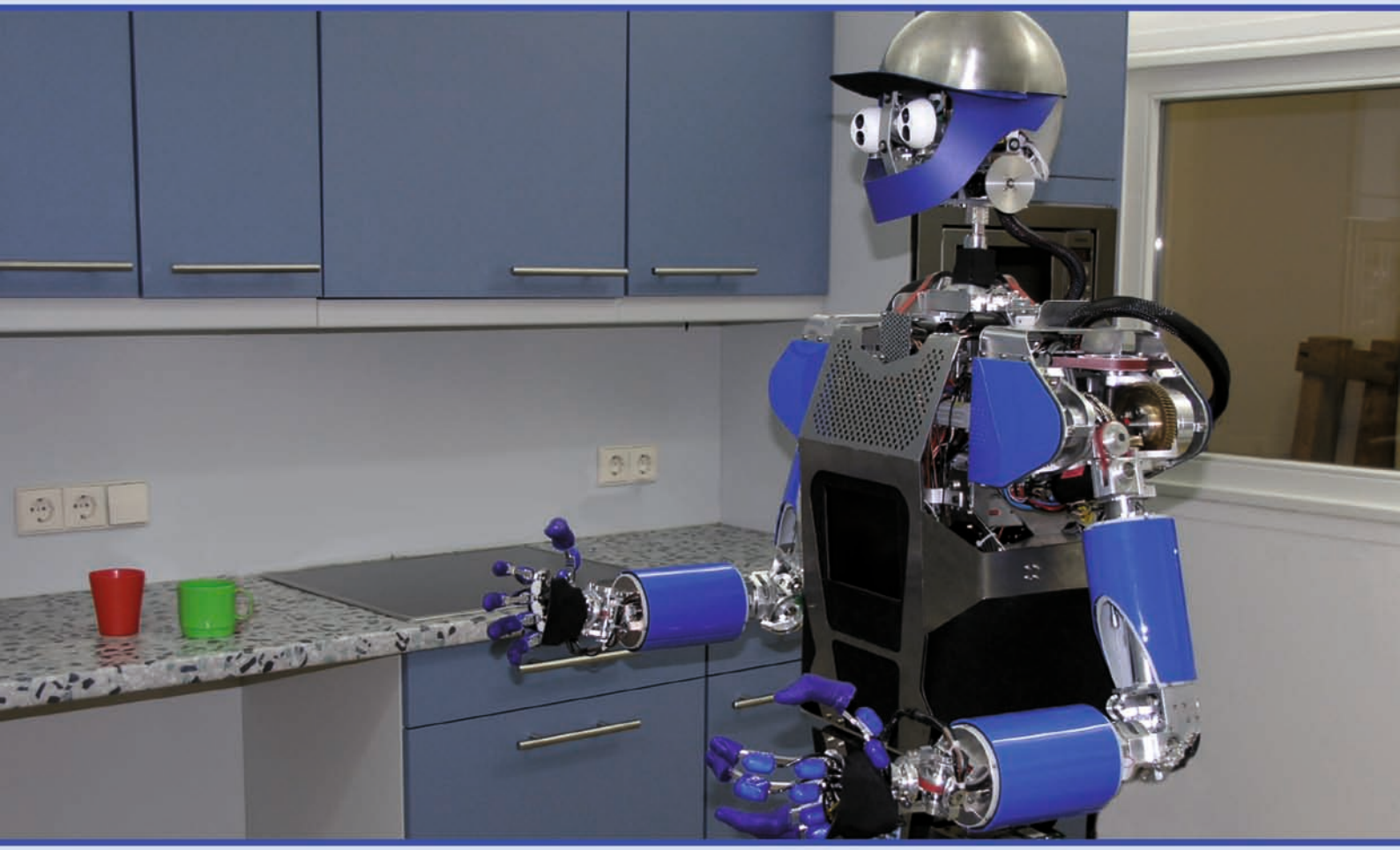
Kristian Regenstein



Modulare, verteilte Hardware-Software-Architektur für humanoide Roboter

KIT Scientific Publishing

Kristian Regenstein

Modulare, verteilte Hardware-Software-Architektur für humanoide Roboter

Modulare, verteilte Hardware-Software-Architektur für humanoide Roboter

von
Kristian Regenstein

KIT Scientific Publishing

Dissertation, Karlsruher Institut für Technologie
Fakultät für Informatik
Tag der mündlichen Prüfung: 27.04.2010
Referenten: Prof. Dr.-Ing. Rüdiger Dillmann, Prof. Dr.-Ing. Heinz Wörn

Impressum

Karlsruher Institut für Technologie (KIT)
KIT Scientific Publishing
Straße am Forum 2
D-76131 Karlsruhe
www.ksp.kit.edu

KIT – Universität des Landes Baden-Württemberg und nationales
Forschungszentrum in der Helmholtz-Gemeinschaft

KIT Scientific Publishing 2010
Print on Demand

ISBN 978-3-86644-527-7

Inhaltsverzeichnis

1. Einleitung

Es ist ein lang gehegter Traum des Menschen, Maschinen zu entwickeln, die ihm die Arbeit erleichtern oder gar abnehmen. Im Bereich der Robotik gibt es bereits seit einiger Zeit vielversprechende Ansätze zur Umsetzung dieses Traums. Es existieren schon zum heutigen Zeitpunkt eine Vielzahl von Robotersystemen, die Arbeiten in der industriellen Produktion oder Arbeiten in einem für Menschen gefährlichen Umfeld übernehmen. Allerdings sind die meisten der heutigen Roboter sehr speziell für eine Anwendungsklasse entworfen und nicht an beliebiger Stelle und bei wechselndem Bedarf einsetzbar. Besonders für die Interaktion zwischen Mensch und Roboter bzw. zum Lernen neuer Fähigkeiten eines Roboters ist dessen Programmierung durch einen Spezialisten notwendig. Der Begriff Roboter wurde durch Karel Čapek geprägt. In seinem Stück „Rossum's Universal Robots" beschreibt Čapek wie menschenähnliche Maschinen eingesetzt werden, um dem Menschen die Arbeit zu erleichtern. Das Wort Roboter leitet sich hierbei aus dem slawischen Wort *robota*, was soviel wie Fronarbeit oder schwere Arbeit bedeutet, ab [98].

Die VDI-Richtlinie 2860 definiert:

> „Industrieroboter sind universell einsetzbare Bewegungsautomaten mit mehreren Achsen, deren Bewegungen hinsichtlich Bewegungsfolge und Wegen bzw. Winkeln frei (d.h. ohne mechanischen Eingriff) programmierbar und gegebenenfalls sensorgeführt sind. Sie sind mit Greifern, Werkzeugen oder anderen Fertigungsmitteln ausrüstbar und können Handhabungs- und/oder Fertigungsaufgaben ausführen."
>
> *[156]*

1.1. Humanoide Roboter

Humanoide Roboter sind Robotersysteme, die in ihrem Aussehen und in ihrem Verhalten dem Menschen nachempfunden sind. Im Gegensatz zu Industrierobotern sind die Anforderungen an diese Roboter darauf ausgerichtet, dass Mensch und Maschine im gleichen Arbeitsraum sicher miteinander agieren können.

Üblicherweise verfügt ein humanoider Roboter über einen Torso mit Kopf, zwei Arme und zwei Beine. Allerdings ist diese Form der Realisierung nicht zwingend. So gibt es auch humanoide Roboter die nur Teilaspekte realisieren. Beispielsweise gibt es einige Roboter bei denen die Beine durch eine mobile Plattform ersetzt sind. Abhängig vom Einsatzgebiet des humano-

iden Roboters wird der Schwerpunkt auf einzelne Teilaspekte gelegt. Eigenschaften, die zumindest in Teilen erfüllt sein müssen, damit ein Roboter als humanoider Roboter eingeordnet wird, sind Autonomie, Lernfähigkeit, menschenähnliche Gestalt und Verhalten, Interaktions- und Kooperationsfähigkeit und hohe Systemsicherheit. Die Anwendungsmöglichkeiten für humanoide Roboter sind vielfältig und bei weitem noch nicht erschlossen. Sie reichen vom Einsatz im Haushalt als interaktiv kooperierendes System oder zu Unterhaltungszwecken in privaten oder öffentlichen Bereichen über Assistenzaufgaben in der industriellen Fertigung bis hin zu Einsätzen in für den Menschen gefährlichen Umgebungen.

Bei Robotern, bei denen die Manipulation von Objekten im Mittelpunkt steht, wird beispielsweise auf die Bewegungsmöglichkeiten der Arme und die Manipulationsfähigkeiten des Greifers bzw. der Hände mehr Wert gelegt als bei einem Roboter für den der Fokus auf zweibeinigem Laufen liegt. Im Bereich humanoider Roboter lassen sich als Schwerpunkte die folgenden charakteristischen Merkmale und Fähigkeiten ausmachen:

- Lokomotion
- Manipulation und Greifen
- Multimodale Interaktion
- Lernen von Handlungen, Objekt- und Raumrepräsentationen
- Kooperation

Bekannte Beispiele für zweibeiniges Laufen sind die humanoiden Roboter *ASIMO*, *HRP-II*, *Wabian-2*, *JOHNNIE* und *KHR-2* [19]. Manipulative Fähigkeiten sind bei diesen humanoiden Robotern nur sehr rudimentär ausgeprägt. Andererseits existieren bereits eine Reihe zweiarmiger Robotersysteme, ausgestattet mit sensorischen Köpfen und mehrfingrigen Händen, meist nur als Torso auf einer mobilen Plattform. Typische Beispiele sind die Systeme *Justin* (DLR) oder *ARMAR-III* (Universität Karlsruhe), die Kraft-Momenten-geregelgte Zweiarm-Manipulationen ausführen können. In Kapitel 2 wird auf die wichtigsten dieser Roboter – unter dem Gesichtspunkt der Hardware-Software-Architektur – näher eingegangen.

1.2. Aufgabenstellung und Beitrag der Arbeit

Die technischen Fortschritte auf den Gebieten der Rechnertechnologie und Mechatronik stellen die wichtigsten Basistechnologien, um humanoide Roboter als mechatronische Systeme mit eingebetteten Rechnern realisieren zu können. Die meisten der heutigen Systemimplementierungen konzentrieren sich vor allem auf die Realisierung von Einzelfähigkeiten wie zweibeiniges Laufen, Bildverarbeitung, Sprachverstehen und die Regelung von Bewegungsabläufen.

Der Entwurf einer generischen Hardware-Software-Architektur zur systematischen Integration dieser Fähigkeiten und der zur Robotersteuerung notwendigen Rechenprozesse bezüglich Sensorik, Datenverarbeitung und Aktorik zu einem robusten und flexiblen Gesamtsystem findet bisher wenig Beachtung.

Ziel dieser Arbeit sind Untersuchungen zur Abbildung funktionaler Roboterarchitekturen auf dezidierte Rechnerarchitekturen sowie deren softwaretechnische Umsetzung auf eine modulare und verteilte Art und Weise. Die resultierende Hardware-Software-Architektur soll am Beispiel eines humanoiden Roboters mit den oben beschriebenen Fähigkeiten illustriert und auf ihre Leistungsfähigkeit hin untersucht werden. Hierbei wird nicht nur eine abstrakte Rechnerarchitektur an sich in die Überlegung mit einbezogen, sondern darüber hinaus – als erster Schritt – eine Steuerungsarchitektur entworfen. Ausgehend von dieser Steuerungsarchitektur wird unter Berücksichtigung spezifischer Randbedingungen bei humanoiden Robotern die Rechnerarchitektur abgeleitet und die Realisierung in Hardware beschrieben. Die Architektur wird modular konzipiert, so dass sie generisch erweiterbar ist und auch auf zukünftige Anforderungen hin erweitert werden kann. Besonderes Augenmerk gilt der Integration von Aktorik und einer Vielzahl unterschiedlicher Sensoren, die im Roboter benötigt werden. Mit Hilfe der Modellierung dieser Komponenten als abstrakte Sensoren bzw. Aktoren, lässt sich sowohl eine Softwarearchitektur als auch eine Hardwarestruktur ableiten. Im Folgenden werden die Fragestellungen, die für die Aspekte Steuerungsarchitektur, Softwarerahmenwerk und die Abbildung auf eine Hardware-Software-Architektur behandelt werden, näher beschrieben:

Steuerungsarchitektur Eine Steuerungsarchitektur beschreibt die Struktur der Steuerung eines Robotersystems, also deren Organisationsprinzip, die erforderlichen Datentypen und sowohl die Operationen auf diesen als auch die Datenflüsse zwischen den an der Steuerung beteiligten Prozessen. Aus der Sicht der Robotersteuerung besteht ein Robotersystem aus den drei Grundprozessen: Wahrnehmung, Planung und Handlung – in der Literatur werden diese Prozesse üblicherweise auch als *sense-plan-act* bezeichnet. Dabei steht *sense* für alle Teilprozesse der Umweltwahrnehmung, die durch Sensoren erfasst werden, wie zum Beispiel Positionssensoren, Drehmomentsensoren oder auch visuelle Sensoren wie Kameras. Unter *plan* versteht man den Teil der Steuerung, der aus den mittels der Sensoren erfassten Daten und aus weiteren Informationen eine zukünftige Handlung plant und daraus Ausführungsanweisungen an die Aktoren generiert. In der *act*-Phase werden die geplanten Handlungen und Bewegungen durch die Aktoren umgesetzt. Diese drei Bereiche *sense-plan-act* können auf unterschiedliche Weisen miteinander verknüpft werden. Als Steuerungsarchitekturen sind reaktive, deliberative und hybride Strukturen in der Literatur vorgeschlagen worden, die auf die jeweiligen Anwendungsszenarien zugeschnitten sind. Zur Steuerung humanoider Roboter müssen zahlreiche Teilprozesse und Aufgaben auf unterschiedlichen Abstraktionsebenen bearbeitet werden. In dieser Arbeit

wird untersucht, welche Struktur die Steuerungsarchitektur zur robusten, funktional-logischen Steuerung eines humanoiden Roboters besitzen muss. Es wird eine Steuerungsarchitektur vorgeschlagen und beispielhaft auf ein humanoides Robotersystem angewandt.

Softwarerahmenwerke Softwarerahmenwerke stellen eine Abstraktionsmethode sowie Werkzeuge zur Programmierung von komplexen Anwendungen dar. In einem Softwarerahmenwerk werden generische Funktionalitäten vorgegeben, die durch Anwendungssoftware näher spezifiziert werden können. Softwarerahmenwerke ähneln Softwarebibliotheken in dem Sinne, dass sie dem Nutzer wiederverwendbaren Code in einer klar definierten API zur Verfügung stellen. Allerdings findet bei Softwarerahmenwerken eine Umkehrung der Steuerung (*Inversion of Control*) statt. Das heißt im Gegensatz zur Verwendung von Bibliotheken, wird der Kontrollfluss nicht durch den Anwendungsprogrammierer vorgegeben, sondern durch das Softwarerahmenwerk. Eine wichtige Aufgabe des Softwarerahmenwerks ist es also, ein klares Entwurfsmuster vorzugeben und Schnittstellen zu definieren. Durch die Vorgabe des Kontrollflusses und die Bereitstellung des Zugriffs auf Basisfunktionalitäten, wird der Entwurf von Anwendungssoftware für komplexere Probleme deutlich erleichtert und bis zu einem gewissen Grad von der konkreten Hardware, auf der die Funktionalität realisiert werden soll, entkoppelt. In humanoiden Robotersystemen werden eine Vielzahl heterogener Hardwarekomponenten zu einem komplexen System zusammengefügt. Um aus Sicht der Software einen vereinheitlichten Zugriff auf die Hardwarekomponenten, der den unterschiedlichen Anforderungen an Echtzeitfähigkeit und Rechenleistung gerecht wird, zu ermöglichen, ist ein entsprechend strukturiertes Softwarerahmenwerk und eine leistungsfähige Middleware erforderlich. Ein Softwarerahmenwerk für humanoide Roboter sollte insbesondere in der Lage sein, Zugriff auf die Basisfunktionalitäten und -prozesse des Roboters zu ermöglichen. Ein wichtiger Aspekt, der beim Entwurf des Softwarerahmenwerks berücksichtigt werden muss, ist die Abstimmung des Kontrollflusses des Softwarerahmenwerks auf die durch die funktional-logische Architektur vorgegebenen Randbedingungen.

Abbildung auf Hardware-Software-Architektur Neben der funktional-logischen Steuerung und der Unterstützung des Anwendungsentwurfs durch ein Softwarerahmenwerk, muss noch eine Architektur für die tatsächlich verwendete Rechenplattform entworfen werden. Hier soll im Sinne eines Hardware-Software-Codesigns, parallel zu der Steuerungsarchitektur und zum Softwarerahmenwerk die Hardwarearchitektur für humanoide Roboter entwickelt werden. Aufgrund dieser gegenseitigen Wechselbeziehungen wird in dieser Arbeit der Begriff Hardware-Software-Architektur für die Gesamtheit aus Steuerungsarchitektur, Softwarerahmenwerk und Hardwarearchitektur verwendet. Seitens der Hardwarearchitektur besteht die Aufgabe darin, einerseits auf struktureller Ebene festzulegen, wie Komponenten miteinander interagieren, andererseits muss die Realisierbarkeit

auf konkreten Rechnerressourcen sichergestellt werden und Komponenten ausgewählt bzw. entworfen werden. Auf struktureller Seite wird die Fragestellung betrachtet, wie die Verteilung der Rechenressourcen gelöst wird und welche Verbindungen zwischen ihnen bestehen. Um die Topologie der Recheneinheiten festzulegen, muss festgelegt werden, an welcher Stelle im Roboter welche Rechenleistung zur Verfügung gestellt werden muss und von wo nach wo welche Datenflüsse mit den zugehörigen Zykluszeiten erforderlich sind. Es muss geklärt werden, über welche Schnittstellen Sensordaten in das Rechnersystem eingespeist werden können und wie der Zugriff auf die Aktorik erfolgen soll. Ebenso müssen Bandbreiten sowie Synchronisationsmechanismen und Protokolle spezifiziert werden. Auf der Realisierungsseite ergeben sich die gleichen Fragen, nur mit dem Fokus auf der konkreten Auswahl von Einheiten. Das heißt, mit welchen verfügbaren Komponenten lassen sich die erforderlichen Rechenleistungen erreichen, welches Bussystem erfüllt die Bandbreitenanforderungen und gegebenenfalls Echtzeitanforderungen der Datenübertragung, über welche Verfahren können Sensorwerte gewonnen werden, über welche Schnittstellen bzw. Protokolle werden sie dem Rechnersystem in digitaler Form bereitgestellt und mittels welcher Aktoren bzw. Leistungselektronik werden die generierten Bewegungsbefehle in die Tat umgesetzt?

Bei dem Entwurf von Steuerungssystemen für humanoide Roboter müssen neben der Sensor-Aktor-Koppelung auch Wahrnehmungsprozesse und letztlich auch kognitive Systemstrukturen implementiert werden. Klassische Ansätze aus dem Bereich der künstlichen Intelligenz weisen eine hohe Komplexität bezüglich der internen Umwelt-, Planungs- und Situationsrepräsentation auf. Lernverfahren erfordern erhebliche Gedächtnisleistungen und -strukturen. Schließlich werden auch aus der Biologie, der Bionik und der Verhaltenspsychologie Forderungen nach einem „Robotergehirn“ formuliert, die bisher an fehlenden Rechnerressourcen für humanoide Roboter scheitern. Während *offline*-Planungsverfahren und Situationsinterpretationsalgorithmen durchaus sinnvolle Ergebnisse zulassen, ist eine *online*-Umsetzung noch nicht in Sicht. Allerdings lassen sich auch hier die wichtigsten Prozesse, Datenflüsse und Repräsentationen strukturiert darstellen, so dass Rückschlüsse auf die notwendige Systemarchitektur gezogen werden können. Große Hoffnungen werden in diesem Zusammenhang auf konfigurierbare Systeme auf einem Chip (*SoC*) und auf die sich rasch entwickelnde *Multicore*-Technologie gesetzt. Bei klaren Robotersystemarchitekturansätzen lässt sich diese Technologie mit dem Ziel von Generik, Robustheit und Flexibilität nutzen!

1.3. Gliederung und Aufbau

Im Folgenden wird der Aufbau der Arbeit aufgezeigt. In Kapitel 2 werden bekannte Robotersystemarchitekturen beschrieben. Dazu werden unterschiedliche Softwarearchitekturen für autonome Roboter, verwendete Hardwarerealisierungen und Steuerungsarchitekturen diskutiert.

Weiterhin wird für ausgewählte humanoide Robotikprojekte das Zusammenspiel dieser drei Aspekte in der Hardware-Software-Architektur näher erläutert.

In Kapitel 3 wird auf die Vorgaben, die an einen humanoiden Roboter gestellt werden, eingegangen. Ausgehend von diesen Vorgaben wird erarbeitet welche Fähigkeiten der Roboter zur Erfüllung dieser Vorgaben bereitstellen muss. Aus den geforderten Roboterfähigkeiten wird das Anforderungsprofil abgeleitet, welches im weiteren als Grundlage für den Entwurf der Hardware-Software-Architektur dient.

Beginnend mit der funktional-logischen Steuerungsarchitektur wird in Kapitel 4 das Konzept für die modulare verteilte Hardware-Software-Architektur für humanoide Roboter vorgestellt. Hierzu wird analysiert, welche Funktionsblöcke die Steuerungsarchitektur bieten muss und wie Datenflüsse zwischen diesen Blöcken realisiert werden. In Anlehnung an die Steuerungsarchitektur werden die beiden weiteren Aspekte Softwarearchitektur und Hardwarearchitektur beschrieben. Ein Schwerpunkt liegt auf den Anforderungen im Bereich der Hardwarearchitektur und der Fragestellung, wie die Systemintegration von Recheneinheiten, Sensoren und Aktoren gelöst werden kann.

Kapitel 5 beschreibt die Entwicklung des humanoiden Robotersystems *ARMAR-III* als Evaluationsplattform. Erst wird der mechanische Systemaufbau und das Sensor-Aktor-System vorgestellt. Im Anschluss wird die konkrete Umsetzung der vorgestellten Hardware-Software-Architektur auf *ARMAR-III* beschrieben. Anhand eines komplexen Demonstrationsszenarios und anhand von Benchmarks wird die Leistungsfähigkeit der Hardware-Software-Architektur dargelegt.

Kapitel 6 gibt eine Zusammenfassung der wesentlichen Aspekte der Arbeit. Den Abschluss bildet ein Ausblick auf zukünftige Arbeiten.

2. Stand der Forschung

In diesem Kapitel werden die wichtigsten Arbeiten und Ansätze zu Hardware-Software-Steuerungen bei humanoiden Robotern vorgestellt. Zunächst wird ein Überblick über aktuelle Softwarerahmenwerke zur Programmierung der Roboterfähigkeiten gegeben. Danach wird aufgezeigt, welche Möglichkeiten zur Realisierung dieser Fähigkeiten und der dazu notwendigen Rechenprozesse in Hardware existieren. Im darauf folgenden Abschnitt wird auf unterschiedliche Ansätze zur Steuerung von Robotern aus funktional-logischer Sicht eingegangen. Anschließend werden die wichtigsten auf humanoiden Robotern implementierten Robotersteuerungen hinsichtlich ihrer Systemarchitektur und Rechnerressourcen untersucht und bewertet.

2.1. Software-Rahmenwerke und Middleware für autonome Roboter

Humanoide Roboter erfordern die Integration einer Vielzahl heterogener Hardwarekomponenten in einem komplexen System. Um aus Sicht der Softwaresystementwicklung einen vereinheitlichten Zugriff auf die heterogenen Hardwarekomponenten und Rechenmittel, der den unterschiedlichen Anforderungen an Echtzeitfähigkeit und Rechenleistung gerecht wird, zu ermöglichen, ist ein strukturiertes Softwarerahmenwerk und eine leistungsfähige Middleware erforderlich. In diesem Abschnitt wird eine Übersicht über domänenspezifische Softwarerahmenwerke und Middlewarelösungen gegeben.

Für den Entwurf komplexer Softwareprojekte kommen etablierte Vorgehensmodelle [17, 165], wie zum Beispiel das Wasserfallmodell, das Spiralmodell oder das in der VDI-Richtlinie 2206 „Entwicklungsmethodik für mechatronische Systeme“ beschriebene V-Modell [155] zum Einsatz. Diese Modelle beschreiben, wie der Prozess der Softwareentwicklung abläuft und strukturieren diesen in verschiedene Phasen und geben somit für die im Prozess auftretenden Aufgabenstellungen eine Ordnung vor. Auf der Seite der konkreten Softwareentwicklung existieren für die Umsetzung Entwurfsmuster [60, 59], die Lösungsansätze für wiederkehrende Entwurfsprobleme anbieten und somit eine Vorlage zur Problemlösung darstellen. Durch die Verwendung von Entwurfsmustern, die sich für ähnliche Problemstellungen wiederverwenden lassen, kann der Aufbau modular erfolgen, wie es zum Beispiel in [120, 122, 121] vorgeschlagen wird.

2.1.1. aRD - Agile Robot Development

Bei dem „Deutschen Zentrum für Luft- und Raumfahrt“ (DLR) in Oberpfaffenhofen wurde zur Entwicklung einer Steuerungssoftware für das dort realisierte Robotersystem *Justin*, bestehend aus zwei Roboterarmen und zwei Fünf-Finger-Händen, das Rahmenwerk „Agile Robot Development“ (aRD) eingeführt [33]. Die Systemkomplexität von *Justin* und anderen raumfahrttauglichen Systemen erforderte ein neues Softwarekonzept zur Entwicklung und zum Betrieb dieser Robotersysteme [116, 24]. aRD stellt einen flexiblen, modularen und verteilten Softwareentwurf zur Entwicklung komplexer mechatronischer Systeme sicher. Die Systemkomplexität der betrachteten Systeme liegt vor allem in deren großer Anzahl an Bewegungsfreiheitsgraden, der Vielzahl an Aktoren und Sensoren und den Interaktionen sämtlicher Teilsysteme.

aRD - Systemstruktur

Bei der Entwicklung von aRD wurde Echtzeitfähigkeit, Skalierbarkeit und die Möglichkeit zur verteilten Ausführung der Programme als zentrale Ziele vorgegeben [35]. aRD basiert auf einem auf die technischen Systemkomponenten zugeschnittenen Ansatz. Dies unterstützt Flexibilität bei dem Softwareentwurf, erlaubt die Wiederverwendbarkeit von Code und die Verteilbarkeit der Software auf mehrere Rechner [68]. aRD kann für mechatronische Systeme mit bis zu 100 Bewegungsfreiheitsgraden, dezentraler Elektronik zur Sensorauswertung und Aktoransteuerung Zykluszeiten im Bereich von 1 ms bis hinunter zu 0,1 ms im Echtzeitbereich realisieren. Das Robotersystem wird dabei als dezentrales Netz aus Recheneinheiten und Verbindungseinrichtungen gesehen.

Mit *Multicore*-Rechnern mit 3 GHz-Taktung und Bussystemen mit Übertragungsraten im Bereich von 4 Mbit bis 1 Gbit, wie zum Beispiel *switched* Gigabit-Ethernet, ist der Aufbau verteilter Computerarchitekturen, die auch unter Echtzeitanforderungen skalierbar sind, möglich [33]. Es wird eine Entkopplung der Systemarchitektur von den Details der Hardwarerealisierung angestrebt, wodurch eine funktionale Sicht auf das Robotersystem möglich wird. Das Robotersystem besteht aus Funktionskomponenten, die auf das Echtzeitsystem, in dem die Steuerungskomponenten implementiert sind, zugreifen, und höheren Planungs- und Entscheidungsebenen inklusive der Benutzerschnittstelle, die auf einem Netzwerk von Rechenmitteln unter weniger strengen Echtzeitbedingungen arbeiten (Abb. 2.1). aRD stellt Werkzeuge zur Entwicklung dieser funktionalen Aspekte und zur Übertragung der Steuerungssoftware auf die Ziel-Hardware zur Verfügung. Die Funktionalität wird in so genannte Blöcke geordnet, wobei die Granularität der Blöcke im Echtzeitteil feiner ist als im Nicht-Echtzeitteil. Für den Aufbau der Blöcke gelten die folgenden Entwurfs-Entscheidungen: Blöcke sind gleichartig, das heißt es wird keine Client-Server-Struktur aufgebaut, sondern Blöcke können sowohl Datenlieferanten als auch Datenempfänger sein. Bei übereinstimmendem Datenformat kann ein Ausgangsport zu einem beliebigen Eingangsport eines anderen Blocks verbunden werden. Blöcke stellen eine Ausfüh-

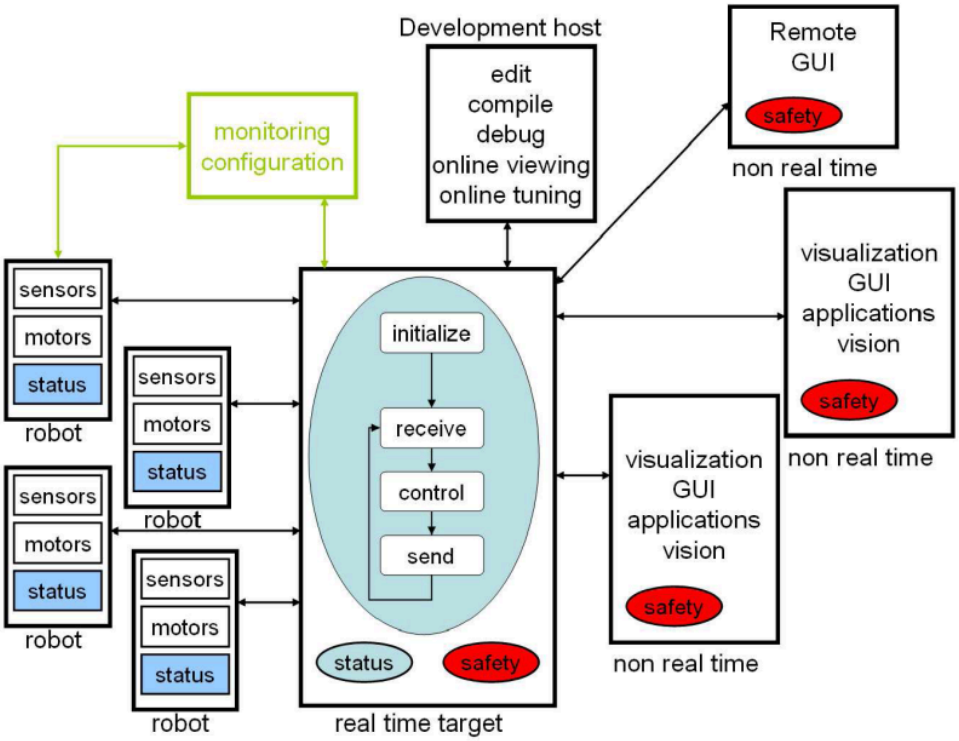


Abb. 2.1.: Systemstruktur von aRD - Verbindung mehrerer Roboterkomponenten mit dem Echtzeitteil und Netzwerk der nicht echtzeitfähigen Rechnerressourcen zur Handlungsplanung und Benutzerinteraktion (Quelle: [33])

rungseinheit dar, der eine feste Priorität zugewiesen ist. Die Ausführungsreihenfolge mehrerer Blöcke wird damit implizit durch die Prioritäten vorgegeben. Mehrere Blöcke können zu Gruppen zusammengefasst werden, wobei sich diese Gruppe dann nach außen hin als eine Ausführungseinheit darstellt und intern die Ausführung der Blöcke übernimmt. Die Ausführung kann sowohl zyklisch als auch durch Dateneingang getriggert erfolgen. Prinzipiell sind unterschiedliche Zykluszeiten möglich, da die Blöcke im Echtzeitteil aber synchronisiert ausgeführt werden müssen, können nur Zykluszeiten verwendet werden, die ein ganzzahliges Vielfaches der kürzesten Zykluszeit sind. Das Datenformat der Blöcke ist variabel, kann zur Laufzeit aber nicht geändert werden. Das gleiche gilt für die Verbindung der Blöcke untereinander. Da Blöcke über ihre Eingangsports aktiviert und deaktiviert werden können, lässt sich trotzdem durch feste Verdrahtung mehrerer Alternativen ein Verhalten, wie bei Verwendung veränderlicher Verbindungen, erreichen.

aRD - Kommunikation

Die Kommunikation zwischen den Blöcken erfolgt ungepuffert und nicht blockierend, so dass bei Blöcken mit unterschiedlichen Zykluszeiten nur das letzte gültige Datum gelesen wird. Der Datenversand erfolgt ausschließlich durch den Sender initiiert, also durch ein *push* der Daten auf den Ausgangsport des Blocks. aRD stellt die Infrastruktur für das Netz der Blö-

cke zur Verfügung, die Funktionalität muss in den Blöcken selbst implementiert werden. Es wird eine Schnittstelle zur Programmierung der Blöcke und zur Integration in das Netz der Blöcke zur Verfügung gestellt. Mit aRDnet existiert eine Software-Suite zur komfortablen Implementierung der Ein- und Ausgangsports der Blöcke. aRDnet stellt eine C/C++-Schnittstelle zur Verfügung und bietet die fünf Funktionen: `create`, `init` zur Erzeugung und Initialisierung der Ein- und Ausgangsports, `send` für nichtblockierendes Senden über den Ausgangsport, `rec` und `tryrec` für blockierendes bzw. nichtblockierendes Empfangen über den Eingangsport [34]. Verbindungen zwischen Blöcken werden zwischen gleichnamigen Ports hergestellt. Da für Eingangsports und Ausgangsports Namen global nur einmal vergeben werden, ist diese Verbindung eindeutig. Die Kommunikation unter Blöcken auf unterschiedlichen, heterogenen Rechnern wird durch die Verwendung kompatibler Basisdatentypen ermöglicht und durch das Programm *ardnet* [33] hergestellt, das auf beiden Rechnern als Netzwerkbrücke läuft und über UDP Daten überträgt. *ardnet* fungiert zusätzlich als Portmultiplikator für Fälle, in denen die Daten einer Quelle an mehreren Eingängen benötigt werden. Dazu empfängt *ardnet* an seinem Eingang die Daten und kopiert sie auf mehrere seiner Ausgänge.

aRD - Programmierumgebung

Zur Implementierung der Funktionalität wird eine Werkzeugkette aus Matlab/Simulink/RTW (Realtime Workshop)[1] und RTLab[2] verwendet. Mittels RTLab können Simulink-Modelle halbautomatisch für mehrere CPUs parallelisiert werden. Dazu muss das Simulink-Modell derart in Teilmodelle untergliedert sein, dass diese Untergliederung der Aufteilung auf die verteilten Rechnerressourcen entspricht. Sämtliche Kommunikations- und Synchronisationsverbindungen der verteilt ausgeführten Blöcke werden automatisch durch RTLab erzeugt. Die Aufteilung des Simulink-Modells ergibt sich bei Robotersystemen durch deren Teilsysteme. Das aRD-Konzept beinhaltet einen *Wrapper* für Simulink-Funktionen und erlaubt somit eine leichte Integration von Simulink-Code auf das Robotersystem.

