht diskutierte Eigenarten auf:

1. Die Auflagen an Subsysteme sind gering.

   Da Subsysteme in eigenen *threads* ausgeführt werden, sind wenigstens durch das Rahmenwerk dem Subsystem keine zusätzlichen Bedingungen (Antwortzeit, Synchronisationsbedingungen) auferlegt.

2. Die entkoppelte Kommunikation zwischen Subsystemen wird unterstützt.

   Diese Eigenschaft ist essentiell, da verschiedene Einheiten der Verarbeitung je nach Rechenaufwand und Bezug zu Hardware mit ganz unterschiedlichen Raten ablaufen und ein Blockieren der Empfänger bestimmter Daten nicht stattfinden darf.

3. Die Transparenz und Abstraktion steigt mit Einführung des Kanalkonzepts: Es kann mehrere Quellen für einen Kanalinhalt geben.

   Kanäle sind nicht ausschließlich als Punkt-zu-Punkt-Verbindung zwischen zwei Subsystemen zu betrachten, sondern als "themenbezogener *broadcast*". Grundsätzlich können auf diese Weise *mehrere* Subsysteme gleichwertige Beiträge zu einer solchen Kommunikation leisten, ohne daß ein Empfängersubsystem Kenntnis über den oder die Produzenten der jeweiligen Daten besitzen muß.

4. Die allgemeine Robustheit steigt erheblich.

   Die Möglichkeit eines temporären oder vollständigen Versagens eines Subsystems ist nicht nur durch Programmierfehler, sondern auch durch fehlerhafte oder vorübergehend nicht betriebsbereite Hardware grundsätzlich gegeben. Ein daraus resultierendes lokales Stocken der Verarbeitung wird wegen der Entkopplung keine schädlichen Auswirkungen auf verbundene Subsysteme haben, da

   (a) eingangsseitig Daten vom Metaarchitektursystem gepuffert werden, und somit keine Aktivität des Subsystems erforderlich wird, um die Funktion des Kanals aufrechtzuerhalten,

   (b) der Ausfall eines Subsystems u.U. weitere Quellen des Ausgabekanals für äquivalente Daten beläßt, die für eine Funktion des Gesamtsystems ausreichend sind.

5. Eine Laufzeitdynamik ist vorgesehen.

   Genau wie ein Roboter oftmals durch Zuschalten von Aggregaten komponentenweise in Betrieb genommen werden kann, so kann auch zur Laufzeit ein Subsystem "nachgeladen" werden, ohne daß eine explizite Verknüpfung mit seinen Kommunikationspartnern geschaltet werden muß: sofern alle benötigten Eingabekanäle (dies kann sogar eine Untermenge der im Subsystem vorgesehenen Eingabekanäle sein!) Daten führen, wird das hinzugeladene Subsystem arbeiten, und entweder die entsprechenden Ausgabekanäle anlegen oder, wenn sie

bereits vorhanden sind, durch seine Ausgaben ergänzen. Auf ähnliche Art und Weise ist auch das Entladen von Subsystemen zur Laufzeit leicht möglich, ohne die Gesamtsystemfunktion vollständig zu unterbrechen.

6. Eine funktionserhaltende Optimierung ist möglich: Alle Systemfähigkeiten können "überladen" und angepaßt werden.

   Die Fähigkeit zum *upgrade* von Kanälen bringt die Option auf späte funktionserhaltende Optimierung bereits bestehender Systeme, ohne daß eine explizite Rekonfiguration oder gar eine erneute Übersetzung des Codes betroffener Subsysteme erforderlich wird. Sämtliche Daten in einem Kommunikationskanal lassen sich modifizieren, sodaß nicht nur die später zugeladenen, sondern auch die bereits vorhandenen und mit dem Kanal verbundenen Subsysteme zu Empfängern ausschließlich der modifizierten Information werden. Der Prozeß ist reversibel, sodaß sich das einen Kanal überladende Subsystem auch wieder transparent entfernen läßt. Diese Option gestattet beispielsweise ohne weiteres, die Daten zusätzlicher Sensoren in bestehende, funktionierende Strukturen nutzbringend zu integrieren, um die Systemleistung zu steigern, ohne daß eine explizite strukturelle Änderung der vorhandenen Architektur erforderlich wird.

7. Das System ist leicht erweiterbar.

   Weitere Subsysteme lassen sich zur Übersetzungs- und zur Laufzeit einfach hinzufügen.

Diese zusätzlichen Eigenschaften lassen das konzipierte System für Anwendung in der Robotik besonders geeignet erscheinen.

4.4.3 Realisierung

Im Vordergrund stehen hier nicht mehr Art und Umfang der Anforderungen an die in 4.4.2 vorgestellten Systemkomponenten, sondern nur mehr die Umsetzbarkeit der gestellten Forderungen, und die daraus resultierende technische Schnittstelle, die beim Entwurf von Subsystemen als Architekturkomponenten und Gesamtarchitekturen in Erscheinung tritt. In den folgenden Unterabschnitten 4.4.3.1–4.4.3.4 soll beginnend bei den verwendeten Betriebssystemdiensten über die Schnittstelle der Subsysteme und die

Implementierung der Kanäle bis zum Zusammenwirken der Teile der Metaarchitektur eine kurze Übersicht über die technische Realisierung gegeben werden. Nach Beschreibung der technischen Grundlagen kann schließlich in 4.4.3.5 auf relativ abstrakter Ebene kurz auf das eigentliche Verfahren für die Instantiierung einer konkreten Architektur eingegangen werden. Es zeigt sich dabei, daß die eigentliche Komplexität dieser Aufgabe nahezu vollständig auf die zum Entwurf bereitgestellten Komponenten verlagert wird.

4.4.3.1 Betriebssystemschnittstelle

Die technische Umsetzung der in 4.4.2 spezifizierten Systemfähigkeiten erfordert einen Rückgriff auf bestimmte durch das Betriebssystem verfügbare Dienste. Das Erzeugen von *threads* und das zur Synchronisation erforderliche *locking* sind hier besonders hervorzuheben.

Semaphore. Die Kommunikation zwischen parallel ausgeführten Subsystemen bedarf der Synchronisation, da nicht notwendig nur atomare Werte über Kommunikationskanäle transferiert werden. Während eines noch andauernden Schreibvorgangs darf kein Lesezugriff stattfinden und umgekehrt. Zur Sicherung dieser kritischer Abschnitte während der Aktualisierung von Kanalvariablen werden Semaphore eingesetzt.

Zur Vereinfachung der Handhabung wurden auch diese Semaphore in eine Objektstruktur überführt. Auf diese Weise kann im Destruktor selbständig die Freigabe des Semaphore mit dem jeweiligen Identifikator erreicht werden. Damit ist zudem der Einsatz spezieller *lock*-Objekte möglich, deren Konstruktion erst mit der erfolgreichen Ausführung von `p()` abgeschlossen wird, und deren auch implizite Destruktion eine `v()`-Operation ausführt. Dies stellt keine Erweiterung der erzielbaren Funktionalität dar, trägt aber zu einem fehlerfreien Entwurf bei.

Threads. Zur parallelen Ausführung der Subsystemen in *threads* wird die *pthread*-Bibliothek genutzt. Da die Funktionen dieser Bibliothek nur auf der Ebene der Programmiersprache C zur Verfügung stehen, müssen weitere Maßnahmen ergriffen werden, damit auch Instanzen von objektorientierten Klassen *threads* enthalten können. Statt der obligatorischen Verwendung einer nicht-*member*-Funktion wird ein eigenes Interface verwendet, das ähnlich wie in der Sprache JAVA [Fla96, S. 161ff] die Nutzung beliebiger Klassen mit einer `run`-Methode erlaubt. Das technische Problem dabei liegt in der

Bestimmung der Adresse einer instanzspezifischen *member*-Funktion dieser Klassen. Um der Schnittstelle der *pthread*-Bibliothek eine statische Funktionsadresse liefern zu können, verfügt die abstrakte Elternklasse `thread` entsprechend über eine statische *member*-Funktion der Signatur `static void* sfp(void* dfp)`. Wegen ihrer statischen Deklaration wird sie mit feststehender Adresse zwischen allen Instanzen geteilt. Die Erzeugung des *thread* geschieht mit dem statischen Funktionszeiger `sfp`, der als Parameter jedoch den Verweis auf das Objekt erhält, dessen *run*-Methode auszuführen ist. Die bereits parallel zur Ausführung kommende Funktion `sfp` ist damit in der Lage, die rein virtuelle (doch in der abgeleiteten Klasse vorhandene) *run*-Methode über ein `((thread*)dfp)->run()` auszuführen. Mit dieser Maßnahme lassen sich Klassen realisieren, deren Instantiierung dynamisch zur Erzeugung eines weiteren parallelen *threads* führt. Die Instanzen solcher Klassen besitzen die für Subsysteme geforderte Eigenschaft, voneinander unabhängig ablauffähig zu sein.

4.4.3.2 Eigenschaften der Subsysteme

Subsysteme werden instantiiert, beginnen mit der Kommunikation, beziehen Daten, liefern Daten und lösen sich schließlich aus eigenem Entschluß oder durch externe Signale veranlaßt aus dem Systemverbund.

Neben der parallelen Ausführung mit der Synchronisation der Kommunikationszugriffe ist aus Sicht der Metaarchitektur die Fähigkeit zur Einhaltung eines bestimmten Protokolls bei Abruf und Übermittlung von Eingabe- bzw. Ausgabedaten wesentlich. Die dazu erforderlichen wenigen Methoden werden in einer abstrakten Basisklasse angelegt, um hier eine klare Schnittstellenspezifikation zu gewährleisten. Fünf einfache Methoden (drei für die Herstellung sämtlicher Kommunikationsverbindungen und je eine zum Lesen bzw. Schreiben von Daten) stellen alle erforderliche Funktionalität bereit.

Während der Instantiierung eines Subsystemobjekts wird zunächst eine Verbindung zur verwaltenden Metaarchitektur geknüpft. Diese Registrierung geschieht in der Konstruktorfunktion der Basisklasse `subsys`. Es ist nicht möglich, ein Subsystem ohne ein Rahmenwerk zu instantiieren, wohl aber, mehrere solcher Rahmenwerke unabhängig und nebeneinander zu verwenden, um mehrere Systeme betreiben zu können.[12] Erst die Registrierung macht das Subsystem der Metaarchitektur bekannt, führt aber noch

[12] Insbesondere kann auch ein Subsystem intern eine private Metaarchitekturinstanz verwenden. Diese Möglichkeit bietet sich an, um eine feinere strukturelle Untergliederung komplexer Subsysteme zu schaffen.

keine Anknüpfung an ihre Kommunikationsinfrastruktur durch. Die Verbindung des instantiierten Subsystems mit den entsprechenden Kanälen der Kommunikation geschieht durch aus dem Konstruktor des Subsystemobjekts vorgenommene Aufrufe der (aus der Basisklasse ererbten) Funktionen `consume`, `provide` und ggf. `replace` unter Angabe der jeweiligen Kanalsignaturen, um als Konsument, möglicherweise zusätzlicher Produzent bzw. als exklusiver Produzent der Kanaldaten registriert zu werden.

Während des Betriebs besteht die Möglichkeit, über die ebenfalls aus der Basisklasse ererbten Funktionen `read` und `write` indirekten Zugriff auf genau die bei der Konstruktion spezifizierten Kanäle zu nehmen. Die Ausführung dieser Funktionen wird über die in 4.4.3.1 genannten Semaphore in Bezug auf den bezeichneten Kanal (nicht die Gesamtarchitektur) serialisiert.

Die Destruktion des Subsystemobjekts löst implizit eine Abmeldung bei sowohl den konsumierten als auch den versorgten Kanälen aus. Dies veranlaßt das Rahmenwerk unter Umständen, unbelieferte Kanäle zu entfernen und ggf. Konsumenten eines wegfallenden Kanals wieder einem *downgrade* zu unterwerfen, sofern der wegfallende Kanal Resultat eines *upgrade*-Vorgangs war. In der technischen Realisierung wird der Vorgang der Deregistrierung nicht aus der Destruktorfunktion, sondern aus einer gewöhnlichen *member*-Funktion ausgelöst, die auch vom Destruktor aufgerufen wird. Der Grund dafür liegt in der Eigenart begründet, daß die Destruktion eines Subsystemobjekts nicht grundsätzlich mit dem Ausführungsende der *thread*-Funktion zusammenfallen muß. Hier sind zwei Fälle zu unterscheiden:

1. Externe Terminierung.

   Im Fall einer Destruktion durch ein Verlassen des *extent* der Variablen oder eine `delete`-Anweisung muß die *thread*-Funktion benachrichtigt werden, damit sie so rasch wie möglich terminiert. Die Destruktorfunktion terminiert nicht, bevor dies nicht geschehen ist. Diese Benachrichtigung geschieht durch eine einfache Signalvariable. Um zu verhindern, daß eine vom Entwickler des Subsystems unterlassene Prüfung des Signals zu einer Fehlfunktion führt, werden die Methoden zum Senden und Empfangen von Kanaldaten automatisch inaktiviert. Danach hat das Subsystem nur mehr die Möglichkeit, auf lokale Instanzen (wie eine ein bestimmtes Gerät repräsentierende Instanz einer Bibliotheksklasse; beispielsweise ein Roboterobjekt) einzuwirken, um einen definierten Abschlußzustand zu schaffen.

2. Selbständige Terminierung.

Der Fall eines vorzeitigen Terminierens der *thread*-Funktion durch einen Fehlerzustand oder programmgesteuertes Beenden kann andererseits noch keinen unmittelbaren Destruktoraufruf nach sich ziehen. In diesen Fällen wird die Abmeldung von allen referenzierten Kanälen über die Ausführung der "Prädestruktorfunktion" erreicht und zusätzlich vermerkt, daß bei Aufruf des Destruktors diese Aktionen nicht ein weiteres Mal vorgenommen werden müssen.

Ein gewisses Konfliktpotential entsteht im zweiten Fall dadurch, daß auf diese Weise *member*-Objekte der Klasse nicht automatisch destruiert werden. Handelt es sich dabei beispielsweise um ein Hardware-*proxy*-Objekt aus einer Roboterklasse, wäre der Zugang zum Gerät für ein nachträglich zugeladenes Subsystem (z.B. eine neue Instanz des gleichen Subsystems, was Sinn ergibt, um durch einen *reload* eine Fehlersituation zu beheben) durch die inaktive, doch zunächst existent bleibende Instanz verwehrt. Eine Abhilfe kann nur durch explizite Berücksichtigung dieses Sachverhalts im Prädestruktor oder durch eine Verlagerung der Instantiierung kritischer Klassen lokal innerhalb der *thread*-Funktion geschehen; hierfür automatisch Vorsorge zu treffen liegt außerhalb der Möglichkeiten der Programmiersprachensemantik.

4.4.3.3 Systemexterne Kommunikation

Subsysteme werden ab Konstruktion des Subsystemobjekts in einem eigenen Ausführungs-*thread* ausgeführt. Dieser *thread* muß praktisch zu beliebigen Zeiten über die vermittelnde Metaarchitektur Daten erhalten und übergeben können.

Transportcodierung. Art und Umfang der zu übermittelnden Daten sind vorab nicht festgelegt. Als gemeinsamer Typ zur Datenübermittlung über Kanäle ist `std::string` vorgesehen, der im Wesentlichen der Aufnahme von Zeichensequenzen dient. Dies setzt voraus, daß zu übermittelnde Daten in eine Transportrepräsentation dieses Formats konvertiert werden können, und eine sinnvolle Rückumwandlung in den die Daten empfangenden Subsystemen möglich ist. Erst die Verwendung einer intermediären Transportrepräsentation gestattet die Aufbewahrung und Pufferung von Daten durch die Kanalinstanzen, denen die bei der Kommunikation verwendeten Typen nicht bekannt sein müssen.

Für alle Basisdatentypen kann eine entsprechende Umwandlung durch Verwendung der jeweiligen Streamoperatoren ("<<" und ">>") geschehen. Für

die meisten ADTs der Bibliothek (Vektoren, Matrizen, Listen, Bäume, ...) ist ebenfalls durch überladene Streamoperatoren eine effiziente Umwandlung in Text und zurück in die interne Repräsentation verfügbar, die auch zur Ausgabe und persistenten Speicherung solcher Daten auf Datenträgern eingesetzt wird. Vergleichbare Operatorfunktionen existieren zudem bereits für alle Basisdatentypen. Der Aufwand für die Konversion ist hier vernachlässigbar gering; das Verfahren weist zudem die Vorzüge auf, daß erstens die in Textform erfolgende Kommunikation eines Kanals leicht durch Einfügen eines Debugger-Subsystems protokolliert werden kann, und daß zweitens wegen des allgemeinverständlichen Formats keine Verpflichtung zur Verwendung der eingeführten Klassenbibliothek und ihrer binären Datenstrukturen besteht (der genaue Typ muß auf der Empfängerseite nicht bekannt sein). Für Massendaten führt diese Methode der Umwandlung in eine textuelle Repräsentation allerdings zu einem nicht mehr vertretbaren Mehraufwand. Als einziger Typ werden die Bilddaten mit ca. 1.3 MByte pro Einzelbild daher tatsächlich in Form von Binärblöcken versandt.

Generell obliegt die Entscheidung über die Wahl einer geeigneten Transportcodierung den Entwicklern der Subsysteme, die kommunizieren sollen. Diese Entscheidung wird maßgeblich vom Aufwand der Konversion bestimmt; der Aufwand in Bezug auf die Kanalinstanzen ist allein auf die dynamische Speicherallokation das jeweilige Umkopieren des Inhalts beschränkt. Grundsätzlich können so sämtliche Typen (auch funktionsfähige Zeigertypen, da die *threads* einen Adreßraum teilen) übermittelt werden.

Das Problem der Umcodierung bzw. das des erforderlichen Typ-*cast* läßt sich nur dem ersten Anschein nach durch Einführung von `template<T>`-Kanälen entschärfen, die genau Daten eines bestimmten Typs `T` transportieren können (vgl. die Syntax der Programmiersprache Occam [Sch88, S. 88], aus technischen Gründen können hier allerdings nur Basisdatentypen übertragen werden). Bei Fehlen eines geeigneten `T`-Kopieroperators oder einer nur einseitigen und somit fehlerhaften Änderung des Protokolls bestünde keine Option mehr auf eine Überprüfung zur Laufzeit. Die während der Rückumwandlung in eine interne Repräsentation stattfindende syntaktische Analyse stellt also auch eine Sicherung der Kommunikation her.

Schnittstelle und Synchronisation. Da im Zusammenhang mit der Kanalkommunikation eine Abstimmung des Lese- und Schreibverhaltens erforderlich ist – weder ein- noch ausgangsseitig darf es möglich sein, daß während einer laufenden Datenentnahme partiell Werte überschrieben werden – werden die Methoden zum Lesen und Schreiben von Daten über die

abstrakte Subsystemklasse vorgegeben. Dies garantiert, daß nur mehr die Auslösung des Lese- bzw. Schreibvorgangs in der Kompetenz des jeweiligen Subsystems liegt, aber nicht mehr der eigentliche Vorgang der Synchronisation. Durch Schreiben einer vom vorgegebenen Subsystemrahmen abgeleiteten Klasse ist es damit nicht möglich, eine dauerhafte Blockierung eines Kanals vorzunehmen. Technisch werden nun Lese- und Schreibzugriffe gegeneinander so durch dem Kanal zugehörige Semaphore abgesichert, daß weder gleichzeitige Schreibzugriffe mehrerer beitragender Produzentensubsysteme noch ein gleichzeitiger Lese- und Schreibzugriff eines Konsumenten- und eines Produzentensubsystems erfolgen kann. Ein Lese- oder Schreibwunsch führt also im ungünstigsten Fall zu einem kurzfristigen Anhalten des *threads* des Subsystems. Da im durch Semaphore geschützten Bereich allerdings nur Daten aus der kanaleigenen in eine *thread*-eigene Variable oder umgekehrt kopiert werden müssen, kann die durch diese Maßnahme bedingte Verzögerung praktisch vernachlässigt werden.