Zum Start des dezentralen Netzes der Blöcke nutzt aRDnet eine Reihe hierarchisch gegliederter Shell-Skripte. Über das `ardstart`-Kommando können Blöcke auf entfernten Rechnern gestartet werden. Mittels des `ardkill`-Kommandos werden Blöcke ordnungsgemäß beendet. Um beim Herunterfahren des Robotersystems gegenseitige Abhängigkeiten der Blöcke zu berücksichtigen und die `ardkill`-Kommandos in der richtigen Reihenfolge auf die ausgeführten Blöcke anzuwenden, wird protokolliert welcher Block auf welcher Recheneinheit gestartet wurde.

[1] `http://www.mathworks.com`
[2] `http://www.opal-rt.com`

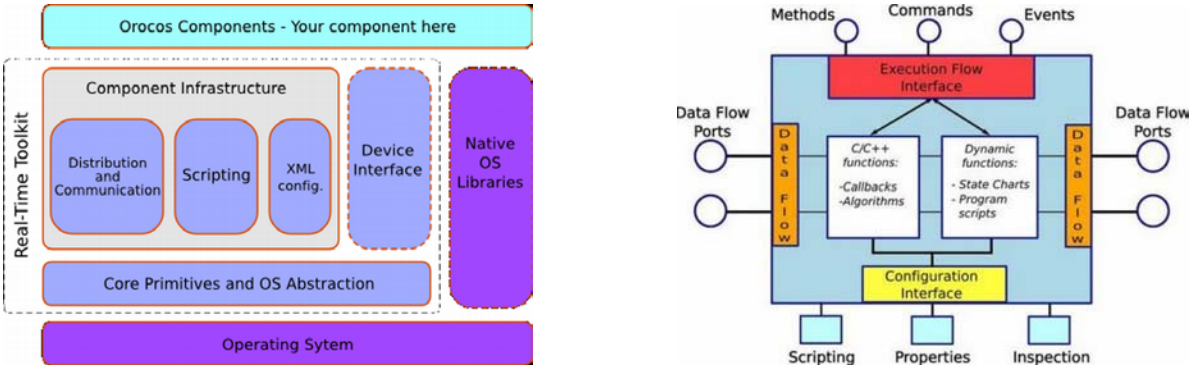


Abb. 2.2.: Strukturbild OROCOS (Quelle [144])

2.1.2. OROCOS - Open Robot Control Software

„Open Robot Control Software“ (OROCOS[3]) ist ein komponentenbasiertes Softwarerahmenwerk zur Unterstützung des Entwurfs von Robotersteuerungssoftware, das im Rahmen eines EU-Projektes an der K.U. Leuven, am „Laboratoire d’Analyse et d’Architecture des Systèmes“ (LAAS) des „Centre National de la Recherche Scientifique“ (CNRS) und an der „Kungl Tekniska Högskolan“ (KTH, Schweden) entwickelt wurde.

OROCOS - Systemstruktur

Bei OROCOS wird eine Standardspezifikation als Schlüssel zur Wiederverwendbarkeit von Softwaremodulen gesehen [108]. Es soll Entwicklern ermöglicht werden, bestehenden Code zu nutzen bzw. Beiträge zu Funktionalitäten zu leisten, ohne die Details des Rahmenwerks kennen zu müssen. Um dies zu erreichen, wurden folgende wesentliche Designentscheidungen getroffen: eine klare Trennung von Struktur und Funktionalität, ein generischer Kern, auf dem weitere Anwendungen aufbauen können, das Bereitstellen von portierbaren Schnittstellen zu Betriebssystemen und Geräten und eine ereignisgesteuerte Ablaufsteuerung [32]. Die Struktur von OROCOS ist komplett modular [31, 143, 145]. Es besteht einerseits aus Hilfsmodulen, die keinen direkten Robotikbezug haben, wie 3D-Visualisierung, numerischen Bibliotheken oder Dokumentationstools. Auf der anderen Seite stehen die Robotikmodule, die die tatsächliche Funktionalität des Roboters beinhalten, beispielsweise Module für Dynamikberechnung, Kinematikberechnung, Servoregelung oder Bewegungsplanung. OROCOS-Komponenten basieren auf dem bereitgestellten *Real-Time Toolkit* (Abb. 2.2).

Als Grundlage wird ein generischer Roboter definiert, dessen Komponenten sich standardisiert verhalten sollen, vollständig kompatibel zueinander aufgebaut sein sollen und von außen parametrisierbar und steuerbar sind. Komponenten entsprechen einem Softwareobjekt, das seine Dienste über eine sprachenunabhängige Schnittstelle zur Verfügung stellt und dynamisch mit

[3] `http://www.orocos.org/`

anderen Komponenten zu einem Netzwerk verbunden bzw. wieder entfernt werden kann.

OROCOS - Kommunikation

Um die Dienste und Steuerungskomponenten rechnerunabhängig zu gestalten, wird die Kommunikation außerhalb der Komponenten realisiert. Mittels einer Komponentenbeschreibungssprache wird der Kommunikationsbibliothek bekannt gegeben, welche Dienste die Komponente zur Verfügung stellt und wie das Datenaustauschformat aussieht. Die Kommunikationsbibliothek stellt weiterhin einen *shared memory* Bereich zur Verfügung in dem Komponenten globale Variablen auslesen bzw. modifizieren können. Neben der Plattformunabhängigkeit fordert OROCOS prinzipiell auch keine bestimmte Programmiersprache, üblicherweise wird jedoch für den echtzeitfähigen Teil C und ansonsten C++ als Programmiersprache verwendet.

Die Ablaufsteuerung lässt sich als gerichteter Graph darstellen und ist frei von Schleifen. Die Kommunikation zwischen einzelnen Komponenten ist auf einer *producer-consumer* Beziehung aufgebaut, eine Verteilung auf unterschiedliche Rechner lässt sich somit durch Auftrennen des Graphen an jeder dieser *producer-consumer*-Relationen erreichen. Somit lässt sich OROCOS verteilt und plattformübergreifend einsetzen.

OROCOS - Programmierumgebung

Die Implementierung der Komponenten stützt sich auf etablierte Standards wie CORBA und die dazugehörige „Interface Description Language“ (IDL), das Internet-Inter-Orb-Protocol und auf die Echtzeiterweiterung TAO[4]. Zur Generierung dieser Komponenten existiert das Rahmenwerk $G^{en}{}_{o}M$ [108]. In diesem Modell bestehen Komponenten aus folgenden drei Einheiten: einem Satz von Algorithmen, einer Ausführungseinheit und einer Kommunikationsbibliothek. Die Algorithmen werden in Form so genannter *Codels* (*Code elements*) implementiert. Ein *Codel* entspricht einem nichtunterbrechbaren und somit atomaren Softwareteil. Die *Codels* können zu *Codel*-Bibliotheken zusammengefasst werden und stellen den ohne Anpassungen portierbaren Teil der Software dar. Der Zugriff auf die *Codels* erfolgt vergleichbar einem Funktionsaufruf, sie erhalten Eingabe-Daten und liefern eine Ausgabe zurück. Die Ausführungseinheit überwacht die Ausführungsabfolge der *Codels* bzw. der Dienste und überwacht den internen Zustand der Komponenten und den Steuerungsdatenfluss. Der Aufruf der Dienste kann durch den Benutzer oder durch eine Ablaufsteuerung, beispielsweise als endlicher Automat oder als Petrinetz, realisiert werden. Grundsätzlich werden Dienste ereignisgesteuert aktiviert. Durch diese Struktur wird jedoch noch keine Steuerungsarchitektur vorgegeben, es lassen sich sowohl verhaltensbasierte, deliberative als auch reaktive Architekturen realisieren.

[4] `http://www.cs.wustl.edu/~schmidt/TAO.html`

2.1.3. Player

Player[5] wurde im Robotics Research Lab der University of Southern California (USC) zur Entwicklung von Steuerungssoftware für die dort eingesetzte mobile Roboterplattform *Pioneer*[6] der Firma ActivMedia entwickelt. Der Realisierung von Player lag die Vorstellung einer generischen Roboterschnittstelle zugrunde, die den Zugriff auf die Roboterhardware abstrahiert und dem Entwickler der Steuerungssoftware in einem Client-Server-basierten Ansatz über abstrakte Funktionen präsentiert [61, 154]. Dabei läuft der Server auf dem Roboter und dient als Hardwareabstraktionsebene zur tatsächlichen Roboterhardware.

Player - Systemstruktur

Die zentrale Abstraktionsschicht wird als *Player Abstract Device Interface* bezeichnet. Die Nutzerprogramme zur Steuerung des Roboters verbinden sich als Clients mit dem Server und erhalten über diesen Zugriff auf die Aktoren und Sensoren des Roboters. Die Kommunikation zwischen Client und Server erfolgt über TCP-Sockets. Player ist in C++ implementiert und verwendet eine POSIX-kompatible *pthread*-Schnittstelle für Multithread-Anwendungen. Durch die Client-Server-Aufteilung können Clients auf jedem Rechner implementiert werden, der über eine Ethernet-Schnittstelle verfügt und TCP-Sockets unterstützt.

Serverseitig folgt die Sichtweise auf die Roboterhardware dem *device-as-file*-Modell [53], welches Geräte in der Behandlung mit Dateien gleichsetzt. Im Fall der Sensoren und Aktoren des Roboters handelt es sich um zeichenorientierte Geräte, die mit Datenströmen arbeiten. Für den Zugriff auf diese Geräte implementiert Player folgende Befehle: `open`, `close`, `read`, `write` und `ioctl`. Zum Auslesen eines Sensorwerts wird mit `open` das Gerät geöffnet, anschließend über `read` der Sensorwert ausgelesen und nach erfolgtem Zugriff das Gerät mit `close` geschlossen. Analog gilt dieses Vorgehen für das Schreiben von Kommandos an die Aktoren. Neben den expliziten Lese- und Schreibzugriffen auf die Geräte implementiert Player noch eine weitere Art der Datenübertragung über das `ioctl`-Kommando (input/output-control). Das `ioctl`-Kommando stellt einen synchronen Zugriff auf persistente Daten, wie beispielsweise interne Konfigurationsdaten der Geräte zur Verfügung. Für jedes Gerät läuft ein eigener Thread, wobei der Datenaustausch unter den einzelnen Threads über einen gemeinsam genutzten globalen Adressbereich erfolgt (Abb. 2.3). In der aktuellen Player-Implementierung ist der simultane Zugriff mehrerer Clients auf ein und dasselbe Gerät möglich. Da das *device-as-file*-Modell für zeichenorientierte Geräte keinen Interruptmechanismus vorsieht, müssen Clients sämtliche Geräte zyklisch abfragen, um aktuelle Daten zu erhalten. Player unterstützt dafür Zykluszeiten von 200 ms bis hinunter zu 10 ms.

[5] `http://playerstage.sourceforge.net/`
[6] `http://www.activrobots.com/ROBOTS/p2dx.html`

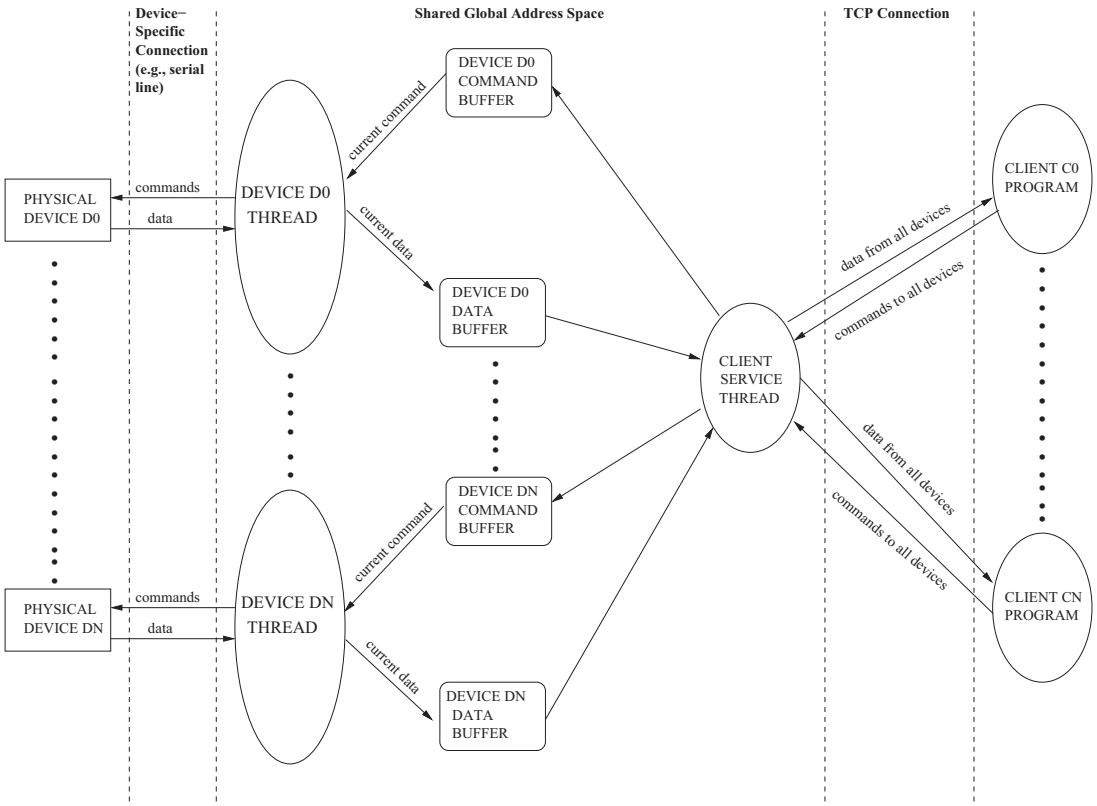


Abb. 2.3.: Systemstruktur von Player (Quelle: [61])

Player - Kommunikation

Als Abstraktion zwischen Player-Server und der Roboterhardware werden Gerätetreiber verwendet, die eine für Geräteklassen einheitliche Schnittstelle zur Verfügung stellen. Die Aufgabe des Gerätetreibers ist es einerseits, die Sensordaten in für die Schnittstelle angepasste Eingabedatenströme umzuwandeln und andererseits, den Ausgabedatenstrom in Kommandos für die Aktoren umzusetzen. Eine Designentscheidung beim Entwurf der Schnittstelle ist, welche Eigenschaften des Geräts über die Schnittstelle verfügbar gemacht werden. Üblicherweise wird versucht, die in der Geräteklasse gemeinsam vorhandenen Eigenschaften über die Standardschnittstelle über `read` und `write` bereitzustellen und den Zugriff auf spezielle Eigenschaften nur über geräteabhängige `ioctl`-Kommandos zu erlauben. Dadurch wird eine größtmögliche Geräteunabhängigkeit erreicht, da Anwendungsprogramme clientseitig in Bezug auf die Schnittstelle programmiert werden können und keinen gerätespezifischen Code enthalten. Im Sinne des ISO-OSI-Schichtenmodells [147] definiert Player die Schichten 4-7, also Transportschicht, Sitzungsschicht, Darstellungsschicht und Anwendungsschicht.

Player - Programmierumgebung

Für Player gibt es clientseitig bereits einige *Utilities*, die unter anderem in C++, Tcl, Java oder Python implementiert sind. Als Simulationsumgebungen stehen Stage für Indoor-Umgebungen in 2D und Gazebo für Outdoor-Umgebungen in 3D zur Verfügung. Dank der klaren Client-Server-Aufteilung und der Schnittstellendefinition ist der Portierungsaufwand von Anwendungsprogrammen zwischen Robotern ähnlicher Hardware und von in der Simulation entwickelten Programmen gering.

Durch die Struktur des Servers kommt es neben den Latenzen der Netzwerkverbindung durch die Verarbeitung der Daten im Server noch zu weiteren Verzögerungen im Datenaustausch. Diese betragen erfahrungsgemäß circa die Hälfte der verwendeten Zykluszeit. Üblicherweise läuft Player mit einer Updaterate von 10 Hz, was zu einer Latenz von 50 ms führt [62]. Seitens der Fehlertoleranz gibt es keine nennenswerten Mechanismen. Im Robotics Research Laboratory der University of Southern California wurden mit Player unter anderem die Steuerung für die Roboter *Pioneer 2* und *RWI* entwickelt.

2.1.4. MARIE - Mobile and Autonomous Robotics Integration Environment

Das „Mobile and Autonomous Robotics Integration Environment“ (MARIE[7]) wurde in Kanada an der Université de Sherbrooke entwickelt und verfolgt zur Steigerung der Wiederverwendbarkeit von Code einen komponentenorientierten Ansatz. MARIE unterstützt die Integration von Software aus anderen Robotikprojekten und kann selbst andere Softwarerahmenwerke einbinden. Bei der Entwicklung von komplexen Robotersystemen wurde die Integration unterschiedlicher Softwarekomponenten als ein zentraler Punkt erkannt [48, 109, 46, 47, 105]. Neben dem oben genannten Aspekt der Software-Wiederverwendbarkeit sollen noch die folgenden Ziele erreicht werden: Unterstützung verteilten Rechnens auf heterogenen Plattformen, Verwendbarkeit unterschiedlicher Kommunikationsprotokolle und eine klare Definition von Schnittstellen, Rahmenwerken und Abstraktionsebenen.

MARIE - Systemstruktur

Zum Erreichen dieser Zielsetzung setzt MARIE auf folgende Designentscheidungen: Anwendung eines Vermittler-Entwurfsmusters, Schichtenmodells und die Abstraktion des Kommunikationsprotokolls. Anstatt, wie andere Softwarerahmenwerke ein spezielles Kommunikationsverfahren auszuwählen [49], setzt MARIE zur Interaktion der Komponenten untereinander auf das Vermittler-Entwurfsmuster (*Mediator Design Pattern* [60, 59]). Der Mediator ist die zentrale Einheit, über den der Datenaustausch unter den Komponenten abgewickelt wird (Abb. 2.4). Durch den Einsatz des Mediators wird die Kommunikation zwischen einzelnen Komponenten

[7] `http://marie.sourceforge.net/mediawiki/index.php/Main_Page`

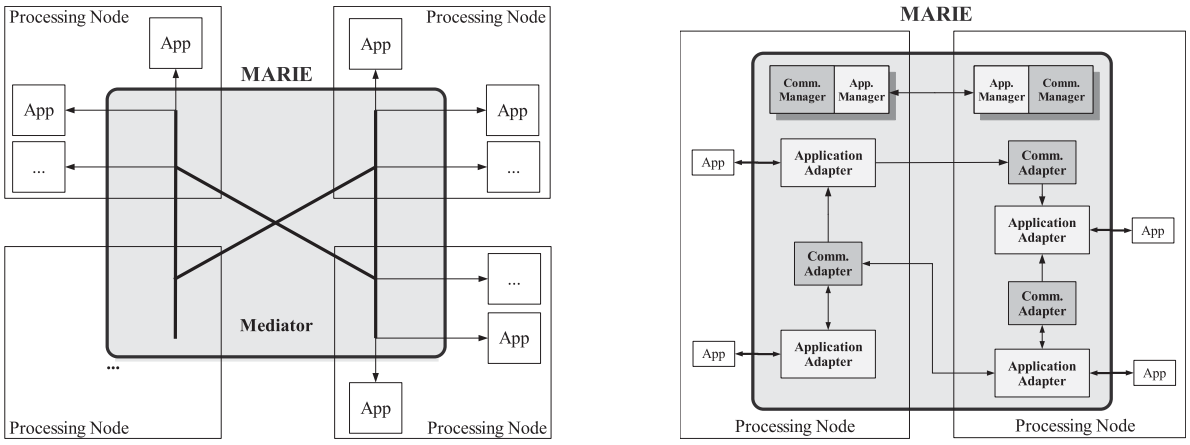


Abb. 2.4.: Strukturbild MARIE (Quelle [49])

ersetzt durch die Kommunikation von Komponente_A mit Mediator und Mediator mit Komponente_B. Dadurch muss nicht jede Komponente alle Kommunikationsprotokolle ihrer potenziellen Kommunikationspartner unterstützen, sondern es muss nur erfüllt sein, dass jede Komponente ein Kommunikationsprotokoll implementiert, welches der Mediator unterstützt. Allerdings wird durch dieses Vorgehen der Mediator selbst sehr komplex, was zu schlechter Code-Verständlichkeit, Wartbarkeit und Erweiterbarkeit führen kann.

Auf Seiten der Softwarearchitektur verwendet MARIE eine Aufteilung in drei Abstraktionsebenen. Auf der so genannten *Core Layer* wird die Unterstützung für Kommunikationsdienste, Datenverarbeitung, verteiltes Rechnen und *low-level* Betriebssystemoperationen wie Speicherzugriff, I/O und Threadkontrolle bereitgestellt. Oberhalb dieser Ebene spezifiziert die *Component Layer* Rahmenwerke zur Implementierung von Komponenten. Die *Application Layer* stellt Tools zum Erstellen und Überwachen von Anwendungsprogrammen zur Robotersteuerung bereit und realisiert den Zugriff auf die in den unteren zwei Schichten verfügbaren Komponenten.

MARIE - Kommunikation

MARIE ist nicht auf ein Kommunikationsprotokoll festgelegt, sondern implementiert unterschiedliche Kommunikationsmechanismen. Dies geschieht mit der als *port* bezeichneten Abstraktion des Kommunikationsprozesses. Der Client greift zum Senden und Empfangen von Daten auf einen so genannten *concrete port* zu, welcher die übermittelten Daten in das Format des so genannten *abstract ports* bringt. Dieser *abstract port* ist dann für die Ausführung und Überwachung des Kommunikationsprotokolls zuständig. Zur Implementierung der Kommunikationsprotokolle wird das *ADAPTIVE Communication Environment* (ACE[8]) verwendet. Üblicherweise kommt als Protokoll TCP mit Sockets zum Einsatz.

[8] `http://www.cs.wustl.edu/~schmidt/ACE.html`

MARIE - Programmierumgebung

Neben dem Systementwurf, den MARIE beschreibt, existieren noch die beiden Tools RobotFlow und FlowDesigner, die auf Seite der Anwendungsprogrammierung den Entwurf von wiederverwendbarer Software unterstützen. Die Kombination aus RobotFlow und FlowDesigner erlaubt den Entwurf und die Visualisierung von Programmblöcken. Durch die Darstellung der Komponentenverbindungen und die Anzeige der über diese Verbindungen übertragenen Daten wird der Debugprozess unterstützt. Das Einbringen und Entfernen dieser Anzeigeelemente erfolgt ohne compilativen Übersetzungsvorgang und beschleunigt somit die Entwicklung. FlowDesigner strukturiert das Anwendungsprogramm in wiederverwendbare Funktionsblöcke, die über klar definierte Schnittstellen verfügen (*input*, *output*, *parameters* und *scheduling*). Eine Vernetzung mehrerer Funktionsblöcke erfüllt komplexere Aufgaben und kann ihrerseits als so genannter super-block wiederum in anderen Kombinationen aus Funktionsblöcken wie ein normaler Funktionsblock eingesetzt werden.

In den Funktionsblöcken werden die Ein- und Ausgabewerte einmal je Zyklus berechnet und dann gepuffert, so dass bei einer Ausgabeanforderung nicht erst die Berechnung gestartet werden muss. Das Scheduling erfolgt implizit, indem jeder Funktionsblock rekursiv Daten für alle seine Eingänge anfordert. Durch dieses implizite Scheduling können Netzwerke aus vielen kleinen Funktionsblöcken ohne Effizienzprobleme aufgebaut werden. Durch diese Struktur ist RobotFlow besonders geeignet für sequenzielle Datenflüsse. Threads können in Blöcken realisiert werden und erlauben parallele bzw. verteilte Berechnungen mit unterschiedlichen Updateraten. Durch eine bereits implementierte Schnittstelle zu Player/Stage (Abschnitt 2.1.3) unterstützt RobotFlow alle mit Player/Stage steuerbaren Roboter.

2.1.5. MCA2 - Modular Controller Architecture

Die „Modular Controller Architecture“ (MCA2[9]) wurde am „Forschungszentrum Informatik“ (FZI) in Karlsruhe entwickelt [56]. Bei der Entstehung von MCA2 stand die Unterstützung für die schnelle Entwicklung unterschiedlicher Roboterprototypen unter Echtzeitanforderungen im Vordergrund [136]. Durch die komponentenbasierte Struktur des Softwarerahmenwerks mit klar definierten Schnittstellen wird die Unterteilung des Gesamtsystems in kleine wiederverwendbare Funktionsblöcke ermöglicht. Die Kommunikation und Synchronisation zwischen diesen Funktionsblöcken wird vollständig von dem Rahmenwerk übernommen. MCA2 verwendet als Programmiersprache C++ und kann unter den Betriebssystemen Windows, Linux und MacOS eingesetzt werden. Zur Erfüllung der Echtzeitanforderungen kommt RTAI/LXRT[10] zum Einsatz. Die grafische Oberfläche für die Benutzerschnittstelle ist mittels QT implementiert.

[9] `http://www.mca2.org`
[10] `https://www.rtai.org/`

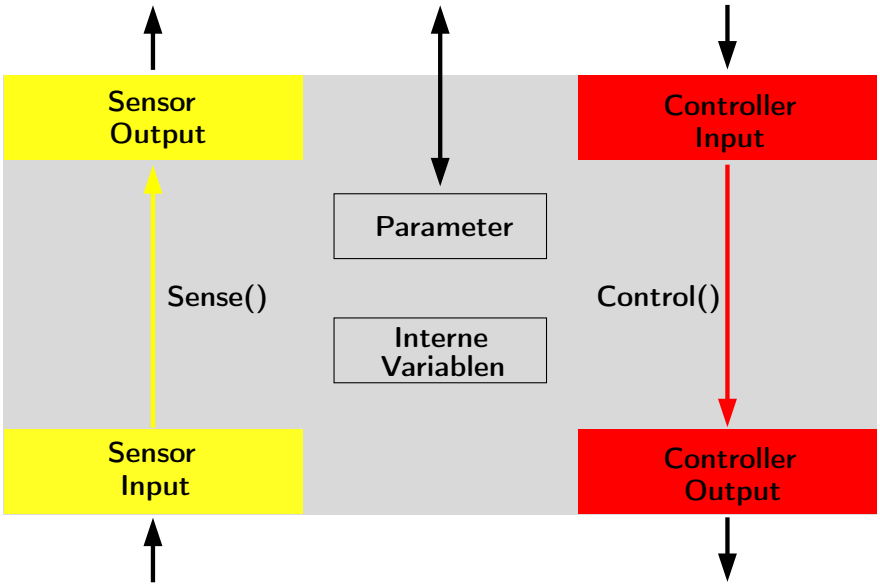


Abb. 2.5.: MCA-Modul: Innerhalb des Moduls werden zyklisch die Funktionen `Sense()` und `Control()` ausgeführt. Diese erhalten jeweils an der Eingangsschnittstelle neue Daten und schreiben ihre Ergebnisse in die Ausgangsschnittstelle. Über die Parameterschnittstelle können die beiden Funktionen parametrisiert werden.

MCA2 - Systemstruktur

Da MCA2 für die Softwareentwicklung auf unterer und mittlerer Robotersteuerungsebene ausgelegt ist, wird ein streng hierarchisch und zeitgesteuerter Ansatz verfolgt [135, 150]. Die unterste Hierarchieebene entspricht dabei der Schnittstelle zur Roboterhardware bzw. den Gerätetreibern. Module, die in der Hierarchie weiter oben angesiedelt sind, erfüllen komplexere und abstraktere Aufgaben. Der Datenfluss im System untergliedert sich im Wesentlichen in zwei Datentypen: Sensordaten, die von unten durch die Hierarchieebenen von Modul zu Modul in der Hierarchie nach oben weitergeleitet und dabei modifiziert werden können, und Steuerungsdaten, die von oben durch die Hierarchieebenen von Modul zu Modul in der Hierarchie nach unten weitergeleitet bzw. verändert werden.

Die wesentlichen Bausteine von MCA2 sind: Module, Gruppen und Kanten. Dabei sind Module die kleinste funktionale Einheit. Die Kommunikation zwischen den Modulen erfolgt über die so genannten Kanten (Abb. 2.5). Mehrere Module können zu einer Gruppe verknüpft werden und somit komplexere Funktionen erfüllen. Innerhalb eines Moduls werden die beiden Funktionen `Sense()` und `Control()` periodisch ausgeführt. Für den Sensordatenfluss gibt es die Schnittstellen *Sensor Input* und *Sensor Output*. Innerhalb der `Sense()`-Funktion werden die Daten vom *Sensor Input* eingelesen, weiterverarbeitet und über *Sensor Output* in der Hierarchie aufwärts zum nächsten Modul weitergegeben. Für den Kontrolldatenfluss ist analog die Funktion `Control()` zuständig, die Daten vom *Controller Input* einliest, weiterverarbeitet und über den *Controller Output* in der Hierarchie nach unten weiterleitet. Der Datenaustausch zwischen

`Sense()` und `Control()` innerhalb des Moduls erfolgt über lokale Variablen, die gleichzeitig den internen Zustand des Moduls darstellen. Der Aufruf der `Sense()`- und `Control()`-Funktion erfolgt periodisch in frei konfigurierbaren Zeitintervallen. Als Datentyp der Modulschnittstellen werden *double*-Werte verwendet. Neben der eigentlichen Datenschnittstelle existiert noch eine Schnittstellenbeschreibung die textuell die Bedeutung der Kante beschreibt. Über ein *changed*-Flag wird den Funktionen `Sense()` und `Control()` mitgeteilt, ob sich die Daten an der Schnittstelle seit dem letzten Zyklus verändert haben. So muss die Funktion nur ausgeführt werden, wenn sich an den Eingängen etwas verändert hat. Für Daten, die sich während der Laufzeit üblicherweise nicht verändern, wie zum Beispiel Konfigurationsdaten für die Programmausführung, existiert noch die Parameterschnittstelle. Beispielsweise lassen sich über Parameter die Zykluszeiten des Programms, Reglerparameter oder Maximalgeschwindigkeiten einstellen. Parameterwerte können einerseits bereits in das Programm hinein kompiliert sein oder beim Start über eine Konfigurationsdatei geladen werden, andererseits können sie sogar noch zur Laufzeit modifiziert werden. Da Module definitionsgemäß einen geringen Komplexitätsgrad haben sollen und nur kleine Funktionsblöcke beinhalten sollen, müssen zum Erreichen komplexerer Funktionalitäten mehrere Module kombiniert werden. Die Verbindung mehrerer Module wird als Gruppe bezeichnet und erfolgt immer nach dem Schema, dass der Output eines Moduls mit dem zugehörigen Input eines oder mehrerer anderer Module verbunden wird. Es wird also *Sensor Output* mit *Sensor Input* und *Controller Output* mit *Controller Input* verbunden. Innerhalb der Gruppe kann so eine mehrstufige Hierarchie aufgebaut werden (Abb. 2.6). Nach außen hin verhalten sich Gruppen genau so wie ein einzelnes Modul. Sie verfügen ihrerseits wieder über die Schnittstellen *Sensor Input/Output* und *Controller Input/Output*, jedoch über keine Parameterschnittstelle. Die Abarbeitung der Module der Gruppe wird von der `Sense()`- und der `Control()`-Funktion der Gruppe erledigt. Da Gruppen über die gleichen Schnittstellen verfügen wie ein Modul, können sogar Gruppen zusammen mit anderen Modulen eine komplexere Gruppe bilden. Die Abarbeitungsreihenfolge der Module innerhalb der `Sense()`-Funktion erfolgt ebenenweise vom Modul links unten zum Modul rechts oben. Die `Control()`-Funktion wird in umgekehrter Reihenfolge ausgeführt.

MCA2 - Kommunikation

Die grundlegende Ausführungseinheit in MCA2 ist der Part; er ist zuständig für die Initialisierung der Ausführungsumgebung, die Kommunikation mit anderen Parts über TCP/IP und die Ausführung der Haupt-Ausführungsschleife – der *Control Loop*. Parts stellen die Schnittstellen der Gruppe anderen Parts über so genannte *inter-part*-Kanten zur Verfügung. Die *Control Loop* des Parts wird zyklisch mit einer einstellbaren Frequenz ausgeführt. Um Multithreading zu erreichen, können MCA-Anwendungen mit Hilfe von *Thread-Containern* in mehrere Threads aufgeteilt werden. Jeder *Thread-Container* ist für die Ausführung eines einzigen Moduls bzw.

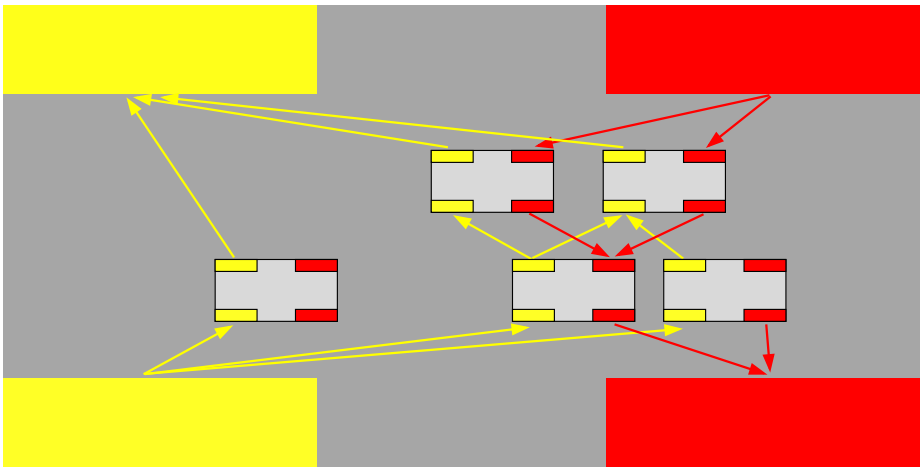


Abb. 2.6.: Verbindung mehrerer MCA-Module zu einer MCA-Gruppe

einer Gruppe zuständig. Für Daten, die sich nicht sinnvoll über die auf *double*-Werte beschränkte Modulschnittstelle austauschen lassen, existieren *Blackboards*. Über diese können größere Datenmengen übertragen werden, wie zum Beispiel Kamerabilder. Der Zugriff auf die *Blackboards* erfolgt netzwerktransparent und wird durch *read/write-Locks* kontrolliert.

MCA2 - Programmierumgebung

Als Tools zum Visualisieren und Debuggen existieren die MCA-Anwendungen *mcagui* und *mcabrowser*. Mittels *mcagui* lässt sich auf oberster Hierarchieebene eine Benutzerschnittstelle zusammenstellen. Es existiert eine Vielzahl von Standardwidgets, wie Buttons, Zählern oder Oszilloskop-Anzeigen. Darüber hinaus lassen sich über Plug-ins weiter Anzeigen wie beispielsweise eine 3D-Roboter-Sicht realisieren. Mit *mcabrowser* kann man die Struktur der Module grafisch analysieren. Zur Laufzeit erhält man Zugriff auf alle Module, Gruppen, Kanten, Parts, Schnittstellen und Blackboards. Module können gestoppt und gestartet, Debugnachrichten ein- und ausgeschaltet werden. *mcabrowser* ermöglicht somit die volle Kontrolle über alle Aspekte des laufenden MCA-Systems. MCA2 ist echtzeitfähig und netzwerktransparent. Da der Datenaustausch nach einem *producer-consumer*-Schema erfolgt, ist es für die Implementierung der Module unerheblich, ob sie im gleichen Thread, Prozess oder auf unterschiedlichen Rechnern ausgeführt werden. Dadurch ist die Verteilung der Programme über mehrere Rechner möglich und macht MCA2 zu einem skalierbaren und modularen Softwarerahmenwerk für Roboteranwendungen. Neben der Robotersteuerung sind auch Aufgaben im Inspektions- und Diagnosebereich möglich. MCA2 wird bereits erfolgreich auf einer Reihe von Robotern verwendet (*MakroPlus*/*Kairo-II* [90, 137, 22], *LAURON* [58], *Odete*, *ARMAR-III*, LTC2-Telelift, Smart), die teilweise auch im industriellen Umfeld eingesetzt werden.

2.1.6. Vergleich und Bewertung

Neben den hier vorgestellten Softwarerahmenwerken, gibt es noch eine Vielzahl weiterer, die hier jedoch nicht im Detail beschrieben werden sollen, da sie entweder den vorgestellten Softwarerahmenwerken in ihrer Struktur zu sehr ähneln oder aufgrund ihrer Eigenschaften für den Einsatz im humanoiden Roboter nicht in Frage kommen. Der Vollständigkeit halber sollen hier die Bekanntesten zumindest noch erwähnt werden: Orca2 [27, 28, 107], RT-Middleware [4, 3], MIRO [51, 152], Microsoft Robotics Studio, CLARAty [111, 158, 157], ORCCAD [141, 142] und SPQR-RDK/OpenRDK [52, 41, 40].

In Tab. 2.1 werden die Zielsetzungen und die verwendeten Technologien der besprochenen Softwarerahmenwerke zusammengefasst. Die näher beschriebenen Rahmenwerke verfolgen einen komponentenbasierten Ansatz. Die Hauptunterschiede liegen in der Art, wie das System in diese Komponenten zerlegt und wie die Kommunikation unter diesen Komponenten realisiert wird. Ein weiterer wichtiger Aspekt ist die Unterstützung des Sofwareentwurfs durch geeignete Tools – sowohl bei der Programmierung, als auch im laufenden Betrieb und zu Debugzwecken. Unter den Softwarerahmenwerken in der engeren Auswahl zeichnen sich besonders aRD und MCA2 durch ihre spezielle Ausrichtung auf die Steuerung mobiler Roboter mit vielen Bewegungsfreiheitsgraden unter Echtzeitbedingungen aus.

Das Konzept von aRD klingt interessant, da es speziell für Roboter mit vielen Bewegungsfreiheitsgraden und der Ausführung unter Echtzeitbedingungen entworfen wurde. Es wird allerdings bisher ausschließlich auf den Robotersystem des DLR verwendet und es ist nicht klar, ob oder unter welchen Lizenzbedingungen dieses Softwarerahmenwerk zur Verfügung gestellt werden soll. Neben diesem Aspekt fällt noch eine starke Ausrichtung auf Matlab/Simulink auf, die nicht für alle Anwendungen zweckmäßig sein muss. Die Struktur der Datenübertragung mittels des *ardnet*-Blocks als Port-Multiplikator zur Bereitstellung der Daten für mehrere Empfänger eignet sich nicht zur Übertragung größerer Datenmengen wie Kamerabilder oder Laserscanner-Daten.

OROCOS wurde zwar bereits erfolgreich auf Robotersystemen mit sechs Bewegungsfreiheitsgraden unter Echtzeitanforderungen eingesetzt. Es bleibt aber fraglich, ob es für den Einsatz für Roboter mit mehr als 20 Bewegungsfreiheitsgraden ebenfalls tauglich ist. Für den Einsatz auf einem Rechner ist OROCOS grundsätzlich echtzeitfähig. Da die Kommunikation zwischen verteilten Komponenten jedoch ausschließlich über CORBA realisiert ist, kann OROCOS nicht unter Echtzeitanforderungen für verteilte Systeme eingesetzt werden.

Das Softwarerahmenwerk Player unterstützt hauptsächlich die Ebene der Gerätetreiber und bietet sich deshalb zum Entwurf einer kompletten Robotersteuerung nicht an. Für den Entwurf höherer Regelungsebenen gibt Player keinen Rahmen vor und überlässt die komplette Implementierung dem Nutzer. Bisherige Implementierungen mit Player beschränken sich auf Simulationen in den Umgebungen Stage und Gazebo bzw. auf tatsächliche Roboter mit sehr wenigen

Softwarerahmenwerk	Vorrangiges Ziel	Verwendete Werkzeuge und Technologien
aRD	Erfüllung harter Echtzeitbedingungen in komplexen Robotersystemen bei verteilter Ausführung	UDP, *synchronous data flow*, *data push*
OROCOS	Definition eines generischen Roboters bzw. generischer Komponenten zur Implementierung von Roboterfunktionalitäten	CORBA/IDL, TAO, *shared memory*
Player	Entwicklungsplattform für unterschiedliche Robotersysteme, Bereitstellen benötigter Dienste, Hardwareabstraktion über generische Roboterschnittstelle	Client-Server, *Three-Tier* Architektur, Proxy-Objekte
Marie	Meta-Rahmenwerk zur Erzeugung von verteilten Komponenten mit Schwerpunkt auf Prototypentwicklung und der Wiederverwendbarkeit des Codes	Vermittler-Entwurfsmuster, Schichtenmodell, ACE
MCA2	Entwurf echtzeitfähiger Roboteranwendungen und Ermöglichung der Wiederverwendbarkeit durch modularen Aufbau	*publish/subscribe*, TCP/IP, RTAI
RT-Middleware [4, 3]	Modularisierung der funktionalen und softwaretechnischen Aspekte von Robotern. Vereinfachung des Roboteraufbaus durch Modulkombination	CORBA
MIRO [51, 152]	Beschleunigung der Softwareentwicklung und Unterstützung heterogener Rechnernetzwerke	CORBA
ORCA [28, 107]	Komponentenbasierte Entwicklung, Verbesserung der Wiederverwendbarkeit von Software	ICE
CLARAty [158, 157]	Modularisierung des Systems. Enge Kopplung von Planung und Ausführung.	Sockets, ACE, TCP

Tab. 2.1.: Überblick über die Zielsetzungen der Softwarerahmenwerke und die verwendeten Technologien. Für die im Text nicht näher beschriebenen Softwarerahmenwerke sind die relevanten Veröffentlichungen in der Tabelle aufgelistet.

Bewegungsfreiheitsgraden. Zusätzlich zu diesen Punkten macht die verwendete Serverarchitektur durch erhebliche Latenzen ein Erreichen der angestrebten Zykluszeiten unmöglich.

MARIE ist kein Softwarerahmenwerk im eigentlichen Sinne, es ist eher ein Meta-Framework bzw. ein Middleware-Framework, das Komponenten integriert, die in anderen Rahmenwerken erstellt wurden

MCA2 zeichnet sich unter den vorgestellten Softwarerahmenwerken besonders durch seine klare Struktur und die vorhandenen Hilfsprogramme *mcabrowser* und *mcagui* aus. Die Möglichkeit, Module zu Gruppen zusammenzufassen und sogar diese Gruppen wieder als Modul in einer noch komplexeren Gruppe einzusetzen, erweist sich beim Entwurf komplexer Roboterfähigkeiten als sehr nützlich. Als verbesserungsfähige Aspekte wurden bei MCA2 folgende Punkte identifiziert: Die Festlegung auf ausschließlich *double*-Werte für die Schnittstellen stellt beispielsweise bei der Übertragung größerer Datenmengen, wie Bilder oder Laserscannerdaten einen Engpass dar und macht die Übertragung von Zeichenketten umständlich. Die strikte Hierarchie ist einerseits positiv, da die Ausführungsreihenfolge somit immer definitiv festgelegt ist, andererseits sind so zyklische Verbindungen für Rückkoppelungen, wie sie in verhaltensbasierten Systemen häufig verwendet werden, nicht realisierbar. MCA2 ist ausschließlich zeitgesteuert, Daten an den Schnittstellen werden also nicht gepuffert, sondern nur die neuesten Daten verwendet. Für manche Aspekte, beispielsweise bei asynchronen Ereignissen, ist diese zeitgesteuerte Kommunikation nicht zweckmäßig. Geplant ist die Aufhebung der Einschränkung auf *double*: zukünftig können alle Schnittstellen komplexe Datenstrukturen enthalten. Diese können komplett oder teilweise verbunden werden. Das Kopieren der Daten über die Kanten wird durch ein direktes *Mappen* auf die Zielschnittstelle ersetzt. Weiterhin sollen Blackboards nicht mehr nur Rohdaten sondern typisierte Schnittstellen enthalten. Um die Struktur der Schnittstellen zur Laufzeit erfassen zu können, müssen die Schnittstellen mit einer IDL beschrieben werden. Hierfür wird der *MCA2 IDL compiler* entworfen, mit dessen Hilfe automatisch der benötigte C++-Code erzeugt werden kann. Damit verhaltensbasierte Ansätze umgesetzt werden können, werden Rückkopplungen zugelassen. Dabei erfolgt die Ausführung der anderen Module nichts desto trotz periodisch. Im überarbeiteten MCA2 werden Ereignisse unterstützt. Ereignisse werden auf der Empfangsseite gepuffert und asynchron, so schnell wie möglich, von der zugehörigen Ereignisbehandlungsroutine bearbeitet.

Bereits im Ausgangszustand stellt sich das Softwarerahmenwerk MCA2 für die Softwareentwicklung für einen humanoiden Roboter unter Echtzeitanforderungen als die zweckmäßigste Wahl dar. Durch die geplanten Erweiterungen in MCA2 werden sogar noch weitere Möglichkeiten eröffnet. Die Unterstützung im Betrieb und der Zugriff auf alle wesentlichen Daten mit dem Tool *mcabrowser* und die Visualisierung der Sensorwerte und der Darstellung eines 3D-Modells des Roboters mittels *mcagui* sind beim Entwurf von Roboterprototypen sehr wertvoll.

2.2. Hardware für autonome Robotersysteme

Beim Aufbau autonomer Robotersysteme steht man immer den gleichen Herausforderungen gegenüber. Es muss eine Vielzahl von Sensorinformationen aufgenommen, übermittelt und verarbeitet werden und anschließend in Kommandos für die Aktoren umgesetzt werden. Da auf dem Markt verfügbare Komponenten zwar meist über eine ausreichende Rechenleistung für diese Aufgaben verfügen, aber nicht genügend Ein- und Ausgänge besitzen, muss hierfür auf Speziallösungen zurückgegriffen werden. In den meisten Fällen gibt es zwar dezidierte Komponenten für vergleichbare Aufgaben, allerdings sind diese aufgrund ihrer weiteren Spezifikationen, wie Abmessungen, Energieverbrauch oder Gewicht nicht für den Einsatz in autonomen Robotersystemen geeignet. Dies führt dazu, dass für diesen Bereich Eigenentwicklungen oft nicht zu vermeiden sind. Auch wenn international viele Forschungsinstitute auf diesem Gebiet forschen, hat sich noch kein Standard etablieren können. Die erarbeiteten Lösungsansätze ähneln sich zwar, unterscheiden sich in Details jedoch trotzdem so sehr, dass sie zueinander nicht kompatibel sind. Im Folgenden wird ein kurzer Überblick über eine Auswahl der verschiedenen Ansätze an namhaften Instituten gegeben, die sich mit dem Thema autonome Roboter beschäftigen. An dieser Stelle wird allgemein und knapp auf autonome Robotersysteme eingegangen, eine detailliertere Vorstellung der Hardwarerealisierungen speziell für humanoide Roboter erfolgt im Abschnitt 2.4.

2.2.1. Hardwarerealisierungen am DLR

Am DLR werden Komponenten wie fünffingrige Hände und Roboterarme entwickelt. Die erste Generation der DLR-Hand wurde mit einer Workstation angesteuert, die über SERCOS mit einer selbst entwickelten Ansteuerplatine mit einer Kombination aus Mikrocontroller und DSP kommunizierte [39]. Für die zweite Generation der Hand wurde für die höhere Regelung eine PowerPC-Steuerung verwendet und auf Ansteuerungsseite eine spezielle FPGA-basierte Platine entworfen [38, 66]. Im DLR-Arm wurde dieses Konzept weiterverfolgt und für jeweils zwei Gelenke eine dezidierte Ansteuerungselektronik entwickelt. Diese basiert auf einer Backplane mit DSP- und FPGA-Steckkarten. Die Kommunikation mit dem PowerPC zur kartesischen Regelung des Arms fand wiederum über SERCOS statt [64, 72, 73].

2.2.2. Hardwarerealisierungen an der EPFL

An der „École Polytechnique Fédérale de Lausanne“ (EPFL) wurden autonome Robotersysteme von mobilen Mikrorobotern in der Größe weniger Kubikzentimeter bis zu etwa menschengroßen mobilen Tourguides aufgebaut. Für den mobilen Mikroroboter *Alice* wurde eine Steuerplatine basierend auf einem PIC-Mikrocontroller der Firma Microchip entworfen, die auch ein proprietäres Funkprotokoll implementiert [42, 43, 44, 45]. Für *Khepera*, einen weiteren mobi-

len Miniroboter, ein Flugzeug für Innenräume und einen Blimp, wurde ein ähnlicher Ansatz gewählt, was aber mit einer Neuentwicklung der Platinen verbunden [164] war. Der autonome Tourguide *Robox*, der auf der Expo 2002 zum Einsatz kam, ist mit einer passiven Busplatine ausgestattet, auf der eine Pentium-III CPU-Karte und eine PowerPC-Karte zum Einsatz kommen. Zusätzlich beherbergt die Backplane weitere I/O-Karten und eine spezielle PIC-basierte Steckkarte zur Systemüberwachung. Zur Interaktion mit dem Roboter und zur Sprachsynthese kommt ein herkömmlicher Industrie-PC zum Einsatz [139, 148, 140, 125, 124, 126].

2.2.3. Hardwarerealisierungen am AIST

Für den humanoiden Roboter *HRP-II* des „National Institute of Advanced Industrial Science and Technology“ (AIST) wird ein backplanebasierter Industrie-PC verwendet. Für diesen wurde für die Ansteuerung der Gelenke und zur Sensorauswertung eine spezielle Schnittstellenkarte, das so genannte HRP-Interfaceboard, entworfen. Neben dem Steuerrechner wird ein backplanebasierter Industrie-PC zur Bildverarbeitung eingesetzt [71, 85, 86]. Das HRP-Interfaceboard wurde an der Universität Tokyo beim Bau des Roboters *Wabian-2* wiederverwendet. In *Wabian-2* kommt diese Schnittstellenkarte zusammen mit einer Pentium-M basierten CPU-Steckkarte auf einer passiven Busplatine zum Einsatz [114].

Im Roboter *JSK-H7*, der im „Jouhou System Kougaku Laboratory“ (JSK-Labor) der Universität Tokyo aufgebaut wurde, kommt ebenfalls ein auf einer passiven Busplatine basierender Rechner zum Einsatz. In dieser Busplatine wird ein Dual-Pentium-III Board als Recheneinheit verwendet. Zur Ansteuerung der Aktoren und zum Einlesen der analogen Sensorwerte werden zwei so genannte Robot-Interface-Karten (RIF-01) eingesetzt, die speziell für diese Anwendung entwickelt wurden [79].

2.2.4. Hardwarerealisierungen am KAIST

Für den Roboter *KHR-2* des „Korea Advanced Institute of Science and Technology“ (KAIST) kommt ein zentrales, echtzeitfähiges PC/104-System zum Einsatz, das mit mehreren dezidierten Sensor- und Aktorknoten über „Controller Area Network“ (CAN-Bus) kommuniziert. Neben den „Joint Motor Controllern“ (JMC) für die Ansteuerung der Motoren mussten noch Kraft-Momenten-Sensoren und Beschleunigungssensoren entwickelt werden [94, 95, 117, 119].

2.2.5. Hardwarerealisierungen bei Honda

Der vom japanischen Automobilkonzern Honda entwickelte humanoide Roboter *ASIMO* nutzt zur Steuerung mehrere PCs. Zur Geräuschlokalisation kommt eine DSP-Karte zum Einsatz. Für die Bewegung des Roboters ist ein dezidierter Prozessor zuständig [134].

An dieser Übersicht ist klar erkennbar, dass für sämtliche Roboter stets neue Hardwarekomponenten entwickelt werden mussten, da für diesen Einsatzzweck keine standardisierten bzw. käuflichen Komponenten existieren. Darüber hinaus findet zwischen einzelnen Projekten kein nennenswerter Austausch der Entwicklungen statt. Eine Ausnahme stellt hier das HRP-Interfaceboard dar, das sowohl in *HRP-II* als auch in *Wabian-2* verwendet wird. Zwischen einigen Projekten ist die Wiederverwendung der Hardware nicht möglich, da die Randbedingung für den Entwurf der Hardware bei einem Mikroroboter gänzlich andere als beispielsweise bei einem humanoiden Roboter sind. Auch wenn sich eine direkte Wiederverwendung der Hardware aufgrund deutlich unterschiedlicher Anforderungen nicht anbietet, sollte das Ziel sein, Konzepte und Teilkomponenten weitgehend zu übernehmen.

2.3. Steuerungsarchitekturen für autonome Robotersysteme

Eine Steuerungsarchitektur beschreibt die Struktur der Steuerung eines Robotersystems. Aus der Sicht der Robotersteuerung besteht ein Robotersystem aus den drei großen Bereichen: Wahrnehmung, Planung und Handlung. Als Steuerungsarchitekturen zur Verknüpfung dieser drei Bereiche haben sich folgende Schemata etabliert:

- Deliberative Architektur
- Reaktive Architektur
- Hybride Architektur

In diesem Abschnitt werden diese Architekturen näher beschrieben und ihre Vorteile und Nachteile dargestellt.

2.3.1. Deliberative Architektur

Die deliberative Architektur ist die älteste Steuerungsarchitektur für autonome Roboter. Bei dieser Architektur werden die Phasen *sense-plan-act* sequenziell durchlaufen, das heißt es wird Information über die Umwelt mittels der Sensoren erfasst und erst nachdem die Planung abgeschlossen ist, eine Anweisung an die Aktoren weitergereicht (siehe Abb. 2.7). Die Grundlage der Planung bei dieser Architektur stellt eine symbolische Abbildung der Realität in ein Modell dar. Hierzu muss im Vorfeld die Umwelt modelliert werden, dieses Modell dient dann als Wissensbasis für den Roboter. Aufgrund des vorhandenen Umweltmodells werden wissensbasiert Entscheidungen getroffen und Pläne für die Ausführung ausgewählt. Der interne Zustand des Roboters ändert sich durch die Wahrnehmung der Umwelt und durch die geplanten Aktionen. Durch die Sensorik wird das vorab erstellte Umweltmodell modifiziert und aktualisiert [7]. Wie bereits erwähnt, ist für die deliberative Architektur eine symbolische Vorabmodellierung der

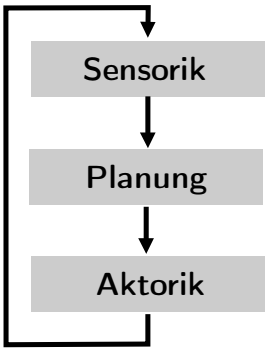


Abb. 2.7.: Schema der deliberativen Steuerungsarchitektur

Umwelt notwendig. Diese symbolische Modellierung ist für viele Aspekte sehr aufwendig und auch speicherintensiv. Hinzu kommt, dass für eine mittel- bis langfristige Planung ein sehr genaues Modell der Umwelt benötigt wird und dies unter Umständen in der nötigen Genauigkeit nicht möglich ist. Beispielsweise sind für komplexe Aspekte der Realwelt, wie menschliche Sprache oder auch die Erkennung von Objekten, sehr rechenaufwendige Algorithmen gefordert. Nachteile dieser Architektur bestehen vor allem darin, dass sowohl die Erstellung als auch Aktualisierung des Umweltmodells sehr hohe Anforderungen an die Rechen- und Speicherkapazität des Robotersystems stellen. Durch die hohe Rechenlast bei der Auswertung und Aktualisierung des Umweltmodells und der Planung sind Reaktionszeiten auf unvorhergesehene Ereignisse oft zu lang. Besonders augenfällig wird dieser Nachteil bei der Vermeidung von unerwartet auftretenden Hindernissen in dynamischen Umgebungen. So nimmt die Erfassung und Erkennung eines Hindernisses, Aktualisierung des Weltmodells und die Erstellung einer Ausweichstrategie unter Umständen zuviel Zeit in Anspruch, als dass die Kollision noch vermieden werden könnte.

2.3.2. Reaktive Architektur

Die Motivation zum Aufbau reaktiver Architekturen kam aus der Erkenntnis, dass sich mit deliberativen Architekturen trotz beachtlich steigender Rechenleistung für gewisse Probleme keine angemessenen Reaktionszeiten erzielen lassen. Neben der zu großen Rechenzeit wurde erkannt, dass auch die Umsetzung geplanter Aktionen aufgrund mechanischer Ungenauigkeiten durch reine Planung nicht möglich ist und eine schnelle Nachregelung notwendig macht, die mit der deliberativen Architektur nicht umgesetzt werden kann. Als Konsequenz wurde der explizite Planer aus dem Ablauf entfernt und eine direkte Verknüpfung der Sensorinformation mit den Aktoren vorgeschlagen. Somit ergibt sich als Steuerschema das in Abb. 2.8 dargestellte *sense-act*. Es kommt also zu einer unmittelbaren Abbildung der Sensordaten auf die Aktorsteuerung. In der striktesten Form der reaktiven Architektur ist somit der Aufbau eines internen Systemzustands nicht möglich. Zudem ist das System nicht in der Lage zu lernen oder ein Modell

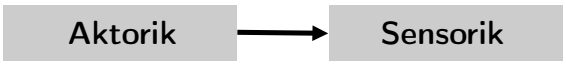


Abb. 2.8.: Schema der reaktiven Steuerungsarchitektur

der Umwelt aufzubauen. Das Verhalten des Gesamtsystems wird also ausschließlich durch die vorab getroffene Auswahl an Verknüpfungen zwischen Sensoren und Aktoren bestimmt. Durch diese direkte Kopplung können reaktiv gesteuerte Systeme sehr schnell auf unvorhergesehene Situationen reagieren.

Da erkannt wurde, dass beim Betrieb des Systems durchaus sensorisch ununterscheidbare Zustände auftreten können, die jedoch unterschiedliche Reaktionen erfordern, wurde das reaktive Konzept um die Möglichkeit erweitert, eine Zustandshistorie aufzubauen. Somit lassen sich auch vorhergehende Zustände in die Sensorauswertung mit einbeziehen.

Durch die Abwesenheit der Planungskomponente muss das gewünschte Systemverhalten durch die Auswahl und die Art der Verknüpfung zwischen den Sensoren und Aktoren implizit in das System gebracht werden. Die Bausteine der reaktiven Architektur werden auch als Verhalten (*behaviour*) bezeichnet. Jedes Verhalten besitzt eine *sense*- und eine *act*-Komponente. Üblicherweise wird ein System aus mehreren Verhalten aufgebaut, die dann konkurrierend arbeiten. Im Idealfall ist ein Sensor nur einem Verhalten zugeordnet, dieses Entwurfsschema wird als *sensor decoupling* bezeichnet. Für den Fall, dass mehrere Verhalten konkurrierend Zugriff auf einen Aktor verlangen, muss eine Koordination vorgesehen werden. Mögliche Lösungen sind hier eine vektorielle Addition der Steuerungsdaten oder eine Unterdrückung eines Verhaltens durch ein hierarchisch höher stehendes Verhalten.

Pionier auf dem Gebiet der verhaltensbasierten Architekturen war unter anderem William Grey Walter, der mit seinen „Machina Speculatrix“-Robotern *Elmer* und *Elsie* als einer der Ersten autonome Roboter aufbaute [159]. Der den BEAM-Robotern[11] zugrundeliegende Mechanismus – elektrische Schaltkreise als künstliche Neuronen zu nutzen – wurde von Mark Tilden etabliert [67]. Valentino Braitenberg wurde durch sein Buch „Vehikel“ [25] bekannt, in dem er beschreibt, wie selbst durch einfache Mechanismen ein komplexes Verhalten des Gesamtsystems erreicht werden kann. Eine der bekanntesten Umsetzungen verhaltensgesteuerter Architekturen ist die 1986 von Rodney A. Brooks vorgeschlagenen Subsumption-Architektur [29, 30].

Die Subsumption-Architektur basiert auf der Unterteilung eines komplexen Verhaltens in hierarchisch angeordnete Module, die Einzelverhalten widerspiegeln. Diese Module sind in so genannten Komptetenzleveln (Abb. 2.9 unten) angeordnet, wobei Module mit einem höheren Kompetenzlevel Einfluss auf die Eingänge und Ausgänge der niedriger angesiedelten Module

[11] `http://www.solarbotics.net/`

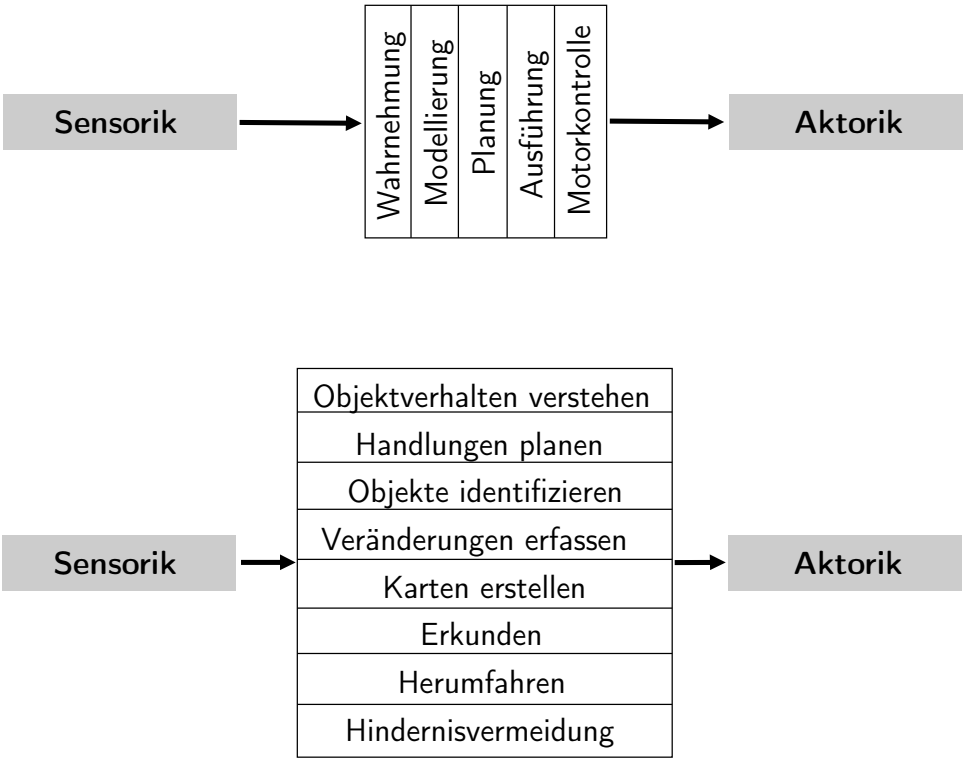


Abb. 2.9.: Module und Schichten der Subsumption-Architektur nach [29]

nehmen können. Hierdurch ist es möglich komplexe Verhalten zu realisieren, die immer auf den schon vorhandenen Verhalten aufbauen und diese weiter abstrahieren.

Nachteile der reaktiven bzw. verhaltensbasierten Architekturen sind, dass sie aufgrund ihrer direkten Sensor-Aktor-Kopplung immer nur auf die aktuelle Situation reagieren und somit kein längerfristiges Ziel verfolgen können. Komplexere Aufgaben wie beispielsweise Objekterkennung lassen sich aufgrund des fehlenden Umweltmodells in diesen Architekturen nicht umsetzen. Für Aufgabenstellungen wie Navigation lässt sich wegen des fehlenden Modells der Umgebung selten der optimale Weg finden, in manchen Fällen liefert die Strategie überhaupt keine Lösung.

2.3.3. Hybride Architektur

Hybride Architekturen bilden eine Mischung aus reaktiver und deliberativer Architektur. Somit vereinen sie die Vorteile beider Ansätze. Wie bereits beschrieben eignen sich reaktive Architekturen besonders für Einsatzgebiete, in denen eine schnelle Reaktion auf eine äußere Veränderung notwendig ist. Deliberative Architekturen sind wiederum bei Aufgaben, für die Lernen bzw. Wissen über die Umwelt notwendig ist, im Vorteil. Durch die Kombination beider Architekturen lässt sich die hybride Architektur auch in Szenarien einsetzen, in denen sowohl plan-

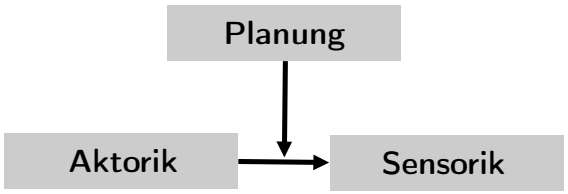


Abb. 2.10.: Schema der hybriden Steuerungsarchitektur

volles Agieren als auch schnelle Reaktionen gefordert sind. Schematisch lässt sich die hybride Architektur wie in Abb. 2.10 darstellen. Im Gegensatz zur deliberativen Architektur befindet sich hier die Planungskomponente nicht zwingend im Regelkreis, sondern sie überwacht nur die direkte Verbindung zwischen Sensorik und Aktorik.

2.4. Hardware-Software-Architekturen bei humanoiden Robotern

In diesem Abschnitt werden aktuelle Robotikprojekte betrachtet. Es wird untersucht, welche Roboterarchitekturen verwendet werden, welche Steuerungsarchitekturen zum Einsatz kommen und wie das Zusammenspiel zwischen mechatronischer Hardware und der Computerarchitektur gelöst wird. Am Ende dieses Abschnitts wird eine Zusammenfassung über den aktuellen Forschungsstand gegeben.