Ein Konflikt wäre bei dieser Anordnung dann denkbar, wenn zeitnah viele Zugriffe auf einen Kanal geschehen sollten. Da Subsysteme allerdings mit vergleichsweise geringen Raten Daten produzieren bzw. nach dem Erhalt von Daten eine längerdauernde Verarbeitung stattfindet, ist von einem Auftreten dieses Konflikts im praktischen Einsatz kaum auszugehen. Um dennoch die Auftretenswahrscheinlichkeit solcher *livelocks* zu verringern, besteht konsumentenseitig die Option, ohne Betreten der durch Semaphore geschützten *critical section* durch Prüfung eines *update*-Flags festzustellen, ob ein Lesezugriff neue Daten liefern würde (dieses Kennzeichen wird nach dem Schreiben der Kanalvariablen gesetzt und muß für jeden angemeldeten Konsumenten separat geführt werden). Auf diese Weise unterbleiben redundante Lesezugriffe auf die Kanalvariable, die bei entsprechend hoher Frequenz eine Verzögerung von Produzentensubsystemen bewirken könnten. Es verbleibt das Problem, daß mit hoher Frequenz aktive Produzenten mit Beiträgen für den selben Kanal sich einerseits gegenseitig, zum anderen auch die konsumierenden Subsysteme verzögern können. Wegen der tatsächlich auftretenden Kommunikationstakte im Bereich einiger Hertz (10 Hz für vollständige SIPs) scheint dieses Problem allerdings von untergeordneter Bedeutung, sodaß kein explizites Scheduling vorgesehen wurde.

4.4.3.4 Metaarchitektur

Eine konkrete Architektur konstituiert sich durch Sammlung einer Reihe beteiligter Subsysteme, die über Kommunikationsverbindungen miteinander interagieren. Um die in 4.4.2.3 beschriebenen Eigenschaften der Metaar-

chitektur technisch umzusetzen wird eine Klasse `arch` formuliert, die dies verwaltet und die erforderliche Funktionalität bereitstellt. Erst die konkrete *Wahl der Teilsysteme und ihrer Verbindungen* determiniert vollständig die resultierende Architektur.

Hinzufügen von Subsystemen. Die Konstruktion eines Subsystemobjekts kann grundsätzlich nur unter Angabe einer Architekturreferenz geschehen. Dies stellt unter normalen Umständen bereits sicher, daß der Gültigkeitsbereich der Subsystemobjekte in dem des Architekturobjekts enthalten ist. Die ersten Maßnahmen, die ein in der Konstruktion begriffenes Subsystem unternimmt, betreffen die Meldung von Bedarf und Leistung, d.h. der für Lese- und Schreibzugriff verwendeten Kanalsignaturen. Dies geschieht durch den Aufruf von Methoden des Architekturobjekts, das auf diese Weise vorübergehend die Kontrolle erhält. Um eine Konsistenz der im Folgenden beschriebenen strukturellen Änderungen zu gewähren wird über ein `arch`-eigenes Semaphore sichergestellt, daß nur eine atomare Hinzufügen-Operation zur Zeit durchgeführt wird; das Hinzufügen von Subsystemen wird also serialisiert.

Die Aufgabe des Architekturobjekts ist im ersten Schritt, die Allokation der Ausgabekanäle (vgl. 4.4.3.3) vorzunehmen, sofern entweder noch kein entsprechend bezeichneter Kanal vorhanden ist oder aber ein Überladen angekündigt wird. Eine Allokation entsprechender Kanäle erfolgt unabhängig davon, ob das Subsystem lesende oder schreibende Nutzung anmeldet, da für beide Zugriffsarten die entsprechenden Kanalvariablen angelegt sein müssen. Die Liste der existierenden Kanäle ist dabei ein privates Attribut der Architekturklasse.

Im zweiten Schritt wird eine Zuordnung des Subsystemobjekts mit den entsprechenden Kanälen vorgenommen. Diese Zuordnung bleibt mindestens solange konstant, bis Subsysteme ge- oder entladen werden. Aus diesem Grund wird sie zur Maximierung der erreichbaren Parallelisierung in Tabellen notiert, die lokal zu den voneinander unabhängigen Kanälen verwaltet werden (vgl. 4.4.3.3). Ausgabeseitig wird das Subsystem dabei stets dem Kanal mit der entsprechenden Signatur zugeordnet, der den höchsten Index trägt; eingabeseitig gilt dies nur, wenn kein Überladen vorgenommen werden soll: Das in Initialisierung begriffene Subsystem selbst bleibt beim Überladen von Kanälen allein Konsument der Daten des ursprünglich vorhandenen Kanals, da es nicht zum Empfänger der eigenen Ausgabe werden darf. Im Falle eines Überladens muß zusätzlich ein Transfer der Konsumenteneinträge des überladenen Kanals zum neu entstandenen Kanal vorge-

nommen werden; diese Maßnahmen bleibt den betroffenen Subsystemen allerdings vollständig verborgen.

Nach diesen Aktionen wird das Semaphore, das eine Änderung an der Architekturstruktur stets nur an einer Stelle zur Zeit zuläßt, wieder dekrementiert, und schließlich der Subsystem-*thread* aktiviert.

Entfernen von Subsystemen. Wenn Subsysteme entfernt werden sollen, werden wiederum aus dem Destruktor des Subsystemobjekts bestimmte Aufrufe von Methoden der Architekturklasse ausgelöst. Die hier notwendigen Vorgänge sind im Einzelnen das Umschreiben der den Kanälen gehörenden Zuordnungstabellen und die Entfernung aller nicht mehr referenzierter Kanäle. Die Änderungen der Zuordnungstabellen umfaßt hauptsächlich das Eliminieren aller Einträge mit Referenz auf das zu entfernende Subsystem, das in Folge nicht mehr als Produzent oder Konsument der Kanaldaten in Erscheinung tritt. Weitere Änderungen sind wiederum erforderlich, sofern das betreffende Subsystem den jeweiligen Kanal durch Überladen zur Verfügung gestellt hat. Sofern dies der Fall ist – und nicht zwischenzeitlich weitere Produzenten hinzugekommen sind –, werden alle eventuell noch verbliebenen Konsumenten durch ein *downgrade* auf den signaturgleichen Kanal mit dem nächstniedrigeren Index umgetragen. Wie bereits im Fall des Hinzufügens von Subsystemen beschrieben wird auch das Entfernen als atomare Transaktion durch Semaphore geschützt ausgeführt.

Destruktion des Architekturobjekts. Wird ein Architekturobjekt aufgelöst, das noch Subsysteme und Kanäle besitzt, muß zunächst ein Anhalten der Subsysteme erzwungen werden. Da die Subsysteme Aufrufe von Methoden des Architekturobjekts vornehmen können und Kanäle referenzieren, hätte es fatale Konsequenzen, diesen Speicherplatz freizugeben, während noch Subsystem.*threads* aktiv sind. In diesem Fall wird über eine Signalisierung eine Beendigung der Subsysteme mit dem oben beschriebenen Prozeß der geordneten Entfernung der Subsysteme aus der Architektur angefordert (externe Terminierung, s. 4.4.3.2). Die Destruktormethode des Architekturobjekts terminiert somit erst, nachdem alle Subsysteme korrekt entladen wurden.

Aktivität des Architekturobjekts. Die Aktivitäten des Architekturobjekts beschränken sich auf die Verwaltung von Kanälen und Subsystemen und die Einhaltung verschiedener Integritätsbedingungen. Alle zur Steuerung

erforderlichen Methoden werden (teilweise in serialisierter Form, sofern strukturelle Änderungen vorgenommen werden) unter Kontrolle der *threads* der jeweiligen Subsysteme aufgerufen. Sofern aus dem Hauptprogramm, das das `arch`-Objekt instantiiert, kein programmgesteuerter Eingriff wie z.B. das Nachladen von bestimmten Subsystemen erforderlich wird, werden zwischen Konstruktor und Destruktor keine weiteren Methodenaufrufe unter Kontrolle des Hauptprogramms getätigt.

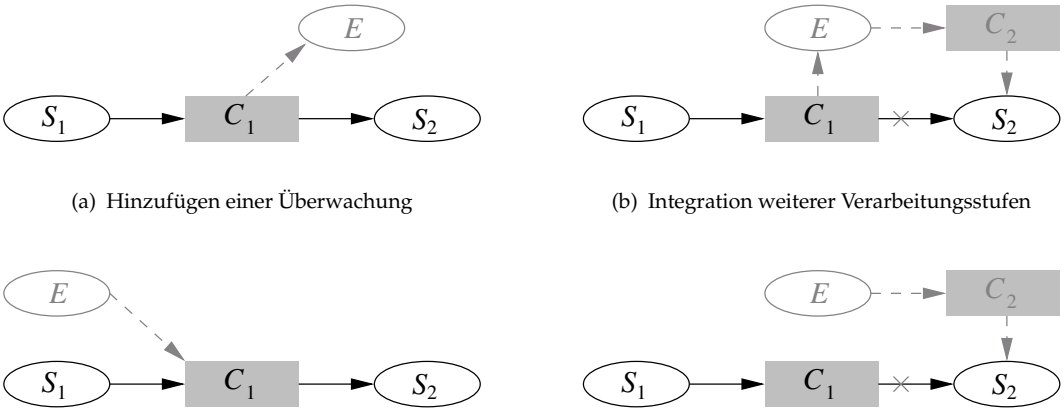
4.4.3.5 Verfahren zum Aufbau einer Architektur

Bislang ist der eigentliche Aufbau einer Architektur noch nicht in das Blickfeld der Untersuchung gerückt. Über Subsysteme, Kanäle und (als verwaltende Instanz dieser beiden Typen) die Metaarchitektur sind jedoch alle Grundlagen zur Konstruktion bereitgestellt.

Um ein konkretes Schema der Kontrolle aufzubauen werden lediglich ausgewählte Subsysteme instantiiert. Die Verbindung der Subsysteme wird unter Berücksichtigung der Kanalsignatur durch die Metaarchitektur vorgenommen. Ein typisches Hauptprogramm instantiiert also ein zunächst leeres Architekturobjekt `arch A` und lädt in Folge Subsysteme `subsys` S_i`(A,` id_i`)` nach. Dieser Vorgang setzt die implizite Allokation der benötigten Kanäle und die Anbindung der Subsysteme an die richtigen Kanäle in Gang, während die einzelnen Subsysteme in separaten *threads* ihre Aktivität entfalten und mit der Roboterhardware interagieren. Das Nachladen weiterer Subsysteme führt zu deren Integration in das bestehende Verbindungsnetzwerk, während das Entfernen von Subsystemen die Funktionalität des Gesamtsystems nur im geringstmöglichen Maße beeinflußt (z.B. durch Wegfall eines Produzenten eines redundant versorgten Kanals oder *downgrade* und Qualitätsverlust im Fall eines überladenen Kanals). Erst der Wegfall von Subsystemen, die unverzichtbare und nicht redundante Information beitragen, führt zu einem Verlust der Gesamtsystemfunktion.

Nachträgliche Architekturänderung. Der Prozeß des Ladens von Subsystemen ist nicht nur zum initialen Aufbau einer Konfiguration einsetzbar. Gerade das späte Hinzufügen von Subsystemen gestattet strukturelle Eingriffe, ohne daß Änderungen an den bestehenden Subsystemen oder ihrer nur implizit formulierten Vernetzung vorgenommen werden müssen:

- Das Einfügen eines *debugger*-Subsystems (Abb. 4.30(a)) für nicht überladene Kanäle. Dieses tritt als Klient eines durch die Signatur spezifi-



(a) Hinzufügen einer Überwachung

(b) Integration weiterer Verarbeitungsstufen

(c) Hinzufügen einer weiteren Datenquelle

(d) Inhibition einer Subsystemausgabe

Abb. 4.30: Optionen zur Architekturänderung: Durch Hinzufügen eines Subsystems E und ggf. eines überladenen Kanals C_2 läßt sich eine bereits vorhandene Verarbeitungsstruktur (S_1, C_1, S_2) auf vielfältige Weise transparent rekonfigurieren.

zierten Kanals auf und kann beispielsweise eine Ausgabe der empfangenen Nachrichten auf Konsole oder in eine Logdatei vornehmen oder bestimmte Typen (z.B. Bilder oder andere Sensorscans in verschiedenen Stadien der Vorverarbeitung) darstellen.

- Die Integration einer vollständig transparenten *Nachverarbeitung* ist durch Hinzufügen eines Subsystems möglich, daß den betreffenden Kanal überlädt (Abb. 4.30(b)). Diese Möglichkeit besteht selbst dann noch, wenn diese Maßnahme bereits mehrfach in der zu ändernden Architektur vorgenommen wurde.

- Das transparente Hinzufügen einer weiteren *neuen Datenquelle* (sei es ein Sensor oder ein Verarbeitungsverfahren) ist ohne weiteres möglich, indem zu bereits verwendeten Kanälen signaturgleiche Daten hinzugefügt werden (Abb. 4.30(c)). Das System profitiert auf diese Weise von nachträglich hinzufügbaren Systemkomponenten, die bei der ursprünglichen Systemkonzeption nicht einmal vorgesehen waren, ohne daß Änderungen der Subsysteme erforderlich werden.

- Das nachträgliche *Inhibieren* als fehlerhaft erkannter Softwaresystemkomponenten läßt sich durch geeignetes Überladen der betreffenden Kanäle bewerkstelligen (Abb. 4.30(d)).

Keine dieser Optionen erfordert, daß Änderungen am Code der eingesetzten Subsysteme vorgenommen werden müssen.

Dynamisch nachladbare Subsysteme. Das nachträgliche Einfügen von Subsystemen erfordert zwar keine Änderungen der vorhandenen Subsysteme, doch noch immer eine Änderung des Quelltexts des Hauptprogramms, in dem die Subsysteme instantiiert werden. Es lassen sich Maßnahmen treffen, um auch dies zu vermeiden. Damit können Architekturen konstruiert werden, die in der beschriebenen Weise änderbar sind, ohne daß überhaupt ein Quellcode verfügbar sein muß.

Die Lösung für dieses Problem liegt im Nachladen ausführbaren Codes. Dies läßt sich betriebssystemseitig über Aufrufe der Funktion der Bibliothek `libdl.so` (die Schnittstelle wird in `dlfcn.h` definiert) lösen, die unter Linux und Solaris standardmäßig effiziente Funktionen zum Zugriff auf den dynamischen Linker zur Verfügung stellt. Insbesondere werden die Funktionen zum Laden von Bibliotheken (`dlopen`) und zum Auflösen von Symbolreferenzen (`dlsym`) benötigt. Da die `libdl.so` allerdings keine Objektorientierung unterstützt, ist nachgeladener Code, der Klassen implementiert, nicht unmittelbar so aus dem Rahmenprogramm nutzbar, als daß konventionell eine Instanz erzeugt werden könnte. Es ist scheinbar nicht möglich, beispielsweise Objekte von Subsystemklassen überhaupt zu instantiieren, wenn nicht der genaue Klassenname (benötigt z.B. für den Konstruktoraufruf) vorab im Rahmenwerk bekannt ist. Das lexikalische Symbol wird allerdings überhaupt erst in der Bibliothek definiert und ist dem Rahmenwerk grundsätzlich unbekannt.

Eine Lösung dieses technischen Problems wurde erzielt, indem in allen dynamischen Subsystembibliotheken eine sog. *factory*-Methode außerhalb aller Klassen implementiert wurde, die einen festgelegten Namen trägt und der gewöhnlichen Namensauflösung mit `dlsym` zugänglich ist. Ein Aufruf dieser Funktion instantiiert das Subsystem und gibt einen Zeiger auf diese Instanz zurück. Eine Verwendung der Instanzen genau wie mit denen statischer Subsystemklassen durch das Metaarchitekturobjekt ist möglich, da ihr festgelegtes Interface (Konstruktor, Destruktor, `run`-, `read`- und `write`-Methoden) von einer abstrakten Klasse abgeleitet ist, auf der das Metaarchitekturobjekt operiert; die bei einem Aufruf erfolgende Auflösung der *pure virtual*-Methoden führt zur Ausführung des nachgeladenen Codes.

Um auf diese Weise eine Architektur mit nachladbaren Subsystemen rekonfigurieren zu können, ist es offensichtlich erforderlich, daß die technischen Voraussetzungen dazu (d.h. das Laden von Bibliotheken mit dynamischen

Subsystemen) zwar explizit vorgesehen werden müssen, jedoch nicht Art und Umfang der Änderungen selbst: Im äußersten Fall wäre denkbar, das Hauptprogramm so zu formulieren, daß die Architektur ausschließlich aus nachzuladenden dynamischen Subsystemen zusammengesetzt wird. Ebenso ist es möglich, die Strukturkonfiguration selbst in die Verantwortung eines speziellen (statischen) Subsystems zu legen, das über eine Kanaleingabe Konfigurationsanweisungen erhält und ausführt.

4.4.4 Prototypen für Gesamtsysteme

Der grundsätzliche Nachweis der Tauglichkeit des Konzepts, Architekturen aus Subsystemen aufzubauen, soll an der prototypischen Umsetzung der Lösung für ein Testproblem erfolgen. Um zu demonstrieren, daß der gewählte Ansatz für die Umsetzung gemäß der verschiedenen Entwurfsparadigmen (Kap. 2) angemessen ist, soll ein zur Untersuchung herangezogenes Anwendungsproblem exemplarisch in deliberativer, reaktiver und hybrider Manier bearbeitet werden. Im Vordergrund steht hier nicht die Angemessenheit der jeweiligen Entwurfsparadigmen und die resultierende Leistung der erstellten Systeme: Ziel ist, die Eignung des vorgestellten Konzepts für die Umsetzung der verschiedenen Architekturen nachzuweisen.

Um einen ähnlichen Schwierigkeitsgrad der Umsetzung aller Versionen zu erreichen wurde das Problem der autonomen Exploration in unbekannter und strukturierter, aber stets nur partiell sensorisch erfaßbarer Umgebung gewählt. Die Bearbeitung dieses Problems kann als Grundlage für nachfolgende Aufgaben der Erkundung, Kartierung, Selbstlokalisierung und für beispielsweise das Auffinden von aufgabenrelevanten Objekten initial unbekannter Position angesehen werden [BBE+98, BEH+99]. Dieses Problem soll jeweils so bearbeitet werden, daß unter Verwendung der gewählten Gestaltungsmittel eine reaktive (4.4.4.1), deliberative (4.4.4.2) bzw. hybride Architektur (4.4.4.3) entsteht.

Szenario. Um für alle entstehenden Problemlösungen eine gemeinsame Grundlage zu schaffen, wird eine einfache simulierte "Welt" (Abb. 4.31) verwendet, in der sich der Roboter bewegen kann. Sie bietet einen zu explorierenden Platz von ca. 250 m^2. Sowohl für die Bewegung wie auch den Einsatz der Sensoren werden Modelle eingesetzt, die die Charakteristik der entsprechenden Geräte in Bezug auf z.B. Positioniergenauigkeit und Sensorrauschen widerspiegeln. Die Regelung der Parameter Rotations- und Translationsgeschwindigkeit erlaubt eine Lenkung des Roboters (von einer Nutzung

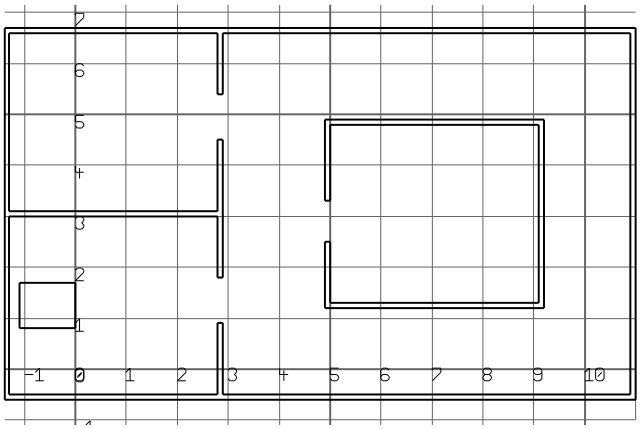


Abb. 4.31: Simulierte Testumgebung für die Umsetzung von Architekturen: Der Roboter startet in $(0, 0)$, Aufgabe ist die Exploration der Umgebung.

der *direct motion commands* wurde abgesehen, da diese der reaktiven Lösung nicht zugänglich sind). Die Verwendung eines Simulationsmodells hat gegenüber der Realität in diesem Fall den Vorzug, daß zu jedem Zeitpunkt eine fehlerfreie Aufnahme des Ist-Zustandes möglich ist; d.h. die tatsächliche Position der Plattform ist für Zwecke der Ergebnissicherung stets bekannt. Das Koordinatensystem wird willkürlich so festgelegt, daß die Startposition des Roboters zu $(0, 0)$ und die initiale Orientierung zu $0°$(in Richtung der x-Achse) bestimmt ist. Die Aufgabe der simulierten Plattform ist dabei lediglich, einen möglichst großen Teil der Umgebung sensorisch zu erfassen, ohne dabei mit Hindernissen in Berührung zu kommen.

4.4.4.1 Reaktive Kollisionsvermeidung

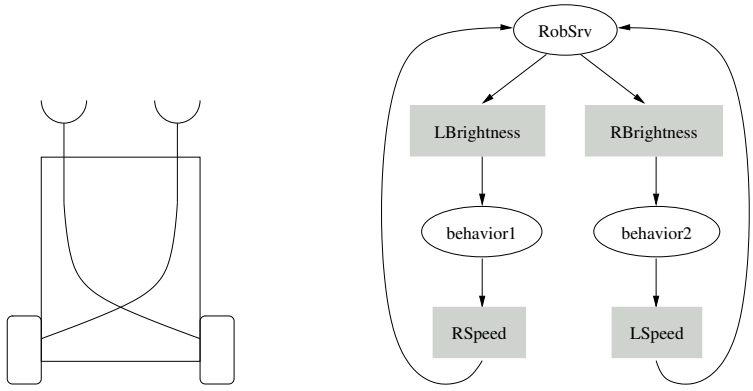
Reaktive Systeme gelten als "aufgabengetrieben", weisen aber grundsätzlich keine explizite Codierung von Abläufen auf, die der Lösung des Anwendungsproblems dienen. Beim Entwurf solcher Systeme werden Subsysteme als Mittel der Implementierung einer idealerweise zustandsfreien Datentransformation eingesetzt. Sie beschreiben somit einen Aspekt des Verhaltens des Gesamtsystems. Die Sicherung der Lokalität eines eventuell erforderlichen Zustands zu dem jeweiligen Subsystem entspricht dem erweiterten reaktiven Paradigma.

Spezifikation reaktiver Systeme. Die ursprünglichen Umsetzungen reaktiver Architekturen sahen als Designrichtlinie vor, daß die *behaviors* unabhängig voneinander funktionieren und idealerweise über lokale Sensoren und Effektoren verfügen sollten. Dies gewährleistet, daß die einzelnen Verhalten tatsächlich – einmal abgesehen von Varianten der Subsumption – unabhängig voneinander ablaufen; das Gesamtverhalten kann damit in keinem Fall anders als emergent entstehen. Die Freiheit der Subsysteme von Zustand und Weltrepräsentation liegt in der Verantwortung des Subsystementwicklers und wird durch keine besonderen strukturellen Mittel erzwungen. Die Existenz einer Vielzahl klassenlokaler Variablen innerhalb des Subsystems bzw. lokaler Variablen der `run`-Methode wäre ein Hinweis auf einen nicht dem reaktiven Paradigma entsprechenden Entwurf.

Während die Bedingung der Entkoppelung der *behaviors* durch den Ausführungsmodus der Subsysteme ermöglicht wird und somit eine unmittelbare Abbildung von *behaviors* auf Subsysteme gestattet ist, kann die Designrichtlinie der Nutzung vorwiegend lokaler Sensoren und Effektoren aus technischen Gründen zumindest nicht mit beliebigen Plattformen eingehalten werden. Für den Einsatz nicht für reaktive Architekturen konstruierter Hardwaresysteme ist unter Umständen eine Abstimmung des Zugriffs auf verschiedene Hardwaresystemkomponenten beispielsweise durch Serialisierung erforderlich. Zur Lösung dieses nicht durch die Kontrollarchitektur, sondern durch den technischen Aufbau bedingten Problems wird hier und für den folgenden deliberativen bzw. hybriden Entwurf ein besonderes Subsystem eingeführt, das eine Kommunikation mit den Sensoren und Effektoren vermittelt. In Abb. 4.32 und den folgenden Abbildungen ist dieses Subsystem mit `RobSrv` bezeichnet; es erhält Zugang zu den referenzierten Hardwarekomponenten über eine spezielle Bibliothek, die die Hardware-*proxy*-Instanzen zur Verfügung stellt (4.2.2.2).

Ein Beispiel zur Umsetzung einer einfachen reaktiven Architektur unter dieser Randbedingung zeigt Abbildung 4.32: Beide durch jeweils ein Subsystem implementierte *behaviors* nutzen exklusiv je einen Sensor und einen Effektor, bedienen sich jedoch der gemeinsamen Schnittstelle des `RobSrv`-Subsystems. Setzen die *behaviors* die Regel $\beta\text{Speed} := c \cdot \alpha\text{Brightness}$ mit einer Skalierungskonstanten $c > 0$ und $\alpha, \beta \in \{\text{L}, \text{R}\}$ um, ist das emergente Verhalten des Systems, sich Lichtquellen zu nähern.

Ein weiteres Charakteristikum reaktiver Komponenten ist, daß in der Kommunikation der Subsysteme dem reaktiven Konzept entsprechend *relative* Werte den größten Anteil stellen, da absolute Angaben – Koordinaten mit Bezugspunkt außerhalb des Roboters, Winkel- und Streckenabgaben – nur



(a) *"Vehicle 2b"* [Bra84]

(b) Schema der reaktiven Systemarchitektur zu (a)

Abb. 4.32: Umsetzung eines BRAITENBERG-*vehicle* (s. Text).

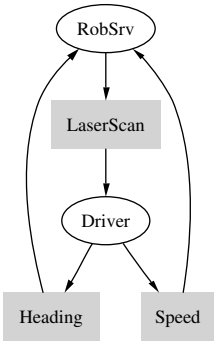
in Bezug auf einen in reaktiven Systemen nicht vorhandenen Systemzustand Bedeutung tragen.

Identifizierte Subsysteme. Im Beispiel der reaktiven Umgebungsexploration ist aus dem genannten Grund der Verwendung relativer Koordinaten insbesondere der Odometriesensor nicht sinnvoll einsetzbar. Die Daten des LRF hingegen werden in robozentrischen Koordinaten ermittelt. Zur Steuerung der Effektoren jedoch muß eine Regelung der Parameter Translations- und Rotationsgeschwindigkeit unternommen werden, da das Einstellen von Zielwinkel und -position eine zustandsbehaftete Überwachung voraussetzen müßte.

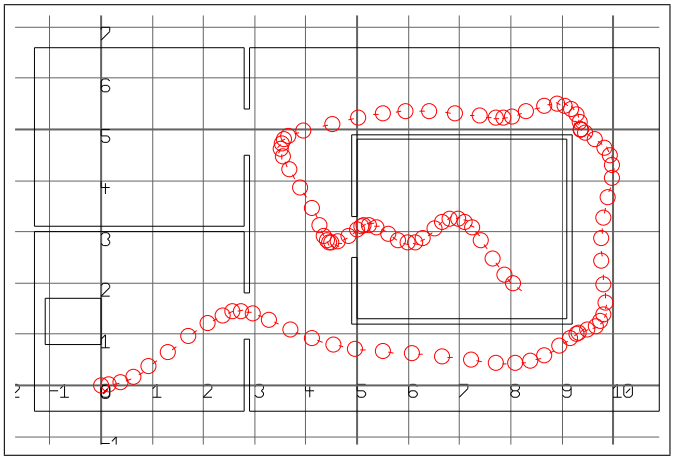
Für eine Umgebungsexploration wird eigentlich nur ein einziges Subsystem erforderlich, das die Parameter der Translations- und Rotationsgeschwindigkeit entsprechend der in robozentrischen Koordinaten erhaltenen Hindernisse aus einem LRF- oder Sonarscan regelt. Eine Aufteilung in zwei unabhängige Subsysteme für diese Teilaufgaben wäre möglich, verspricht aber keine verbesserte Leistung. Dieses als `Driver` bezeichnete System unternimmt die reaktive Kopplung[13] von Sensordaten auf die Effektorsteuerung.

Das Subsystem `RobSrv` (s. Abb. 4.33(a)) bildet eine Schnittstelle zur Roboterhardware und nimmt im Betrieb die einzustellenden Geschwindigkeits-

[13] Allein aus praktischen Gründen ist das Subsystem zustandsbehaftet, um durch Glättung der Steuerkommandos der Masse der Plattform Rechnung zu tragen.



(a) Architekturschema für kollisionsfreie LRF-Navigation



(b) Resultat (Zustand nach 100 Schritten): Die lokale Wahl von Werten für Winkel- und Translationsgeschwindigkeit, die eine möglichst ungestörte Fahrt gestatten, führt zum emergenten Verhalten der kollisionsfreien Exploration.

Abb. 4.33: Beispiel für die Umsetzung einer reaktiven Architektur: `Driver` ermittelt eine Abbildung der LRF-Scan-Daten auf Winkel- und Translationsgeschwindigkeiten.

werte entgegen. Die Verwendung eines separaten Subsystems ist hier technisch nicht erforderlich, schafft aber die Grundlage zur Integration weiterer Subsysteme, die auf diese Weise ebenfalls die Möglichkeit zur Steuerung der Roboterbewegung erlangen können. Für eine Abstimmung des gemeinsamen Zugriffs auf die selben Effektoren müßte noch ein weiteres Subsystem eingefügt werden, das eine wie auch immer geartete Kombination der Steuerinformation (Mittelung, Auswahl über Priorität der Quelle etc.) unternimmt. Eine weitere Aufgabe von `RobSrv` ist, LRF-Scans durchzuführen und im entsprechenden Kanal allen Interessenten zur Verfügung zu stellen.

Interaktion und Gesamtfunktion. Das Subsystem `RobSrv` führt im Konstruktor einen Verbindungsaufbau mit der Roboterhardware durch und stellt anschließend kontinuierlich LRF-Messungen im `LaserScan`-Kanal zur Verfügung. Weitere selbständige Aktion wird nicht unternommen. Der Destruktoraufruf würde schließlich einen geordneten Abbau aller Verbindungen auslösen.

Sobald `Driver` eine Liste der LRF-Meßpunkte erhält, wird der Winkel im Scanbereich ermittelt, unter dem eine maximale Fahrstrecke in mindestens

der Breite des Roboters zur Verfügung steht (hier steckt aller Reaktivität zum Trotz ein unverzichtbarer Parameter). Entsprechend des relativen Winkels wird eine Winkelgeschwindigkeit der Plattform im Kanal `Heading` gesetzt; die absolute Größe ist für die korrekte Funktion des Systems praktisch irrelevant und lediglich so zu wählen, daß bei der erzielbaren Rate von `Driver`-Aufrufen noch nicht die vollständige zu steuernde Winkeldifferenz abgearbeitet wurde. Aus mechanischen Gründen ist es zweckmäßig, für kleine Richtungsänderungen eine geringere Winkelgeschwindigkeit und umgekehrt zu wählen, und zudem eine zustandsbehaftete Dämpfung durchzuführen. Die Regelung der Translationsgeschwindigkeit erfolgt in Abhängigkeit von der Distanz des nächsten Hindernisses, das in einem entsprechenden Scanausschnitt zwischen 0°und dem ermittelten "idealen" Winkel liegt. Eine entsprechende Angabe wird in `Speed` übermittelt und von `RobSrv` an die Roboterplattform übermittelt.

Das emergente Systemverhalten entspricht einer Drehung in Richtung des längsten freien Fahrabschnittes und weist eine Geschwindigkeitsabnahme bis zum Stillstand entsprechend der Hindernisentfernung auf. Dies gestattet, eine Drehung selbst vor einem nahen Hindernis zu vollenden, indem die Rotationskomponente scheinbar ein größeres Gewicht gewinnt, obgleich keinerlei Abstimmung der beiden Teile des Gesamtverhaltens vorgenommen wird. Bei großer Entfernung zu in Fahrtrichtung befindlichen Hindernissen wird die Geschwindigkeit entsprechend erhöht (vgl. Abb. 4.33, die Positionsmarkierungen wurden in zeitlich gleichen Abständen notiert, um die Geschwindigkeitsvariation sichtbar zu machen). Die Exploration ist kollisionsfrei, aber notwendig nicht zielgerichtet, da in der reaktiven Umsetzung kein Modell eines durch das System erkennbaren Ziels vorhanden sein kann.

4.4.4.2 Deliberative Exploration und Kartierung

Deliberative Systementwürfe sind von der Nutzung einer internen Repräsentation der äußeren Welt geprägt. Dieses Modell gestattet, zustandsbehaftet Entscheidungen über die zu einem bestimmten Zeitpunkt angemessene Aktion zu treffen, die über eine Betrachtung der aktuellen Sensordaten hinausgehen.

Spezifikation deliberativer Systeme. Im Fall des Entwurfs deliberativer Systeme werden bestimmte Subsysteme die Funktionen der Pflege eines Weltmodells ausüben, also mit dessen Aufbau, Ergänzung und evtl. Prüfung der Konsistenz mit den aktuellen Sensordaten und der Ableitung von

Planungszielen beschäftigt sein. Das Erreichen von solchen Zielen der Planung kann einschrittig durch eine atomare Effektoraktion möglich sein, erfordert aber eventuell auch die Einführung von Zwischenzielen, von denen allein das erste überhaupt auf direkte Weise erreichbar ist.

Die Implementierung des benötigten Zustands in Subsystemen geschieht durch Verwendung von subsystemlokalen Variablen. Die Steuerung eines mehrschrittigen Ablaufs kann durch Standardmethoden, z.B. die Verwaltung einer Queue von Teilzielen geschehen; die Überwachung des aktuellen Teilziels auf Erreichen oder endgültige Nichterreichbarkeit mit der dann eintretenden Notwendigkeit einer Neuplanung wird üblicherweise durch einen endlichen Automaten vorgenommen. Eine Persistenz von Planung und Zustand über die aktuelle *session* hinaus kann sehr einfach durch geeignete Spezifikation von Konstruktur und Destruktor der Subsystemklasse erreicht werden.

Die asynchrone Kanalkopplung gestattet den datengenerierenden Subsystemen ein kontinuierliches Weiterarbeiten. Entsprechend der deliberativen Grundstruktur ist ein Sensordaten verarbeitendes Subsystem jedoch in keiner Weise verpflichtet, während der Durchführung von Berechnungen neu eintreffende Eingabedaten überhaupt zur Kenntnis zu nehmen. Sollte aus irgendeinem Grunde eine vollständige Speicherung und Auswertung aller Kanaldaten erforderlich sein, so wäre ein eingangsseitiges Pufferungssubsystem einzusetzen, das mit einem bidirektionalen Protokoll (zwei Kanäle) zur sequentiellen Herausgabe aller Datensätze veranlaßt werden könnte.

Identifizierte Subsysteme. Im Fall der deliberativen Exploration kann im Gegensatz zu den reaktiven Systemen die Odometrieinformation mit Gewinn genutzt werden, um aktuelle Sensordaten unter Kenntnis der Systemposition in Bezug auf einen ortsunveränderlichen Bezugspunkt einem wachsenden Weltmodell hinzuzufügen, das nicht nur den aktuell sensorisch verifizierbaren Zustand widerspiegelt. Mit dieser Hinzufügung tritt zum ersten Mal ein Problem mit dynamischer Umwelt auf. Die Berücksichtigung des Alters der im Weltmodell notierten Information wäre ein hierfür ein denkbarer Lösungsansatz, soll hier aber aus Gründen der Übersichtlichkeit des Beispiels unterbleiben.

Eine Umgebungskarte erscheint als das geeignete Weltmodell, um bei der Exploration im Testszenario den Fortgang der Erkundung festzuhalten und Routen zu noch nicht besuchten Punkten zu planen, wird im begrenzten Szenario jedoch nicht zur Durchführung einer Lokalisierung genutzt (vgl. [CN01]). Als Form der Datenspeicherung wird das *occupancy grid* ge-

wählt, das pro Rasterzelle einen der Zustände "frei", "belegt", "unbekannt (unerreichbar)" oder "unbekannt (erreichbar)" (im Folgenden abgekürzt durch die Symbole F, B, Uu bzw. Ue) speichern soll. Dieses Modell wird im Subsystem Mapper verwaltet (Abb. 4.34(a)). Die Driver-Komponente ist gegenüber der in 4.4.4.1 verwendeten leicht abgewandelt: sie entnimmt nicht mehr einem LRF-Scan unmittelbar den vorläufigen Zielpunkt, sondern wird vom Mapper entsprechend versorgt. Da keine Notwendigkeit zur ausschließlichen Verwendung robozentrischer Koordinaten besteht, ist die so erhaltene Information in Bezug auf ein globales Koordinatensystem formuliert. Der Driver benötigt zur Auslösung der angemessenen Aktion also zusätzlich den aktuellen Aufenthaltsort der Roboterplattform. Das unverändert anzusteuernde Subsystem RobSrv stellt wiederum die Schnittstelle zu den benutzten Sensoren und Effektoren zur Verfügung. Es entspricht in praktisch jedem Aspekt dem des bereits in 4.4.4.1 beschriebenen. Die einzige Abweichung ist, daß auch die Odometrieinformation über einen eigenen Kanal zur Verfügung steht. Das Subsystem ist nach wie vor zustandsfrei (in der zugrundeliegenden Bibliothek arbeiten allerdings notwendigerweise einige Protokollautomaten) und wird wiederverwendet.