2.4.1. ASIMO

Die Forschung an humanoiden Robotern hat bei dem japanischen Automobilkonzern Honda eine lange Tradition [69, 70]. Bereits seit 1986 wird dort das zweibeinige Laufen erforscht und es werden humanoide Assistenzroboter entwickelt (Abb. 2.11). Die Motivation bei Honda war es, eine Maschine zu entwerfen, die in der Lage ist, dem Menschen in einer für den Menschen konzipierten Umgebung, monotone Arbeiten abzunehmen. Dabei wurde ein Schwerpunkt auf die zweibeinige Fortbewegung gelegt, die notwendig ist, um Hindernisse wie zum Beispiel Treppen zu überwinden. Am Anfang der Entwicklung beschränkte man sich mit den experimentellen Modellen E0 und E1 auf reine zweibeinige Laufmaschinen ohne Oberkörper. Auf diesen wurde ausschließlich statisches Laufen implementiert. Auf den Modellen E2-E4 wurde das dynamische Laufen untersucht, welches allerdings nur auf ebenem Untergrund funktionsfähig war. Für die Modelle E5-E6 wurde das dynamische Laufen weiter verbessert und weitere Techniken zur Stabilisierung des Laufens, wie die Anpassung der Fußstellung an den Untergrund und der Schrittweiten, implementiert. Ab dem folgenden Prototypmodell P1 wurde damit begonnen, die zweibeinige Laufmaschine zusätzlich mit einem Oberkörper, Kopf und Armen mit Manipulator

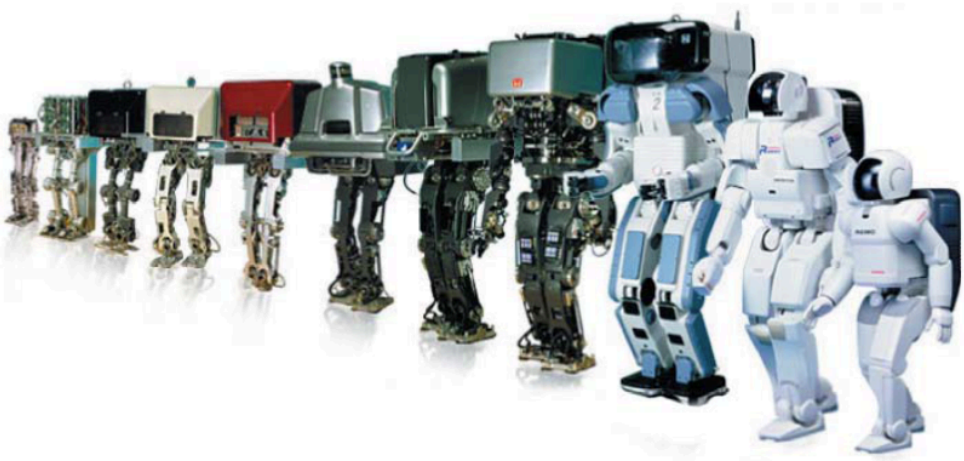


Abb. 2.11.: Der humanoide Roboter ASIMO und seine Vorgänger (Quelle: [75])

auszustatten. P1 hatte bei einer Größe 1,9 m, ein Gewicht von 175 kg und war bereits in der Lage, Schalter und Türgriffe zu bedienen. Die Stromversorgung und der Steuerungsrechner waren in einer Art Rucksack am Rücken des Roboters befestigt. Mit dem P2 wurde begonnen, die Größe des Roboters zu reduzieren, dieser ist noch 1,8 m groß, wiegt allerdings mit 210 kg mehr als sein Vorgänger, was an der höheren Integration und der Verkleidung liegt. P2 verfügt über einen Steuerrechner mit vier Microspec CPUs und wurde mittels eines drahtlosen Netzwerks von den Laborrechnern aus gesteuert. Der humanoide Roboter P3 ist mit 1,6 m deutlich kleiner als P2 und wiegt dank anderer Materialwahl bei der Stützstruktur nur noch 130 kg. Die Steuerung wird von 2 Microspec CPUs der zweiten Generation erledigt. Die Bewegungskontrolle wird auf dem Roboter selbst durchgeführt, die Planung und Vorgabe von Zielkoordinaten wird weiterhin durch eine Workstation durchgeführt.

Die Tendenz, den Roboter kompakter, leichter und kleiner zu gestalten, wurde mit dem aktuellen Forschungsprojekt „Advanced Step in Innovative Mobility“ (ASIMO) konsequent weitergeführt. *ASIMO* (Abb. 2.12) wiegt bei einer Größe von 1,2 m nur noch 52 kg. Die 26 Servomotoren zur Ansteuerung der 26 Bewegungsfreiheitsgrade des Roboters können von der mitgeführten Batterie circa eine halbe Stunde mit Strom versorgt werden.

ASIMO - Software und Steuerungsarchitektur

Die Planungsarchitektur auf *ASIMO* ist verhaltensbasiert und hat sowohl deliberative als auch reaktive Komponenten [134]. Die deliberativen Komponenten sind zuständig für höhere Verhalten, wie Reaktion auf Sprachbefehle des Menschen. Zur Bewältigung der sich schnell verändernden Umgebung, beispielsweise bei der Navigation in dynamischen Umgebungen, werden reaktive Verhalten eingesetzt. Die Kommunikation auf dieser Ebene erfolgt ereignisgesteuert und damit asynchron.

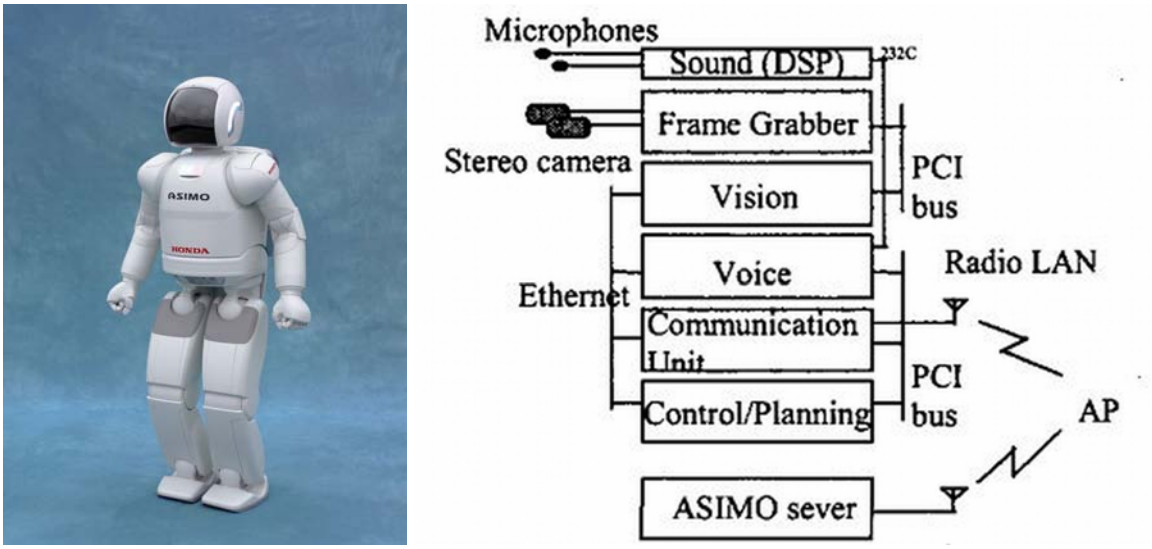


Abb. 2.12.: Der humanoide Roboter ASIMO und die verwendete Architektur [134]

ASIMO - Hardware und Systemstruktur

Das verwendete Rechnersystem besteht aus einem PC zur Bildverarbeitung, einem PC zur Spracherkennung und Sprachgenerierung, einem dezidierten Prozessor für Steuerungsaufgaben und einem DSP-Board zur Lokalisation von Geräuschquellen.

ASIMO - Datentransfer und Peripherie

Der Datenaustausch zwischen den Komponenten erfolgt über Ethernet und wird durch einen so genannten *communication server* kontrolliert.

2.4.2. H7

Der humanoide Roboter *JSK-H7* wurde im JSK-Labor der Universität Tokyo als Nachfolger des *JSK-H6* [113, 78, 80] aufgebaut. Mit diesen Robotersystemen soll die Kopplung zwischen Wahrnehmung und Aktion untersucht werden. Neben der Fortbewegung auf zwei Beinen steht die Untersuchung von koordinierten Bewegungen des gesamten Körpers im Vordergrund [79, 112]. *JSK-H7* soll ebenfalls als Testplattform für unterschiedliche Aspekte humanoider Roboter eingesetzt werden. Auf dem Roboter sollen unter anderem 3D-Objekterkennungsalgorithmen, eine sensitive Haut und Algorithmen zur Bewegungsplanung getestet werden. Besonders die Möglichkeit zur koordinierten Bewegung des gesamten Körpers stellt hohe Anforderungen beim Entwurf und bei der Integration der notwendigen Sensorik.

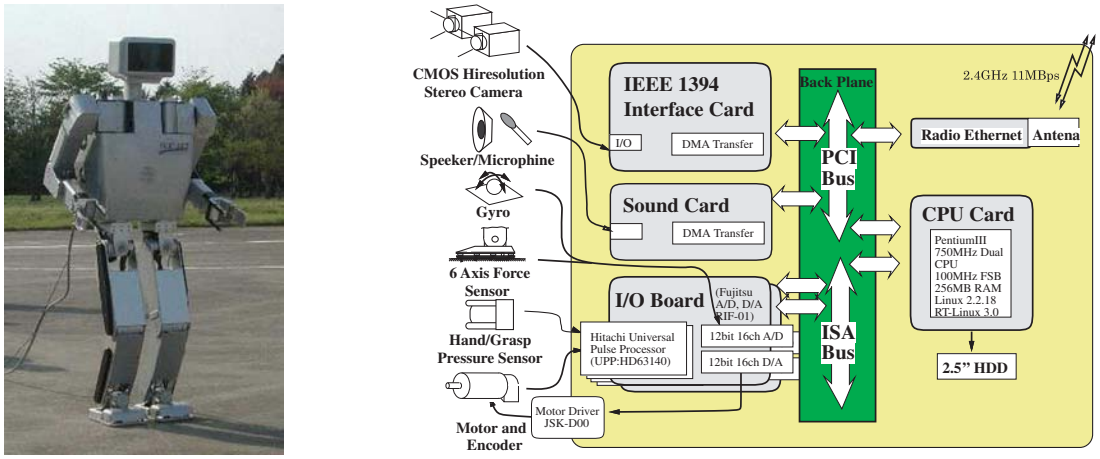


Abb. 2.13.: Der humanoide Roboter H7 und die verwendete Rechnerarchitektur (Quelle: [79])

H7 - Software und Steuerungsarchitektur

Als Betriebssystem kommt auf *JSK-H7* RT-Linux zum Einsatz. Die Steuerung des Roboters erfolgt zentral und hierarchisch in Ebenen organisiert. Wie in Abb. 2.14 dargestellt gliedert sich die Steuerungsarchitektur in die Ebenen: Anwendungsebene, Netzwerkebene, die *Onbody*-Ebene und die Kernelebene. Unterhalb dieser Steuerungsebenen befindet sich die Hardwareschicht mit den Ein- und Ausgabe-Platinen, die die Verbindung zu den physikalisch vorhandenen Sensoren und Aktoren herstellt.

H7 - Hardware und Systemstruktur

JSK-H7 hat insgesamt 30 Bewegungsfreiheitsgrade, sieben in jedem Bein, sieben in jedem Arm inklusive des Greifers und zwei im Halsgelenk. *JSK-H7* wiegt 57 kg bei einer Größe von 1,47 m. Die Energieversorgung des Roboters wird mit vier Bleibatterien mit jeweils 2 Ah bei 48 V und einem Gesamtgewicht von 3,44 kg erledigt. Diese Batterien sind im Oberkörper verbaut. Zur Verlängerung der Laufzeit kann optional ein zusätzliches, 6 kg schweres Batterie-Pack aus „Nickel-Metallhydrid-Akkus“ (NiMH) mit 6 Ah auf dem Rücken montiert werden. Im Roboter kommt die in Abb. 2.13 dargestellte Architektur zum Einsatz. Diese basiert auf einem Dual-Pentium-III Board das mit 750 MHz getaktet ist (zur Taktfrequenz finden sich widersprüchliche Angaben in [112] ist von 1,1 GHz die Rede). Das Prozessorboard steckt in einer *Backplane*, an die auch zwei „Robot-Interface-Karten“ (RIF-01) angesteckt sind. Diese Karten verfügen über 16 12 Bit Analog-Digital-Wandler, 16 12 Bit Digital-Analog-Wandler und einen universellen Impulszähler. Die Steuersignale für die Motoren werden von den Digital-Analog-Wandlern an die Motorendstufenplatine – den JSK-D00 Motor-Driver – weitergereicht und somit die Motoren betrieben. Die Motorsensoren und die Position des Greifers werden durch den Impulszähler ausgewertet.

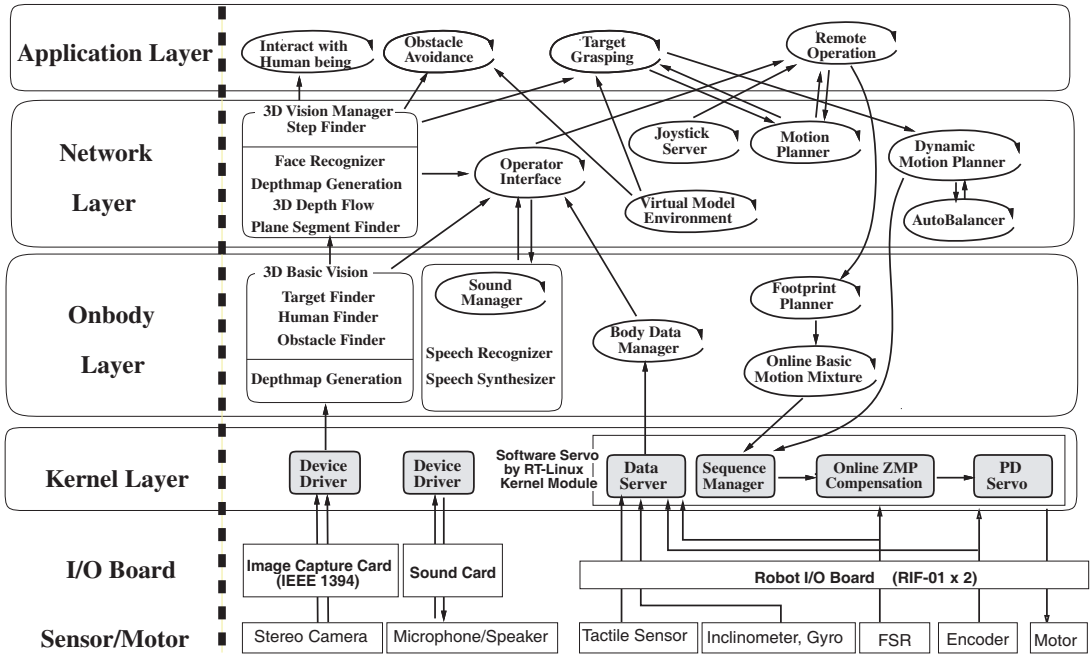


Abb. 2.14.: Steuerungsarchitektur auf dem humanoiden Roboter *JSK-H7* (Quelle: [79])

H7 - Datentransfer und Peripherie

Der Datenaustausch zwischen den RIF-01 und dem Prozessor erfolgt über den PCI-Bus der Backplane. Die unterschiedlichen Sensoren sind mit dem RIF-01 verbunden. Die Auswertung der Kraft-Momenten-Sensoren erfolgt mittels der Analog-Digital-Wandler. Die Ausgaben des Gyroskops werden mit dem universellen Impulszähler ausgewertet. Als Stereokamerasystem kommt das Mega-D von Videre Design[12] zum Einsatz. Diese Kameras bieten eine Auflösung von 1280 $\times$ 1024 Pixeln und werden über IEEE1394 FireWire an das Dual-Pentium-Board angeschlossen. Über eine Soundkarte sind zwei Mikrofone angeschlossen und es besteht die Möglichkeit zur Sprachausgabe mittels eines Lautsprechers.

Die Steuerung des Kopfes erfolgt über ein weiteres Motortreiberboard, JSK-D01, das über I^2C angeschlossen ist [80]. Zur Kommunikation mit den Laborrechnern kommt ein drahtloses Netzwerk nach dem IEEE 802.11b-Standard zum Einsatz.

2.4.3. HRP-2

Das „Humanoid Robotics Project“ (HRP) ist ein gemeinsames Forschungsprojekt des AIST und Kawada Industries. Es wurde von 1998 bis 2002 durch das japanische Wirtschafts- und Forschungsministerium (METI/NEDO) gefördert. Schwerpunkt der Entwicklungen lag auf der Bereitstellung eines modularen Softwarerahmenwerks (OpenHRP) und der Entwicklung eines humanoiden Roboters, der nicht nur zu Unterhaltungszwecken eingesetzt werden kann, sondern

[12] `http://www.videredesign.com/`

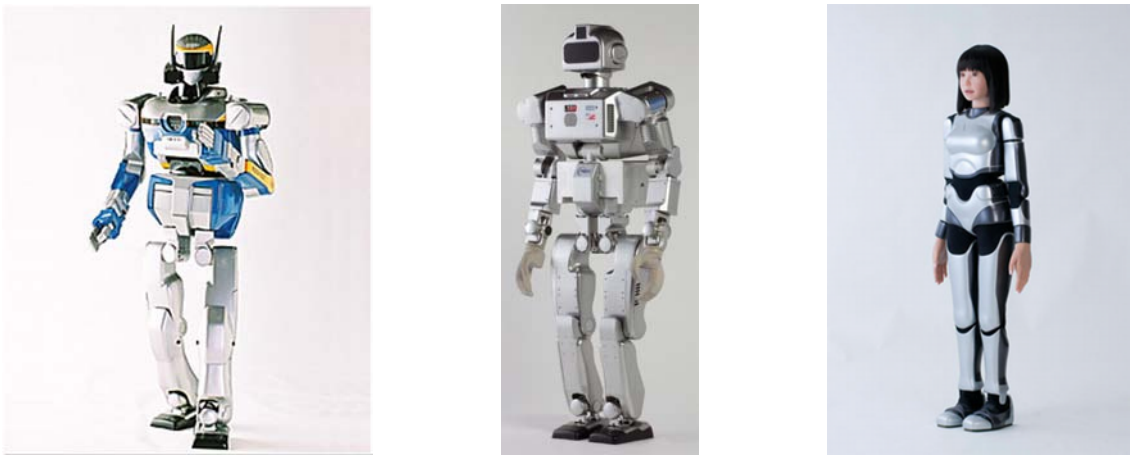


Abb. 2.15.: Der humanoide Roboter *HRP-II*, der aktuelle Prototyp *HRP-III* und der Unterhaltungsroboter *HRP-IVc* (Quelle: [85, 1, 81])

in der Lage ist mit Menschen zu interagieren und für diesen Arbeit zu erledigen [85].

In diesem Rahmen wurde die „humanoid robotics platform“ (*HRP-II*, siehe Abb. 2.15) gemeinsam von der Forschungsgruppe des AIST und Kawada Industries entwickelt und aufgebaut. *HRP-II* verfügt über 30 Bewegungs-Freiheitsgrade und hat bei einer Größe von 1,54 m ein Gewicht von 58 kg.

HRP-2 - Software und Steuerungsarchitektur

Als Softwarearchitektur kommt das modulare aufgebaute OpenHRP [71, 83] zum Einsatz. OpenHRP ist mittels CORBA umgesetzt und wird auf dem Roboter unter ART-Linux realisiert [82].

Bezüglich der Rechnerhardware und der Ansteuerung der Motoren ist *HRP-II* zentral aufgebaut. Die Auswertung der gesamten Sensorwerte und die Berechnung neuer Stellwerte für die Bewegungsachsen erfolgt zentral auf den beiden eingebauten PCs. Die verwendete Softwarearchitektur ist modular aufgebaut und die Steuerung erfolgt unterteilt in die Teilsysteme: Bildverarbeitung, Planung, Bewegungskontrolle und Interaktion. Die Steuerung ist in OpenHRP realisiert, wobei das Robotermodell als Schnittstelle zwischen den einzelnen Teilsystemen dient [115]. Die Abstraktion der tatsächlichen Controller durch OpenHRP ermöglicht den Einsatz des *Integrated Simulation Environment*. Geplante Handlungen werden anhand dieses Robotermodells auf Durchführbarkeit simuliert und im Erfolgsfall werden Vorgaben für die Roboterposition, Fußplatzierung und die Gelenkwinkel an die Ausführung auf dem Roboter weitergegeben. OpenHRP stellt folgende Objekte zur Verfügung: *Model parser, collision checker, controller, view simulator, input device, pattern generator, motion planner* und *dynamics*.

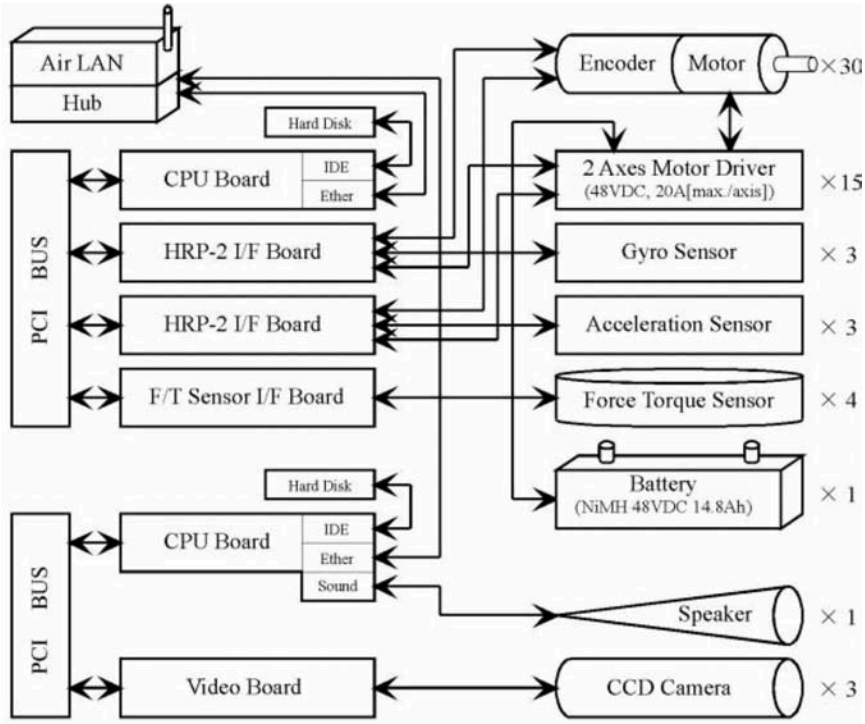


Abb. 2.16.: Rechnerarchitektur auf HRP-2 (Quelle: [85])

HRP-2 - Hardware und Systemstruktur

Das Computersystem besteht aus zwei Pentium-III-basierten PCs mit einer Taktfrequenz von 1 GHz. Die PCs sind mittels Steckkarten-CPU auf einer passiven Busplatine realisiert. Ein PC ist für die Echtzeit-Regelung des gesamten Körpers zuständig, der andere PC für die „Versatile Volumetric Vision“ (VVV) [149] genannte Bildverarbeitung und Spracherkennung bzw. Sprachausgabe. Als Bus-System der Backplane-Rechner kommt der PCI-Bus zum Einsatz. Da die Spezifikation des PCI-Busses nur vier Busteilnehmer zulässt, musste zur Verbindung mit den Motortreiberplatinen und zum Auslesen der Sensoren eine spezielle Schnittstellenkarte entwickelt werden. In Abb. 2.16 ist schematisch die Verbindung der Rechnerkomponenten und der Peripherie dargestellt.

HRP-2 - Datentransfer und Peripherie

Neben den im Roboter integrierten Rechnern verfügt *HRP-II* noch über eine drahtlose Netzwerkverbindung zu den Laborrechnern, über die er neue Aufgaben erhält. Eine Schnittstellenkarte dient zur Verbindung mit den 15 Motortreiberplatinen, die jeweils zwei Servomotoren ansteuern und somit die 30 Bewegungs-Freiheitsgrade des Roboters antreiben. Die Motorencoder sind direkt an das zugehörige HRP-Interfaceboard angeschlossen. Am zweiten HRP-Interfaceboard sind die drei Gyroskope und die drei Beschleunigungssensoren angeschlossen. Eine speziell entwickelte Schnittstellenkarte dient zur Auswertung der vier verbauten Kraft-

Momenten-Sensoren. Sprachausgabe kann über einen an den Soundausgang der CPU-Karte angeschlossenen Lautsprecher erfolgen. Die drei zur Umgebungserfassung eingesetzten CCD-Kameras werden über eine Framegrabberkarte auf der passiven Busplatine ausgelesen.

Aktuell wird *HRP-III* (Abb. 2.15) als Nachfolger des *HRP-II* aufgebaut [1, 84]. Bei *HRP-III* wird von der zentralen Steuerung abgewichen. Aufgrund der festgestellten Probleme mit der zentralen Lösung, wurden spezielle Controller-Knoten entwickelt, die für die dezentrale Ansteuerung der Motoren zuständig sind. Diese Controller sollen über Realtime-Ethernet mit dem zentralen Rechner verbunden werden. Neben diesen auf Manipulationsaufgaben und zweibeiniges Laufen ausgelegten Robotern wird mit *HRP-IVc* [81] ein Roboter entwickelt bei dem der Schwerpunkt eher auf einer möglichst naturgetreuen Nachbildung des Menschen liegt.

2.4.4. KHR

Der zweibeinige Roboter *KHR* wurde an der südkoreanischen Universität KAIST entwickelt. Der Roboter soll sowohl als Assistenzroboter eingesetzt werden können als auch zu Unterhaltungszwecken dienen [92]. Untersuchungen zufolge eignet sich für diese Fälle besonders ein etwa kindsgroßer Roboter [93].

KHR - Software und Steuerungsarchitektur

Auf *KHR* kommt eine zentrale Architektur zum Einsatz. Die Steuerung des Roboters, wird von einem einzelnen Rechner im Oberkörper des Roboters erledigt. Als Betriebssystem kommt auf diesem Rechner das „Microsoft Disk Operating System“ (MS-DOS) zum Einsatz [96]. Der Vorteil von MS-DOS ist, dass es einen direkten Zugriff auf die Rechnerhardware erlaubt und somit eine Echtzeitsteuerung mit Hilfe von Schnittstellenkarten erlaubt.

KHR - Hardware und Systemstruktur

KHR ist 1,2 m groß und wiegt 48 kg. Er verfügt über insgesamt 21 Bewegungsfreiheitsgrade. Davon entfallen sechs auf jedes Bein, einer auf die Hüfte und vier auf jeden Arm. Bewegungsmöglichkeiten für Hals und Hände sind nicht vorgesehen. Sämtliche zum Betrieb des Roboters notwendigen Komponenten, wie Recheneinheiten und Stromversorgung, sind in den Roboterkörper integriert, so dass der Roboter ohne Kabelverbindung autonom agieren kann. In Abb. 2.17 ist die auf *KHR* eingesetzte Rechnerarchitektur dargestellt. Die Rechnerarchitektur ist zentral ausgelegt und die Ansteuerung sämtlicher Motoren und die Auswertung der Sensoren erfolgt sternförmig von einer im Oberkörper integrierten Recheneinheit. Diese besteht aus einem PC/104-System mit einer Pentium-III Prozessorplatine, die mit 500 MHz getaktet ist und zwei Peripheriecontrollern. Diese beiden als „peripheral interface board“ (PIB) bezeichneten Platinen sind über den PC/104-Steckverbinder mit dem CPU-Board verbunden.

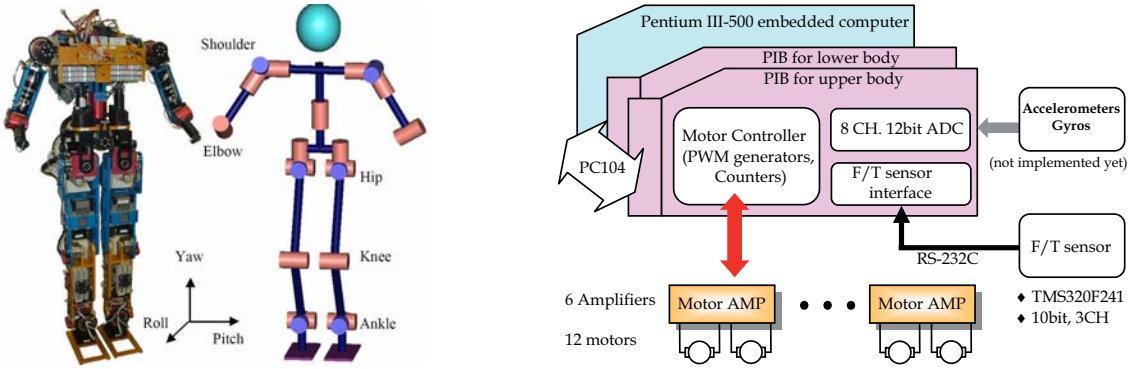


Abb. 2.17.: Der humanoide Roboter *KHR* und die verwendete Rechnerarchitektur (Quelle [93])

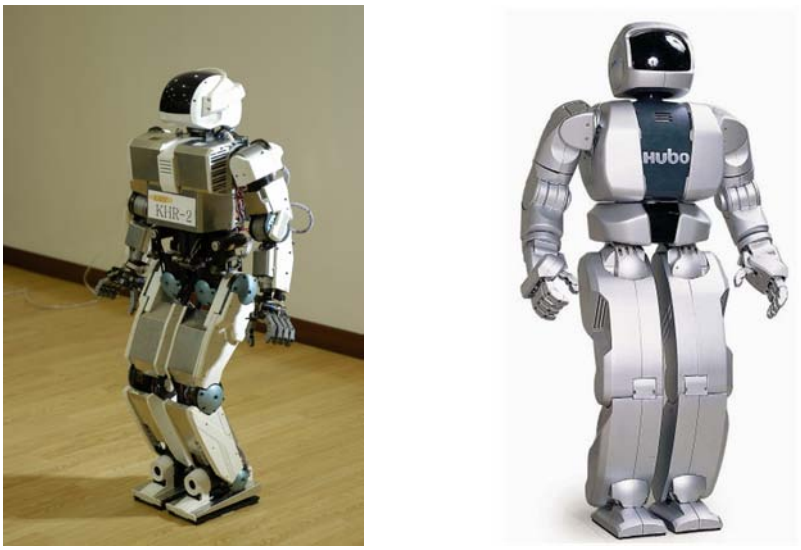


Abb. 2.18.: Die humanoiden Roboter *KHR-2* (links) und *KHR-3*/*HUBO* (rechts)

KHR - Datentransfer und Peripherie

Bei *KHR* befinden sich auf dem PIB zur Ansteuerung der Motoren zwölf Einheiten zur „Pulsweiten-Modulation“ (PWM), Encoderzähler und ein integrierter Mikrocontroller, der über RS-232 mit dem Kraft-Momenten-Sensor in der Fußsohle kommuniziert. Über Analog-Digital-Wandler auf dem PIB werden die Ausgaben der Lage- und Beschleunigungssensoren ausgewertet. Der Ausgang der PWM-Einheiten wird mit den Endstufenplatinen zur Motoransteuerung verbunden. Diese sind jeweils in der Lage zwei Motoren bei 24 V mit bis zu 18 A zu versorgen. An jedes PIB können sechs dieser Motortreiberplatinen angeschlossen werden, was bei den zwei verwendeten PIB eine Maximalzahl von 24 Bewegungsfreiheitsgraden ergibt.

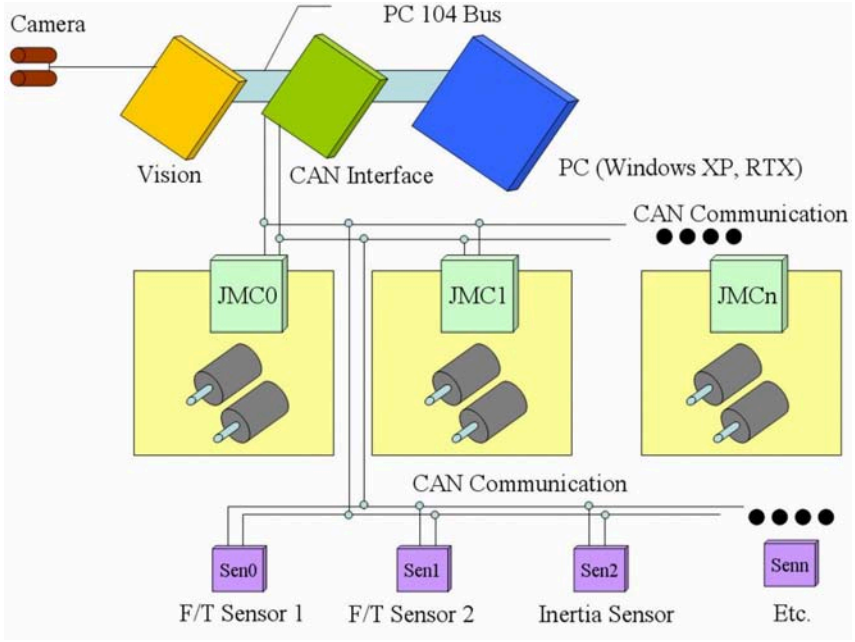


Abb. 2.19.: Rechnerarchitektur im humanoiden Roboter *KHR-2* (Quelle [119])

2.4.5. KHR-2

Parallel zu der vorliegenden Arbeit wurde am KAIST ab etwa 2004 mit der Weiterentwicklung des *KHR* begonnen. Für *KHR-2* wurde auch auf eine verteilte Architektur gesetzt [119, 96]. Die Designentscheidung bezüglich der Robotergröße wurde beibehalten, die Anzahl der Bewegungsfreiheitsgrade wurde jedoch deutlich erhöht (Abb. 2.18). So besitzt *KHR-2* insgesamt 41 Bewegungsfreiheitsgrade und wiegt 56 kg.

KHR-2 - Software und Steuerungsarchitektur

Beim Nachfolger des *KHR*, dem humanoiden Roboter *KHR-2*, kommt eine hierarchische, verteilte Steuerungsarchitektur zum Einsatz (Abb. 2.19). Auf dem zentralen Steuerungsrechner kommt Windows XP mit einer RTX HAL Echtzeiterweiterung zum Einsatz [5, 6]. Der Vorteil dieser Lösung gegenüber der Architektur im *KHR* ist eine Entlastung des Hauptrechners, der bei der größeren Anzahl an Bewegungsfreiheitsgraden nicht leistungsfähig genug für eine Echtzeitsteuerung wäre.

KHR-2 - Hardware und Systemstruktur

KHR-2 verfügt über 41 Bewegungsfreiheitsgrade. Jeder Arm hat sechs Bewegungsfreiheitsgrade, jede Hand fünf, jedes Bein sechs und die Hüfte einen. Der Kopf hat insgesamt sechs Bewegungsfreiheitsgrade, wobei jedes Auge über zwei verfügt und zwei für jedes Halsgelenk verwendet werden. Als zentrale Recheneinheit kommt genau wie im *KHR* ein PC/104-System

zum Einsatz auf dem eine Framegrabber-Karte zur Verarbeitung der Kamerainformation aufgesteckt ist. Zur Kommunikation mit den dezentral über den Roboter verteilten Motortreiberkarten ist eine CAN-Karte in das System integriert.