Interaktion und Gesamtfunktion. Vor Ausführung des ersten Sensorscans sind alle Kartenwerte mit dem Symbol Uu vorinitialisiert. Rasterzellen werden nach Durchführung eines Scans mit F markiert, wenn sie Uu oder Ue waren und von einem Meßstrahl ohne Hindernis durchdrungen werden. Die Markierung mit B erfolgt, wenn auch nur ein Meßpunkt innerhalb des Rasterfelds liegt.[14] Das Subsystem Mapper führt dieses Weltmodell und trägt unter Nutzung der eigenen Positionsdaten den Sensorscan in die Karte ein. Im Fall von Verdeckungen und durch das eingeschränkte Sichtfeld des Sensors entstehen Randbereiche, an denen mit F markierte Felder mit solchen mit unbekannt-unerreichbarem Status Uu benachbart sind. Diese werden als Ue markiert, da ein Weg besteht, in die Nachbarschaft dieses speziellen unbekannten Feldes zu gelangen, um seinen Belegtheitsstatus aufzuklären.

Jeweils das dem Roboter nächste Rasterfeld mit dem Status Ue dient der Plattform als Zwischenziel. Die erste geradlinige Teilstrecke der Route, die über eine einfache *distance propagation* (DP) ermittelt wird (vgl. Abb. 4.34(b)), wird an den Driver übermittelt. Diese Information wird als Ergebnis einer

[14] Die Kartierung ist also u.U. nichtmonoton, da auch F-Felder bei einem später entdeckten Hindernis zu B-Feldern werden. Bei Präsenz bewegter Hindernisse müßte eine spätere Rückumwandlung in F-Felder vorgesehen werden. – Das Verfahren kann gegenüber [ME85, ME87] wegen der hohen Meßgenauigkeit des Sensors stark vereinfacht werden.

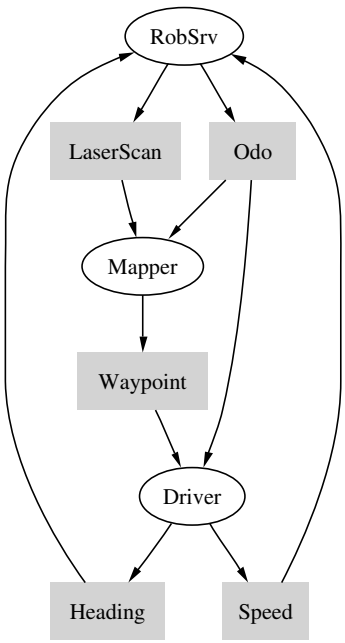
expliziten Planung über den mit `Waypoint` bezeichneten Kanal ausgegeben. Im `Driver`-Subsystem geschieht die Umsetzung der Planungsergebnisse in Effektoraktionen: der zu erreichende `Waypoint` wird durch Steuerung der Größen Translations- und Rotationsgeschwindigkeit angefahren und dann die Bewegung gestoppt. Sobald das zu klärende Feld in den Sichtbereich des Sensors gerät, wird sein Zustand ohne weitere Maßnahmen auf "frei" oder "belegt" geändert. Eine Neuplanung wird unter anderem also auch dann ausgelöst, sobald sich der Status des aufzuklärenden Zielfeldes verändert hat. Ein weiterer Grund für die Durchführung einer Neuplanung kann das Erscheinen eines näheren aufzuklärenden Punktes in der Karte sein.

Durch die Vergröberung auf Zellen des *occupancy grid* entsteht ein Modell, das in Abb. 4.34(b) für das Testszenario skizziert ist (die zur Veranschaulichung unterlegte Karte aus Liniensegmenten gehört nicht zum Weltmodell). Das System navigiert kollisionsfrei in Bezug auf kartierbare Hindernisse und generiert in endlicher, statischer Umwelt eine vollständige Karte. Die vom System dazu zurückgelegte Route ist in Abb. 4.34(c) dargestellt – sie ist nicht notwendig minimal, stellt aber angesichts der Tatsache, daß initial keine Kenntnisse über die Umgebung vorhanden waren, eine gute Näherung dar.

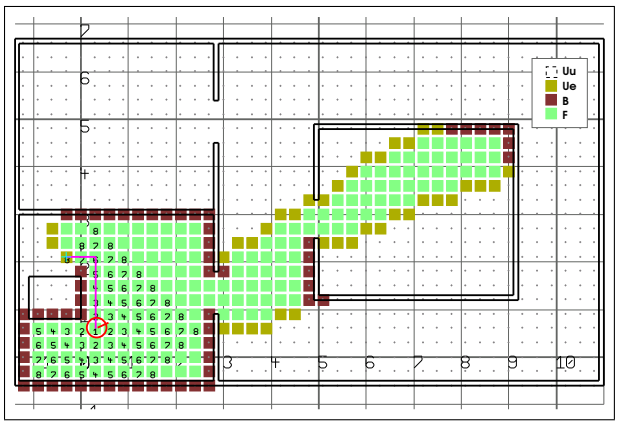
Das gesamte System hat die Eigenschaft, zielgerichtet eine vollständige Kartierung der erreichbaren Umwelt auszuführen. Es trägt alle typischen Kennzeichen deliberativer Systementwürfe: es besitzt ein Weltmodell, trifft Entscheidungen nicht allein auf Basis der Sensordatenlage, verfügt über einen mit vergleichsweise geringer Frequenz aktiven Planungsmechanismus (Neuplanung und Ausgabe nach `Waypoint` finden erst statt, sobald das Planungsziel erreicht wurde) und einen Subplaner, der Teilziele für die direkte Umsetzung generiert.

4.4.4.3 Hybride Exploration

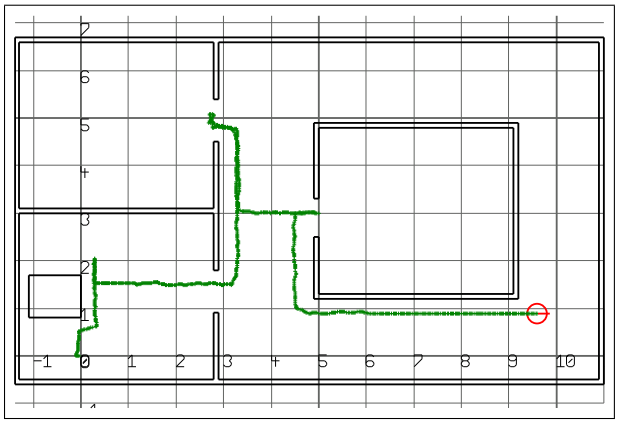
Hybride Systeme sind keinen besonderen Einschränkungen in Bezug auf ihre Binnenstruktur unterworfen. Viele Vertreter dieser Systemklasse bedienen sich der Dienste einer reaktiv implementierten Basisschicht, die durch übergeordnete deliberative Prozesse ggf. neu parametrisiert werden, um ein längerfristiges Planungsziel zu erreichen. Die sensorische Eingabe dient also sowohl der unmittelbaren Reaktion wie – nach Durchführung einer Interpretation – der Modellbildung und Deliberation.



(a) Architekturschema



(b) Zwischenzustand: das nächste als unbekannt/erreichbar (Ue) markierte Feld wird zum Ziel; die Route wird durch *distance propagation* geplant.



(c) Vollständige Route des Systems nach abgeschlossener Exploration.

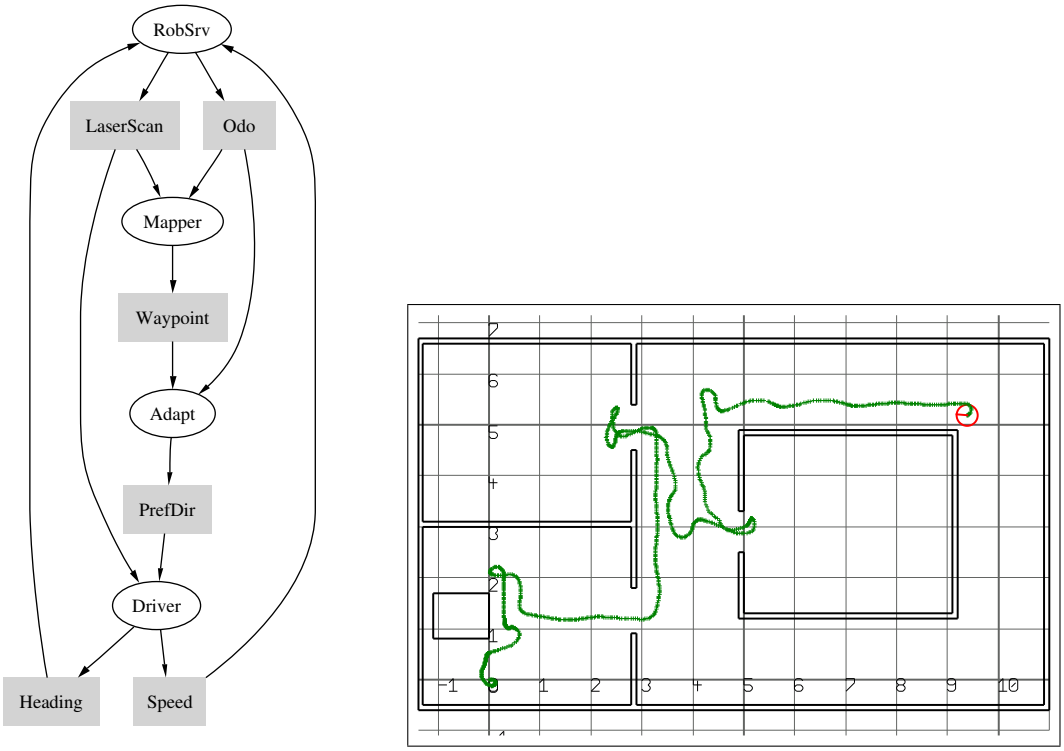
Abb. 4.34: Verhalten des deliberativen Systems im Testszenario.

Spezifikation hybrider Systeme. Die benannte separate Verwendung der Sensordaten zu zwei verschiedenen Zwecken wird strukturell durch die Möglichkeit erfaßt, mehrere voneinander unabhängige Subsysteme zu Konsumenten eines Sensordatenkanals zu machen. Die grob "Schichten" zugeordneten Teilsysteme werden wie in 4.4.4.1 und 4.4.4.2 in reaktiver bzw. deliberativer Manier gestaltet. Eine zusätzliche Verbindung hat zu erfolgen, um eine wechselseitige Interaktion zwischen den Verarbeitungsschichten zu gestatten. Dies kann als "dritte" Schicht (vgl. 2.1.4.4) geschehen; in einfachen Fällen wie dem vorliegenden ist allerdings eine unidirektionale Parameterschnittstelle zum reaktiven Ausführungsteilsystem ausreichend, über die ein deliberatives Steuerungssystem eine Anpassung des reaktiven Verhaltens vornehmen kann.

Identifizierte Subsysteme. Als reaktive Ausführungseinheit sollen genau die Subsysteme aus 4.4.4.1 wiederverwendet werden, die in Kombination bereits eine kollisionsfreie, allerdings noch nicht zielorientierte Navigation ermöglicht hatten. Das Planungssubsystem aus 4.4.4.2, das eine Karte der Umgebung erarbeitet und als Grundlage für die Bestimmung einer zweckmäßigen Route verwendet, soll ebenfalls in unveränderter Form erneut eingesetzt werden. Ebenfalls wird die Roboterschnittstelle als `RobSrv`-Subsystem wiederverwendet.

Die Ausgabe des deliberativen `Mapper`-Subsystem kann nicht unmittelbar zur Steuerung reaktiver Systemkomponenten genutzt werden, da die Daten (die Weltkoordinaten des nächsten Zielpunktes) nicht in robozentrischen Koordinaten ausgegeben werden. Durch den Planer ist allerdings garantiert, daß der nächste Punkt als Planungsteilziel in jedem Fall von der gegenwärtigen Position der Plattform aus in direkter Fahrt erreichbar ist. Ein zusätzliches Subsystem `Adapt` (s. Abb. 4.35(a)), das über das Teilziel informiert ist und gegenwärtige Position und Orientierung der Plattform dem `Odo`-Kanal entnimmt, ist also in der Lage, eine in Bezug auf das lokale Plattformkoordinatensystem angegebene Vorzugsrichtung (Kanal der Bewegung zu errechnen. Diese wird in den Kanal `PrefDir` übermittelt. Einer architekturspezifischen Anpassung wird schließlich das `Driver`-Subsystem unterzogen werden, da die entsprechende reaktive Form noch nicht über die Option einer Übernahme veränderlicher Parameter – in diesem Fall der ermittelten Vorzugsrichtung – verfügte.

Interaktion und Gesamtfunktion. Das Verhalten des Gesamtsystems ohne aktivierten Planer entspricht dem des in 4.4.4.1 vorgestellten Systems:



(a) Schema

(b) Trajektorie der Roboterbewegung bis zum Abschluß der Exploration

Abb. 4.35: Schema und Verhalten des hybriden Systems im Testszenario.

Translations- und Rotationsgeschwindigkeit werden so gewählt, daß eine Bewegung in Richtung des maximalen Freiraums erfolgt. Die Erweiterung des `Driver`-Subsystems wurde so gestaltet, daß ein Empfang von Angaben über die Vorzugsrichtung nicht zwingend erforderlich ist. Dies sichert zum einen im Sinne der Robustheit eine gewisse Basisfunktionalität und ist zum anderen ohnehin sinnvoll, da Angaben durch das Planungssubsystem mit nur vergleichsweise geringer Rate generiert werden können. Dies gilt unter der üblichen Annahme, daß deliberative Planer eine wegen des erforderlichen höheren Abstraktionsniveaus generell langsamere Verarbeitung durchführen als reaktive Komponenten. Die Generierung von `Waypoint`-Daten entspricht unverändert dem vorgestellten deliberativen Beispiel. Nach jeder Ausgabe in diesen Kanal wird das nachgeschaltete `Adapt`-Subsystem unter Kenntnis des Zielpunktes und der aktuellen Position und Ausrichtung der Plattform eine Vorzugsrichtung ermitteln, in deren Richtung eine Bewegung zur Aufklärung des nächsten unbekannt/erreichbaren Punktes erfolgen sollte. Diese Angabe wird im `Driver`-Subsystem als zusätzlicher Parameter verwertet.

Eine weitere Einflußnahme auf die Ausführung besteht seitens der deliberativen Systemkomponenten nicht; insbesondere ist es hier nicht relevant, mit welcher Präzision der an die reaktiven Systemkomponenten nicht einmal übermittelte Zielpunkt erreicht wird, oder ob dies überhaupt geschieht: Alle unmittelbaren Navigationsaufgaben – wie die Wahl einer angemessenen Bewegungsrichtung und Geschwindigkeit – liegen allein in der Kompetenz des reaktiven `Driver`-Moduls. Sollte beispielsweise ein auftretendes Hindernis ein Ausweichen erforderlich machen oder ein attraktiver Pfad eine leicht abweichende Route begünstigen, würde entweder das Teilziel auch von dort in das Sensorblickfeld geraten (was ausreichend ist, sofern das Teilziel die aufzuklärende Rasterzelle und nicht nur einen Zwischenschritt bezeichnet), oder aber eine Neuplanung erfolgen, sobald die tatsächliche Route von der geplanten um mehr als eine Rasterzelle abweicht. Die Plattform bleibt in jedem Fall in Bewegung und wird so nicht grundsätzlich, aber zumindest in vielen Fällen ein angemessenes Verhalten an den Tag legen. In den Fällen, in denen das Verhalten kurzfristig nicht optimal ist und nur eine Neuplanung eine sinnvolle Aktion ermitteln kann, garantieren die Eigenschaften des reaktiven `Driver`-Subsystems, daß die Aktionen nicht zu Kollisionen führen werden.

Das Gesamtsystem nimmt eine reaktive Exploration der Umgebung vor, während aus den aktuellen Sensordaten schrittweise eine Umgebungskarte generiert wird. Die aus der Kartierung resultierende Steuerinformation führt in gewissen Zeitabständen zur Neueinstellung einer Vorzugsrichtung

für die tatsächliche Plattformbewegung, was dazu führt, daß der gesamte erkundbare Bereich sensorisch untersucht wird. Abbildung 4.35(b) zeigt für das Testfeld die Folge der Roboterpositionen, die das Verhalten demonstrieren, das durch diesen deliberativ parametrisierten reaktiven Vorgang entsteht.

4.4.5 Ergebnisse und Bewertung

Mit den Systemprototypen zu einem gemeinsamen Testszenario wurden drei verschiedenartige Lösungen gezeigt, bei denen durch unterschiedliche Zusammenstellung von Subsystemen jeweils eine Systemarchitektur konstruiert wurde. Für eine Bewertung sind nicht die Qualität und Leistung der jeweiligen Lösungen interessant, sondern die Bedingungen der jeweiligen technischen Umsetzung. An dieser Stelle ist zunächst festzuhalten, daß zumindest eine grundsätzliche Eignung zur Formulierung bestimmter Architekturen nach reaktivem, deliberativem oder hybridem Schema nachgewiesen ist.

Aufwand. Der Hauptaufwand gegenüber der konventionellen monolithischen Entwicklung bei der nun erforderlichen Partitionierung in Systemteile und der expliziten Berücksichtigung von Kommunikationsschnittstellen. Dies impliziert, daß insbesondere die Menge der zwischen den Subsystemen zu kommunizierenden Daten und der jeweilige Zugriffsmodus (lesend, schreibend) spezifiziert werden muß. Zwar erfordert der Zugriff selbst lokal nicht mehr als einen einfachen Methodenaufruf an Stelle eines Variablenzugriffs, doch werden immerhin weitergehende Überlegungen zu einer angemessenen Subsystemschnittstelle erforderlich. Die Bestimmung und Offenlegung der Interaktion (benötigt, produziert, überlädt) durch explizite Deklaration ist aus methodischen Gründen jedoch als Gewinn zu betrachten, der den geringen Mehraufwand rechtfertigt. Für die Formulierung der Subsysteme als *thread* und die Nutzung der Architekturkonfiguration ist kein erheblicher Mehraufwand erforderlich, sofern die jeweilige Subsystemklasse von `subsys` abgeleitet wird.

Wiederverwendbarkeit. Bereits die Systemprototypen vermitteln einen Eindruck der Möglichkeiten zur Wiederverwendung ausgewählter Subsysteme im Kontext anderer Konstruktionen: Änderungen mußten vorwiegend am jeweiligen `Driver`-Subsystem vorgenommen werden, das im reaktiven Fall die eigentliche Steuerfunktionalität integrierte und für den de-

liberativen Fall wegen der grundsätzlich abweichenden Form der Eingabedaten (Zielpunkt in Koordinaten eines globalen Koordinatensystems vs. robozentrischer Sensorscan) ausgetauscht werden mußte. Die Leistung der Wiederverwendung wird deutlich beim Einsatz eines praktisch unveränderten `RobSrv`, der Verwendung des reaktiven `Driver`-Subsystems im hybriden Modell und der Integration des gänzlich unveränderten deliberativen Planers, der trotz unpassender Schnittstelle einfach für eine Nutzung im Rahmen des hybriden Prototypen adaptiert werden konnte.

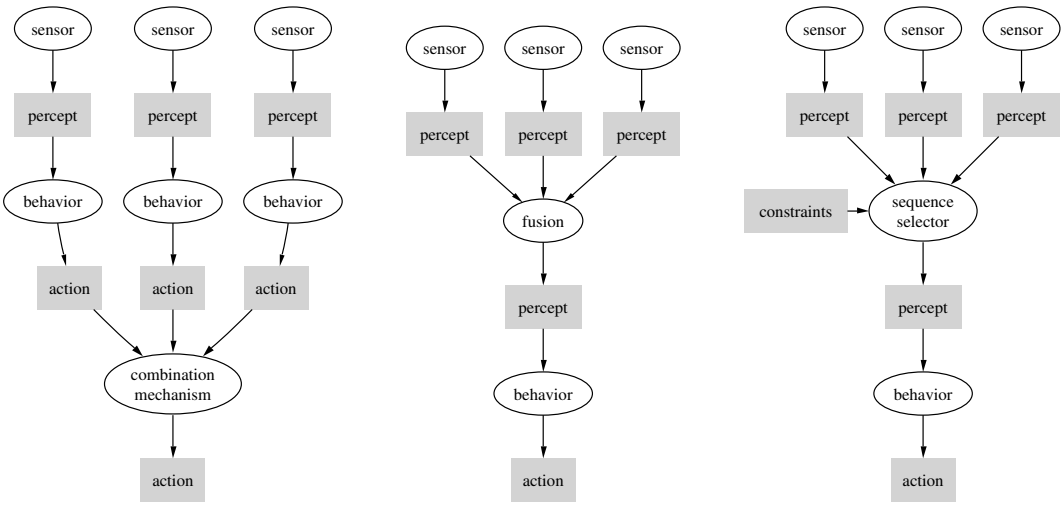
Leistung. Die Leistungseinbußen des beschriebenen Verfahrens gegenüber monolithischen Systemen liegen im Mehraufwand durch die stattfindende kanalgebundene Kommunikation begründet: Für eine Parameterübermittlung zwischen zwei Subsystemen sind zwei Kopieroperationen (in die Kanalvariable, an die endgültige Position) und die Bedienung eines Semaphore erforderlich. Zu der Notwendigkeit einer Namensauflösung des bezeichneten Kanals kommt es in der Regel nicht, da die jeweils gegenwärtige Zuordnung von Kanälen zu Subsystemen in Tabellen notiert wird, die nur unmittelbar nach strukturellen Änderungen durch Hinzuladen oder Entfernen von Subsystemen aktualisiert werden müssen. Der anteilsmäßig größte Mehraufwand ist also – abhängig vom Umfang der zu übermittelnden Daten – durch die Kopieroperation und ggf. die Umwandlung in eine zeigerfreie textuelle Zwischenrepräsentation bedingt. Für die Raten der Sensordatenerfassung, mit denen die Testplattform und das korrespondierende simulierte System arbeiten, kann die durch den Mehraufwand entstehende zusätzliche Zeitverzögerung vernachlässigt werden. Im Fall des Einsatzes in Echtzeitsystemen mit strengeren Anforderungen besteht die Möglichkeit, durch Nutzung des subsystemübergreifenden gemeinsamen Adreßraums eine durch Verzicht auf Umkopieren von Daten erheblich schnellere Kommunikation zu ermöglichen; dies ist grundsätzlich möglich, da beispielsweise auch (gültige) Zeiger zwischen den Subsystemen übermittelt werden können. Bei Nutzung dieser Option kann die Metaarchitektur allerdings keine Sicherung der Kommunikation (z.B. durch *locking*) vornehmen.

Aufgabenabstraktion und Designalternativen. Für die Demonstratorentwürfe wurde an keiner Stelle ein im herkömmlichen Sinne aufgabenspezifisches Subsystem verwendet. Die faktische Granularität des Systems entwickelte sich aus einer intuitiven Partitionierung des Gesamtsystems in Teilaufgaben und wäre durchaus anders vorstellbar, beispielsweise durch Verfeinerung des `Mapper`s in einen passiven kartenführenden Teil, der über

ein bidirektionales Protokoll unter Verwendung von zwei Kanälen abgefragt werden kann, und einen aktiven Teil, der eine `Waypoint`-Angabe für die zweckmäßige Fortsetzung der Exploration generiert. Da ein funktionaler Mehrwert – einmal abgesehen davon, daß der `Mapper` um diese Abfragemöglichkeit erweitert werden müßte, um ihn allgemein einsetzen zu können – im Rahmen des Testszenarios nicht zu erwarten war, wurde diese Verfeinerung zur Erhaltung der Übersichtlichkeit des Beispiels nicht vorgenommen.

Offene Punkte. Nicht alle funktionierenden Designalternativen sind sinnvoll. Im Fall der Entwicklung des hybriden Systems bestand an der Stelle der Transformation zwischen abstrakten Planungsdaten (globale Koordinaten des Zielpunktes) und der Angabe einer robozentrischen Vorzugsrichtung technisch die Option auf ein Überladen der `Mapper`-Ausgabe, um als Ausgabe einen robozentrisch angegebenen Zielpunkt zu generieren, der zur Parametrisierung von `Driver` hätte verwendet werden können. Diese Konfiguration wurde – obgleich grundsätzlich funktionsfähig – nicht gewählt, da ein wenn auch äußerst unwahrscheinlicher Ausfall des `Adapt`-Subsystems dazu geführt hätte, daß dem `Driver` syntaktisch, aber nicht semantisch verständliche (und damit irreführende) Informationen übermittelt würden. Die Verantwortung für das Vornehmen eines angemessenen Überladens kann dem Entwickler also nicht durch eine strukturelle Vorgabe abgenommen werden.

Unterstützung bestimmter Strukturelemente. Im begrenzten Umfeld der Prototypen treten einige Strukturelemente der in Kap. 2 aufgeführten Architekturen nicht auf. Die meisten von ihnen können auf einfache Weise auf die Subsystemstruktur abgebildet werden, z.B. der *arbiter* oder die vektorielle Summation. Weniger offensichtlich ist das Verfahren bei den verschiedenen Strategien der Sensorfusion; Abbildung 4.36 zeigt die entsprechenden Subsystem-Teilarchitekturen für *sensor fission*, *fusion* und *fashion*. Das einzige weitere Strukturelement, das nur vergleichsweise unelegant umgesetzt werden kann, ist das *blackboard*-artige LPS (vgl. 2.1.4.3) des hybriden modellorientierten Architekturstils, da hier den Kommunikationspartnern vorab keine Rollenzuweisung als Produzent oder Konsument zugeschrieben wird. Während Schreibzugriffe gemeinsam über nur einen Kanal vorgenommen werden könnten, wären Lesezugriffe auf einzelne Elemente des Modells nur über ein Abrufprotokoll mit erheblichem Mehraufwand möglich. Von diesem explizit ungeregelten Kommunikationsansatz einmal ab-



(a) *sensor fission*: Kombination der Steuerdaten

(b) *sensor fusion*: Kombination der Sensordaten

(c) *sensor fashion* (nach ARKIN)

Abb. 4.36: Abbildung von Sensorfusionsstrategien nach [Mur00, S. 201] auf Subsystem-/Kanalarchitekturen: die Abbildungen zeigen jeweils den Architekturausschnitt von den Sensorsubsystemen bis zur Generierung einer Aktion. Die Verwendung signaturverschiedener Kanäle erlaubt direkt eine genaue Nachbildung jeder vorgegebenen Teilarchitektur.

gesehen läßt sich allerdings festhalten, daß eine Umsetzung der üblichen Systemstrukturen auf das vorgeschlagene Modell vorzüglich gelingt.

4.5 Zusammenfassung und Bewertung

Im vorgestellten Verfahren wurde eine Aufteilung der Entwurfsschritte auf vier verschiedene Ebenen des Entwurfs vorgeschlagen. Diese Ebenen sind in Anordnung steigender Komplexität mit der Hardwarestruktur und der systeminternen Kommunikation (4.1), der Geräte- und Protokollabstraktion (4.2), der Bildung von Subsystemen (4.3) und schließlich der Bildung von Architekturen unter Verwendung der Subsysteme befaßt (4.4).

Leistung der Schichten. Auf der untersten softwaremäßig implementierten Schicht (s. Abb. 4.2 auf S. 98) dominiert die verfügbare Hardware und deren technische Integration den Entwurfsprozeß nahezu vollständig: An dieser Stelle besteht wegen der technischen Randbedingungen der geringste Freiraum für alternative Entwurfsentscheidungen. Das Ziel der Abstraktion von den Gegebenheiten des technischen Aufbaus konnte durch Einführung von Serverprozessen erreicht werden, die über ein gerätespezifisches Protokoll den Zugriff auf das jeweilige Aggregat ungeachtet seines Ortes und der technischen Schnittstellenanbindung gestatten. Diese Maßnahme ist für Einzelkonstruktionen zwar nur von untergeordneter Bedeutung, gestattet allerdings durch die Abstraktion vom strukturellen Hardwareaufbau sowohl das transparente Vornehmen technischer Änderungen am System als auch die Formation einer äquivalenten Schnittstelle bei verschiedenartigen Systemen und ist so als Schritt zur Entwurfssystematisierung zu begreifen.

Die Entscheidung, auf der untersten Ebene nach Möglichkeit eine Weiterleitung des unveränderten Herstellerprotokolls vorzunehmen hat zur Folge, daß prinzipiell alle Gerätefunktionalität zur Nutzung durch höhere Schichten erhalten bleibt, ohne daß eine entsprechend umfangreiche Softwareschnittstelle entworfen werden muß. Eine Abstraktion von den eigentlichen Geräteprotokollen ist allerdings erforderlich, um dennoch eine Generizität über verschiedene Ausprägungen eines Sensortyps (verschiedene ToF-Sensoren, unterschiedliche Plattformen) erreichen zu können. Dies ist eine Aufgabe der Protokollabstraktionsschicht. Die Verwendung objektorientierter Techniken wie der Einführung einer Vererbungshierarchie gestattet, gestuft detaillierter werdende Instanzen für Hardware aus *proxy*-Klassen zu

generieren. Diese ermöglichen einen Zugang zu den unspezifischen Fähigkeiten eines Sensors, der für einen abstrakten und hochgradig portablen Entwurf oftmals ausreichend ist, bis hin zu einem Zugang zu möglicherweise benötigten gerätespezifischen Details. Eine weitere Aufgabe dieser Schicht ist die Umsetzung des durch technische Notwendigkeit asynchronen Geräteprotokolls und die Kapselung der Protokollautomaten, um eine vereinfachte Aufrufschnittstelle zu erhalten, die der üblichen (d.h. nicht ereignisgesteuerten) Softwareentwicklung angemessener ist.

Nach Schaffung von gerätebezogenen Dienstzugangspunkten steht auf der folgenden Ebene die Implementierung von Teilsystemen im Vordergrund. Diese Teilsysteme dienen der Kapselung von benötigten "Fähigkeiten" des Systems, der Codierung von Lösungen für Teilaspekte von Anwendungsproblemen und der Berücksichtigung der konkreten Zusammenstellung der abstrakten Roboterkomponenten: Zum einen weisen alle Sensoren und Effektoren in Bezug auf die Gesamtplattform eine Position und Ausrichtung auf (die als roboterspezifische, aber nicht gerätebezogene Parameter in jegliche Verwendung einzugehen hat), zum anderen kann gerade aus der (ebenfalls roboterspezifischen) Präsenz von zwei oder mehr technischen Systemen oft ein Mehrwert gezogen werden. Exemplarisch formulierte Subsysteme demonstrieren sowohl die Möglichkeit als auch den erreichbaren Abstraktionsgrad. Es liegt auf der Hand, daß die Menge der Subsysteme nicht nur bei Hinzufügen von Hardwarekomponenten, die Systemfähigkeiten vermitteln sollen, erweitert werden muß, sondern daß auch die Dekomposition von Anwendungsproblemen zur Integration von Teillösungen in weiteren Subsystemen führen wird. Als Leitlinien für die Findung einer sinnvollen Granularität dienen hier sowohl die intuitive Problemdekomposition wie die angestrebte Wiederverwendbarkeit möglichst allgemeiner Subsysteme; für diese Entscheidung kann kein formales Verfahren bereitgestellt werden.

Für die Zusammenstellung eines Gesamtsystems entsprechend einer konkreten und aus einem Anwendungsproblem hervorgehenden Aufgabenstellung wurde die letzte Schicht konzipiert. Wegen der üblichen technischen Randbedingungen wurde zur Entkoppelung der separat entworfenen Subsysteme eine Ausführung in eigenen *threads* vorgesehen. Zusammen mit der Struktur zur Kopplung der Subsysteme, die zum Austausch von Informationen erforderlich ist, entsteht so zur Laufzeit eine Graphenstruktur. Grundsätzlich wurde die Idee, Strukturen einer Architektur mit Hilfe eines gerichteten Graphen zu formulieren bereits in [MA92] verwendet. In der genannten Arbeit, die die hybride Architektur SFX vorstellt, wird diese Technik allerdings allein zu Zwecken der Sensorfusion (*sensor plan*-Graph)

eingesetzt; die Verbindungen der Teilsysteme können zudem nur manuell und explizit geknüpft werden. Im mit dieser Arbeit vorliegenden Fall wurde ein abweichender Ansatz gewählt: Zum einen wird das vollständige System auf diese Weise geknüpft, zum anderen wird die Verknüpfung automatisch anhand der Kanalbezeichnungen vorgenommen. An dieser Stelle sind auch $m : n$-Verknüpfungen zulässig; selbst die Sonderfälle $m = 0$ oder $n = 0$ führen zu einem ablauffähigen, wenn auch in der Leistung eingeschränkten System. Auch im Sinne einer erhöhten Zuverlässigkeit wird die Zuordnung dynamisch zur Laufzeit vorgenommen, sodaß ein Nach- oder Neuladen von Subsystemen selbständig zur Integration in das Gesamtsystem führt. Ein Mechanismus zum "Überladen" von Kanälen erlaubt dabei, nachträgliche Änderungen der entstehenden Struktur vorzunehmen.

Verifikation. Eine separate Verifikation der Funktionalität der untersten Schicht zur Kommunikationsabstraktion wurde wegen des ausgedehnten Einsatzes der Experimentierplattformen in Forschung und Lehre hier nicht erneut unternommen. Der Nachweis der gelungenen Protokollabstraktion ist durch die grundsätzliche Austauschbarkeit des jeweils angesteuerten Zielsystems gegen eine andere Plattform, einen aus einem anderen Projekt hervorgehenden visuellen Simulator [BBBM01] oder die in 4.4.4 verwendeten Simulationsmodelle belegt. Im praktischen Einsatz erscheint allerdings die Restriktion, daß bei Verwendung von Methoden der spezialisierten *proxy*-Klassen (Blattknoten in Abbildung 4.5) eine Austauschbarkeit nur bei Vorhandensein eines gleichartigen Geräts bestehen kann. Eine weitgehende Hardwareabstraktion findet zudem auf der Subsystemebene statt. Die separate Erstellbarkeit von plattform- und teilproblembezogenen Subsystemen wird durch die Entwicklung exemplarischer Vertreter zur Bearbeitung von Aspekten von Anwendungsproblemen belegt; als weitere Arbeit befaßt sich [Küp03] mit einem speziellen dieser Teilprobleme. Um die Möglichkeit zur Erstellung von Systemen entsprechend der Standardarchitekturen mit Hilfe dieser Ausdrucksmittel zu verifizieren wurde für ein Testszenario jeweils ein reaktiver, ein deliberativer und ein hybrider Prototyp erstellt und auf korrekte Funktion getestet. Da sich alle klassischen Entwurfsmuster mit nur geringem Aufwand nachbilden lassen, ist mit dieser Maßnahme die äquivalente Mächtigkeit des Ansatzes zu den klassischen Entwurfsmitteln demonstriert.

Neuartige Leistungsmerkmale. Die Fähigkeit, klassische Architekturentwürfe mit neuen strukturellen Mitteln nachvollziehen zu können, macht für sich allerdings noch keinen besonderen Wert aus. Eine erhebliche (aber

grundsätzlich nur graduelle) Verbesserung im Sinne der Problemstellung wurde auf dem Gebiet der Abstraktion von gegebenen Randbedingungen erzielt. Diese Erhöhung des jeweils erreichbaren Abstraktionsgrades zieht sich durch alle Schichten: vom Hardwareaufbau über die Geräte bis zum getrennten Entwurf separater Subsysteme. Dies ist aus methodischen Gründen bereits zu begrüßen, stellt jedoch noch nicht die wesentliche Differenz des vorgelegten Ansatzes dar, dessen Leistung über die Möglichkeit zur Nachbildung üblicher Architekturen deutlich hinausgeht.

Neuartig ist hingegen der Ansatz zur Wiederverwendbarkeit von Lösungselementen, und dies aus zwei verschiedenen Gründen.

Erstens gestattet die indirekte Kommunikation mit Geräten über möglichst allgemeine Basisklassen von Sensoren und Effektoren einen weitgehend transparenten Austausch von Hardwareelementen (während über Nutzung der detaillierten abgeleiteten Klasse im Spezialfall eine Nutzung des vollen hardwarespezifischen Leistungsumfangs unter Verlust der Portabilität immer noch möglich ist). Zweitens kann durch Nutzung des Protokolls zwischen Subsystemen auf einfache Weise eine Interaktion verschiedenartiger Softwaremodule erreicht werden, die sich überhaupt nicht mehr auf eine bestimmte Hardware beziehen müssen. Diese Möglichkeit zur Modularisierung stellt zudem eine strukturelle Grundlage für separate Entwicklung und Testbarkeit dar.

Eine besondere Stärke des Ansatzes ist die Möglichkeit zur dynamischen Konfiguration, um Systemansätze erweitern und modifizieren zu können: beispielsweise kann selbst durch Nachrüstung entstehende sensorische Redundanz ohne deren explizite Berücksichtigung und ohne Änderungen an einer bestehenden Subsystemkonfiguration genutzt werden, indem auf einfache Weise zusätzliche Produzenten mit einem bereits bestehenden Kanal verbunden werden. In ähnlicher Weise kann auch die Ausfalltoleranz erhöht werden, da wegen der Trennung der Subsysteme durch Kanäle seitens der konsumierenden Subsysteme keine Annahmen über die den Kanal versorgenden Produzenten gemacht werden müssen. Die Inaktivierung eines Produzenten hat im Fall der redundanten Versorgung des Kanals nur die Konsequenz einer graduellen Leistungsverschlechterung, und selbst der Ausfall des letzten Produzenten eines Kanals hinterläßt ein im restlichen Gefüge funktionsfähiges System, das – geschicktes Design vorausgesetzt – eine ausreichende Überlebenswahrscheinlichkeit besitzen kann. Die Vernetzung ist zudem insofern selbstreparierend, als daß das Wiedererscheinen eines z.B. nach einem Ausfall neu gestarteten Subsystems ohne weitere Maßnahmen zur selbsttätigen Integration in das System führt. Dies trägt

insbesondere dem durchaus als experimentell zu bezeichnenden Charakter üblicher Roboteraufbauten Rechnung.

Ein letzter Aspekt betrifft die neu geschaffene Option, nachträgliche Änderung an bestehenden Architekturen vornehmen zu können, ohne daß Eingriffe in den Quelltext der Subsysteme vorgenommen werden müssen: Durch das Überladen von Kanälen ist es möglich, zu einem späterem Zeitpunkt an nahezu jeder Stelle sowohl erkannte systematische Fehler zu korrigieren als auch zwecks Leistungssteigerung zusätzliche Information einfließen zu lassen. Diese Eigenschaft verbessert die Anpaßbarkeit einer Architektur an neue Gegebenheiten erheblich.