KHR-2 - Datentransfer und Peripherie

Für die aktuellen Nachfolger des *KHR* wurde auf einen dezentralen Aufbau gewechselt. Auf der Seite der Rechnerarchitektur wird im *KHR-2* eine verteilte Echtzeitarchitektur, basierend auf CAN-Bus und verteilten Sensor- und Aktor-Knoten, verwendet. Zur Ansteuerung der Motoren und zur Auswertung der Sensorinformation wurde der „Joint Motor Controller“ (JMC) entwickelt. Es gibt zwei Typen des JMC, beide bestehen aus einem Controller-Modul und einer Verstärkerplatine. Der eine Typ ist in der Lage bis zu sieben Motoren mit geringer Leistung anzusteuern und ist somit beispielsweise für die Bewegungsachsen im Kopf geeignet. Der andere Typ ist für leistungsstärkere Motoren, wie die in den Beingelenken ausgelegt, kann dafür aber nur zwei Achsen betreiben. Neben den Motorcontrollern sind noch dezentrale Sensorauswerteeinheiten an den CAN-Bus angeschlossen. Hierzu zählen der Kraft-Momenten-Sensor, der im Fußgelenk montiert ist und die Beschleunigungssensoren.

Laufende Arbeiten werden an dem Nachfolger *KHR-3* bzw. *HUBO* durchgeführt. Aus Architektursicht unterscheidet sich dieser nicht von seinem Vorgänger [117, 118]. Die wesentlichen Unterschiede bestehen in einer verbesserten Mechanik und Anpassungen im äußeren Erscheinungsbild.

2.4.6. Wabian-2

Wabian-2 stammt aus der Waseda Universität in Tokyo. Der Roboter wurde zur Erforschung des zweibeinigen Laufens entwickelt. Er dient zur Nachahmung verschiedener menschlicher Bewegungsabläufe beim Laufen. Es werden Untersuchungen im Bezug auf die Rehabilitation bei Verletzungen des Gehapparats durchgeführt. Beispielsweise wurden Experimente durchgeführt bei denen sich *Wabian-2* auf einen Gehwagen gestützt fortbewegt. Die Bewegungsbereiche der einzelnen Gelenke wurden denen des Menschen nachempfunden [161, 162, 20].

Wabian-2 - Software und Steuerungsarchitektur

Wabian-2 wird hierarchisch gesteuert und als Rechnerarchitektur wird wie bei *HRP-II* eine zentrale Rechnerarchitektur verwendet. Die Steuerung erfolgt über einen zentral verbauten PC. Auf diesem PC wird QNX als Echtzeitbetriebssystem eingesetzt [114].

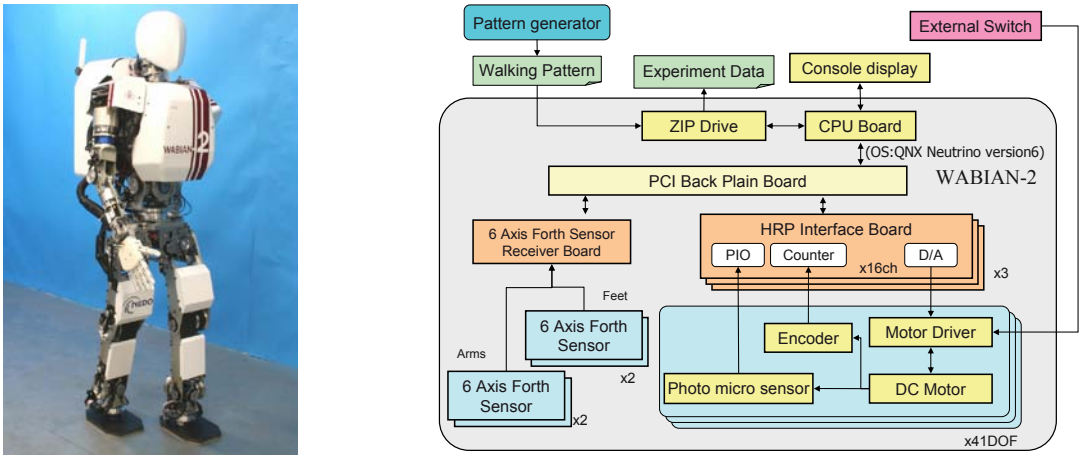


Abb. 2.20.: Der humanoide Roboter *Wabian-2* und die verwendete Rechnerarchitektur (Quelle: [114])

Wabian-2 - Hardware und Systemstruktur

Wabian-2 verfügt über 41 Bewegungsfreiheitsgrade und wiegt 64,5 kg bei einer Größe von 1,48 m. Die Bewegungsfreiheitsgrade verteilen sich wie folgt auf die einzelnen Körpersegmente: sechs Bewegungsfreiheitsgrade je Bein, zwei Bewegungsfreiheitsgrade in der Hüfte, zwei Bewegungsfreiheitsgrade im Bauchbereich, sieben Bewegungsfreiheitsgrade je Arm, drei Bewegungsfreiheitsgrade in den Händen und drei Bewegungsfreiheitsgrade im Halsgelenk. Zusätzlich ist die Fußsohle mit einem passiven Bewegungsfreiheitsgrad ausgestattet.

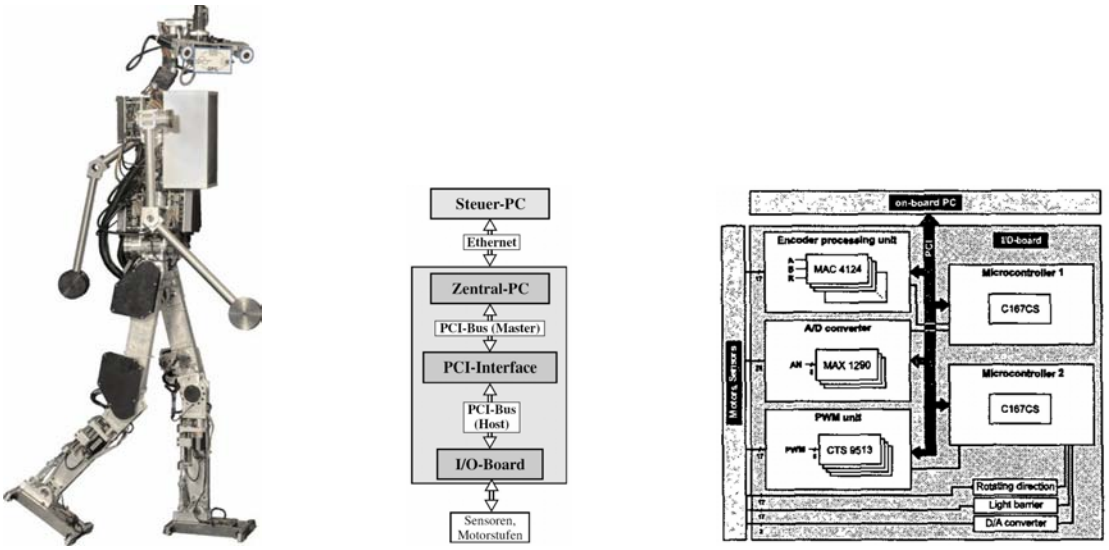
Die Computerhardware besteht aus einem einzigen zentralen Rechner, der vergleichbar einem Rucksack am Oberkörper des Roboters befestigt ist. Als Rechner wird eine Pentium-M basierende CPU-Steckkarte benutzt, die mit dem Roboter über das aus dem *HRP-II* bekannte HRP-Interfaceboard verbunden ist (Abb. 2.20).

Wabian-2 - Datentransfer und Peripherie

Bei *Wabian-2* sind für jede Achse ein Encoder am Motor und ein Encoder an der Achse vorgesehen. Beide Fußgelenke sind mit einem Kraft-Momenten-Sensor ausgestattet. Die Auswertung sämtlicher Sensoren erfolgt zentral über die am PCI-Bus des Rechners eingesteckten HRP-Interfaceboards.

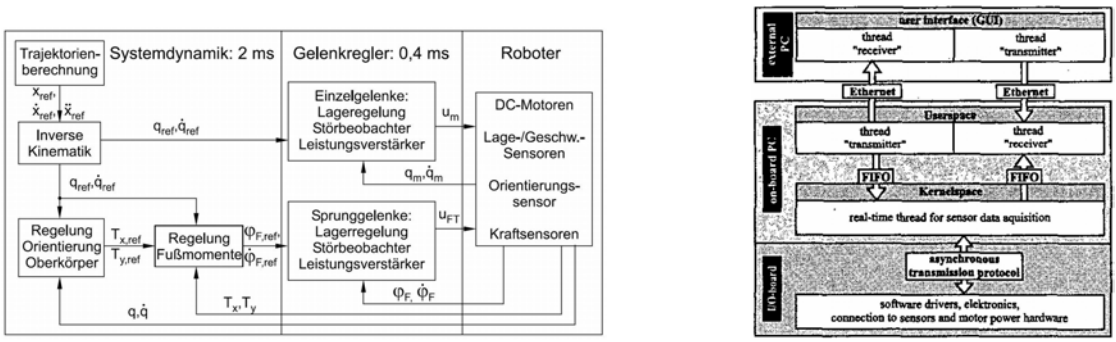
2.4.7. Johnnie

JOHNNIE wird von der „Technischen Universität München“ (TUM) im Rahmen des DFG-Schwerpunktprogramms SPP 1039 „Autonomes Laufen“ entwickelt. In diesem Projekt steht als Ziel der Aufbau eines zweibeinigen, autonomen Roboters im Vordergrund. Mit diesem soll es möglich sein, einen schnellen, dynamisch stabilen Gang zu erreichen. Mit dem aufgebauten



(a) JOHNNIE (b) Rechnerarchitektur (c) I/O-Board

Abb. 2.21.: Der humanoide Roboter JOHNNIE und die verwendete Architektur (Quelle: [103, 151])



(a) Steuerungsarchitektur (b) Softwarestruktur

Abb. 2.22.: Steuerungsarchitektur und Softwarestruktur auf JOHNNIE (Quelle: [151, 103])

System (Abb. 2.21(a)) konnten Gangeschwindigkeiten von 2,4 km/h und Schrittweiten bis zu 0,55 m erreicht werden [100].

Johnnie - Software und Steuerungsarchitektur

Zum Erreichen der geforderten Echtzeitausführung der Regelung wird auf dem Zentralrechner im Roboter RT-Linux als Betriebssystem eingesetzt. Die Softwarearchitektur auf *JOHNNIE* gliedert sich wie in Abb. 2.22 dargestellt in drei Ebenen. Die Ansteuerung der Motoren und die Auswertung der Sensoren wird durch die Treibersoftware realisiert, die asynchron mit einem Kernel-Thread im Echtzeitteil kommuniziert. Über FIFOs können weitere Userspace-Threads Daten mit dem Kernel-Thread austauschen und diese dann auf dem externen PC über das GUI darstellen (Abb. 2.22(b)).

Johnnie - Hardware und Systemstruktur

JOHNNIE verfügt über 17 angetriebene Bewegungsfreiheitsgrade. Er wiegt bei einer Größe von 1,8 m etwa 49 kg. Je Bein verfügt er über sechs Bewegungsfreiheitsgrade. Drei Gelenke befinden sich in der Hüfte, zwei weitere im Sprunggelenk. Die Arme dienen lediglich zur Stabilisierung der Laufbewegung und haben ein Schultergelenk mit zwei Bewegungsfreiheitsgraden. Die Sprunggelenke werden mit Kugelgewindeantrieben angetrieben. Alle andern Achsen werden mit Harmonic Drive Leichtbaugetrieben betrieben.

Am Institut wurden bereits einige Laufmaschinen entwickelt [123]. Für diese Roboter, wie beispielsweise die sechsbeinige Laufmaschine TUM-Laufroboter *Max* [160] bzw. den Kanalinspektionsroboter *Moritz* [163], wurde eine dezentrale Rechnerarchitektur verwendet. Aufgrund der stark gekoppelten Kinematik und Dynamik des zweibeinigen Robotersystems wurde für diesen eine zentrale Architektur bevorzugt [103]. Die für *JOHNNIE* entwickelte Rechnerarchitektur besteht aus einem Standard-PC im Micro-ATX Format mit einer Taktfrequenz von 2,8 GHz, welcher am Rücken des Roboters verbaut ist. Der komplette Roboter wird durch diesen *on-board* Rechner gesteuert. Zur Verbindung mit den Aktoren und Sensoren kommt eine leistungsfähige, selbst entwickelte PCI-Steckkarte zum Einsatz (Abb. 2.21(c)). Auf dieser Karte kommen zwei Infineon C167CS-Mikrocontroller, drei schnelle Analog-Digital-Konverter, 17 Encoder-Auswerteeinheiten und vier PWM-Einheiten zum Einsatz.

Johnnie - Datentransfer und Peripherie

JOHNNIE besitzt für jedes angetriebene Gelenk einen inkrementellen Quadratur-Encoder auf der Motorwelle. Zur initialen Positionierung befinden sich in den Gelenken Lichtschranken, die eine bestimmte Position im Arbeitsbereich auszeichnen [104, 102]. Die Auswertung dieser Encodersignale erfolgt zentral auf der PCI-Steckkarte im zentralen PC. Zur Bestimmung der Roboterpose im Raum wird eine Kombination aus drei Beschleunigungssensoren und drei Gyroskopen verwendet. Die Auswertung dieser Sensoren erfolgt ebenfalls auf der PCI-Steckkarte. Die Signale der 6D-Kraft-Momenten-Sensoren in den Fußgelenken werden von den Analog-Digital-Wandlern auf der Steckkarte ausgewertet.

Der Roboter verfügt durch die Integration des Steuerrechners im Rumpf des Roboters über ein hohes Maß an Autonomie. Lediglich die Stromversorgung und ein Ethernetkabel müssen nach außen geführt werden (Abb. 2.21(b)). Über das Netzwerkkabel ist *JOHNNIE* mit einem Laborrechner verbunden von dem abstrakte Bewegungsbefehle kommen und auf dem der Roboterzustand überwacht werden kann.

2.5. Zusammenfassung und Bewertung

In diesem Kapitel wurden Verfahren zur Hardware-Software-Steuerung humanoider Roboter vorgestellt. Für den Betrieb eines humanoiden Roboters ist es von grundlegender Bedeutung die Aspekte Softwarerahmenwerk, Hardwarerealisierung und funktional-logische Steuerung aufeinander abzustimmen.

Bei der Analyse der vorgestellten Lösungen fällt auf, dass sie nur ein sehr spezielles, auf den konkreten Anwendungsfall bezogenes Problem adressieren. Die analysierten Rahmenwerke, Hardwarerealisierungen und Steuerungsarchitekturen sind jeweils auf das konkrete Robotersystem ausgelegt. Der Überblick über die realisierten humanoiden Roboter zeigt, dass sich diese in ihrer Auslegung und auch in der mit ihnen untersuchten Fragestellung deutlich unterscheiden (Tab. 2.2). Damit sind auch die Anforderungen an das System sehr heterogen und speziell. Somit sind auch die Lösungsansätze nicht allgemeingültig und lassen sich nicht für den Aufbau anderer humanoider Roboter übertragen. Bisher existieren keine Standards oder Entwurfsmethoden für den Aufbau humanoider Roboter. Damit eine Hardware-Software-Steuerung auf unterschiedlichen humanoiden Robotersystemen eingesetzt werden kann, muss sie einerseits eine Reihe von Grundanforderungen erfüllen und flexibel an zusätzliche, roboterspezifische Anforderungen angepasst werden können. Andererseits sollte für den Entwurfsprozess und für die Anpassung an weitere Anforderungen eine klare Vorgehensweise beschrieben werden.

Robotersystem	Vorrangiges Ziel	Software	Hardware	Kommunikation
Asimo	Erforschung zweibeinigen Laufens	agentenbasiertes, ereignisgesteuertes verteiltes System	2 × PC, dezidierter Prozessor für Steuerungsaufgaben , DSP für Akustiklokalisation	Ethernet, WLAN
H7	Untersuchung der Kopplung zwischen Wahrnehmung und Aktion	RT-Linux, zentrale hierarchische Steuerung	750 MHz PC, Robot-Interface-Card	PCI-Bus, FireWire, I2C, WLAN
HRP-2	Zweibeiniges Laufen und Interaktion mit dem Menschen	ART-Linux, OpenHRP, CORBA	2 × 1 GHz Steckkarten-CPU	Ethernet, WLAN
KHR / KHR-2	Einsatz als Assistenzroboter und zu Unterhaltungszwecken	MS-DOS bzw. XP mit RT-Erweiterung, zentrale Architektur	1 × 500 MHz PC/104-System, Mikrocontroller	CAN-Bus, WLAN
Wabian-2	Nachahmung verschiedener Bewegungsabläufe des menschlichen Ganges	QNX, hierarchische Steuerung, offline generierte Laufmuster	1 × Steckkarten-CPU, HRP-Interfaceboards	PCI
Johnnie	Zweibeiniges Laufen und dynamisch stabiler Gang	RT-Linux, zentrale Architektur	PC, Mikrocontroller	Ethernet, proprietäres Protokoll

Tab. 2.2.: Überblick über die Forschungsschwerpunkte und verwendeten Technologien in den beschriebenen Robotikprojekten

3. Vorgaben und Anforderungen

Die Analyse der Anforderungen an die Hardware-Software-Architektur erfolgt ausgehend von den Vorgaben, die durch die Vorstellungen, die der Nutzer von einem humanoiden Roboter hat, definiert sind. Zur Erfüllung dieser Vorgaben muss der Roboter eine Vielzahl von Fähigkeiten besitzen. Die Anforderungen, die diese Fähigkeiten an die drei Säulen der Hardware-Software-Architektur – also die Steuerungsarchitektur, die Softwarearchitektur und die Hardwarearchitektur – stellen, sollen in diesem Kapitel vorgestellt, strukturiert und formalisiert werden.

3.1. Vorgaben für humanoide Roboter

An einen humanoiden Roboter wird eine Vielzahl von Anforderungen gestellt. Diese Anforderungen ergeben sich hauptsächlich aus den Vorstellungen, die der Mensch bezüglich der Interaktion mit einem humanoiden Roboter hat. Damit sich der Roboter beispielsweise sicher in der Umgebung des Menschen aufhalten kann, müssen viele Voraussetzungen erfüllt sein. Der Roboter soll in der Lage sein, autonom in einer für den Menschen geschaffenen Umgebung agieren zu können. Er muss über die notwendige Mobilität verfügen, um seine jeweiligen Einsatzorte zu erreichen und um Hol- oder Bringdienste ausführen zu können. Da nicht alle benötigten Fähigkeiten bereits beim Aufbau des Roboters bekannt sind und die Möglichkeit bestehen soll, zur Laufzeit neue Fähigkeiten zu erwerben, muss der Roboter in der Lage sein zu lernen. Untersuchungen haben ergeben, dass für Menschen die Hemmschwelle für die Interaktion mit einem Roboter geringer ist, wenn dessen äußere Form vertraut ist und sie das Verhalten des Roboters gut einschätzen können [127], wobei es allerdings kulturelle Unterschiede gibt [87]. Dies lässt sich am besten dadurch erreichen, dass man dem Roboter eine menschenähnliche Gestalt gibt und auch sein Verhalten dem des Menschen nachempfindet. Damit eine Integration des Roboters in alltägliche Abläufe des Menschen nahtlos möglich ist, muss Interaktion mit dem Roboter über dem Menschen geläufige Modalitäten erfolgen. Dies sind vor allem natürliche Sprache, aber auch nonverbale Kommunikation durch Gesten oder auch Interaktion über haptischen Kontakt. Neben der Interaktion zur Kommunikation soll es auch möglich sein, in Kooperation mit dem Roboter gemeinsam Aufgaben zu erledigen, wie zum Beispiel das Tragen einer schweren oder sperrigen Last. Nicht zuletzt muss sichergestellt werden, dass vom Roboter keine Gefahr für den Menschen ausgeht. Dazu muss gewährleistet werden, dass der Roboter ständig in einem sicheren Zustand ist. Einsatzszenarien für humanoide Roboter sind beispielsweise als Assistenzroboter im Haushalt, so kann der Roboter zum Beispiel alltägliche Aufgaben in der Küchenum-

Anforderung	Beschreibung
Autonomie	Roboter soll in der Lage sein, eigenständig und unabhängig zu agieren
Mobilität	Roboter soll Ziele eigenständig erreichen können und Hindernisse vermeiden
Lernfähigkeit	Roboter soll neue Fähigkeiten erwerben können
Menschenähnlichkeit	Absenkung der Hemmschwelle beim Umgang mit dem Roboter
Interaktionsfähigkeit	Interaktion mit dem Roboter über Sprache, Gestik und Haptik
Kooperationsfähigkeit	Roboter soll zusammen mit dem Menschen Aufgaben erledigen
Sicherheit	Zum Schutz des Menschen muss das System in einem sicheren Zustand sein

Tab. 3.1.: Anforderungen an humanoide Roboter

gebung erledigen. Hier müsste der Roboter Aufgaben wie Tisch decken, Gegenstände aus dem Kühlschrank holen, oder Spülmaschine ausräumen beherrschen. Ein anderes Anwendungsszenario stellt der Einsatz im Bereich der Altenpflege dar, wo der Roboter zur Unterstützung alter Personen herangezogen werden kann und somit deren Eigenständigkeit im häuslichen Umfeld verbessert. Der Einsatz von humanoiden Robotern im industriellen Umfeld ist genauso denkbar wie der Einsatz auf Messen oder in Museen, dort könnte der Roboter Führungen übernehmen oder als mobile Informationsquelle dienen. Dieser Überblick über die Einsatzszenarien verdeutlicht, wie breit das Spektrum möglicher Einsatzbereiche des humanoiden Roboters ist und wie universell dadurch das Anforderungsprofil an den Roboter wird. Zusammenfassend lassen sich die in Tab. 3.1 aufgelisteten Aspekte als Anforderungen formulieren.

3.2. Roboterfähigkeiten

Jede der in Tab. 3.1 aufgelisteten Anforderungen wird feiner unterteilt und es wird untersucht über welche Fähigkeiten der Roboter verfügen muss, um die gestellten Anforderungen zu erfüllen. Dabei wird auch darauf eingegangen, welche Sensoren und Aktoren zum Erreichen dieser Fähigkeiten notwendig sind. Anhand der sich aus dieser Analyse ergebenden Liste der geforderten Fähigkeiten wird dann im folgenden Abschnitt abgeleitet, welche Anforderungen diese an die Hardware-Software-Architektur stellen.

Am Beispiel der Anforderung Autonomie soll hier das Vorgehen skizziert werden, das für alle Anforderungen durchgeführt wurde. Damit ein Roboter autonom sein kann, muss er über eine Vielzahl an Fähigkeiten verfügen. Auf technische Aspekte wie beispielsweise eine mitgeführte Stromversorgung soll hier nicht eingegangen werden, sondern nur auf Aspekte, die Roboterfähigkeiten beschreiben. Damit der Roboter autonom, das heißt selbständig, in und mit

seiner Umgebung interagieren kann, muss er beispielsweise in der Lage sein, sich in dieser Umgebung zu bewegen also zu navigieren. Für die Fähigkeit Navigation muss einerseits eine Wahrnehmung der Umwelt erfolgen. Dazu dienen navigationsbezogene Abstandssensoren wie Laserscanner oder Ultraschallsensoren, aber auch Kamerasysteme. Andererseits muss sich der Roboter in seiner Umgebung fortbewegen können, sei es wie der Mensch mit zwei Beinen oder mittels einer mobilen Plattform. In beiden Fällen ist für die Fortbewegung Aktorik nötig, die geregelt werden muss. Das heißt, es muss sensoriell überwacht werden, ob die Bewegungsbefehle erfolgreich umgesetzt werden. Zur Erfassung von Bewegungen kommen Sensoren zur Positions- und Geschwindigkeitsbestimmung zum Einsatz. Neben diesen beiden Größen ist oft auch eine Überwachung von Kräften und Drehmomenten oder im Falle von Elektromotoren die Kontrolle der Motorströme sinnvoll. Damit der Roboter Einfluss auf die Umwelt nehmen kann, muss er über Manipulationsfähigkeiten verfügen. Zur Manipulation gehört vor allem die Bewegung der Arme und der Hände, wozu wiederum einerseits Aktorik und andererseits Sensorik zur Regelung benötigt wird. Als Aktoren für den Antrieb der Arme und Hände kommen vor allem Elektromotoren in Frage, aber auch pneumatische oder hydraulische Systeme. Auf Sensorikseite werden hier auch Positions-, Geschwindigkeits-, Kraft- und Drehmomentsensoren benötigt. Damit die Manipulation erfolgreich sein kann, müssen noch Informationen über das zu greifende Objekt ermittelt werden. Hierzu ist die Fähigkeit Objekterkennung notwendig. Die Objekterkennung liefert über bildgebende Sensoren einerseits Auskunft über die Art des Objekts und andererseits über die Position und Lage im Raum. Da der Roboter mit Menschen interagieren soll, muss er auch die Fähigkeit besitzen, Menschen zu erkennen und diese im Fokus zu behalten. Die beiden Fähigkeiten Personenerkennung und Personenverfolgung beruhen also hauptsächlich auf bildgebenden Sensoren und der Fähigkeit diese so auszurichten, dass sie den Menschen wahrnehmen können. Die Fähigkeit Kollisionsvermeidung bezieht sich auf die Vermeidung von Kollisionen bei der Navigation, bei der Manipulation und auf die Vermeidung von Eigenkollisionen. Für diese Fähigkeit werden Informationen aus den gleichen Sensoren, wie sie zur Navigation und Manipulation verwendet werden, herangezogen. Die bisher genannten Punkte beziehen sich alle direkt auf eine Interaktion mit der Umwelt, neben diesen gibt es noch Fähigkeiten, die rein roboterintern einen wichtigen Beitrag dazu leisten, dass dieser autonom in der Umgebung agieren kann. Der Aufbau eines Umweltmodells und das Erstellen eines Handlungsplans dienen zur Planung von Handlungen und zum Aufbau von Wissen über die Umgebung ohne das der Roboter nicht autonom agieren könnte.

In Tab. 3.2 wird dargestellt welche Fähigkeiten zur Erfüllung der vorgestellten Anforderungen notwendig sind. Zusätzlich wird eine Bewertung abgegeben, wie wichtig eine Fähigkeit zur Erfüllung der jeweiligen Anforderung ist und die Fähigkeiten gruppiert und einer der vier Domänen Bildverarbeitung, Akustik, Planung oder Regelung zugeordnet. Ein weiteres Ergebnis der Aufschlüsselung nach Fähigkeiten war die Erkenntnis, welche Sensoren und Aktoren für

diese benötigt werden. Diese werden im Abschnitt 4.3 näher beschrieben.

3.3. Anforderungen an die Hardware-Software-Architektur

Die Hardware-Software-Architektur bildet die Grundlage für die Roboterfähigkeiten, die zum Erreichen der Vorgaben benötigt werden. Die Hardware-Software-Architektur stützt sich dazu auf drei Säulen: die Steuerungsarchitektur, die Softwarearchitektur und die Hardwarearchitektur. In den folgenden Abschnitten werden für jede dieser drei Säulen die Anforderungen in den Domänen Bildverarbeitung, Akustik, Regelung und Planung erörtert.

3.3.1. Anforderungen an die Steuerungsarchitektur

Aufgabe der Steuerungsarchitektur ist es, die drei Aspekte *sense-plan-act* geeignet miteinander zu verbinden und die logische Struktur der Steuerung vorzugeben (Abb. 3.1). Grundsätzlich müssen im Humanoiden sowohl reaktive als auch deliberative Komponenten vorhanden sein, da einerseits auf bestimmte Situationen reaktiv und schnell reagiert werden muss und andererseits komplexere Handlungen nicht ohne planendes Element zu realisieren sind. Beispielsweise ist die Kollisionsvermeidung oder auch das Vermeiden von Überlastsituationen nur mit reaktivem Verhalten zufriedenstellend zu erreichen. Andererseits sind die Anforderungen an den humanoiden Roboter so komplex, dass er in der Lage sein muss zu planen und ein Modell der Umwelt aufzubauen. Die gewünschten Manipulationsaufgaben und interaktiven sowie kooperativen Handlungen sind ohne deliberativen Anteil in der Steuerungsarchitektur nicht zu bewerkstelligen. Bezüglich der Steuerungsarchitektur bedeutet das, dass eine hybride Architektur verwendet werden muss. Wie diese genau aufgebaut wird, wird in Kapitel 4 beschrieben. Für die Steuerungsarchitektur ergeben sich in den einzelnen Domänen die in den folgenden Absätzen beschriebenen Anforderungen.

Steuerungsarchitektur für Bildverarbeitung

Im Bereich der Bildverarbeitung muss die Steuerungsarchitektur deliberative Komponenten besitzen, um ein Modell der Umwelt aufbauen zu können. Datenbanken müssen unter anderem für die Speicherung von Umweltwissen, wie relevante Objekteigenschaften oder Merkmale von zu erkennenden Personen, bereitgestellt werden. Strukturell ist die Bildverarbeitung im Perzeptions-Teil der Steuerungsarchitektur zu finden. Über eine Abstraktion und gegebenenfalls Fusion mit anderen Sensordaten kann ein Modell der Umwelt aufgebaut werden, welches über Datenbanken der Planungsebene zugänglich gemacht wird.

	Autonomie	Mobilität	Lernfähigkeit	Menschenähnlichkeit	Interaktionsfähigkeit	Kooperationsfähigkeit	Sicherheit	
Objekterkennung	+	+	+	○	+	+	+	Bildverarbeitung
Personenerkennung	+	○	○	+	+	+	+	
Personenvervolgung	+	○	○	+	+	+	+	
Gestenerkennung	○	○	+	+	+	+	+	
Spracherkennung	○	○	+	+	+	+	+	Akustik
Sprachverstehen	○	○	+	+	+	+	+	
Akustiklokalisation	○	○	○	+	+	+	○	
Sprechertrennung	○	○	○	+	+	+	○	
Sprachausgabe	○	○	○	+	+	+	○	
Roboterbewegungen	++	+	○	+	+	+	○	Regelung
Navigation	++	++	○	+	○	+	○	
Objektmanipulation	+	○	○	○	+	+	○	
Bewegliche Plattform	+	++	○	○	○	+	○	
Taktile Interaktion	○	○	+	+	+	+	+	
Umweltmodell	+	+	++	○	○	○	+	Planung
Kollisionsvermeidung	+	+	○	○	++	++	++	
Proaktives Handeln	○	○	+	+	+	+	++	
Handlungsplan	+	○	++	+	+	++	○	
Navigation	+	+	○	○	○	○	○	

Wichtigkeit der Fähigkeiten: ++: sehr wichtig +: wichtig ○: nicht wichtig

Tab. 3.2.: Bewertung der Wichtigkeit einzelner Fähigkeiten zur Erfüllung der Anforderungen und die Zuordnung zu den vier Domänen

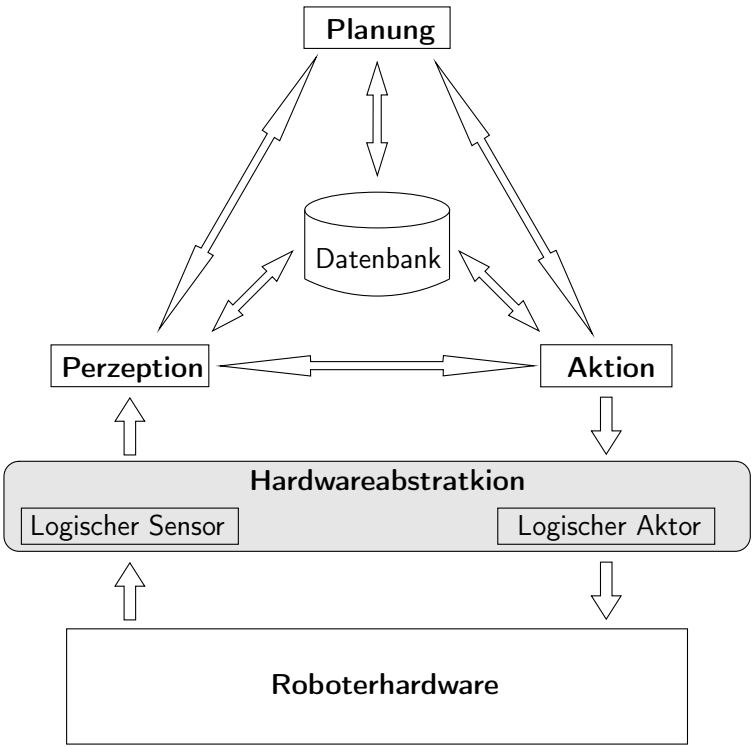


Abb. 3.1.: Abstrakte Darstellung der Steuerungsarchitektur und der Relationen der drei Bereiche Perzeption, Planung und Aktion. Der Zugriff auf die Roboterhardware wird durch logische Sensoren und Aktoren abstrahiert.

Steuerungsarchitektur für Akustik

Für die Akustik, also die Interaktion über Sprache und die Erkennung und Ortung von Geräuschen, reicht eine rein reaktive Steuerungsarchitektur nicht aus, weil für die Analyse und Synthese von Sprache und zur Erkennung von Wörtern und Sinnzusammenhängen bereits bekannte oder erlernte Begriffe in Datenbanken vorgehalten werden müssen. Eine rein reaktive Verarbeitung von Akustiksignalen ist ohnehin nur in wenigen Fällen sinnvoll. Die Akustik ist im Perzeptions-Teil der Steuerungsarchitektur angesiedelt und kann einerseits als direkter Kanal zur Benutzerinteraktion verstanden werden, andererseits können erfasste Signale im Zusammenhang mit anderen Sensorinformationen über Datenbanken der Planung zur Verfügung gestellt werden.

Steuerungsarchitektur für Regelung

Die Regelung stellt eine reaktive Komponente in der Steuerungsarchitektur dar. Für die Regelung ist ein schneller und zuverlässiger Zugriff auf die Sensordaten unabdingbar. Neben den reaktiven bzw. reflexartigen Komponenten der Regelung gibt es jedoch auch Teile der Regelung, die für eine Koordination und Synchronisation komplexerer Bewegungsabfolgen zuständig sind. Dieser Teil benötigt Zugriff auf abstrahierte und fusionierte Sensordaten unterschiedli-

cher Quellen, die beispielsweise in einer Datenbank vorgehalten werden können. Die Regelung ist im Aktions-Teil der Steuerungsarchitektur zu finden und realisiert über abstrahierte Aktorschnittstellen den Zugriff auf die Roboterhardware und ist somit für die tatsächliche Ausführung von Aktionen zuständig. Hierzu wird auf reflexartige reaktive Vorgaben zurückgegriffen, die direkt aus Sensorwerten Steuersignale für die Aktorik generieren.

Steuerungsarchitektur für Planung

Aufgrund des komplexen Anforderungsprofils und der in Tab. 3.1 explizit geforderten Lernfähigkeit des humanoiden Roboters, muss die Architektur über eine Planungskomponente verfügen. Da wie unter Regelung beschrieben auch eine reaktive Komponente in der Steuerungsarchitektur benötigt wird, ist diese nicht als rein deliberativ aufzubauen sondern als hybrid. Neben der direkten reaktiven Sensor-Aktor-Kopplung stellt die Planung das Bindeglied zwischen Sensorik und Aktorik dar. Die Planungsebene benötigt Zugriff auf mit Hilfe der Sensorik aufgebauter Modelle der Umwelt und auf ein Handlungsrepertoire. Die Funktion der Planungsebene ist es, eine durch den Nutzer oder eine übergeordnete Entscheidungskomponente vorgegebene Aufgabe auf die Roboterfähigkeiten abzubilden und durch Zugriff auf die Sensorik und Aktorik die Ausführung zu überwachen.

3.3.2. Anforderungen an die Softwarearchitektur

Das Softwarerahmenwerk für einen humanoiden Roboter muss einen abstrahierten Zugriff auf das komplexe Hardwaresystem humanoider Roboter zur Verfügung stellen. Dazu zählt das Auswerten vielfältiger und unterschiedlich anzusprechender Sensoren sowie die Ansteuerung unterschiedlicher Aktoren. Außerdem muss es die Programmierung komplexer Handlungen unterstützen, indem es diese in wiederverwendbare Module untergliedert. Domänenabhängig werden an das Softwarerahmenwerk sehr unterschiedliche Anforderungen an die Behandlung und Übertragung von Datenblöcken gestellt. Für die Softwarearchitektur ergeben sich in den einzelnen Domänen die in den folgende Abschnitten beschriebenen Anforderungen.

Softwarearchitektur für Bildverarbeitung

Für die Bildverarbeitung muss durch die Softwarearchitektur die Möglichkeit geschaffen werden, zwischen den an der Auswertung beteiligten Modulen große Datenmengen auszutauschen. Die Bildverarbeitung benötigt Schnittstellen zu Datenbanken zur Speicherung und zum Abgleich erkannter Objekte und eine Verbindung zu Prozessen der Planung. Für die Sensorfusion müssen Schnittstellen zu anderen Prozessen der Perzeption geschaffen werden. Bezüglich der Echtzeitfähigkeit werden an die Prozesse der Bildverarbeitung keine harten Anforderungen gestellt.

Softwarearchitektur für Akustik

Ebenso wie bei der Bildverarbeitung muss die Softwarearchitektur für die akustische Perzeption die Möglichkeit zum Austausch größerer Datenmengen bereitstellen. Für die Ortung von Schallquellen ist eine zeitlich exakt synchronisierte Verarbeitung der einzelnen Signalquellen von großer Bedeutung. Für die akustische Perzeption werden Schnittstellen zu Datenbanken, zu Prozessen der Planung und zu weiteren Prozessen der Perzeption benötigt. Die Anforderungen an die Echtzeitfähigkeit der Prozesse sind gering.

Softwarearchitektur für Regelung

Für die Regelung müssen aufgrund der verteilten Struktur der Sensoren und Aktoren Schnittstellen über ein geeignetes Bussystem zwischen den einzelnen Recheneinheiten hergestellt werden. Die Regelung benötigt Schnittstellen zu den Prozessen der Perzeption und der Planung. Der Schwerpunkt liegt in diesem Fall nicht auf der Übertragung größerer Datenmengen, sondern auf der hohen Zuverlässigkeit dieser Schnittstellen. Für die Prozesse der Regelung ist eine Ausführung in Echtzeit von hoher Relevanz.

Softwarearchitektur für Planung

Die Planung benötigt Zugriff auf unterschiedliche Informationsquellen. Es werden Schnittstellen zu den Datenbanken der Umweltmodelle und des Handlungswissens benötigt. Weiterhin müssen Schnittstellen zu den Recheneinheiten der Regelung realisiert werden. Wie unter dem Punkt Regelung beschrieben müssen diese Schnittstellen eine hohe Zuverlässigkeit haben und Echtzeitkriterien erfüllen. Die nicht regelungsbezogenen Planungsprozesse müssen nicht echtzeitfähig ausgelegt werden.

3.3.3. Anforderungen an die Hardwarearchitektur

Die Hardwarearchitektur beschreibt die Recheneinheiten und Verbindungseinrichtungen, mit denen die durch die Steuerungsarchitektur und Softwarearchitektur vorgegebenen Strukturen in Hardware realisiert werden. Hierzu zählt die Auswahl konkreter Hardwarekomponenten, die Definition von Bussystemen für den Datenaustausch zwischen Komponenten und das Bereitstellen von Schnittstellen zu Sensoren und Aktoren. Die topologische Verteilung der Komponenten wird maßgeblich durch die als menschenähnlich vorgegebene äußere Gestalt des humanoiden Roboters bestimmt. Mit folgenden Aspekten der Hardwarearchitektur lassen sich die Anforderungen quantitativ beschreiben:

Rechenleistung: Maß für die Anzahl an Rechenoperationen, die pro Zeiteinheit ausgeführt werden können. Ist besonders für rechenintensive Prozesse relevant.

Zykluszeit: Beschreibt, wie oft pro Zeiteinheit ein Prozess ausgeführt werden muss.

Speicherbedarf: Gibt an, wie viel Speicherplatz ein Prozess lokal auf der jeweiligen Recheneinheit benötigt.

Verbindungsbandbreite: Gibt an, welche Datenmengen zwischen Prozessen auf unterschiedlichen Recheneinheiten ausgetauscht werden müssen.

Die folgenden Abschnitte erörtern die konkreten Anforderungen an die Hardwarearchitektur in den vier Domänen.

Hardwarearchitektur für Bildverarbeitung

Für die visuelle Erfassung der Umgebung kommt auf dem Roboter üblicherweise ein System aus mindestens zwei Kameras zum Einsatz. Um für Manipulationsaufgaben eine ausreichende Präzision zu erreichen, müssen die Bilder der Kameras sehr hoch aufgelöst sein. Das führt dazu, dass zur Bildverarbeitung große Datenmengen unkomprimiert im Speicher der Recheneinheit vorgehalten werden müssen. Neben dem großen Speicherbedarf stellen die bei der Bildverarbeitung benötigten Operationen sehr rechenintensive Prozesse dar. Die Zykluszeit der Bildverarbeitung wird durch die bei Kameras üblichen Frameraten im Bereich von 20 bis 30 fps vorgeben und liegt somit im Bereich von 33 bis 50 ms. Die bei der Bildverarbeitung anfallenden Datenmengen können lokal auf der verarbeitenden Recheneinheit verbleiben und müssen nicht zu anderen Recheneinheiten übertragen werden. Datenübertragung findet in Form von stark abstrahierten Ergebnissen der Bildverarbeitung, wie erkanntes Objekt und 6D-Position im Raum, statt, welche im Datenvolumen sehr begrenzt sind.

Hardwarearchitektur für Akustik

Zur Interaktion über Sprache und zur Ortung des Sprechers bzw. von Geräuschen werden mehrere Mikrofone eingesetzt. Zur Geräuschortung ist es wichtig, die Signale der Mikrofone zeitsynchron aufzunehmen und zu verarbeiten. Hierzu müssen die zeitsynchron digitalisierten Akustiksignale im Arbeitsspeicher vorgehalten werden und dort segmentiert und analysiert werden. Sowohl die Analyse der Akustiksignale zur Spracherkennung als auch die Analyse der Korrelation der Signale zueinander zur Ortung des Geräuschursprungs stellen rechenintensive Prozesse dar. Für die Digitalisierung des Signals kommen üblicherweise Abtastraten von 40 kHz zum Einsatz. Allerdings sind die für das Rechnersystem relevanten Zykluszeiten deutlich niedriger, da hier jeweils mit längeren Segmenten des Signals gearbeitet wird. Wie bei der Bildverarbeitung findet auch bei der Spracherkennung und Geräuschortung eine Datenübertragung zu anderen Recheneinheiten nur mit geringer Bandbreite in Form von abstrahierten Ergebnissen statt. Die Sprachsynthese stellt in keiner der vier Aspekte nennenswerte Anforderungen an die Hardwarearchitektur.

Hardwarearchitektur für Regelung

Damit der Roboter motorisch mit der Umwelt interagieren kann, muss er über Aktoren und Sensoren verfügen. Als Aktoren kommen hauptsächlich Elektromotoren zum Einsatz. Damit die Bewegungen des Roboters kontrolliert erfolgen, muss der aktuelle Zustand des Roboters über Sensoren erfasst werden und in den Regelkreis einbezogen werden. Die wichtigsten, sensorisch erfassbaren Größen sind in diesem Zusammenhang Gelenkpositionen, Gelenkgeschwindigkeiten, Drehmomente und Motorströme. Vorgegeben durch die physikalische Struktur des Roboters sind sowohl Aktoren als auch Sensoren räumlich über den Roboter verteilt. Somit ist es zweckmäßig, die für die Bewegung des Roboters zuständigen Recheneinheiten analog zu den Aktoren und Sensoren auf dem Roboter zu verteilen. Die Hauptanforderung der Regelung an die Hardwarearchitektur liegt nicht in den Bereichen des Speicherbedarfs oder der Übertragungsbandbreite, sondern in der Garantierung kurzer Zykluszeiten im Bereich von 1 ms. Die Anforderungen an die Rechenleistung werden durch die strikten und kurzen Zyklen zur Herausforderung. Da die Recheneinheiten zur Regelung über den Roboter verteilt sein sollen, kommt der Übertragung von Sollwerten und Istwerten zwischen den Regelungskomponenten und den Planungskomponenten eine hohe Bedeutung zu. Vor dem Hintergrund der drastischen Folgen einer Fehleinschätzung des Roboterzustandes oder der Vorgabe falscher Sollpositionen ist an dieser Stelle die Zuverlässigkeit der Verbindungsstrecke von hoher Wichtigkeit. Die Anforderung an die verfügbare Bandbreite in der Größenordnung von 1 MB/s nimmt dazu im Gegensatz eine untergeordnete Rolle ein.

Hardwarearchitektur für Planung

Zur Ausführung komplexer Handlungen ist auf dem Roboter eine Komponente zur Planung erforderlich. Zur Planung gehört der Aufbau von Repräsentationen der Umwelt, die hohe Anforderungen an den verfügbaren Speicherplatz stellen. Die Berechnung komplexer Handlungsstrategien und der Aufbau und die Aktualisierung des Umweltmodells stellen rechenintensive Prozesse dar. Die Neuberechnung von Handlungsplänen stellt keine harten Anforderungen an die Zykluszeiten. Im Bezug auf den Zugriff und die Aktualisierung der Umweltmodelle, lassen sich diese so untergliedern, dass nur für einen kleinen Teil des Umweltmodells – also den für die aktuelle Handlungsaufgabe relevanten Teil – kurze Zykluszeiten gefordert werden. Eine Datenübertragung zu anderen Recheneinheiten lässt sich vermeiden, wenn die Planungskomponente und die Umweltmodelle lokal auf der gleichen Recheneinheit realisiert werden.

3.4. Zusammenfassung

In diesem Kapitel wurden die Vorgaben an einen humanoiden Roboter analysiert und daraus abgeleitet, welche Fähigkeiten der Roboter zur Erfüllung dieser Vorgaben benötigt. Die Fähig-

	Steuerungsarchitektur	Softwarearchitektur	Hardwarearchitektur
Charakteristische Eigenschaften	Struktur	Struktur	Rechenleistung
	Logische Verbindungen	Schnittstellen	Verbindungsbandbreite
		Modularisierung	Speicherbedarf
		Echtzeitfähigkeit	Echtzeitfähigkeit
		Zykluszeit	Zykluszeit

Tab. 3.3.: Quantitative Anforderungscharakteristika der drei Säulen der Hardware-Software-Architektur

keiten wurden klassifiziert und konnten den vier Domänen Bildverarbeitung, Akustik, Regelung und Planung zugeordnet werden. Für jede der drei Säulen der Hardware-Software-Architektur wurde dargelegt durch welche Eigenschaften sich die Anforderungen der jeweiligen Domänen charakterisieren lassen. Für die Steuerungsarchitektur sind das die Struktur und die logischen Schnittstellen. Die Anforderungen an die Softwarearchitektur lassen sich mit der Struktur, den Schnittstellen, der Modularisierbarkeit, der Echtzeitfähigkeit und der Zykluszeit beschreiben. Für die Hardwarearchitektur wurden als quantifizierbare Merkmale die Rechenleistung, die Zykluszeiten, die Echtzeitfähigkeit, der Speicherbedarf und die Verbindungsbandbreite identifiziert. Eine Übersicht ist in Tab. 3.3 dargestellt.

Die Steuerungsarchitektur muss hybrid aufgebaut sein, da durch das Anforderungsprofil der einzelnen Domänen sowohl reaktive als auch deliberative Aspekte abgedeckt werden müssen. Da die Steuerungsarchitektur zum Aufbau von Modellen, beispielsweise der Umwelt, nicht zustandslos sein kann, muss die Struktur der Steuerungsarchitektur Datenbanken beinhalten. Strukturell muss die Steuerungsarchitektur sowohl auf Sensorik- als auch auf Aktorikseite einen abstrahierten Zugriff auf die Roboterhardware bereitstellen. Als funktional-logische Verknüpfungen muss sie Verbindungen zwischen den Komponenten der Perzeption, der Planung und der Aktion zu Datenbanken und direkte Verbindungen unter diesen drei realisieren. Zusätzlich dazu muss eine Schnittstelle zu einer nicht näher spezifizierten übergeordneten Entscheidungskomponente bzw. zur Nutzerinteraktion geschaffen werden, über die die Planung ihre Eingaben erhält.

Die Softwarearchitektur orientiert sich in ihrer Struktur an der funktional-logischen Steuerung. Sie muss einen Rahmen für die Implementierung der Roboterfunktionalitäten setzen und die geforderten Schnittstellen zur Verfügung stellen. Aufgrund der topologischen Struktur des Roboters muss das Softwarerahmenwerk verteilte Prozesse unterstützen. Um die Entwicklungsarbeit auf die Entwicklung von Funktionalitäten zu konzentrieren, muss das Softwarerahmenwerk modular aufgebaut sein, so dass komplexe Funktionalitäten untergliedert und somit wiederverwendet werden können. Für bestimmte Prozesse muss eine echtzeitfähige Ausführung

	Datenaufkommen	Zykluszeiten	Rechenleistung
Bildverarbeitung (Stereokamerasystem mit 25 fps)	ca. 50 Mbit/s	ca. 40 ms	hoch
Mikrofonsystem bestehend aus 6 Mikrofonen	ca. 2 Mbit/s	ca. 1000 ms	hoch
Regelungsbezogen (20 Bewegungsfreiheitsgrade mit je zwei Sensoren)	ca. 1 Mbit/s	ca. 1 ms	mittel
Planung	ca. 10 Mbit/s	ca. 1000 ms	mittel-hoch

Tab. 3.4.: Domänenspezifische Anforderungen an das Rechnersystem

garantiert werden können.

Die Anforderungen an die Hardwarearchitektur stellen sich am heterogensten dar. Abhängig von der Domäne schwanken die Anforderungen sehr stark. Einerseits gibt es Komponenten, die ein sehr hohes Datenaufkommen produzieren und hohe Anforderungen an die verfügbare Rechenleistung stellen, dabei jedoch keine harten Anforderungen an die Echtzeitfähigkeit stellen und lange Zykluszeiten tolerieren. Andererseits gibt es Komponenten, die ein vergleichsweise geringes Datenaufkommen haben und mit überschaubarer Rechenkapazität beherrschbar sind, dafür allerdings harte Echtzeitbedingungen und sehr kurze Zykluszeiten fordern. Für einen exemplarischen Roboter sind in Tab. 3.4 und im Anhang A konkrete Zahlenwerte für die Anforderungen in den jeweiligen Domänen dargestellt, die diese Heterogenität verdeutlichen.

4. Konzeption der Hardware-Software-Architektur

Die Steuerung eines humanoiden Roboters und die Erfüllung der in Kapitel 3 aufgestellten Vorgaben lassen sich mit der Kombination

- einer in drei Hierarchieebenen gegliederten hybriden Steuerungsarchitektur,
- einer modularen Softwarearchitektur und
- einer verteilten, modularen und weitgehend auf Standardkomponenten basierenden Hardwarearchitektur

realisieren.

4.1. Funktional-logische Steuerungsarchitektur

Die Entscheidung über die Strukturierung der funktional-logischen Steuerung eines humanoiden Roboters hat maßgeblichen Einfluss auf die Software- und Hardwarearchitektur. Aus diesem Grund werden in dieser Arbeit die Aspekte der Hardware-Software-Architektur in der folgenden Reihenfolge behandelt: funktional-logische Architektur, Softwarearchitektur und Hardwarearchitektur.

Grundsätzlich ist es Aufgabe der Steuerungsarchitektur, die Verbindung zwischen den drei Bereichen Perzeption, Planung und Aktion herzustellen und deren konkrete Ausprägung zu beschreiben. Für rein reaktive Steuerungsarchitekturen entfällt als Sonderfall der Bereich Planung. Da, wie in Kapitel 3 dargelegt, im humanoiden Roboter zwingend der Bereich Planung vertreten sein muss, wird eine Steuerungsarchitektur des Typs hybride Steuerungsarchitektur vorgeschlagen. Im Folgenden soll – vorerst noch sehr abstrakt und generisch – beschrieben werden, welche Prozesse auf einem humanoiden Roboter in den drei Bereichen Perzeption, Planung und Aktion ablaufen müssen.

Die Steuerungsarchitektur beschreibt einerseits die Interaktionen und die logische Verknüpfung zwischen den drei Bereichen Perzeption, Planung und Aktion. Andererseits beschreibt sie auch die Funktionsabläufe innerhalb dieser drei Bereiche. An dieser Stelle wird aufgeführt, welche generischen Funktionsblöcke zum Betrieb eines Roboters, der die in Kapitel 3 gestellten Anforderungen erfüllen kann, notwendig sind. In Abb. 4.1 wird ein Überblick über die benötigten Funktionsblöcke der Steuerungsarchitektur gegeben. Im Folgenden werden diese, getrennt nach den Bereichen Perzeption, Planung und Aktion, näher beschrieben.

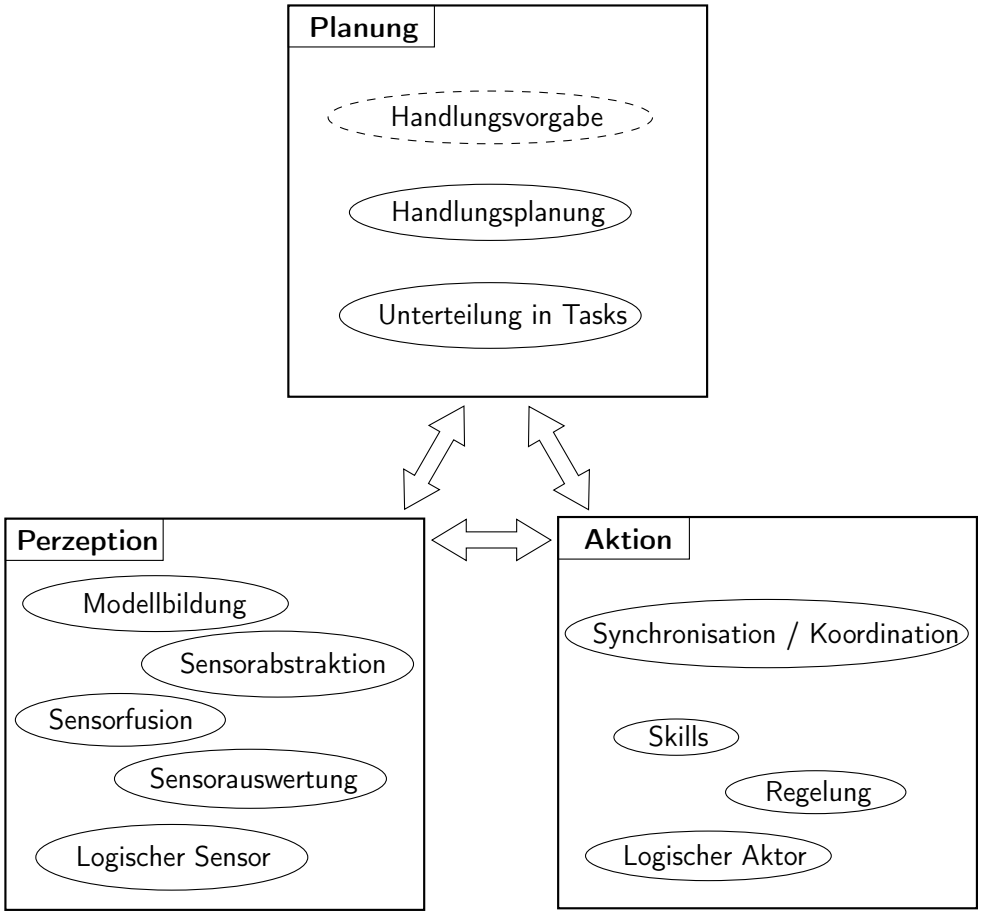


Abb. 4.1.: Funktionsblöcke in den drei Bereichen Perzeption, Planung und Aktion

4.1.1. Prozesse im Bereich der Perzeption

Die Signale der unterschiedlichen Sensortypen, wie zum Beispiel Positionssensoren, Kraftsensoren, Kameras oder Laserscanner müssen jeweils elektrisch erfasst werden und der Steuerung verfügbar gemacht werden. In Abschnitt 4.3.3 wird näher auf unterschiedliche Sensortypen eingegangen. Abhängig vom konkreten Sensor findet die Erfassung des Messwerts auf unterschiedliche Weise statt. Dies hat zur Folge, dass der Messwert in unterschiedlichen Repräsentationen vorliegen kann. Beispielsweise kann ein Positionsmesswert je nach Messverfahren als Spannungswert oder digitaler Zählerstand vorliegen. Seitens der Steuerung ist jedoch nur das vom Messverfahren unabhängige Messergebnis relevant. Um also eine einheitliche Repräsentation der Sensorwerte in der Steuerung zu gewährleisten, ist es notwendig, vom Messverfahren zu abstrahieren und einen so genannten logischen Sensor zu definieren. Dieser logische Sensor stellt der Steuerung dann unabhängig von der tatsächlichen Ausprägung des Sensors die relevante Information zur Verfügung. Um genauere bzw. neue Informationen aus den Sensorwerten zu gewinnen, können Sensorwerte fusioniert werden. Aufbauend auf den logischen Sensorwerten bzw. den Ergebnissen der Sensorfusion werden die Sensordaten interpretiert, ihnen also eine Bedeutung zugewiesen. Nach dem Zusammenführen unterschiedlicher Sensordaten, können an-

hand der semantischen Bedeutung der Sensorwerte aus diesen Modelle aufgebaut werden. Mittels der internen Sensorik kann ein Modell des Roboterzustandes aufgebaut bzw. aktualisiert werden. Mit Hilfe externer Sensorik, wie Kameras, können Informationen über die Umwelt erfasst werden und so Modelle der Umwelt oder von Objekten erstellt werden. Zusammengefasst fallen in den Bereich der Perzeption folgende Prozesse:

- Sensorwerterfassung
- Logischer Sensor als Schnittstelle zu konkreten Sensoren
- Fusion verschiedener Sensorwerte
- Interpretation der Sensordaten
- Modellbildung anhand der Sensordaten

4.1.2. Prozesse im Bereich der Planung

Im Bereich der Planung geht es darum, die Handlung des Roboters zu planen und zu entscheiden, welche einzelnen Handlungsabschnitte durchgeführt werden müssen, um diese Handlung abzuschließen. Der Anstoß bzw. der Antrieb eine Handlung zu starten, wird in dieser Arbeit als extern gegeben angenommen. Dieser äußere Antrieb wird beispielsweise durch eine Nutzerinteraktion mit dem Roboter gegeben. Einen wichtigen Kommunikationskanal für diese direkte Interaktion mit dem Nutzer stellt die Kommandierung per Sprache dar. Hier bekommt der Roboter vom Nutzer Befehle erteilt, die er dann in eine Handlung umsetzt. Eine weitere Interaktionsmöglichkeit stellt die Bedienung des Roboters über eine Rechnerschnittstelle dar, bei der der Nutzer sich über einen Rechner mit dem Roboter verbindet und den Roboter über eine grafische Bedienoberfläche steuert. Diese Art der Steuerung kann vor Ort oder im Sinne einer Telepräsenz auch über alternative Eingabemodalitäten, wie zum Beispiel einen Datenhandschuh zur Steuerung der Hand, erfolgen. Als Instanz zur Generierung von Handlungsabsichten kommt auch eine übergeordnete Entscheidungskomponente in Frage [36, 37]. Diese entscheidet, welche Handlung durchgeführt werden soll und interagiert nur bei Bedarf mit dem Nutzer. Zu den drei Möglichkeiten zum Anstoß einer Handlung – Nutzerinteraktion, Telepräsenz und übergeordnete Entscheidungskomponente – wird eine Schnittstelle vorgesehen. Über diese Schnittstelle wird vermittelt, welche Handlung auszuführen ist. Dieser Arbeit liegen folgende Begriffsdefinitionen zugrunde:

Handlung Eine komplexe Aufgabe, die der Roboter ausführen soll, wie zum Beispiel das Einräumen eines Gegenstandes in die Spülmaschine

Task Untergliederung der Handlung in einzelne Handlungsabschnitte, wie zum Beispiel das Greifen eines Objekts, das Öffnen der Spülmaschine und das Platzieren des Objektes in der Spülmaschine

Skill Elementare Fähigkeiten des Roboters, die zur Abarbeitung des Tasks benötigt werden, wie zum Beispiel bildbasiertes Greifen eines Objektes

In der Planung wird diese Handlung analysiert und basierend auf Modellwissen und den erfassten Sensordaten die Durchführbarkeit bewertet. Für durchführbare Handlungen wird untersucht welche Tasks zur Ausführung notwendig sind. Diese Information wird an den Bereich Aktion weitergeleitet und dort weiterverarbeitet. Zusammengefasst fallen in den Bereich der Planung folgende Prozesse:

- Antrieb zu Handlungen wird als extern angenommen (Übergeordnete Entscheidungskomponente, Nutzerinteraktion, Telepräsenz).
- Schnittstelle zu Nutzerinteraktion, Telepräsenz oder übergeordneter Entscheidungskomponente
- Handlungsplanung basierend auf Sensordaten und Modellwissen
- Abbildung der Handlung auf konkrete Tasks

4.1.3. Prozesse im Bereich der Aktion

Im Bereich der Aktion werden die von der Planung vorgegebenen Tasks weiter in die vom Roboter bereitgestellten Basisfähigkeiten, so genannte Skills (vergleiche Abb. 4.2), unterteilt. Um eine koordinierte Ausführung der Handlung zu gewährleisten, müssen die Tasks synchronisiert und koordiniert ausgeführt werden. Bei der Ausführung von Tasks wird oft konkurrierend der Zugriff auf diese Skills benötigt. Da den Skills konkret der Zugriff auf eine Roboterressource zugeordnet ist und diese nur exklusiv einem einzigen Task zur Verfügung gestellt werden kann, muss im Rahmen der Synchronisation und Koordination zusätzlich entschieden werden, welcher Task wann Zugriff auf die Hardware bekommt. Es wird also eine Ressourcenverwaltung benötigt, die den Zugriff auf die Skills bzw. die Roboterhardware koordiniert. Die Sensor-Motor-Skills greifen ihrerseits direkt auf die Regelung der Hardware zu. Die Regelung der Sensor-Motor-Ebene realisiert unterschiedliche Regelalgorithmen für die Achsen des Roboters. Um unabhängig vom tatsächlich verwendeten Aktortyp zu sein, wird ein so genannter logischer Aktor als Schnittstelle zur tatsächlichen Hardware eingesetzt. Zusammengefasst fallen in den Bereich der Aktion folgende Prozesse:

- Koordination und Synchronisation

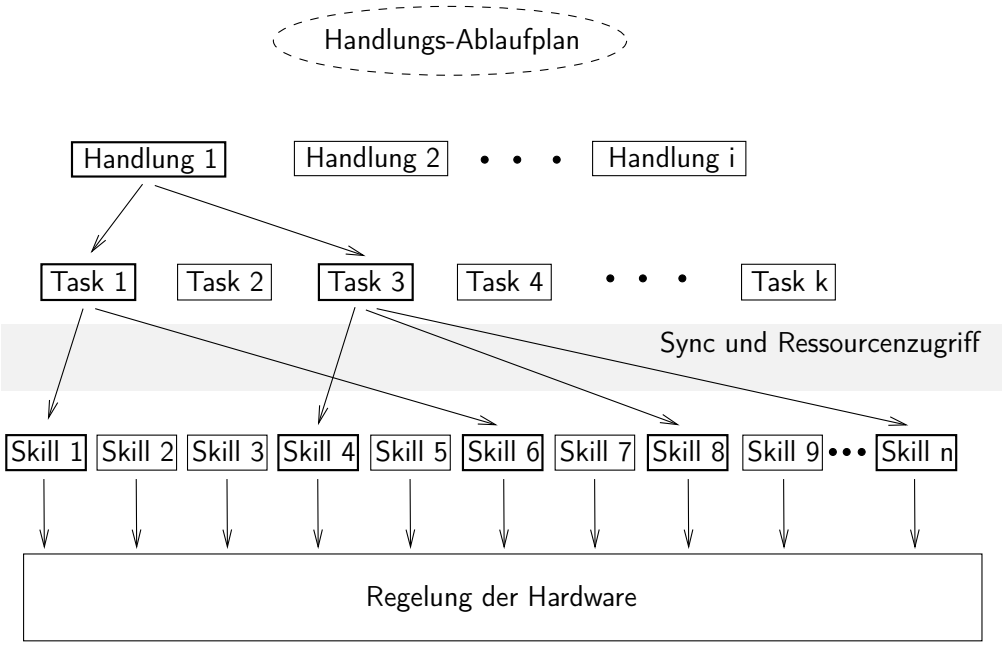


Abb. 4.2.: Untergliederung einer Handlung: Vom Ablaufplan wird eine Handlung ausgewählt, zu deren Ausführung bestimmte Tasks durchgeführt werden müssen. Diese Tasks lassen sich über so genannte Skills realisieren. Skills stellen elementare Fähigkeiten des Roboters bereit bzw. realisieren den Zugriff auf die Roboterhardware und die Aktorregelung.

- Roboterressourcenverwaltung (Zugriff auf Skills und Roboterhardware)
- Regelung auf Sensor-Motor-Ebene
- Logischer Aktor als Schnittstelle zur konkreten Aktorik
- Aktoransteuerung

Die oben beschriebene Untergliederung einer Handlung (Abb. 4.2) soll hier an einem kurzen Beispiel näher erläutert werden. Auf Ebene der Planung wird davon ausgegangen, dass eine konkrete Handlungsaufgabe über eine Nutzterschnittstelle, eine Telepräsenzsteuerung oder eine übergeordnete Entscheidungskomponente gestellt wird. Als Beispiel soll die Handlungsaufgabe „Stell die rote Tasse in die Spülmaschine“ dienen. Aufgabe der Planung ist es dann die Ausführung dieser Handlungsaufgabe zu planen und in einzelne Abschnitte so genannte Tasks zu zerlegen. Also im Beispiel: „Lokalisiere Tasse“, „Fahre zu Tasse“, „Greife Tasse“, „Lokalisiere Spülmaschine“, „Fahre zu Spülmaschine“, „Öffne Spülmaschine“, „Stelle Tasse ab“. Diese Tasks müssen weiter auf Basisfähigkeiten, die so genannten Skills, abgebildet werden. Skills sind beispielsweise „Bewege Plattform um x cm nach vorne und y cm zur Seite“, „Winkle Ellenbogen auf Winkel θ an“ oder komplexere Kombinationen daraus „Bringe TCP in Position x, y, z und Orientierung ψ, θ, ϕ“. Für die Ausführung solcher komplexerer Skills ist die Synchronisation und Koordination einzelner Skills notwendig. Um den TCP an eine feste Raumposition

und Orientierung zu bringen, muss eine Vielzahl von einzelnen Achsen in die entsprechende Position gebracht werden. Hierfür müssen die Achsen geregelt werden. Je nach Anforderung muss die Regelung unterschiedliche Aspekte kontrollieren wie zum Beispiel Position, Geschwindigkeit oder Drehmoment. Für den Zugriff der Regelung auf die tatsächlichen Aktoren, wie Elektromotoren unterschiedlicher Ausführung, Hydraulik oder Pneumatik, muss wiederum eine abstrakte Schnittstelle geschaffen werden, die als logischer Aktor bezeichnet wird. Dieser muss abstrakte Bewegungsbefehle an die unterschiedliche Aktor-Hardware, wie Elektromotoren, Hydraulikantriebe oder Pneumatikantriebe übermitteln. Der logische Aktor muss einen Satz von Bewegungsbefehlen zur Verfügung stellen, der unabhängig von der angesteuerten Hardware ist. Aufgabe der Abstraktionsschicht im logischen Aktor ist dann, diese Bewegungsbefehle wieder auf die konkreten Aktoren bzw. deren Ansteuerelektronik abzubilden.