Kapitel 5

Abschluß

Ausgangspunkt. Die vorgefundene Situation in der Robotik unter speziell der Berücksichtigung autonomer mobiler Systeme stellt sich ausgesprochen inhomogen dar. Weder die hardwarebezogene Robotertechnik noch die softwarebezogene Architekturkonstruktion scheinen derzeit rasch zu einer stabilen Position zu konvergieren. Dies wäre ohnehin untypisch für eine vergleichsweise junge Disziplin. Auch das Feld der zu bearbeitenden Anwendungsaufgaben wird noch vorwiegend von der erreichbaren faktischen Systemleistung beschränkt, und nicht etwa von den Wünschen und Ideen für den Einsatz von Robotern. In den letzten Jahrzehnten wurden grundverschiedene Entwürfe für die Systemstruktur mobiler Roboter vorgelegt und unter Bezugnahme auf jeweils bestimmte Anwendungsprobleme teils ausgesprochen kontrovers diskutiert. Neben der Entwicklung der *Systemstruktur* (die vorwiegend in der Architekturdebatte vorangetrieben wird) macht die Entwicklung von *Verfahren* einen Schwerpunkt aus, der wesentlichen Einfluß auf die erreichbare Systemleistung robotischer Konstruktionen besitzt. Von diesem Ausgangspunkt betrachtet kann festgestellt werden, daß die Antwort auf die auftretenden Fragen nach Verallgemeinerung und Systematisierung, Erhöhung der Systemleistung und Reichweite der Problemlösungskapazität nicht allein in der Vorlage einer weiteren Systemarchitektur bestehen kann.

Ziel und Lösung. Entsprechend der vorgefundenen Situation ist das allgemeine Ziel, einen systematischen Entwurf vorzulegen, der in weitestgehender Unabhängigkeit von technischen Randbedingungen verschiedener Plattformen gestattet, auch nachträglich neue Verfahren zu integrieren und mit ihnen praktisch beliebige Strukturen für problemangemessene System-

architekturen aufzubauen. Um den gewünschten höheren Grad an Allgemeinheit erreichen und überhaupt eine Portabilität von aufwendigen Verfahrensimplementierungen erzielen zu können, muß nicht nur eine Trennung zwischen der technischen Basis und der Ebene der Verfahren, sondern auch zwischen den Verfahren und der umfassenden Systemstruktur erreicht werden. Weiterhin ist die Eliminierung jeder unerwünschten wechselseitigen Beeinflussung der Verfahren nötig.

Welche Architektur – und welche Auswahl von Verfahren – in der Lösung eines zur Konstruktionszeit eines solchen Systemrahmens noch nicht bestimmten Anwendungsproblems Verwendung finden werden, kann aus offensichtlichen Gründen nicht vorhergesehen werden. Dabei ist nicht die grundsätzliche Mächtigkeit einer Architektur die entscheidende Frage, sondern es sind die der *Angemessenheit*, der *Sparsamkeit* und der *Eleganz* des Entwurfs. Diese drei Begriffe können zwar auch als praktisch synonym angesehen werden, sollen sich hier jedoch auf die Aspekte der effektiven System*fähigkeit*, der durch Hardware und eingesetzte Verfahren determinierten *Leistung* und die *Adaptierbarkeit* auch an veränderte Einsatz- und Aufgabenbedingungen beziehen. Selbst die hybriden Systeme, die grundsätzlich das größte Problemlösungspotential besitzen, sind oft schwer zu einer bestimmten Leistung zu bewegen, da die deliberativen symbolischen Planungsvorgaben einer hinreichend komplexen reaktiven Ebene schwer verständlich zu machen sind. Mitunter stellen also auch diese Systeme keine effiziente Lösung dar.

Unter diesen Bedingungen der Variabilität von Hardwarebasis und Softwaresuperstruktur (d.h. der Architektur) besteht das Ziel dieser Arbeit darin, genau die Strukturierungsmittel bereitzustellen, die erfolgreich zum Aufbau rekonfigurierbarer Systemarchitekturen eingesetzt werden können. Die vorgestellte Lösung sieht zunächst eine konventionelle Abstraktionsschicht vor, die von den konstruktionsbedingten und durch den Hardwareaufbau vermittelten technischen Randbedingungen verallgemeinern läßt. Eine Reihe von Serverprozessen vermitteln zwischen dem jeweiligen Gerätetreiber und einem Zugangspunkt, der eine Verwendung der Roboteraggregate ohne Rücksicht auf die gerätespezifische Schnittstellenanbindung zuläßt. Nach dieser Abstraktion von der vorwiegend durch die Zahl und Eigenschaften der vorhandenen Schnittstellen bestimmten akzidentellen Hardwarestruktur wird auf einer weiteren Ebene eine Protokollabstraktion vorgenommen. Hier findet eine Kapselung der gerätespezifischen und unter Umständen komplexen Protokollautomaten statt, um eine Bedienung durch eine weitgehend zustandslose und damit einfache Aufrufschnittstelle zu ermöglichen. Geräte werden hier softwaremäßig über Hardware-*proxy*-

Klassen einer speziellen **Bibliothek** abgebildet, die entsprechende Abfrage- und Manipulationsmethoden zur Verfügung stellen. Die Verwendung einer durch Vererbungsmechanismen gebildeten Gerätehierarchie gestattet dabei oft die Implementierung – wenigstens jedoch die Schnittstellenspezifikation – von abstrakten Methoden und Eigenschaften, die allen abgeleiteten Geräteklassen zukommen. Auf diese Weise kann zum Zweck der Verallgemeinerung durch Verwendung der wurzelnahen Klassen eine weitgehende Abstraktion von der konkreten Hardware erreicht werden, während für besondere Anforderungen weiterhin die spezialisierten (und nicht portablen) Methoden der abgeleiteten, dann zunehmend gerätespezifischer werdenden Klassen zur Verfügung stehen. Diese Geräteklassen dienen also der Präsentation von weitgehend abstrakten "Fähigkeiten" einzelner technischer Systemkomponenten. Die auf einen Aspekt einer Problemlösung gerichtete Verwendung von Komponenten zu einem bestimmten Zweck ist allerdings nicht mehr in dieser kanonischen Form lösbar, da der Einsatz mit dem konkreten *Zweck* als auch mit Details der jeweiligen Roboter*struktur* (z.B. der Präsenz mehrerer Aggregate oder ihrer Montageposition) verbunden ist. Diese komplexen Systemfähigkeiten, die oft mit Parametern oder internen Zuständen verknüpft sein können, werden auf der Ebene der sog. **Subsysteme** implementiert. Subsysteme dienen im reaktiven Fall der Transformation einer Eingabe in eine Ausgabe; beispielsweise von Sensordaten oder Ausgaben eines anderen Subsystems in Kommandos zur Effektorsteuerung oder Eingaben eines weiteren Subsystems. Um eine unbeabsichtigte wechselseitige Beeinflussung der als unabhängig voneinander betrachteten Subsysteme zu vermeiden, werden diese in separaten Einheiten der Ausführung konkurrent bearbeitet und können nur über eine spezielle Kommunikationsschnittstelle Daten austauschen und so interagieren. Da wegen der Beteiligung von vielfältiger experimenteller Hardware und möglicherweise auch unzuverlässiger Subsysteme auch mit temporären wie permanenten Ausfällen von Subsystemen zu rechnen ist, muß die entwickelte Kommunikationsschnittstelle dieser Eigenschaft besondere Rechnung tragen, um ein Blockieren der nicht vom Fehler unmittelbar betroffenen Subsysteme auszuschließen. Auch aus diesem Grunde wurde eine dynamische Strukturaggregation innerhalb einer **Metaarchitektur** vorgesehen, statt wie in konventionellen Lösungen die Verarbeitungsstruktur statisch als Graph zu spezifizieren. Die dynamische und automatische Herstellung der Verknüpfung von einzelnen Subsystemen anhand der von ihnen verwendeten Signaturen ihrer Kommunikationskanäle gestattet auf einfache Weise, die in bislang vorgelegten Architekturschemata verwendeten Kommunikationsmodelle elegant abzubilden. Dabei kommen allen so spezifizierten Architekturen immanent die zusätzlichen Eigenschaften von sowohl Ausfalltoleranz als auch

Erweiterbarkeit zu. Darüber hinaus sichert eine weitere Eigenschaft der Metaarchitektur, die Option zum Überladen von Kanalinformation durch später hinzugefügte Subsysteme, die strukturelle Anpaßbarkeit bereits formulierter Architekturschemata und die Austauschbarkeit von Subsystemen gegen erweiterte oder neue Verfahren selbst dann, wenn kein Eingriff in bestehende komplexe Systemstrukturen vorgenommen werden kann.

Methode der Evaluation. Den beteiligten Entwurfsebenen entsprechend muß eine Evaluation ebenfalls auf verschiedenen Ebenen erfolgen. Im Fall der unteren, vorwiegend hardwarebezogenen Schichten wurden über den Lauf der Entwicklung und des Einsatzes konventionelle Funktionstests vorgenommen, deren Methodik im hier behandelten Kontext allerdings nicht von besonderem Interesse sein wird. Für die erforderliche Verifikation und Bewertung der auf der Ebene einzelner Subsysteme erzielten Resultate sei auf die entsprechenden Einzelveröffentlichungen (und natürlich Abschnitt 4.3) verwiesen.

Einen Sonderfall stellt die Evaluation des Entwurfs von Strukturen allgemeiner Systeme dar. Naturgemäß entzieht sich dieser gerade in Ermangelung vergleichbarer Ansätze jeder sinnvollen quantitativen Analyse, da eine solche Auswertung stets auf spezielle Fälle der Anwendung beschränkt bleiben müßte (von denen, wie angedeutet, beliebig viele existieren). Aus diesem Grunde muß die Bewertung hier qualitativen Kriterien folgen. Da die drei Basisarchitekturschemata derzeit jeweils für gewisse spezielle Probleme als beste verfügbare Lösung erachtet werden, muß zumindest sichergestellt sein, daß eine angemessene – d.h. leicht erhältliche und performant funktionierende – Umsetzung der korrespondierenden Architekturentwürfe innerhalb des vorgestellten Systems möglich ist. Ein Gelingen dieser Umsetzung, also die strukturelle Subsumption der klassischen Vertreter von Robotersystemarchitekturen, wäre bereits als Fortschritt zu vermerken, da hierbei für alle Modelle immerhin eine einheitliche Grundlage verwendet würde, die (geeignete Programmierung selbstverständlich vorausgesetzt) einen transparenten Austausch der technischen Plattform gegen ein äquivalentes Modell zuließe. Ein weiteres Testkriterium stellt der ebenfalls nur qualitativ sinnvoll faßbare Aufwand für spätere Designänderungen der auf diese Weise entwickelten Systeme dar, wobei besonders der dabei vorhandene Freiraum, aber auch der Grad der Wiederverwendbarkeit von Einzelkomponenten von Bedeutung ist.

Um eine Evaluation vorzunehmen, wurde für ein Standardproblem, die

Umgebungsexploration,[1] jeweils eine reaktive, deliberative bzw. hybride Lösung unter Nutzung des in dieser Arbeit entwickelten Lösungsrahmens formuliert. Dies sichert zunächst, daß die vorgestellte Struktur der Metaarchitektur in der Lage ist, tatsächlich eine Subsumption gängiger Lösungsansätze vorzunehmen. Nur exemplarisch kann auf diese Weise belegt werden, daß bei der Konstruktion kein unüblicher Aufwand getrieben werden muß, und daß insbesondere keine ungewöhnlichen aus dem Metaarchitekturkonzept erwachsenden neuen Randbedingungen beim Entwurf berücksichtigt werden müssen. Die Wahl eines für alle Entwürfe gleichen Anwendungsproblems gestattet weiterhin, Untersuchungen über die strukturelle Adaptierbarkeit von Architekturentwürfen anzustellen. Zur Illustration der Flexibilität der Entwürfe wird jeweils der reaktive Entwurf als Grundlage zur Entwicklung des deliberativen und dieser im nächsten Schritt als Basis des hybriden Systems herangezogen. Neben der so demonstrierten Adaptierbarkeit der Struktur wird auch die Anpaßbarkeit von Subsystemen durch Erweiterung bzw. Einsatz in anderem Kontext für eine spätere Wiederverwendung gezeigt.

Beitrag der Arbeit. In dieser Arbeit wird ein Schema präsentiert, das die Konstruktion von Robotersystemen auf eine systematische Weise gestattet. Der Fokus liegt dabei nicht auf der Entwicklung eines zu einer bestimmten Aufgabe optimalen Systems, sondern auf der eines möglichst universellen und leicht adaptierbaren Systemaufbaus, um eine Vielzahl von zum Zeitpunkt der Systemkonstruktion noch nicht festgelegten Anwendungsaufgaben möglichst erfolgreich bearbeiten zu können. Durch weitestmögliche Abstraktion sowohl von der zur Verfügung stehenden Hardware als auch von durch einen konkreten Anwendungskontext vorgegebenen Aufgabenstellungen wird der aus beiden Richtungen kommende Einfluß soweit minimiert, sodaß allgemeine und portable Systementwürfe konstruiert werden können. Der Vorgang der Spezifikation eines zur Bearbeitung einer Anwendungsaufgabe vorgesehenen Systems geschieht dabei durch Zusammenstellung und Vernetzung einer Reihe von zu diesem Zweck selektierten Subsystemen, die einzeln zur Bearbeitung von bestimmten Teilaspekten der Anwendungsprobleme eingesetzt werden. Eine als Metaarchitektur bezeichnete probleminvariante Softwaresuperstruktur dient dabei der anwendungsproblemvarianten Kombination von und Kommunikation zwischen den einzelnen Subsystemen.

[1] Dieses Problem ist absichtsvoll unterspezifiziert gestellt; es handelt sich somit nicht um ein Problem, das zu einem bestimmten Architekturparadigma spezifisch gestellt wäre.

Die vorgelegte Lösung gestattet die Subsumption der klassischen Architekturentwürfe auf einer einheitlichen Grundlage, ohne dabei besonderen Aufwand zu erfordern oder der Konstruktion ungewöhnliche Randbedingungen aufzuerlegen. Die Möglichkeit zur Abstraktion von allen nicht wesentlichen Gegebenheiten der technischen Plattform ermöglicht eine einfache Übertragbarkeit von Problemlösungen auch auf andere Plattformen. Die Formulierung von spezifischen Aspekten der Lösung in separaten Subsystemen, die allein über explizit erfolgende Kommunikation interagieren, unterstützt insbesondere Testbarkeit und Wiederverwendung der eingesetzten Verfahrensimplementierungen auf elegante Weise. Die Lösung zeichnet sich darüber hinaus dadurch aus, daß eine einfache Änder- und Erweiterbarkeit vorhandener Systeme durch die Integration weiterer Lösungskomponenten, den Austausch verwendeter Subsysteme und die transparente Erweiterung von Art und Umfang der internen Kommunikation möglich ist. Damit können Systeme mit äußerst geringem Aufwand so adaptiert werden, daß sie einer Konfigurationsänderung der Hardwareplattform gerecht werden oder in einem veränderten Kontext einsetzbar sind. Neben der so erzielten Universalität und Orthogonalität des Entwurfs ist neuartig, daß die Verbindung der Subsysteme vollständig dynamisch zur Laufzeit erfolgt und auch Veränderungen unterworfen werden kann, während das System sich im Einsatz befindet. Auf diese Weise können jederzeit sowohl eine dynamische Rekonfiguration vorgenommen werden als auch Hardwareausfällen sowie -erweiterungen einfach Rechnung getragen werden.

Ausblick. Der Entwurf spezialisierter Systeme zur Bearbeitung bestimmter, einzelner Aufgaben in einem reduzierten Anwendungskontext besitzt offensichtliche Berechtigung, sofern die Laufzeitperformanz das ausschlaggebende Kriterium bei der Entwicklung ist. Aus ähnlichen Gründen kann auch kein bestimmter Architekturentwurf allein als vertretbare Lösung aller Anwendungsprobleme – auch der noch nicht spezifizierten – angeführt werden, wie innovativ er auch sei. Der vorgelegte Entwurf stellt allerdings eine Abstraktion von den Einflüssen der Plattform wie von der konkreten Anwendungsaufgabe (und der durch sie implizit bestimmten Idealarchitektur) zur Verfügung, die wenigstens im üblichen Fall keinen wesentlichen Verlust an Performanz mit sich bringt. Auf diese Weise ist der einfachen Wiederverwendung nicht nur von Verfahren, sondern auch von Implementierungen erstmals eine zweckmäßige Möglichkeit eröffnet. Die dynamische Rekonfiguration innerhalb einer Metaarchitektur läßt zudem den eleganten Entwurf von Systemen und ihre einfache Adaption an veränderte Verhält-

nisse zu. Dies ist ein wesentlicher Schritt in Richtung der mittelfristig zu erwartenden Entwicklung allgemeiner, nicht mehr aufgabengebundener Systeme. Als offensichtliche Weiterentwicklungsmöglichkeit ist die automatisierte Distribution von Subsystemen auf verschiedene Rechner zu nennen: Da die einzelnen Subsysteme ohnehin bereits unabhängig voneinander ablaufen und keine weitere Synchronisation erforderlich ist, sind die Grundlagen hier bereits gelegt. Mit besonderem Aufwand ist hierbei allerdings zu rechnen, sofern eine optimale automatische Systempartitionierung entsprechend Kommunikationsvolumen und noch tolerierter Verzögerung vorgenommen werden sollte. Die Möglichkeit zur dynamischen Instantiierung von Subsystemen und ihrer automatischen Einbettung in den Systemkontext gestattet weiterhin, die Systemadaption teilweise oder vollständig programmgesteuert vorzunehmen. Dies schließt die Möglichkeit zur selbständigen adaptiven Architekturformierung ein, sofern softwaregesteuert eine geeignete Problemanalyse, -partitionierung und anschließende Leistungsbewertung vorgenommen werden kann. Die limitierenden Faktoren für solche Ansätze liegen allerdings nicht innerhalb der Kernrobotik, sondern in den von ihr berührten Disziplinen begründet.