4.1.4. Datenspeicherung und Datenaustausch

Für den Datenaustausch unter den Komponenten und zum Aufbau von Umweltmodellen und Handlungswissen müssen Daten in Datenbanken vorgehalten werden. Hierbei sind globale und lokale Datenbanken notwendig. In den globalen Datenbanken wird eine umfassende Sicht der Umwelt abgelegt. Zugriffe auf diese Datenbank sind nicht echtzeitkritisch. Ihr Inhalt ist für die aktuell ausgeführte Handlung nicht zwingend oder direkt notwendig. Im Gegensatz dazu werden in den lokalen Datenbanken Inhalte vorgehalten, auf die ein echtzeitfähiger Zugriff notwendig ist. Diese Daten werden direkt für die gerade ausgeführte Handlungsaufgabe benötigt und müssen sich im schnellen Zugriff befinden. Da die Modelle, die in dieser Datenbank vorgehalten werden, jeweils für die aktuelle Handlung relevant sind und nur einen Ausschnitt der gesamten Modelle darstellen, werden diese als aktive Modelle [11] bezeichnet.

4.1.5. Datenflüsse in der Steuerungsarchitektur

Neben den Funktionsblöcken müssen die Datenflüsse zwischen diesen Funktionsblöcken beschrieben werden und eine Vorgabe zum Austausch und Vorhalten von Daten gegeben werden. In Abb. 4.3 wird ein Überblick über die Funktionsblöcke, die Datenströme und Datenbanken gegeben. Um bei der heterogenen Sensorik im humanoiden Robotern unabhängig von der konkreten Sensorrealisierung, also zum Beispiel dem verwendeten Messverfahren oder dem Protokoll zur Datenübertragung, zu werden, wird in der Roboterabstraktionsebene der logische Sensor definiert. Dieser logische Sensor präsentiert gegenüber der Steuerungsarchitektur eine standardisierte Sensorschnittstelle. Diese Abstraktion vom tatsächlichen Sensor ermöglicht eine Anpassung der Sensorik bzw. einen Austausch von Roboterhardware, ohne dass dies eine Änderung in der Steuerungsarchitektur nach sich zieht. Somit kann auch Software, die auf den Sensorwerten aufbaut, unverändert weiterverwendet werden. Für bestimmte Sensortypen liefert der logische Sensor bereits Informationen, die nicht weiter ausgewertet werden müssen und

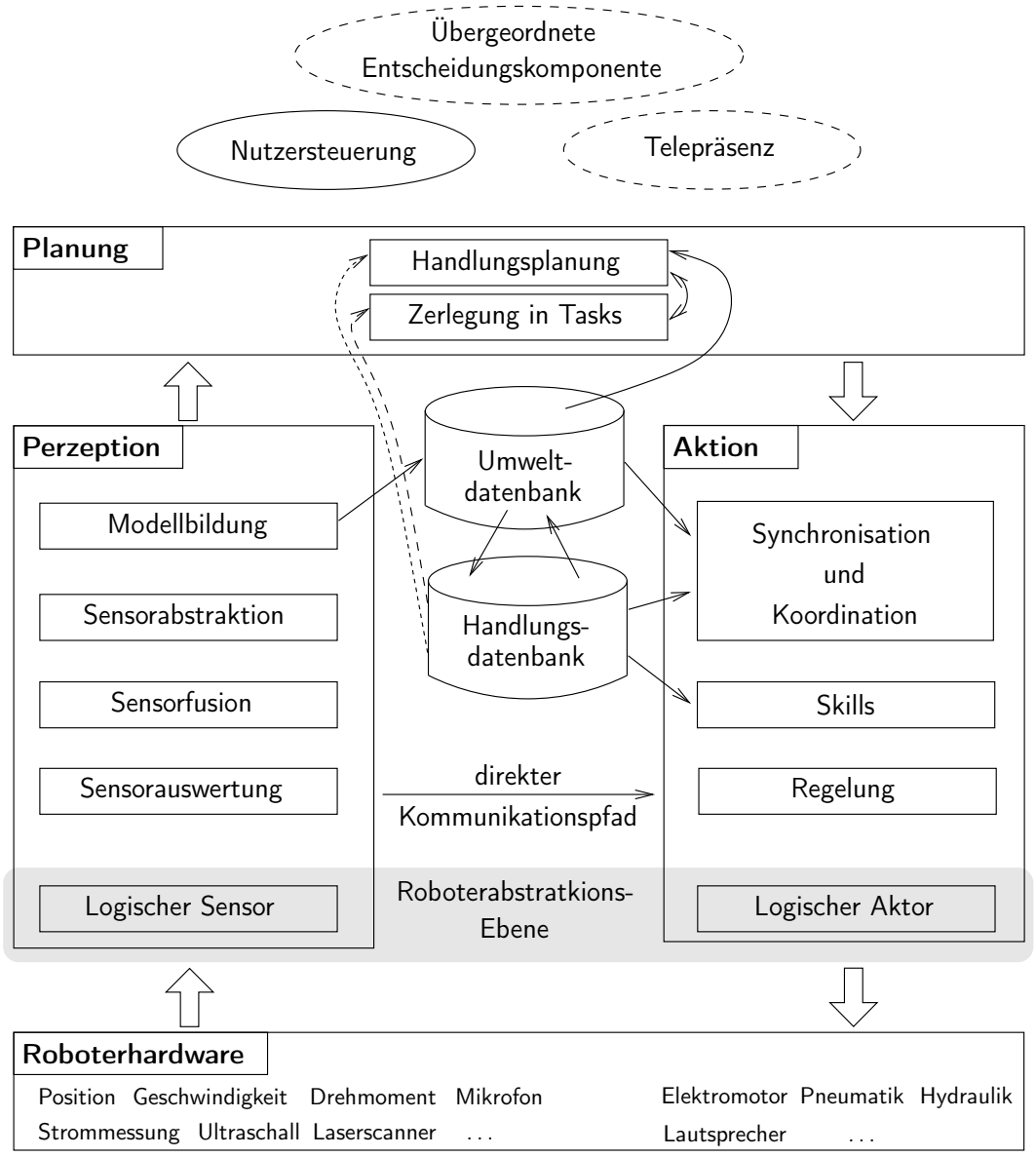


Abb. 4.3.: Funktionale Sicht auf das System humanoider Roboter

direkt weiterverarbeitet werden können. Beispielsweise gibt der logische Sensor für Positions- oder Geschwindigkeitsmessung direkt die benötigten Werte aus. Für komplexere Sensorwerte, wie beispielsweise Kamerabilder oder Punktwolken eines Laserscanners muss die Informationsextratktion noch auf den standardisierten Ausgaben des logischen Sensors erfolgen. Die Ergebnisse dieser Sensorauswertung werden je nach Verfügbarkeitsanforderung an die jeweiligen Daten in der Umweltdatenbank abgelegt bzw. direkt an die Aktorik kommuniziert. Durch Sensorfusion kann die Genauigkeit einer Messung durch Kombination von Sensoren, die auf die Bestimmung der gleichen Größe ausgerichtet sind, gesteigert werden. Ein klassisches Beispiel für diese Art der Sensorfusion ist die Kombination odometrischer Daten mit inertialen Messverfahren zur Positionsbestimmung. Somit kann durch die Sensorfusion ein besseres, in

diesem Fall genaueres, Messergebnis erreicht werden. Eine andere Form der Sensorfusion erlangt durch Fusion von Sensoren, die unterschiedliche Größen messen, neue Informationen. Als Beispiel kann hier die Kombination von Laserscannerdaten mit Kamerabildern angeführt werden. Hier liefert der Laserscanner eine Punktwolke und die Kamerabilder eine Textur – durch die Fusion lässt sich eine texturierte 3D-Ansicht des Objekts generieren. Im Rahmen der Sensorabstratktion werden die Ergebnisse der Sensorfusion bzw. direkt die Ergebnisse der Sensorauswertung analysiert und ihre Bedeutung ermittelt. An dieser Stelle findet also eine Verknüpfung des Messergebnisses mit Domänenwissen statt. Dies ist eng mit dem Aufbau von Modellen bzw. der Nutzung von Wissen aus dem Modell verbunden. Ausgehend von einem vorgegebenem Modell, kann die Sensorinformation abstrahiert und zur Aktualisierung des bestehenden Modells genutzt werden. Darüber hinaus kann das bestehende Umweltmodell über Lernprozesse, die sich auf sensorische Wahrnehmung stützen, erweitert werden. Das aktualisierte Modell der Umwelt wird wiederum in der Umweltdatenbank abgelegt, wo es auch den Prozessen der Planung und Aktion zur Verfügung steht.

Die Planung erhält Handlungsvorgaben aus einer vorerst als extern angenommenen Instanz. Diese kann durch eine übergeordnete Entscheidungskomponente realisiert werden, die eigenständig Handlungsaufgaben aufgrund von Kontextinformationen generiert. Für die angestrebte enge Zusammenarbeit mit dem Menschen eignet sich auch eine Variante, in der der Roboter Handlungen nicht aus eigenem Antrieb beginnt, sondern in der Handlungsabläufe explizit durch eine Nutzerinteraktion initiiert werden. Diese Nutzerinteraktion sollte über Modalitäten erfolgen, die dem Menschen geläufig sind. Eine intuitive Form der Interaktion mit dem Roboter ist die Kommandierung des Roboters über Sprache. Durch Sprachbefehle werden Handlungsaufgaben angestoßen, die dann in der Planung weiterverarbeitet werden.

Wenn der Roboter telepräsent bedient werden soll, sind weitere Installationen in der Infrastruktur des Raums notwendig. Als Rückkoppelung zur Steuerung sind zum Beispiel Kameras im Raum notwendig, die dem Nutzer den aktuellen Zustand des Roboters wiedergeben. Im Rahmen der Telepräsenz ergeben sich zusätzlich zur grafischen Benutzeroberfläche noch weitere Eingabemöglichkeiten wie zum Beispiel die Steuerung des Roboterarmes durch Handtracking des Benutzers [132]. Ausgehend von der über diese Kanäle initiierten Handlungsaufgabe wird im Rahmen der Planung ein Handlungsplan erstellt. Hierzu greift die Planung auf das in der Umweltdatenbank abgelegte Wissen über den Roboterzustand und die Umwelt zurück. Zusammen mit dem in der Handlungsdatenbank abgelegten Handlungswissen kann die Handlungsaufgabe in einzelne Tasks untergliedert werden. Diese Tasks werden dann zur Ausführung an den Bereich der Aktion übergeben.

Mit den Informationen aus Handlungsdatenbank und Umweltdatenbank wird im Bereich Aktion eine weiter Unterteilung auf die Skills vorgenommen. Diese stellen elementare Roboterfähigkeiten dar und greifen auf die Roboterhardware zu. Da bestimmte Skills und die Roboter-

hardware nur exklusiv verfügbar sind und von den Tasks konkurrierend angefordert werden können, muss auf dieser Ebene der Aktion eine Ressourcenverwaltung eingeführt werden. Diese wird im Rahmen der Synchronisation und Koordination der Tasks und Skills bereitgestellt. Das Wissen, welche Skills zur Erfüllung der Tasks benötigt werden und welche Restriktionen bezüglich der Reihenfolge und Priorisierung beachtet werden müssen, sind in der Handlungsdatenbank abgelegt. Die für den Zugriff auf die Roboterhardware zuständigen Skills greifen zur Roboterregelung auf Informationen der Perzeptionsseite zu. Diese Sensorinformationen werden genutzt, um unterschiedliche Regelalgorithmen zur Sensor-Motor-Regelung der Roboterachsen zu implementieren. Zu den Basisregelalgorithmen, die zur Robotersteuerung benötigt werden, zählen Positions- und Geschwindigkeitsregler sowie Drehmomentregler. Genau wie für die Sensorik gilt auch auf der Aktorikseite, dass es eine Vielzahl unterschiedlicher möglicher Aktoren zum Antrieb der Roboterachsen gibt. Typische Antriebe im humanoiden Roboter sind unterschiedliche Varianten elektrischer Motoren wie bürstenbehaftete und bürstenlose Motoren, aber auch pneumatische oder hydraulische Antriebe. Um im Bereich der Aktion weitgehend unabhängig vom tatsächlich verwendeten Aktor zu sein, greift die Regelung über eine standardisierte Schnittstelle – den logischen Aktor – auf die Hardware zu. Der logische Aktor abstrahiert von der Hardware und stellt der Regelung eine klar definierte Schnittstelle zur Verfügung.

In der hier vorgestellten Steuerungsarchitektur wird der deliberative Teil der Steuerungsarchitektur über verschiedene Hierarchieebenen von der Sensorik zur Planung und von der Planung über die Hierarchieebenen der Aktion realisiert. Über miteinander verknüpfte Datenbanken wird das Umweltmodell aufgebaut und durch die sensorische Umweltwahrnehmung aktualisiert und erweitert. In der Handlungsdatenbank sind Informationen über das Handlungsrepertoire des Roboters abgelegt, welches durch Lernprozesse erweitert werden kann. Reaktive Strukturen lassen sich mit der vorgestellten Steuerungsarchitektur über den schnellen Pfad von der Sensorauswertung zur Regelung realisieren. Auf diese Weise können neben dem deliberativen Anteil, der zur planvollen Ausführung der komplexen Aufgaben eines humanoiden Roboters unabdingbar ist, auch reflexartige Verhalten implementiert werden.

4.2. Softwarearchitektur

Für den Begriff Softwarearchitektur gibt es eine Vielzahl von unterschiedlichen Definitionen. Das Software Engineering Institute der Carnegie Mellon Universität führt eine Liste der gebräuchlichsten Sichtweisen[1].

Im Folgenden soll die von Bass et al. stammende verwendet werden:

> „The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the external properties of those components, and the relationship among them“
>
> *[18]*

Mit anderen Worten lässt sich die Aufgabe der Softwarearchitektur wie folgt zusammenfassen: Die Softwarearchitektur legt die übergeordnete Organisation des Softwaresystems inklusive der globalen Kontrollstrukturen fest. Sie beschreibt die Softwarekomponenten im System und deren Schnittstellen, die sie nach außen bereit stellt. Das aus den Softwarekomponenten aufgebaute System erfüllt alle an das System gestellten fachlichen und technischen Anforderungen. Sie legt die Interaktionen zwischen den einzelnen Softwarekomponenten fest [50].

Humanoide Roboter erfordern die Integration einer Vielzahl heterogener Hardwarekomponenten in einem komplexen System. Um aus Sicht der Softwaresystementwicklung einen vereinheitlichten Zugriff auf die heterogenen Hardwarekomponenten und Rechenmittel, der den unterschiedlichen Anforderungen an Echtzeitfähigkeit und Rechenleistung gerecht wird, zu ermöglichen, ist ein strukturiertes Softwarerahmenwerk und eine leistungsfähige Middleware erforderlich. Die Softwarearchitektur muss die Struktur der funktional-logischen Architektur widerspiegeln. Sie muss also auf Softwareseite die Abstraktion der tatsächlichen Sensorik und Aktorik realisieren und entsprechende Module für die Hierarchieebenen Task-Ausführung, Task-Koordination und Task-Planung bereitstellen [106]. Als Softwarerahmenwerk soll das am FZI entwickelte MCA2 [135, 150] verwendet werden.

4.2.1. Rahmenwerk MCA2

Als strukturgebender Rahmen für die Softwarearchitektur soll das am FZI entstandene Softwarerahmenwerk MCA2 dienen. MCA2 unterstützt durch seine komponentenbasierte Struktur mit klar definierten Schnittstellen die Unterteilung eines komplexen Gesamtsystems in wiederverwendbare Funktionsblöcke. Durch die Vorgabe von Kommunikations- und Synchronisationsmechanismen durch das Rahmenwerk kann der Schwerpunkt der Entwicklungsarbeiten auf die Bereitstellung von Roboterfähigkeiten gelegt werden. Da MCA2 als Programmiersprache C++ zugrundeliegt und für die Betriebssysteme Windows, Linux und MacOS zur Verfügung steht,

[1] `http://www.sei.cmu.edu/architecture/published_definitions.html`

wird ein hohes Maß an Plattformunabhängigkeit erreicht. Unter Linux lassen sich mit MCA2 unter Verwendung von beispielsweise RTAI/LXRT echtzeitfähige Prozesse realisieren.

Das Rahmenwerk macht für die kleinste Gestaltungseinheit – das Modul – klare Vorgaben über die bereit zu stellenden Schnittstellen und die Verbindungsmechanismen zwischen einzelnen Modulen. Innerhalb des Moduls wird die zyklische Ausführung der zwei Funktionen `Sense()` und `Control()` vorgeschrieben. Der Inhalt bzw. die Implementierung dieser beiden Funktionen bestimmt die Eigenschaften des Moduls. Die Ausführung der Funktionen `Sense()` und `Control()` lässt sich zur Laufzeit über die Parameterschnittstelle beeinflussen. Auf weitere interne Variablen innerhalb des Moduls besteht von außerhalb des Moduls kein Zugriff. Aus Implementierungssicht entspricht ein MCA-Modul einer C++-Klasse und stellt über entsprechende Funktionen die folgenden Schnittstellen zur Verfügung:

Controller Input Über diese Schnittstelle empfängt das Modul von einem in der Hierarchie weiter oben angesiedelten Modul Kontrolldaten. Diese werden dann innerhalb der Funktion `Control()` verarbeitet.

Controller Output Diese Schnittstelle stellt den Ausgang für die Kontrolldaten dar. Über sie können die Ergebnisse aus der Ausführung der Funktion `Control()` an andere Module, die weiter unten in der Hierarchie angesiedelt sind, übertragen werden.

Sensor Input Schnittstelle zum Modul, die entweder Daten eines physikalischen Sensors empfängt bzw. vorverarbeitete Sensordaten aus anderen MCA-Modulen niedrigerer Hierarchieebenen entgegennimmt. Diese Daten werden daraufhin in der Funktion `Sense()` verarbeitet.

Sensor Output Das Ergebnis der Funktion `Sense()` kann über diese Schnittstelle weiteren, hierarchisch höher liegenden Modulen, zur Verfügung gestellt werden.

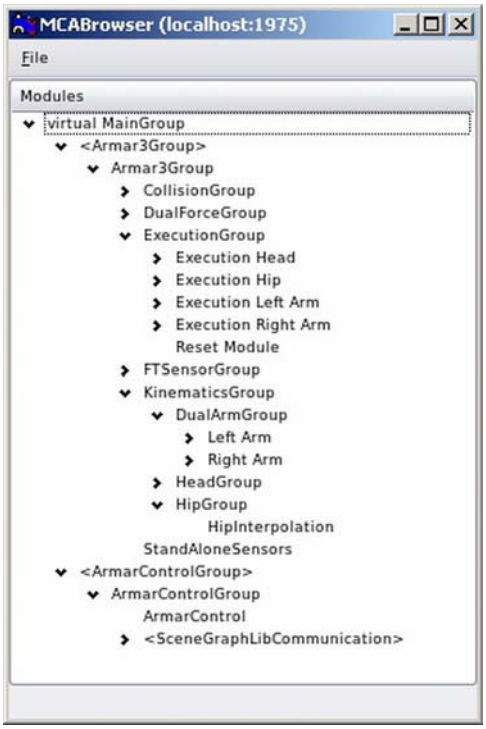
Parameter Über diese Schnittstelle lassen sich die internen Parameter des Moduls verändern.

MCA2 schreibt eine ebenenweise, hierarchische Anordnung der Module vor und lässt über die Schnittstellen nur ebenenübergreifende Kommunikation zu. Der Datenaustausch über die Schnittstellen findet in Form von Arrays aus *double*-Werten, so genannten Vektoren, statt. Unter Berücksichtigung dieser Gestaltungsregeln lässt sich aus den Modulen eine Softwarearchitektur aufbauen. Dies wird durch Verbinden der Ausgangsschnittstellen der Module mit den Eingangsschnittstellen anderer Module über die so genannten Kanten realisiert. Um eine möglichst gute Wiederverwendbarkeit der Module zu gewährleisten, muss die Modularisierung sehr feingranular erfolgen. Allerdings würde dies dazu führen, dass eine sehr große Anzahl von Modulen verschaltet werden müsste. Um das System übersichtlich zu halten, sieht MCA2 hierfür vor, mehrere Module zu einer so genannten Gruppe zusammen zu fassen. Dadurch wird sowohl eine gute Wiederverwendbarkeit einzelner Module gewährleistet als auch die Übersichtlichkeit

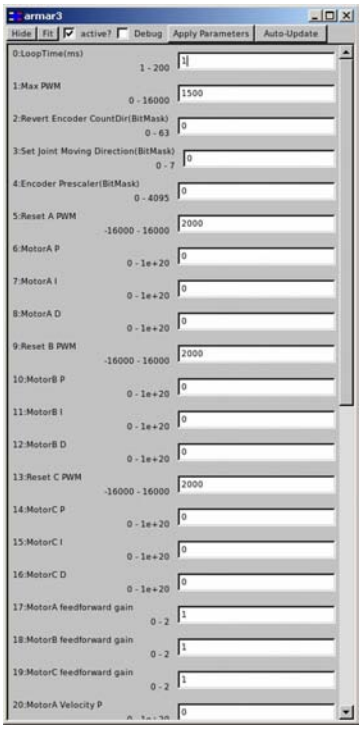
der Gesamtarchitektur sichergestellt. Eine Gruppe verhält sich selbst wie ein Modul, das heißt, sie verfügt bis auf die Parameterschnittstelle über die gleichen Kanten wie ein normales Modul. Innerhalb der Gruppe werden die beinhalteten Module ebenenweise angeordnet und wie angegeben verschaltet (Siehe hierzu Abb. 2.5 und Abb. 2.6). Die Gruppe stellt, wie ein Modul, die Funktionen `Sense()` und `Control()` zur Verfügung. Beim Aufruf dieser Funktionen der Gruppe werden sukzessiv die Funktionen der beinhalteten Module aufgerufen. Für den Kontrolldatenfluss erfolgt der Aufruf der `Control()`-Funktion der Module ebenenweise in der Hierarchie von oben nach unten. Analog wird für den Sensordatenfluss die `Sense()`-Funktion der Module ebenenweise von unten nach oben aufgerufen. Da sich Gruppen nach außen wie Module verhalten, können sie ihrerseits wieder zu Gruppen zusammengefasst werden. Auf diese Weise lässt sich die Gesamtsteuerung als eine große Gruppe realisieren, die durch Verschachteln von Modulen und Gruppen aufgebaut ist.

Um Teile der Steuerung asynchron zum Rest ausführen bzw. um für einzelne Teile der Steuerung unterschiedliche Zykluszeiten realisieren zu können, besteht die Möglichkeit, Module in so genannte *TaskContainer* einzubetten. Innerhalb des *TaskContainers* werden die Funktionen `Control()` und `Sense()` des Moduls in einem eigenen Thread aufgerufen. Der *TaskContainer* wird anstelle des Moduls in die Gruppe eingebunden. Er präsentiert nach außen hin die gleichen Schnittstellen wie ein Modul und verfügt auch über die beiden Funktionen `Control()` und `Sense()`. Allerdings werden beim Aufruf der Funktionen des *TaskContainers* nicht die Funktionen des Moduls ausgeführt, sondern nur die jeweils aktuellen Werte aus der letzten Ausführung synchronisiert. Durch dieses Verfahren wird ein nichtblockierender Synchronisationsmechanismus zwischen dem im *TaskContainer* eingebetteten Modul und der restlichen Steuerung geschaffen.

Die grundlegende Ausführungseinheit in MCA2 ist der Part; er ist zuständig für die Initialisierung der Ausführungsumgebung, die Kommunikation mit anderen Parts über TCP/IP und die Ausführung der Haupt-Ausführungsschleife – der *Control Loop*. Parts stellen die Schnittstellen der Gruppe anderen Parts über so genannte *inter-part*-Kanten zur Verfügung. Die *Control Loop* des Parts wird zyklisch mit einer einstellbaren Frequenz ausgeführt. Um Multithreading zu erreichen, können MCA-Anwendungen mit Hilfe von *Thread-Containern* in mehrere Threads aufgeteilt werden. Jeder *Thread-Container* ist für die Ausführung eines einzigen Moduls bzw. einer Gruppe zuständig. Für Daten, die sich nicht sinnvoll über die auf *double*-Werte beschränkte Modulschnittstelle austauschen lassen, existieren *Blackboards*. Über diese können größere Datenmengen wie zum Beispiel Kamerabilder übertragen werden. Der Zugriff auf die *Blackboards* erfolgt netzwerktransparent und wird durch *read/write-Locks* kontrolliert.



(a) Darstellung der ausgeführten MCA-Gruppen im *mcabrowser*



(b) Eingabefelder zur Anpassung der Parameter des UCoM-Programms *armar3.fl.S*

Abb. 4.4.: Screenshots des Programms *mcabrowser*

Werkzeuge

Für das Testen und für die Fehlersuche bietet MCA2 zwei Werkzeuge – *mcabrowser* und *mcagui*. Diese dienen dazu, die korrekte Verbindung der Module zu kontrollieren und zur Laufzeit Zugriff auf die Werte aller Schnittstellen zu erhalten. Mit *mcagui* existiert eine grafische Benutzerschnittstelle zur Robotersteuerung. Die beiden Tools verbinden sich über TCP-Sockets mit den aktuell ausgeführten Steuerungsprogrammen und stellen somit deren Informationen zur Verfügung.

Mit Hilfe von *mcabrowser* erhält der Nutzer zur Laufzeit Zugriff auf alle relevanten Informationen der realisierten Architektur. Mit diesem Tool lässt sich die Verbindungsstruktur grafisch analysieren und auf Fehler untersuchen. Hierbei wird einerseits die hierarchische Struktur der Modulanordnung dargestellt und andererseits der Zugriff auf sämtliche Schnittstellen der Module ermöglicht (Abb. 4.4(a)). Somit lassen sich zur Laufzeit die Werte der Kanten anzeigen und verändern und die Parameter zur Modifikation der Modulausführung anpassen (Abb. 4.4(b)). Mit *mcabrowser* lassen sich Module starten und stoppen, Debugnachrichten ein- und ausschalten und somit alle Aspekte des laufenden MCA-Systems kontrollieren.

Die grafische Benutzerschnittstelle von MCA2 wird durch das Programm *mcagui* realisiert. Auf oberster Hierarchieebene lässt sich mit den bereitgestellten Standardwidgets die Benut-

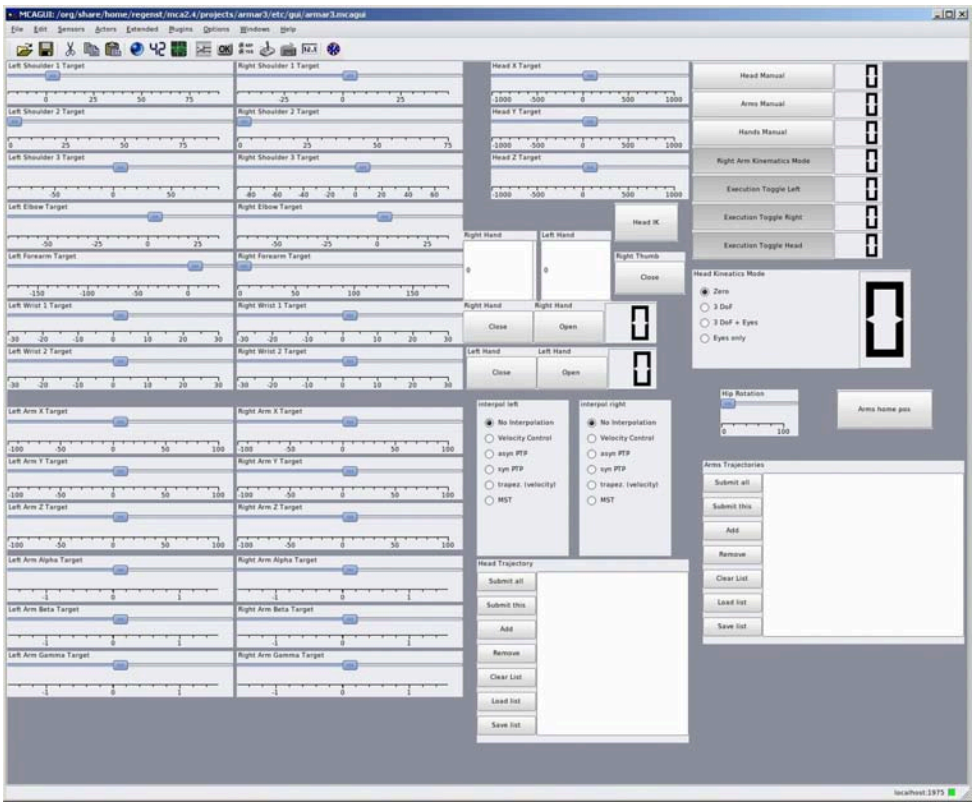


Abb. 4.5.: Beispiel für mcagui, wie es für die Ansteuerung des humanoiden Roboters *ARMAR-III* verwendet wird

zerschnittstelle zusammenstellen (Abb. 4.5). Standardmäßig sind Widgets wie Buttons, Zähler, Oszilloskop-Anzeigen und Eingabefelder implementiert. Über weitere Plug-ins lassen sich komplexere Darstellungen realisieren. So existiert beispielsweise ein Plug-in, das eine dreidimensionale Roboterdarstellung ermöglicht.

4.2.2. Formale Darstellung der MCA-Struktur in UML

Um die Verknüpfung der MCA-Module zu einem Gesamtsystem und die inneren Zusammenhänge der Softwarearchitektur auf eine standardisierte Weise formal darzustellen, soll anstelle der in Abschnitt 2.1.5 verwendeten Darstellung die „Unified Modeling Language“ (UML) verwendet werden. UML wurde im Oktober 1994 von Grady Booch und Jim Rumbaugh bei der Rational Software Corporation entwickelt. Damit sollten ihre erfolgreichen Methoden zu einem einheitlichen Industriestandard weiterentwickelt werden. Seit Herbst 1995 ist auch Ivar Jacobson an der Entwicklung beteiligt und integrierte seine OOSE-Methode in die UML. Der UML-Standard UML 1.0 wurde im Januar 1997 verabschiedet [16]. Seitdem wurde der UML-Standard ständig weiterentwickelt und um weitere Diagrammtypen erweitert. Aktuell ist die UML 2.1.2[2] spezifiziert.

[2] `http://www.omg.org/docs/formal/07-11-04.pdf`

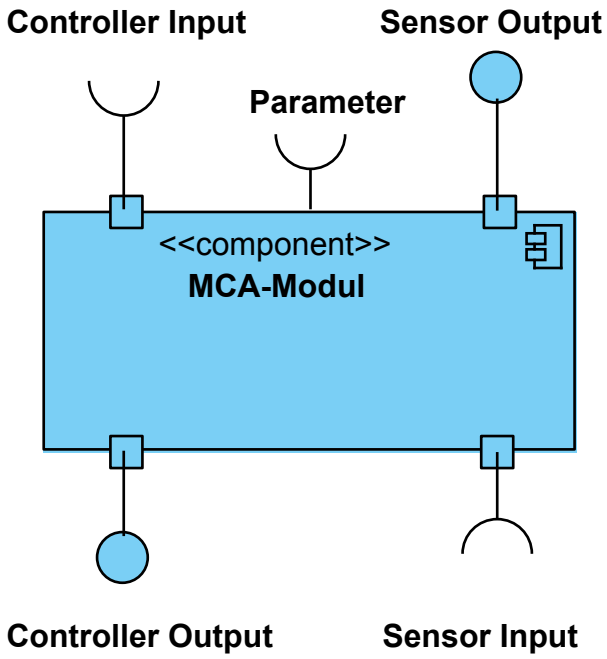


Abb. 4.6.: Modellierung eines MCA-Moduls als Komponente in einem UML-Komponentendiagramm

Mit UML soll die Struktur der MCA-Module modelliert werden. Aus UML-Sicht handelt es sich dabei um den *design view*. Für diesen Zweck, also für die Modellierung der MCA-Struktur, ist das Komponentendiagramm am besten geeignet. Im Komponentendiagramm werden auf abstrakte Weise die Interaktionen zwischen den einzelnen Komponenten dargestellt. Die hauptsächlich verwendeten Konzepte, Bausteine und Verbindungen im Komponentendiagramm sind:

Komponente (*component*) Ein modularer Teil des Gesamtsystems. Die Komponente verbirgt ihre konkrete Implementierung und stellt nach außen hin wohldefinierte Schnittstellen zur Verfügung.

Angebotene Schnittstellen (*provided interface*) Schnittstellen, an denen die Komponente anderen Teilnehmern Dienste zur Verfügung stellt.

Erforderliche Schnittstelle (*required interface*) Schnittstelle, an der die Komponente Zugriff auf einen Dienst erwartet.

Abhängigkeit (*dependency*) Stellt eine Beziehung zwischen zwei Elementen her. Dabei liefert das eine Element Inhalte, die das andere Element benötigt. Die häufigste Anwendung dieser Beziehung besteht zwischen bereitgestellter und benötigter Schnittstelle zweier Komponenten.

Port (*port*) Ein Port ist ein strukturelles Merkmal eines Classifiers und stellt einen Interaktionspunkt auf der Hülle des Classifiers dar.

Delegation (*delegation*) Über diese Beziehung wird eine Verbindung zwischen den Ports auf dem Rand der Komponenten und weiteren Elementen im inneren der Komponente hergestellt.