Literatur- und Quellenverzeichnis

[AB97] ARKIN, Ronald C. ; BALCH, Tucker R.: AuRA: principles and practice in review. In: *Journal of experimental & theoretical artificial intelligence (JETAI)* 9 (1997), Nr. 2-3, S. 175–189

[AC89] AGRE, Philip E. ; CHAPMAN, David. *What are plans for?* A.I. Memo 1050a. October 1989

[ACF+98] ALAMI, R. ; CHATILA, R. ; FLEURY, S. ; GHALLAB, M. ; INGRAND, F.: An Architecture for Autonomy. In: *International Journal of Robotics Research* 17 (1998), April, Nr. 4, S. 315–337

[Act00] ACTIVMEDIA ROBOTICS LLC: *Pioneer 2 Mobile Robots with Pioneer 2 Operating System Servers: Operations Manual*. Manual v5. : ActivMedia Inc., 2000

[AHT98] ALVAREZ, C. G. ; HASHIMOTO, Yukio ; TSUCHIYA, Takeshi: Ultrasonic Range Sensor with Phase Detection And Variable Transmission Frequency. In: [KWS98a], S. 2–5

[Alb81] ALBUS, James S.: *Brains, Behaviour, and Robotics*. Byte Books, Peterborough, N.H., 1981

[Alb89] ALBUS, James S.: A Control Architecture for Cooperative Intelligent Robots. In: [DSA89], S. 713–744

[AM99] AMIR, Eyal ; MAYNARD-REID II, Pedrito: Logic-Based Subsumption Architecture. In: *Proc. 6th Intl. Joint Conf. on Artificial Intelligence (IJCAI-99)*, 1999, S. 147–152

[ARH87] ARKIN, Ronald C. ; RISEMAN, Edard M. ; HANSON, Allan R.: AuRA: An Architecture for Vision-Based Robot Navigation. In: *Proc. DARPA Image Understanding Workshop*, 1987

[Ark93] ARKIN, Ronald C.: Survivable robotic systems: Reactive and homeostatic control. In: JAMSHIDA, M. (Hrsg.) ; EICKER, P. (Hrsg.): *Robotics and remote systems for hazardous environments* Bd. 1. Englewood Cliffs, New Jersey : Prentice Hall, 1993, S. 135–154

[Ark95] ARKIN, Ronald C. *Just what is a robot architecture anyway? Turing equivalency versus organizing principles*. 1995

[AV90] ARKIN, Ronald C. ; VACHTSEVANOS, George: Techniques for Robot Survivability. In: *Proc. 3rd International Symposium on Robotics and Manufacturing*. Vancouver, BC, 1990, S. 383–388

[Bab00] BABOS, Martin. *Untersuchung von Verfahren zur einfachen, robusten Kalibrierung eines Stereokamerakopfes*. Universität Hamburg, Fachbereich Informatik, Studienarbeit. 2000

[BB01] BUNGENSTOCK, Michael ; BAUDRY, Andreas: *Entwicklung einer dreidimensionalen VR- und Simulationsumgebung für mobile Roboter und ihre Sensorikkomponenten*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 2001

[BBBM01] BUNGENSTOCK, Michael ; BAUDRY, Andreas ; BITTERLING, Johannes ; MERTSCHING, Bärbel: Development of a Simulation Framework for Mobile Robots. In: [GS01], S. 89–92

[BBE+98] BLUM, Stefan ; BURSCHKA, Darius ; EBERST, Christof ; EINSELE, Tobias ; HAUCK, Alexa ; STÖFFLER, Norbert O. ; FÄRBER, Georg: Autonome Exploration von Innenräumen mit der Multisensorik-Plattform MARVIN. In: [WDH98], S. 138–147

[BBGG99] BARTHEL, R. ; BESCHNIDT, J. ; GERN, T. ; GILLES, E.D.: Konfliktanalyse auf Binnenwasserstraßen. In: [SHF99], S. 56–65

[BC86] BOGAERT, M. ; CORNEZ, P. Calibration and segmentation algorithms for an accurate and fast range-finder. In: FAUGERAS, O. (Hrsg.) ; KELLEY, R. (Hrsg.): *Proc. SPIE 595, Computer Vision for Robots*, 1986

[BEF96] BORENSTEIN, Johann ; EVERETT, H. R. ; FENG, L. *Where am I? – Systems and Methods for Mobile Robot Positioning*. März 1996

[BEH+99] BLUM, S. ; EINSELE, T. ; HAUCK, A. ; STÖFFLER, N. O. ; FÄRBER, G. ; SCHMITT, T. ; ZIERL, C. ; RADIG, B.: Eine konfigurierbare

Systemarchitektur zur geometrisch-topologischen Exploration von Innenräumen. In: [SHF99], S. 376–387

[BF95] BORENSTEIN, J. ; FENG, L.: Correction of Systematic Odometry Errors in Mobile Robots. In: [IRO95], S. 569–574

[BF96] BORENSTEIN, J. ; FENG, L.: Measurement and Correction of Systematic Odometry Errors in Mobile Robots. In: *IEEE Journal of Robotics and Automation* 12 (1996), Dezember, Nr. 6, S. 869–880

[BFG+97] BONASSO, R. P. ; FIRBY, James ; GAT, Erann ; KORTENKAMP, David ; MILLER, David P. ; SLACK, Marc G.: Experiences with an Architecture for Intelligent, Reactive Agents, 1997, S. 237–256

[BH99] BRIECHLE, Kai ; HANEBECK, Uwe D.: Simultane Lokalisierung und Kartenaufbau für einen mobilen Serviceroboter. In: [SHF99], S. 200–210

[BHJ+99] BOLLMANN, Maik ; HOISCHEN, Rainer ; JESIKIEWICZ, Michael ; JUSTKOWSKI, Christoph ; MERTSCHING, Bärbel: Playing Domino: A Case Study for an Active Vision System. In: [Chr99], S. 392–411

[Bit99] BITTERLING, Johannes. *FPGA-Redesign eines Signalverarbeitungsalgorithmus zur gehörgerechten Sprachvorverarbeitung*. Universität Hamburg, Fachbereich Informatik, Studienarbeit. Juli 1999

[Bit00] BITTERLING, Johannes: *Konzeption und Realisierung einer Systemarchitektur für einen mobilen sicht- und sonargestützten Roboter*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 2000

[BJ83] BLUME, Christian ; JACOB, Wilfried: *Programmiersprachen für Industrieroboter: Konzepte und Sprachen; AL, VAL, HELP, SIGLA, ROBEX*. 1. Ausgabe. Vogel-Buchverlag Würzburg, 1983

[BJP99] BERG, Markus ; JÖRG, Klaus-Werner ; PAULICK, Jan-Peter: 3D Ultraschall-Entfernungsmessung mit Pseudo-Random Sequenzen. In: [SHF99], S. 78–87

[BK95] BORENSTEIN, Johann ; KOREN, Yoram: Error Eliminating Rapid Ultrasonic Firing for Mobile Robot Obstacle Avoidance. In: *IEEE Transactions on Robotics and Automation* 11 (1995), February, Nr. 1, S. 132–138

[BKM03a] BITTERLING, Johannes ; KÜPPER, Sebastian ; MERTSCHING, Bärbel: Incremental Feature-based Pose Tracking for a Mobile Robot. In: *Proc. IEEE 11th International Conference on Advanced Robotics 2003 (ICAR)*, 2003, S. 1038–1043

[BKM03b] BITTERLING, Johannes ; KÜPPER, Sebastian ; MERTSCHING, Bärbel: Incremental Localization and Environment Mapping for a Mobile Robot. In: *Proc. IEEE International Conference on Robotics, Vision, Information and Signal Processing (ROVISP)*, 2003, S. 378–383

[BM00] BITTERLING, Johannes ; MERTSCHING, Bärbel: Sichtgestützte Navigation für einen dominospielenden autonomen mobilen Roboter. In: [Gro00], S. 22–30

[BM01] BITTERLING, Johannes ; MERTSCHING, Bärbel: A Vision-Guided Robot for Manipulation of Domino Tokens. In: [LS01], S. 99–103

[BM02] BITTERLING, Johannes ; MERTSCHING, Bärbel: Self-controlled Sensor-/Platform-Adjustment for a Mobile Robot. In: [IRO02],

[Bou95] BOURGIN, David. *Colorspace FAQ*. Internet: www.aim-dtp.net/aim/technology/color-spaces%5Ffaq/color-space.faq.txt. 1995

[Bra84] BRAITENBERG, Valentino: *Vehicles*. MIT Press, Cambridge MA, 1984, 1984

[Bra97] BRADSHAW, Jeffrey M. (Hrsg.): *Software Agents*. Cambridge, Massachusetts : MIT Press, 1997

[Bro86a] BROOKS, Rodney A. *Achieving Artificial Intelligence Through Building Robots*. A.I. Memo 899. May 1986

[Bro86b] BROOKS, Rodney A.: A Robust Layered Control System for a Mobile Robot. In: *IEEE Journal of Robotics and Automation* 2 (1986), März, Nr. 1, S. 14–23

[Bro89] BROOKS, Rodney A.: A Robot that Walks; Emergent Behavior from a Carefully Evolved Network. In: *Neural Computation* 1 (1989), Nr. 2, S. 253–262

[Bro90a] BROOKS, Rodney A.: *The Behavior Language, User Guide*. Artificial Intelligence Laboratory: Massachusetts Institute of Technology, April 1990

[Bro90b] BROOKS, Rodney A.: Challenges for Complete Creature Architectures. In: *Proc. 1st International Conference on Simulation of Adaptive Behavior*. Paris, September 1990, S. 434–443

[Bro90c] BROOKS, Rodney A.: Elephants Don't Play Chess. In: *Robotics and Autonomous Systems* 6 (1990), Juni, Nr. 1&2, S. 3–15

[Bro91a] BROOKS, Rodney A.: How to Build Complete Creatures Rather than Isolated Cognitive Simulators. In: VANLEHN, Kurt (Hrsg.): *Architectures for Intelligence*. Pittsburgh : Lawrence Erlbaum Associates, 1991 (22nd Carnegie Mellon Symposion on Cognition)

[Bro91b] BROOKS, Rodney A.: Integrated Systems Based on Behaviors. In: *SIGART Bulletin* 2 (1991), Nr. 4, S. 46–50

[Bro91c] BROOKS, Rodney A.: Intelligence Without Reason. In: MYOPOULOS, John (Hrsg.) ; REITER, Ray (Hrsg.): *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*. Sydney, Australia : Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991, S. 569–595

[BSK02] BUTTKUS, Joshua ; SCHRÖDER-KROLL, Roland. *Image Segmentation Algorithms for the Collection of Target Objects with a Mobile Robot*. Universität Hamburg, Fachbereich Informatik, Baccalaureatsarbeit. 2002

[BZD01] BERTAGNOLLI, Frank ; ZIEGLER, Marcus ; DILLMANN, Rüdiger: Autonome mobile Systeme auf dem Weg zur CE-Zertifizierung am Beispiel eines Messroboters. In: [LS01], S. 91–98

[Chr99] CHRISTENSEN, H. I. (Hrsg.): *Computer Vision Systems (ICVS)*. Springer, Berlin, 1999

[CN01] CHOSET, Howie ; NAGATANI, Keiji: Topological Simultaneous Localization and Mapping (SLAM): Toward Exact Localization Without Explicit Localization. In: *IEEE Transactions on Robotics and Automation* 17 (2001), April, Nr. 2

[Con92] CONNELL, J.: SSS: A Hybrid Architecture Applied to Robot Navigation. In: *Proc. of the 1992 IEEE International Conf. on Robotics and Automation (ICRA-92)*, 1992, S. 2719–2724

[DFBT99] DELLAERT, Frank ; FOX, Dieter ; BURGARD, Wolfram ; THRUN, Sebastian: Monte Carlo Localization for Mobile Robots. In: [ICR99],

[DFDN99] DREXLER, Ch. ; FRANK, C. ; DENZLER, J. ; NIEMANN, H.: Probabilistisch modellierte Blicksteuerung zur Selbstlokalisation anhand natürlicher Landmarken. In: [SHF99], S. 221–230

[Dir00] DIRECTED PERCEPTION. *Computer Controlled Pan-Tilt Unit, Model PTU, User's Manual*. 2000

[DPPFB96] DUTOIT, T. ; PAGEL, V. ; PIERRET, N. ; F. BATAILLE, O. Van der V.: The MBROLA Project: Towards a Set of High-Quality Speech Synthesizers Free of Use for Non-Commercial Purposes. In: *Proc. ICSLP 96, Philadelphia* Bd. 3, 1996, S. 1393–1396

[DSA89] DARIO, Paolo (Hrsg.) ; SANDINI, Giulio (Hrsg.) ; AEBISCHER, Patrick (Hrsg.): *Robots and Biological Systems: Towards a New Bionics?* Bd. *102*. Springer Verlag, Berlin, in Kooperation mit der NATO Scientific Affairs Division, 1989 (NATO ASI Series F: Computer and System Sciences)

[Ebn98] EBNER, Marc: Evolution of a control architecture for a mobile robot. In: SIPPER, Moshe (Hrsg.) ; MANGE, Daniel (Hrsg.) ; PÉREZ-URIBE, Andrés (Hrsg.): *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES 98)* Bd. 1478. Lausanne, Switzerland : Springer Verlag, 23-25 1998, S. 303–310

[ED99] V. EHR, Markus ; DILLMANN, Rüdiger: Planung von Messpositionen zur automatischen und autonomen Oberflächenvermessung. In: [SHF99], S. 262–271

[Edl96] EDLINGER, Thomas: *Hierarchische Steuerung für einen mobilen Roboter zur autonomen Erkundung seiner Einsatzumgebung*, Universität Kaiserslautern, Diss., 1996

[Eve95] EVERETT, H. R.: *Sensors for mobile Robots – Theory and Application*. A K Peters, Ltd., 1995 (Intelligent Robots and Autonomous Agents)

[Fau88] FAUGERAS, O. A few Steps toward Artificial 3D Vision / INRIA. Le Chesnay, France, 1988. – Rapports de Recherche (N° 790)

[Fer92] FERGUSON, Innes A.: *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. Cambridge, UK, Diss., 1992

[Fey00] FEYRER, Stefan: *Detektion, Lokalisierung und Verfolgung von Personen mit einem mobilen Serviceroboter*, Eberhard-Karls-Universität Tübingen, Fakultät für Informatik, Diss., 2000

[FG96] FRANKLIN, S. ; GRAESSER, A.: Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents. In: *Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96)* Bd. 1193. Berlin, Germany : Springer-Verlag, 1996

[Fir89] FIRBY, R. J.: *Adaptive Execution in Complex Dynamic Domains*, Yale University, Diss., January 1989

[Fir94] FIRBY, R. J.: Architecture, Representation and Integration: An Example from Robot Navigation. In: *Proceedings of the 1994 AAAI Fall Symposium Series Workshop on the Control of the Physical World by Intelligent Agents*, New Orleans LA, November 1994

[Fis03] FISCHER, Björn: *Implementierung einer Disparitätsschätzung für stereoskope Bildsignale eines mobilen Roboters*, Universität der Bundeswehr Hamburg, Diplomarbeit, 2003

[Fla96] FLANAGAN, David: *Java in a Nutshell*. 1st edition. O'Reilly and Associates, Inc., 1996

[FPS99] FIRBY, R. J. ; PROKOPOWICZ, Peter N. ; SWAIN, Michael J.: The Animate Agent Architecture. In: KORTENKAMP, David (Hrsg.) ; BONASSO, Russell P. (Hrsg.) ; MURPHY, Robin (Hrsg.): *Artificial Intelligence and Mobile Robots – Case Studies of Successful Robot*

Systems. Menlo Park, California, Cambridge, Massachusetts, London, England : AAAI Press/The MIT Press, 1999, Kapitel 10, S. 243–275

[FS01] FÖLSCH, Florian ; SCHÖNFELDER, Jochen. *Teilautonome Navigation eines mobilen Roboters anhand interaktiv generierter Karten innerhalb einer verteilten Architektur*. Universität Hamburg, Fachbereich Informatik, Baccalaureatsarbeit. 2001

[Gat92] GAT, Erann: Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling a Real-world Mobile Robot. In: *Proc. of the Tenth National Conference on Artificial Intelligence*. San Mateo, CA : Morgan Kaufmann, 1992

[Gat97] GAT, Erann: On three-layer architectures. In: *Artificial Intelligence and Mobile Robots* (1997). – MIT/AAAI Press

[Gat99] GAT, Erann: Three-Layer Architectures. In: KORTENKAMP, David (Hrsg.) ; BONASSO, Russell P. (Hrsg.) ; MURPHY, Robin (Hrsg.): *Artificial Intelligence and Mobile Robots – Case Studies of Successful Robot Systems*. Menlo Park, California, Cambridge, Massachusetts, London, England : AAAI Press/The MIT Press, 1999, Kapitel 8, S. 195–210

[GED98] GRAF, René ; VON EHR, Markus ; DILLMANN, Rüdiger: Automatische Vermessung großer Objekte mittels eines mobilen Roboters. In: [WDH98], S. 263–270

[Gei02] GEISTHÖVEL, Roman. *Projektbericht: Mobile Roboter*. Universität Hamburg, Fachbereich Informatik: Interner Bericht in zwei Teilen zu Projekt 18.331 im Wintersemester 2001/2002. 2002

[Gen94] GENGENBACH, Volker: *Einsatz von Rückkopplungen in der Bildauswertung bei einem Hand-Auge-System zur automatischen Demontage*, Karlsruhe, Univ., Diss., 1994

[GHJV95] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Design Patterns*. 1st Edition. Addison-Wesley, 1995

[GL87] GEORGEFF, M. P. ; LANSKY, A. L.: Reactive reasoning and planning. In: *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, 1987, S. 677–682

[Gro00] GROSS, H. M. (Hrsg.): *Selbstorganisation von adaptivem Verhalten (SOAVE)*. Bd. *10*. VDI-Verlag, 2000

[GRS99] GOEL, Puneet ; ROUMELIOTIS, Stergios I. ; SUKHATME, Gaurav S.: Robust localization using relative and absolute position estimates. In: [IRO99],

[GS96] GUTMANN, J.-S. ; SCHLEGEL, C.: Amos: Comparison of scan matching approaches for self-localization in indoor environments. In: *Proceedings of the 1st Euromicro Workshop on Advanced Mobile Robots, IEEE Computer Society Press*, 1996

[GS01] GIAGOURTA, Venetia (Hrsg.) ; STRINTZIS, Michael G. (Hrsg.): *Proc. International Conference on Augmented Virtual Environments and Three Dimensional Imaging (EUROIMAGE ICAV3D)*. The European Project INTERFACE IST, 2001

[GS02] GOLDBERG, Ken (Hrsg.) ; SIEGWART, Roland (Hrsg.): *Beyond Webcams – An Introduction to Online Robots*. 1. Ausgabe. The MIT Press, Cambridge, Massachusetts, London, England, 2002

[GS03] GAWORSKI, Björn ; STAHLBERG, Alexander. *Projektbericht Mobile Roboter*. Universität Hamburg, Fachbereich Informatik, Projektarbeit. 2003

[GWN99] GUTMANN, J. ; WEIGEL, T. ; NEBEL, B.: Fast, accurate, and robust selflocalization in the robocup environment. In: *Robocup 1999 (at IJCAI-99)*, 1999, S. 304–317

[Har89] HARALICK, R.M.: Monocular Vision Using Inverse Perspective Projection Geometry: Analytic Relations. In: *CVPR* 89 (1989), S. 370–378

[HCBK00] HAGER, Gregory D. (Hrsg.) ; CHRISTENSEN, Henrik I. (Hrsg.) ; BUNKE, Horst (Hrsg.) ; KLEIN, Rolf (Hrsg.): *Sensor Based Intelligent Robots*. Springer Verlag, Berlin, Heidelberg, 2000 (Lecture Notes in Computer Science (LNCS), Nr. 2238)

[HKWW01] HERZOG, Rainer ; KÜPPER, Sebastian ; WALDAU, Björn ; WITT, Stefan. *Dokumentation Projektseminar Mobile Roboter*. Universität Hamburg, Fachbereich Informatik: Interner Bericht, Wintersemester 2000/2001. 2001

[Hol03] HOLST, Robert: *Entwurf und Implementierung einer Bildkompression für eine Roboter-Steuerung*, Universität der Bundeswehr Hamburg, Diplomarbeit, 2003

[Hor86] HORN, Berthold Klaus P.: *Robot vision*. Cambridge, Mass.: MIT Press, 1986

[HPR+99] HAUCK, Alexa ; PASSIG, Georg ; RÜTTINGER, Johanna ; SORG, Michael ; FÄRBER, Georg: Biologically Motivated Hand-Eye Coordination for the Autonomous Grasping of Unknown Objects. In: [SHF99], S. 140–49

[HSDK97] HAMPSHIRE, J. ; SALIDO, J. ; DOLAN, J. ; KHOSLA, P.: A modified reactive control framework for cooperative mobile robots. In: *Proceedings on SPIE International Symposium on Sensor Fusion and Decentralized Control in Autonomous Robotic Systems* Bd. 3209, 1997, S. 90–100

[ICR88] ICRA (Hrsg.): *Proc. IEEE International Conference on Robotics and Automation (ICRA)*. 1988

[ICR99] ICRA (Hrsg.): *Proc. IEEE International Conference on Robotics and Automation (ICRA)*. 1999

[Ioc99] IOCCHI, Luca: *Design and Development of Cognitive Robots*, Università degli Studi di Roma La Sapienza, Diss., März 1999

[IRO94] IROS (Hrsg.): *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 1994

[IRO95] IROS (Hrsg.): *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Pittsburgh, Pennsylvania, 1995

[IRO99] IROS (Hrsg.): *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 1999

[IRO02] IROS (Hrsg.): *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland : Omnipress, 2002

[Jen99] JENSFELT, Patric: Localization Using Laser Scanning and Minimalistic Environmental Models. 1999. – Forschungsbericht

[Jen01] JENSFELT, Patric: *Approaches to Mobile Robot Localization in Indoor Environments*, Department of Signals, Sensors and Systems (S3), Royal Institute of Technology (KTH) Stockholm, Sweden, Diss., 2001

[JHG99] JÄHNE, Bernd (Hrsg.) ; HAUSSECKER, Horst (Hrsg.) ; GEISSLER, Peter (Hrsg.): *Handbook of Computer Vision and Applications*. Academic Press, 1999

[Jäh97] JÄHNE, Bernd: *Digitale Bildverarbeitung*. Springer, 1997

[KA98] KOHNO, Michimune ; ANZAI, Yuichiro: An Adaptive Sensor Network System for Complex Environments. In: [KWS98a], S. 21–28

[KBM98] KORTENKAMP, David (Hrsg.) ; BONASSO, Russell P. (Hrsg.) ; MURPHY, Robin (Hrsg.): *Artificial Intelligence and Mobile Robots – Case Studies of Successful Robot Systems*. Menlo Park, California, Cambridge, Massachusetts, London, England : AAAI Press/The MIT Press, 1998

[Kha86] KHATIB, Oussama: Real-time obstacle avoidance for manipulators and mobile robots. In: *International Journal of Robotics Research* 5 (1986), Nr. 1, S. 90–98

[KJ99] KALENKA, S. ; JENNINGS, N. R.: Socially Responsible Decision Making by Autonomous Agents. In: KORTA, K. (Hrsg.) ; SOSA, E. (Hrsg.) ; ARRAZOLA, X. (Hrsg.): *Cognition, Agency and Rationality*. Kluwer Academic publishers, 1999, S. 135–149

[KM99] KONOLIGE, Kurt ; MYERS, Karen: The Saphira Architecture for Autonomous Robots. In: KORTENKAMP, David (Hrsg.) ; BONASSO, Russell P. (Hrsg.) ; MURPHY, Robin (Hrsg.): *Artificial Intelligence and Mobile Robots – Case Studies of Successful Robot Systems*. Menlo Park, California, Cambridge, Massachusetts, London, England : AAAI Press/The MIT Press, 1999, Kapitel 9, S. 211–242

[KM03] KIRITZ, Michael ; MALZAHN, Martin. *Projektbericht: Mobile Roboter*. Universität Hamburg, Fachbereich Informatik, Projektarbeit. 2003

[KMRS97] KONOLIGE, K. ; MYERS, K. L. ; RUSPINI, E. H. ; SAFFIOTTI, A.: The Saphira Architecture: A Design for Autonomy. In: *Journal*

of experimental & theoretical artificial intelligence (JETAI) 9 (1997), Nr. 1, S. 215–235

[Kon96] KONOLIGE, Kurt G.: *Pioneer 1 Mobile Robot Software Manual*. Version 4.1.2. : ActivMedia Inc., 1996

[Kon97] KONOLIGE, Kurt G.: *Saphira Software Manual*. Version 6.1. : SRI International (former Stanford Research Institute), 1997

[Kon00] KONOLIGE, Kurt G.: *Pioneer 2 Mobile Robot Operations Manual*. v5. : ActivMedia Robotics, LLC, Januar 2000

[Kro86] KROGH, Bruce H.: Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles. In: *Proceedings of 1986 IEEE International Conference on Robotics and Automation (ICRA '86)*, 1986, S. 1664–1669

[KW99] KRUSE, E. ; WAHL, F. M.: SiVCAT-Sichtsystemfunktionen für die automatisierte Montage: 3D-Lagemessungen. In: [SHF99], S. 293–302

[KWS98a] KAKAZU, Y. (Hrsg.) ; WADA, M. (Hrsg.) ; SATO, T. (Hrsg.): *Intelligent Autonomous Systems (IAS-5)*. IOS Press, Amsterdam, Berlin, Oxford, Tokyo, Washingtion DC, 1998 (IAS 5)

[KWS+98b] KRABBES, M. ; WEBER, S. ; STEPHAN, V. ; BÖHME, H.-J. ; GROSS, H.-M.: Monokulare visuelle Hindernisdetektion auf Basis merkmalsbasierter Bildsegmentierung. In: [WDH98], S. 85–92

[Küp01] KÜPPER, Sebastian. *Verhaltenssteuerung bei autonomen mobilen Robotern am Beispiel eines fiktiven Kleintieres*. Universität Hamburg, Fachbereich Informatik, Studienarbeit. 2001

[Küp03] KÜPPER, Sebastian: *Gewinnung einer Positionshypothese für mobile Systeme durch konsekutive Auswertung extrinsischer Sensorinformationen*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 2003

[Lag87] LAGEMANN, Klaus: *Rechnerstrukturen – Verhaltensbeschreibung und Entwurfsebenen*. Springer-Verlag, 1987

[LH92] LYONS, D.M. ; HENDRIKS, A.: Planning for Reactive Robot Behavior. In: *IEEE Int. Conf. on Robotics and Automation Nice*, 1992

[LM94] LU, F. ; MILIOS, E.: Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. In: *Proc. CVPR*, 1994, S. 935–938

[LN02] LAKEMEYER, Gerhard (Hrsg.) ; NEBEL, Bernhard (Hrsg.): *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann Publishers, 2002

[Log98] LOGAN, Brian: Classifying Agent Systems. In: *Proc. of the AAAI-98 Workshop on Software Tools for Developing Agents*. Wisconsin, USA, 1998

[LP00] LIPSON, Hod ; POLLACK, Jordan B.: Automatic Design and Manufacture of Robot Lifeforms. In: *Nature* (2000), Nr. 406, S. 974–978

[LS01] LEVI, Paul (Hrsg.) ; SCHANZ, Michael (Hrsg.): *Autonome Mobile Systeme (AMS)*. Springer Verlag, 2001 (GI Informatik aktuell)

[LSF89] LIN, Long-Ji ; SIMMONS, Reid ; FEDOR, Christopher. *Experience with a Task Control Architecture for Mobile Robots*. tech. report CMU-RI-TR-89-29. December 1989

[Lus81] LUSTGARTEN, Eli S.: Robotics and its Relationship to the Automated Factory. In: *Proc. Robotics Workshop of the Congressional Office of Technology Assessment*, US Government Printing Office, 1981, S. 118–136

[LVSG99] LANG, Oliver ; VOGEL, Ralf ; SIEBELS, Nils T. ; GRÄSER, Axel: Vergleich verschiedener bildbasierter Regler zur Realisierung teilautonomer Greifvorgänge. In: [SHF99], S. 188–197

[MA92] MURPHY, Robin ; ARKIN, Ronald C.: SFX: An Architecture for Action-Oriented Sensor Fusion. In: *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1992, S. 1079–1086

[ME85] MORAVEC, H. ; ELFES, A.: High-resolution maps from wide-angle sonar. In: *Proc. IEEE ICRA*, 1985, S. 116–121

[ME87] MATTHIES, L. H. ; ELFES, A.: Integration of Sonar and Stereo Range Data Using a Grid-Based Representation. In: [ICR88],

[Mey91] MEYSTEL, A.: *World Scientific Series in Automation*. Bd. 1: *Autonomous Mobile Robots – Vehicles with Cognitive Control*. World Scientific, 1991

[MLD96] MICHAUD, F. ; LACHIVER, G. ; DINH, C. T. L.: A new control architecture combining reactivity, deliberation and motivation for situated autonomous agent. In: *Proc. of the Int. Conf. on Simulation and Adaptive Behavior (SAB)*. Cap Cod, 1996, S. 245–254

[Mor00] MORGENSTERN, Anika: *Etablierung eines* in vitro-*Kultursystems für* Pennisetum glaucum *(L.) R. Br.*, Universität Hamburg, Diplomarbeit, 2000

[MPN+02] MUTZ, Darren H. ; PETRAS, Hari Das ND R. ; NESNAS, Issa A. ; VOLPE, Richard ; ESTLIN, Tara A.: Technical Support Package on Coupled-Layer Architecture for Advanced Software for Robots / National Aeronautics and Space Administration. 2002. – Nasa Tech Briefs Vol. 26 No. 12 from JPL New Technology Report NPO 21218

[Mur00] MURPHY, Robin R.: *Introduction to AI Robotics*. MIT Press Cambridge, Massachusetts, London, England, 2000 (Intelligent Robots and Autonomous Agents)

[Neh00] NEHMZOW, U.: *Mobile Robotics: A Practical Introduction*. Springer, 2000 (Applied Computing)

[Neh02] NEHMZOW, Ulrich: *Mobile Robotik*. Springer Verlag, Berlin et al., 2002

[Nei74] NEISSER, Ulric: *Kognitive Psychologie*. 1. Ausgabe. Ernst Klett Verlag, 1974

[Nil82] NILSSON, Nils J.: *Principles of artificial intelligence*. Berlin : Springer, 1982 (Symbolic Computation)

[Nil98] NILSSON, Nils J.: *Artificial Intelligence: A New Synthesis*. Stanford University : Morgan Kaufman Publishers, Inc., 1998

[Nol02] NOLFI, Stefano: Power and limits of reactive agents. In: *Neurocomputing* (2002), Nr. 49, S. 119–145

[OWH01] OOI, Teng L. ; WU, Bing ; HE, Zijiang J.: Distance determined by the angular declination below the horizon. In: *Nature* (2001), November, Nr. 414, S. 197–200

[Par98] PARKER, L. E.: ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. In: *IEEE Transactions on Robotics and Automation* 14(2) (1998), S. 220–240

[PFPB99] PAULY, M. ; FINKE, M. ; PETERS, L. ; BECK, K.: Echtzeitfähige Positionskorrektur auf Basis natürlicher Landmarken. In: [SHF99], S. 98–107

[PH00] PYEATT, Larry D. ; HOWE, Adele E.: Evaluating robustness in a two layer simulated robot architecture. In: *Journal of experimental & theoretical artificial intelligence (JETAI)* 12 (2000), Nr. 2, S. 213–234

[PHTO+00] PIRJANIAN, P. ; HUNTSBERGER, T. ; TREBI-OLLENNU, A. ; AGHAZARIAN, H. ; DAS, H. ; JOSHI, S. ; SCHENKER, P.: CAMPOUT: a control architecture for multirobot planetary outposts. In: *Proc. SPIE Conf. Sensor Fusion and Decentralized Control in Robotic Systems III*, 2000

[Pir97] PIRJANIAN, Paolo: An Overview of System Architectures for Action Selection in Mobile Robotics / Laboratory of Image Analysis, Aalborg University. 1997. – Forschungsbericht

[Poy98] POYNTON, Charles. *Color FAQ*. Internet: www.poynton.com/ColorFAQ.html. 1998

[PSW90] PORTELE, T. ; SENDLMEIER, W.F. ; W., Hess: HADIFIX: A system for German speech synthesis based on demisyllables, diphones, and suffixes. In: *Proc. ESCA-Workshop on Speech Synthesis, Autrans, France* Institut für Kommunikationsforschung und Phonetik, Universität Bonn, 1990, S. 161–164

[RK03] RANGANATHAN, Ananth ; KOENIG, Sven: A Reactive Robot Architecture with Planning on Demand. In: *Proc. IROS*. College of Computing Georgia Institute of Technology Atlanta, 2003

[Ros95] ROSENBLATT, J.: DAMN: A Distributed Architecture for Mobile Navigation. In: HEXMOOR, H. (Hrsg.) ; KORTENKAMP, D. (Hrsg.): *Proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*, AAAI Press, Menlo Park, CA, March 1995, S. 27–29

[Ros97] ROSENBLATT, Julio K.: *DAMN: A Distributed Architecture for Mobile Navigation*, Carnegie Mellon University, Robotics Institute, Diss., 1997

[Ros98] ROSENTHAL, Michael: *Kollisionsfreie Navigation eines mobilen Roboters durch Auswertung von Sonar- und Bilddaten*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 1998

[RP89] ROSENBLATT, J.K. ; PAYTON, D.W.: A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control. In: *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks* Bd. 2, 1989, S. 317–324

[RRN02] ROWE, Anthony ; ROSENBERG, Charles ; NOURBAKHSH, Illah: A Low Cost Embedded Color Vision System. In: [IRO02],

[SA01] STOYTCHEV, Alexander ; ARKIN, Ronald C.: Combining Deliberation, Reactivity, and Motivation in the Context of a Behavior-Based Robot Architecture. In: *In Proc. IEEE International Symposium on Computational Intelligence in Robotics and Automation (IEEE CIRA 2001)*. Banff, Canada, 2001

[Sah94] SAHOTA, Michael K.: Reactive Deliberation: An Architecture for Real-Time Intelligent Control in Dynamic Environments. In: *Proc. of the AAAI Spring Symp. on Lessons Learned from Implememted Software Architectures for Physical agents*. Stanford, CA, 1994, S. 1303–1308

[Sah95] SAHOTA, M. K.: Reactive Deliberation: An Architecture for Real-Time Intelligent Control in Dynamic Environments. In: *Proc. of the AAAI Spring Symp. on Lessons Learned from Implememted Software Architectures for Physical agents*. Stanford, CA, 1995

[San03] SANDER, Tony: *Entwurf und Implementierung einer WLAN-Übertragung von stereoskopen Bildsignalen*, Universität der Bundeswehr Hamburg, Diplomarbeit, 2003

[SB03] SCHWEMANN, Timm ; BECK, Mark. *Projektbericht: Mobile Roboter*. Universität Hamburg, Fachbereich Informatik, Projektarbeit. 2003

[Sch88] SCHÜTTE, Alois: *Programmieren in Occam*. Addison-Wesley, 1988

[Sch99] SCHMALZ, Steffen. *Aufbau der Funkmodemverbindung zum Pioneer-Roboter*. Interner Bericht. 1999

[SGH+97a] SIMMONS, Reid ; GOODWIN, Richard ; HAIGH, Karen Z. ; KOENIG, Sven ; O'SULLIVAN, Joseph: A Layered Architecture for Office Delivery Robots. In: JOHNSON, W. L. (Hrsg.) ; HAYES-ROTH, Barbara (Hrsg.): *Proc. 1st Intl. Conf. on Autonomous Agents (Agents 1997)*. New York : ACM Press, 1997, S. 245–252

[SGH+97b] SIMMONS, Reid ; GOODWIN, Richard ; HAIGH, Karen Z. ; KOENIG, Sven ; OŚULLIVAN, Joseph. *Xavier: Experience with a layered robot architecture*. 1997

[SHF99] SCHMIDT, G. (Hrsg.) ; HANEBECK, U. (Hrsg.) ; FREYBERGER, F. (Hrsg.): *Autonome Mobile Systeme (AMS)*. Springer Verlag, 1999 (GI Informatik aktuell)

[SIC00] SICK AG, Düsseldorf: *LMS/LMI 400 Definition der Telegramme zwischen Benutzerschnittstelle und LMS- oder LMI 400-System über RS422/RS232*. 4 2000

[Son97] Sony Corp., Tokyo, Japan: *EVI-370/370D/371/371D Service Manual*. 1997

[Ste65] STEINBUCH, Karl: *Automat und Mensch*. Springer-Verlag Berlin, Heidelberg, New York, 1965

[Str00] STROUSTRUP, Bjarne: *Die C++ Programmiersprache*. 4., aktualisierte Auflage (dt. Ausgabe der Special Edition). Addison-Wesley, 2000

[TPP03] TRAPPL, Robert (Hrsg.) ; PETTA, Paolo (Hrsg.) ; PAYR, Sabine (Hrsg.): *Emotions in Humans and Artifacts*. Bradford Books, MIT Press, 2003

[Tro00] TROLLTECH. *Qt Cross-Platform C++ GUI Framework*. Internet: www.trolltech.no. 2000

[UJ00] UMBACH, Dale ; JONES, Kerry N.: A Few Methods for Fitting Circles to Data. In: *IEEE Transactions on Instrumentation and Measurement* XX (2000)

[van02] VAN HEESCH, Dimitri. *Doxygen – a documentation system*. Internet, www.doxygen.org. September 2002

[Č61] ČAPEK, Karel: *R.U.R.* Oxford University Press, 1961

[Ver01] VERBEEK, Christian: *Reaktive Steuerung autonomer mobiler Roboter*, Universität Bielefeld, Diss., Januar 2001

[VNE+00] VOLPE, Richard ; NESNAS, I.A.D. ; ESTLIN, T. ; MUTZ, D. ; PETRAS, R. ; DAS, H.: CLARAty: Coupled Layer Architecture for Robotic Autonomy / Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California. 2000. – JPL Technical Report D-19975

[VNE+01] VOLPE, R. ; NESNAS, I. ; ESTLIN, T. ; MUTZ, D. ; PETRAS, R. ; DAS, H.: The CLARAty architecture for robotic autonomy. In: *Proc. 2001 IEEE Aerospace Conference*. Big Sky, Montana, März 2001

[Wal50] WALTER, W. G.: An Imitation of Life. In: *Scientific American* (1950), Mai, S. 42–45

[Wal51] WALTER, W. G.: A Machine That Learns. In: *Scientific American* (1951), August, S. 60–63

[WDH98] WÖRN, H. (Hrsg.) ; DILLMANN, R. (Hrsg.) ; HENRICH, D. (Hrsg.): *Autonome Mobile Systeme (AMS)*. Springer Verlag, 1998 (GI Informatik Aktuell)

[Wil03] WILKEN, Guido. *Projektbericht: Mobile Roboter*. Universität Hamburg, Fachbereich Informatik, Projektarbeit. 2003

[WKH98] WASSON, Glenn ; KORTENKAMP, David ; HUBER, Eric: Integrating Active Perception with an Autonomous Robot Architecture. In: SYCARA, Katia P. (Hrsg.) ; WOOLDRIDGE, Michael (Hrsg.): *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*. New York : ACM Press, 1998, S. 325–331

[Wol01] WOLF, Jürgen: *Bildbasierte Lokalisierung für mobile Roboter*, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, Arbeitsgruppe Autonome Systeme, Diplomarbeit, 2001

[WWP94] WEISS, Gerhard ; WETZLER, Christopher ; VON PUTTKAMER, Ewald: Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans. In: [IRO94], S. 595–601

Index

T