Mit diesen Bausteinen und Verbindungen lässt sich ein MCA-Modul als Komponente im UML-Komponentendiagramm modellieren. Das MCA-Modul selbst stellt eine Komponente dar. Die fünf Schnittstellen, die ein MCA-Modul bietet, werden entsprechend der Datenflussrichtung als angebotene bzw. als erforderliche Schnittstelle modelliert. In UML erscheint ein MCA-Modul dann wie in Abb. 4.6 gezeigt. An den Schnittstellen *Controller Input*, *Sensor Input* und *Parameter* erwartet das MCA-Modul Eingaben von außen, folglich werden diese als erforderliche Schnittstellen mit dem Halbkreissymbol modelliert. An den Schnittstellen *Controller Output* und *Sensor Output* bietet das MCA-Modul Informationen nach außen hin, weshalb diese mit dem Vollkreis-Symbol als angebotene Schnittstellen dargestellt werden. Die Verbindung zwischen zwei MCA-Modulen erfolgt über die Abhängigkeits-Beziehung. Benötigt der *Controller Input* eines Moduls die Eingaben eines anderen Moduls, so wird dies durch den gestrichelten Abhängigkeitspfeil von diesem *Controller Input* zum *Controller Output* des bereitstellenden MCA-Moduls symbolisiert. Diese Verbindung ist beispielhaft in Abb. 4.8(b) für die dort dargestellten Komponenten Modul 1 und Modul 2 aufgezeigt. In dieser Abbildung hängt CI1 von CO2 ab und SI2 von SO1.

Auch die Zusammenfassung zu einer MCA-Gruppe lässt sich auf die Darstellung in UML übertragen. Dazu werden mehrere Komponenten in eine umschließende Komponente eingebettet. An der Hülle dieser Komponente werden dann die nach außen hin relevanten Verbindungen über einen Port dargestellt. Im Falle einer am Port angebotenen Schnittstelle wird über die Delegations-Beziehung an eine angebotene Schnittstelle der enthaltenen Komponenten verwiesen. Bei erforderlichen Schnittstellen zeigt die Delegationsbeziehung von einer erforderlichen Schnittstelle zum Port der einschließenden Komponente. In Abb. 4.7 ist die bereits in Abb. 2.6 in der bisherigen Notation dargestellte MCA-Gruppe in UML modelliert. Zur übersichtlicheren Darstellung ist in den Gruppen-Diagrammen die Parameterschnittstelle ausgeblendet, da der Zugriff auf die Parameterschnittstelle ohnehin ausschließlich mit dem *mcabrowser* erfolgt. In dieser Abbildung lässt sich beispielsweise an der Abhängigkeitsbeziehung der Module erkennen zwischen welchen Schnittstellen Daten ausgetauscht werden. Durch die Abhängigkeitsbeziehung von SI5 des MCA-Modul5 zu SO2 von MCA-Modul2 und SO3 von MCA-Modul3 wird dargestellt, dass MCA-Modul5 an seinem *Sensor Input* die Ausgaben der *Sensor Outputs* der Komponenten MCA-Modul2 und MCA-Modul3 benötigt. Anhand der Delegations-Beziehungen, die an den Ports der MCA-Gruppe beginnen bzw. enden lässt sich erkennen welche Daten über diesen Port ausgetauscht werden. Beispielsweise fließen über den *Controller Output* der Gruppe die Daten zu den *Controller Outputs* der Komponenten MCA-Modul2 und MCA-Modul3. Mittels dieser Notation lassen sich die in MCA2 üblichen Schachtelungen gut

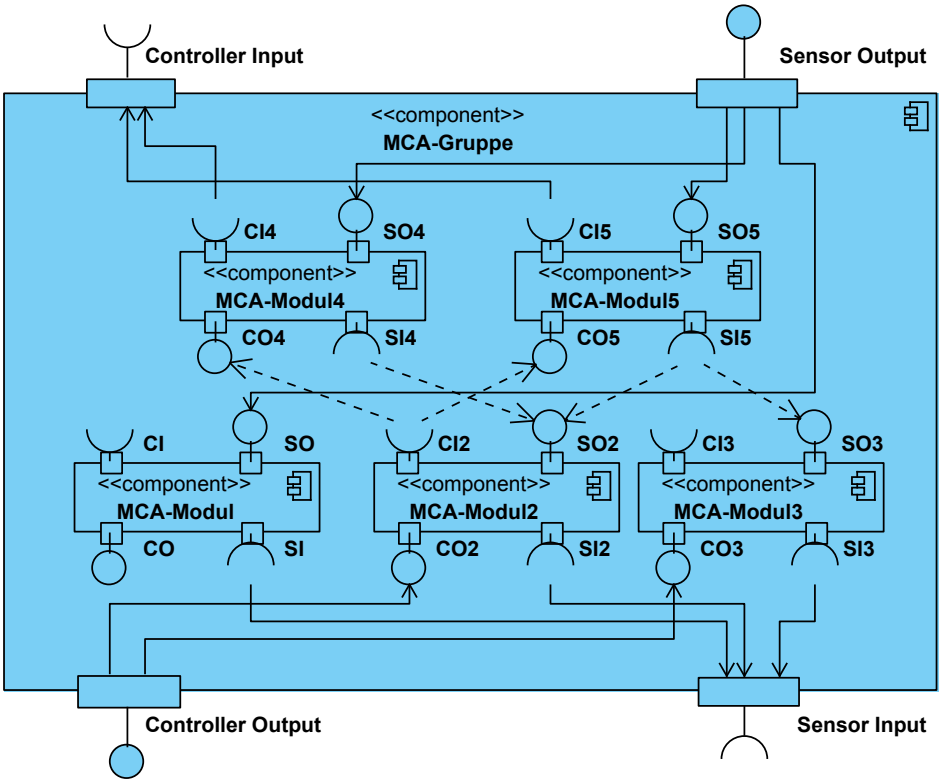
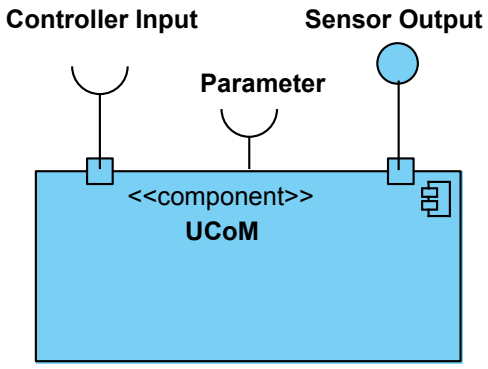


Abb. 4.7.: Modellierung einer MCA-Gruppe als Verbindung mehrerer Komponenten in einem UML-Komponentendiagramm

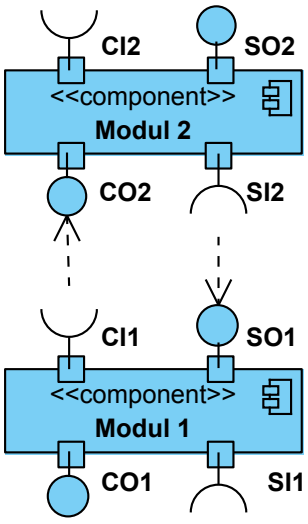
umsetzen und in einer formalen Weise darstellen. Die Abbildung der in Abb. 4.3 dargestellten funktional-logischen Steuerung auf die hier vorgestellte Notation wird in Abschnitt 5.2.2 am konkreten Beispiel des humanoiden Roboters *ARMAR-III* vorgenommen.

4.3. Hardwarearchitektur

Aufgabe der Hardwarearchitektur ist es, die durch die Steuerungsarchitektur und Softwarearchitektur beschriebenen Strukturen auf tatsächliche Hardwarekomponenten und Verbindungseinrichtungen abzubilden. Dabei sind einerseits strukturelle Aspekte zu berücksichtigen, andererseits müssen die in Kapitel 3 aufgestellten Leistungskenndaten in den Bereichen Zykluszeit, Rechenleistung, Speicherbedarf und Verbindungsbandbreite erfüllt werden. Die Hardwarearchitektur beschreibt die Zuordnung der im Rahmen der Steuerungsarchitektur und Softwarearchitektur definierten Prozesse und Module auf Rechnerkomponenten und spiegelt im Idealfall die Modularisierung der Softwarearchitektur wider. Neben den Verbindungen der Rechnerkomponenten untereinander müssen Schnittstellen zur Aktorik und Sensorik bereitgestellt werden. Hier besteht die Herausforderung darin, die Vielzahl unterschiedlicher Sensor- und Aktortypen und deren entsprechende Kommunikationsmechanismen in das Rechnersystem einzubinden. Die topologische Verteilung der Komponenten wird maßgeblich durch die als menschenähnlich



(a) Modellierung einer dezidierten Echtzeitkomponente (UCoM) in einem UML-Komponentendiagramm. Die Komponente hat direkten Zugriff auf die Hardware und bietet somit die Schnittstellen *Controller Output* und *Sensor Input* nicht an. Diese Komponente stellt sich in der Struktur wie ein MCA-Modul dar.



(b) Exemplarische Darstellung der Verbindung zwischen zwei Modulen. Die angebotene Schnittstelle CO2 ist über eine Abhängigkeitsbeziehung mit CI1 verbunden, SI2 benötigt die Schnittstelle SO1.

Abb. 4.8.: Dezidierte Echtzeitkomponente (UCoM) als MCA-Modul und formale Darstellung der Verbindung zwischen zwei Modulen

vorgegebene äußere Gestalt des humanoiden Roboters bestimmt (Abb. 4.9).

Bei der Umsetzung der Steuerungsarchitektur und Softwarearchitektur auf die Hardwarearchitektur soll als Vorgabe gelten, möglichst alle Rechnerkomponenten und Verbindungseinrichtungen mit Standardkomponenten zu realisieren. Durch diese Vorgehensweise kann erheblicher Entwicklungsaufwand eingespart werden. Bei der Verwendung von Standardkomponenten mit standardisierten Schnittstellen ist ein Profitieren an der schnell fortschreitenden Entwicklung der Leistungsfähigkeit von Rechnerkomponenten mit geringem Aufwand möglich. Legt man das Mooresche Gesetz zu Grunde steigt die Leistungsfähigkeit von Rechnerkomponenten alle zwei Jahre auf das Doppelte [110]. Somit lassen sich die verwendeten Rechnerkomponenten durch leistungsfähigere ersetzen, ohne dass eine Neuentwicklung notwendig wird.

4.3.1. Rechnersystem

Das wesentliche Unterscheidungsmerkmal für die Rechnerkomponenten bzw. Verbindungseinrichtungen ist die Frage, ob sie einen echtzeitfähigen Betrieb ermöglichen. Betrachtet man die Anforderungen in den vier Domänen Bildverarbeitung, Akustik, Planung und Regelung, fällt auf, dass Echtzeitfähigkeit besonders im Bereich der Regelung wichtig ist (Tab. 3.4). Die Hardwarearchitektur muss also einerseits Komponenten zur Verfügung stellen, die eine echtzeitfähi-

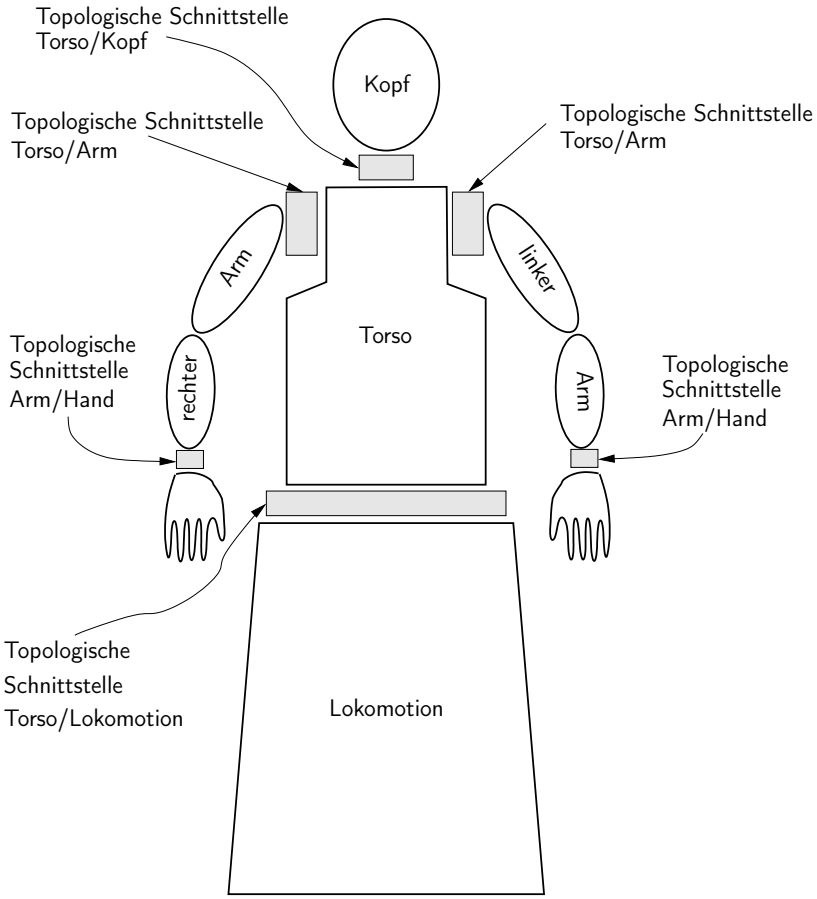


Abb. 4.9.: Topologiesicht auf das Gesamtsystem humanoider Roboter. Das Robotersystem lässt sich topologisch in klar abgegrenzte Teilsysteme zerlegen, die über Schnittstellen verbunden werden.

ge Ausführung von Programmen sicherstellen, für den Fall verteilter Komponenten muss auch die Kommunikation unter diesen Echtzeitkomponenten in Echtzeit erfolgen. Auf der anderen Seite müssen Komponenten mit hoher Leistungsfähigkeit vorgesehen werden, die für den Fall eines verteilten Aufbaus auf eine breitbandige Verbindungsinfrastruktur zugreifen können müssen (Abb. 4.10). Prozesse, die auf Nichtechtzeitkomponenten laufen und mit den Echtzeitkomponenten interagieren, müssen auch hohe Anforderungen an die Vorhersagbarkeit ihrer Ausführung erfüllen. Auch die Verbindung dieser Komponenten mit den Echtzeitkomponenten muss über eine Verbindung mit kalkulierbarer Latenz erfolgen. Sowohl die hohen Anforderungen an die Gesamt-Leistungsfähigkeit der Komponenten als auch die topologischen Randbedingungen im humanoiden Roboter legen eine Realisierung der Hardwarearchitektur mit verteilten Komponenten nahe. Gegenüber einer monolithischen Lösung bietet eine verteilte Realisierung folgende Vorteile:

Performanz Wie sich in der Anforderungsanalyse abzeichnet, sind die Anforderungen an die Rechenleistung des Gesamtsystems so hoch, dass sich diese mit herkömmlichen alleinstehenden Recheneinheiten nicht realisieren lassen. Weiterhin lassen sich Prozesse klarer

von einander trennen und somit kann verhindert werden, dass sehr ressourcenhungrige Prozesse die Ausführung anderer Prozesse mit strikten Zeitanforderungen blockieren.

Skalierbarkeit Durch das Verteilen auf mehrere Recheneinheiten wird eine Infrastruktur geschaffen, das System um weitere Einheiten zu erweitern und auf diese Weise die Leistungsfähigkeit des Gesamtsystems zu steigern.

Ausfallsicherheit Durch das Verteilen der Steuerung auf mehrere Einheiten, führt der Ausfall einer Komponente noch nicht zum Versagen des Gesamtsystems. Es kann durch die Verteilung Redundanz geschaffen werden und somit die Sicherheit des Systems gesteigert werden.

Modularität Eine Verteilung auf mehrere Recheneinheiten unterstützt eine konsequentere Modularisierung des Robotersystems. So lässt sich der Roboter in verschiedene Teilsysteme unterteilen, beispielsweise in topologisch motivierte Teilsysteme oder funktionale Gruppen. Diese Unterteilung kann auch in der Hardwarearchitektur abgebildet werden.

Datenlokalität Durch die Verteilung auf mehrere Recheneinheiten lassen sich durch geeignete Platzierung der Recheneinheiten kurze Verbindungswege zwischen den Datenquellen, Datenverarbeitungseinrichtungen und den Datensenken realisieren. Auf diese Weise müssen das Datenaufkommen der vielfältigen Sensorinformationen und die Aktorstellsignale nur über kurze Strecken übertragen werden und die Verarbeitung kann lokal erfolgen. Dadurch werden einerseits für die Echtzeitfähigkeit störende Latenzen minimiert und das Risiko von Übertragungsfehlern vermindert.

Die Struktur der Hardwarearchitektur soll wie in Abb. 4.10 dargestellt als verteilte Architektur aus Echtzeitkomponenten und Nichtechtzeitkomponenten realisiert werden. Die Echtzeitkomponenten sollen untereinander über ein echtzeitfähiges Bussystem verbunden sein. Die Nichtechtzeitkomponenten sind untereinander über ein Breitband-Bussystem verbunden [131]. Für die Verbindung zwischen Echtzeit- und Nichtechtzeitkomponenten wird das gleiche Bussystem vorgesehen, wie für die Verbindung der Echtzeitkomponenten untereinander.

An dieser Stelle soll beschrieben werden, mit welcher Art von konkreten Komponenten sich die Anforderungen in den einzelnen Domänen, wie sie in Kapitel 3 diskutiert wurde (Tab. 3.4), erfüllen lassen. Die Anforderungsstruktur der Domänen Bildverarbeitung, Akustik und Planung entsprechen dem Leistungsspektrum aktueller PC-Komponenten bzw. Rechnerverbünden, die sich aus PC-Komponenten mit Standardnetzwerkverbindungen aufbauen lassen. In anderen Worten lässt sich mit Standard-PC-Komponenten und beispielsweise Gigabit-Ethernet ein Rechnerverbund aufbauen, der den Leistungsanforderungen aus diesen drei Domänen gerecht wird. Neben den reinen Leistungsdaten wurden jedoch für den Einsatz im humanoiden Roboter noch weitere Anforderungen formuliert, die auf den Formfaktor der Komponenten abzielen und

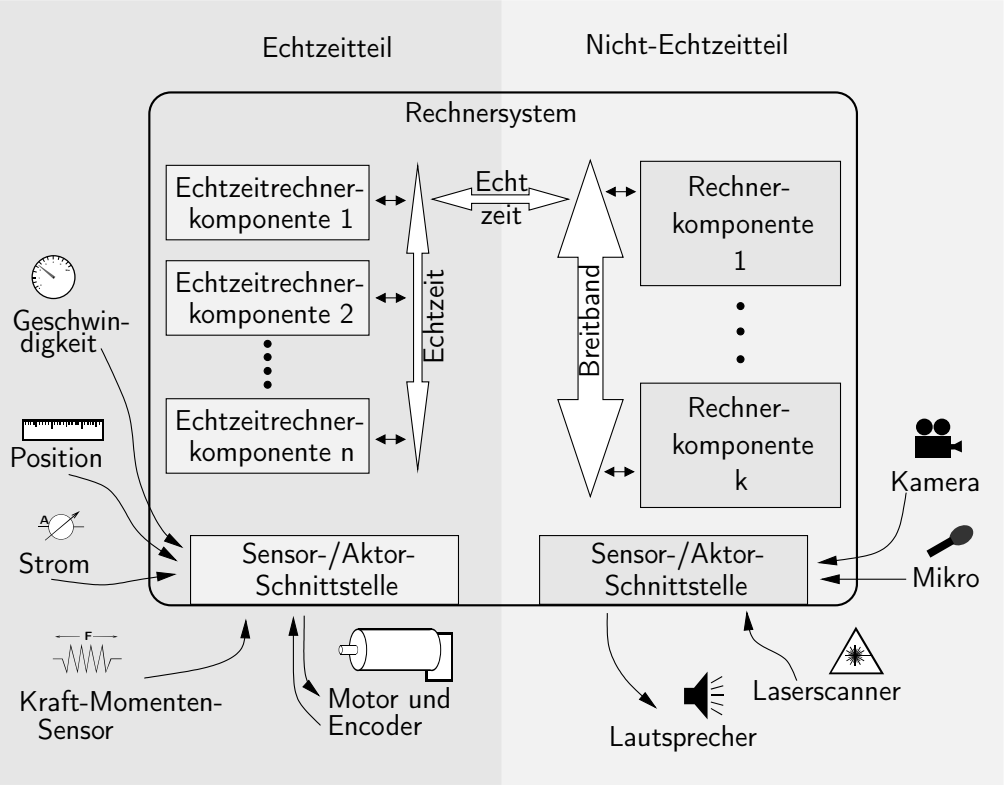


Abb. 4.10.: Komponentensicht auf das Gesamtsystem humanoider Roboter. Das Hauptunterscheidungsmerkmal der Recheneinheiten ist die geforderte Echtzeitfähigkeit. Auf der einen Seite werden echtzeitfähige Komponenten benötigt, die auch über einen Echtzeitbus verbunden sind. Auf der anderen Seite gibt es sehr leistungsfähige, aber nicht echtzeitfähige Recheneinheiten, die über einen Breitbandbus verbunden sind. Die Einbindung der Sensoren und Aktoren erfolgt über eine einheitliche Schnittstelle.

für den autonomen Einsatz einen geringen Energieverbrauch fordern. In den folgenden Absätzen wird ein kurzer Überblick über konkrete Rechnerkomponenten gegeben, die aufgrund ihrer weiteren Spezifikationen für den Einsatz im humanoiden Roboter in Frage kommen.

PC/104

PC/104 ist ein Standard für PC-kompatible Module, die durch das Übereinanderstecken verschiedener Modul-Lagen ein Rechensystem bilden (Abb. 4.11(a)). Der PC/104-Standard[3] wurde durch das PC/104-Konsortium 1992 veröffentlicht. Der Standard schreibt die Modulgröße, die Lage von Erweiterungssteckern und die Pin-Belegung der Steckverbinder zwischen den Modulen vor. Die Modulgröße ist auf 90,17 mm $\times$ 95,89 mm spezifiziert (Abb. 4.12). Übliche Bauhöhen liegen bei etwa 140 mm. Die Verbindungsstecker entsprechen in der ursprünglichen Spezifikation dem ISA-Bus. Folgende Definitionen beinhalten noch zusätzlich den PCI-Bus auf dem Verbindungsstecker (PC/104-plus) bzw. nur noch den PCI-Bus (PCI/104). Da der Bus-

[3] `http://www.pc104.org/pc104_specs.php`

(a) PC/104-System (b) Box-PC (c) Passive Busplatine

Abb. 4.11.: Unterschiedliche Realisierungsformen von Industrie-PCs

Verbindungsstecker über alle Module durchgeleitet wird, kann ohne eine passive Busplatine durch Übereinanderstecken der Module ein Rechensystem zusammengestellt werden. Üblicherweise besteht der Minimalaufbau aus einem Spannungsversorgungsmodul und einem Prozessormodul, welches bereits Standardschnittstellen für VGA, Tastatur, Maus und Netzwerk beinhaltet. Weitere Schnittstellen oder Funktionalitäten können durch weitere Module realisiert werden. So existieren beispielsweise Module für FireWire-Schnittstellen, CAN-Bus und RS422.

Vorteilhaft am PC/104-Standard ist die kleine, kompakte Bauform. Weiterhin lassen sich die PC/104-Systeme durch vielfältige Module flexibel erweitern. Durch den modularen Aufbau des Systems kann beispielsweise, unter Beibehaltung der übrigen Module, nur das Prozessormodul erneuert werden, wenn dort leistungsfähigere Prozessoren verfügbar werden. PC/104-Systeme sind durch ihren geringen Energieverbrauch gut für den autonomen Einsatz im humanoiden Roboter geeignet. Prozessorlagen mit Pentium M-Prozessor und Taktfrequenzen von 1,8 GHz haben eine Leistungsaufnahme von circa 30 W. Ein Nachteil der PC/104-Systeme ist ihr sozusagen offener Aufbau, so sind die Platinen der einzelnen Module direkt zugänglich und auch die Schnittstellen sind oft als Pfostenleisten realisiert. Gerade dieser Umstand macht das System beim Einsatz im humanoiden Roboter anfällig für Kurzschlüsse oder mechanische Beschädigung, besonders bei Wartungsarbeiten am Roboter. Die mechanische Stabilität der Modulverbindungen wird durch Verschraubung mit Abstandshaltern hinreichend gewährleistet, allerdings besteht an den Schnittstellen mangels einer Arretierung der Steckverbindung die Gefahr, dass sich diese durch Vibrationen im Roboter lösen.

Box-PCs

Eine sehr kompakte und robuste Form des Industrie-PCs ist der so genannte Box-PC (Abb. 4.11(b)). Die Abmessungen sind hier nicht klar vorgeschrieben und auch der interne Aufbau der Systeme variiert stark von Hersteller zu Hersteller. Handelsübliche Varianten haben eine Größe von circa 200 mm × 110 mm × 250 mm, so dass bei dieser Bauform noch Standard-PCI oder ISA-Karten verbaut werden können. Noch kompaktere Bauformen werden realisiert indem

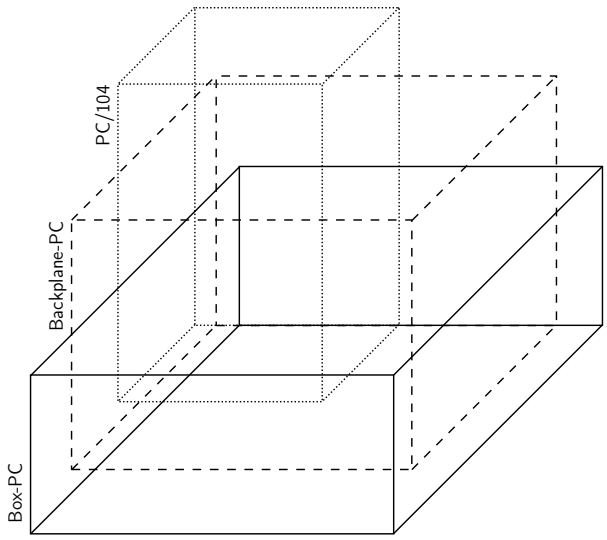


Abb. 4.12.: Vergleich der unterschiedlichen Formfaktoren von Recheneinheiten. Die Außenmaße der jeweiligen Rechnertypen sind als Quader dargestellt. Die Abmessungen des Backplane-PCs sind mit gestrichelten, die des PC/104 mit gepunkteten und die des Box-PCs mit durchgezogenen Linien eingezeichnet.

anstatt durch PCI- oder ISA-Steckplätze Erweiterungsmöglichkeiten durch PC/104-Steckplätze auf der Hauptplatine vorgesehen werden. So lassen sich Abmessungen von 160 mm $\times$ 70 mm $\times$ 260 mm erreichen (Abb. 4.12). Das Leistungsspektrum der Box-PC reicht an das normaler PC-Systeme heran.

Vorteile der Box-PCs sind ihr sehr robuster Aufbau und ein sehr kompaktes Gehäuse. Oft zeichnen sich diese Systeme durch hohe Belastbarkeit durch Vibrationen und Stöße und einen breiten Temperatureinsatzbereich aus. Durch Nutzung des Gehäuses als große Kühlfläche, können diese Systeme teilweise auch passiv gekühlt betrieben werden, was eine höhere Ausfallsicherheit durch Verzicht auf Lüfter als mechanisches Verschleißteil mit sich bringt. Ihr Energieverbrauch liegt bei Nutzung eines Pentium-M-Prozessors mit einer Taktfrequenz von 1,6 GHz unter 30 W. Ein großer Vorteil der Box-PCs ist ihr geschlossener Aufbau in einem robusten Gehäuse, so dass Beschädigung der Rechnerhardware nahezu auszuschließen ist. Sowohl der interne Aufbau des Box-PC als auch die Absicherung der externen Schnittstellen durch Schraubverbindungen machen das System robust gegen Vibrationen. Nachteilig für das Systemupgrade wirkt sich der monolithische Aufbau des Systems aus. Für die Variante, die sich durch PCI- bzw. ISA-Karten erweitern lässt, ist die Bauform für den Einsatz im humanoiden Roboter durch ihre Größe ungünstig.

Backplane-Systeme

Eine weitere Variante von Industrie-PCs sind Systeme, die aus einer Steckkarten-CPU auf einer passiven Busplatine bestehen (Abb. 4.11(c)). Die Abmessungen der passiven Busplatine

sind nicht so klar spezifiziert wie beispielsweise das PC/104-System. Vielmehr gibt es entsprechend der Schnittstellenanforderungen eine große Auswahl an unterschiedlichen Abmessungen. Auf der passiven Busplatine gibt es einen ausgezeichneten Steckplatz für die Steckkarten-CPU und eine flexible Anzahl von ISA- bzw. PCI-Steckplätzen. Bei diesen Rechnersystemen befindet sich die CPU auf einer Steckkarte. Standardmäßig befinden sich auf dieser Steckkarte Anschlüsse, wie sie sonst auf Hauptplatinen zu finden sind, also mindestens ein Prozessorsockel, ein Steckplatz für Arbeitsspeicher, Anschlüsse für Festplatten und meistens ein VGA-Chip und Netzwerk-Controller. Aufgrund der durch die Steckkarten-CPU abgedeckten Funktionen sollten für übliche Anwendungen im humanoiden Roboter neben dem Steckplatz für die Steckkarten-CPU vier weitere Steckplätze ausreichen. Passive Busplatinen mit dieser Anzahl an Steckplätzen haben Abmessungen von circa 150 mm $\times$ 180 mm. Die Höhe wird in diesem Fall durch die Höhe der Steckkarten-CPU bzw. durch die Höhe der verwendeten ISA- oder PCI-Karten bestimmt. Die maximale Höhe von PCI-Karten beträgt 107 mm, die maximale Höhe von ISA-Karten 122 mm, wobei letztere üblicherweise auch unterhalb der maximalen Höhe von PCI-Karten bleiben (Abb. 4.12). Die übliche Leistungsaufnahme einer Steckkarten-CPU ausgestattet mit einem Pentium-M-Prozessor mit 1,6 GHz Taktfrequenz liegt bei etwa 29 W.

Als Vorteile für die Realisierung mittels Steckkarten-CPU auf passiver Busplatine lassen sich der geringe Energieverbrauch, die variabel mit der Schnittstellenanforderung anpassbare kompakte Bauform und die flexible Erweiterbarkeit bzw. Upgrademöglichkeit durch Wechsel der Steckkarten-CPU aufführen. Nachteilig ist, wie beim PC/104-System, der offene Aufbau des Systems. Aufgrund der fehlenden Standardisierung der Busplatinengrößen und der vielen unterschiedlichen Größen in Abhängigkeit der Schnittstellenzahl, gibt es nicht für jede Platine maßgenaue Gehäuse. Somit bestünde zur Kapselung des Systems die Notwendigkeit ein Gehäuse zu entwerfen. Bei offenem Aufbau bestehen die gleichen Gefahren hinsichtlich mechanischer Beschädigung und elektrischer Kurzschlüsse wie bei PC/104-Systemen. Vibrationen wirken sich bei diesem System negativ auf den Kontakt der Steckkarten in der Busplatine aus.

In Tab. 4.1 sind die Vor- und Nachteile der einzelnen Formfaktoren gegenübergestellt. Grundsätzlich kommen alle drei Formfaktoren für den Einsatz im humanoiden Roboter in Frage. Welcher Formfaktor zum Einsatz kommt, hängt noch vom konkreten Einbauort im Roboter ab. In räumlich sehr beengtem Bauraum wird trotz der Nachteile bezüglich der Robustheit dem PC/104-System der Vorzug gegeben. Ist ausreichend Bauraum für den robusteren Box-PC vorhanden sollte dieser verwendet werden. Wenn nicht eine große Anzahl an Schnittstellenkarten wie PCI oder ISA benötigt wird, sollte dem Box-PC gegenüber dem Backplane-System der Vorzug gegeben werden.

	Leistungsfähigkeit	Größe	Robustheit	Erweiterbarkeit
PC/104	+	++	−	+
Box-PC	++	+	++	○
Backplane-PC	++	+	−	+

Tab. 4.1.: Gegenüberstellung der Vor- und Nachteile der unterschiedlichen Realisierungsvarianten von Industrie-PCs

4.3.2. Regelungskomponente UCoM

Für die Sensor-Motor-Regelung des Roboters – also die Auswertung der verteilten Sensoren und die Ansteuerung der verteilten Aktoren – unter Echtzeitbedingungen, sind herkömmliche PCs und auch die beschriebenen Industrie-PCs nicht geeignet. Handelsübliche Motorregler sind aus mehreren Gründen nicht für den Einsatz im humanoiden Roboter geeignet. Die verfügbaren Motorregler sind von ihrem Formfaktor her für die industrielle Anwendung ausgelegt und von ihren Abmessungen her deutlich zu groß für den Einsatz im humanoiden Roboter. Weiterhin sind die Einstellmöglichkeiten für die Reglercharakteristik meist sehr gering. Bei vielen dieser Motorregler erfolgt die Rechneranbindung über serielle Schnittstelle, so dass sich keine Busstruktur aufbauen lässt, sondern jeder Regler sternförmig an den zugehörigen Rechner angeschlossen werden muss. Dies führt zu einem erhöhten Verkabelungsaufwand, der im humanoiden Roboter nach Möglichkeit zu vermeiden ist. Die Nachteile der verfügbaren Motorregler lassen sich in den folgenden Punkte zusammenfassen:

- Bauraum: Übliche Servoregler für einen Motor sind in für industrielle Anwendungen robust und großzügig dimensionierten Gehäusen verbaut (in der Größenordnung von $100\,\text{mm} \times 100\,\text{mm} \times 30\,\text{mm}$).
- Geringe Flexibilität: Mangelnde Konfigurationsmöglichkeiten des Reglers. Oft nur ein bestimmter Reglertyp implementiert, für den sich die Reglerparameter einstellen lassen.
- Rechneranbindung: Oft nur Punkt zu Punkt Verbindungen über serielle Schnittstelle

Aufgrund der nicht vorhandenen Verfügbarkeit von kommerziell erhältlichen Standardkomponenten, ist für die Regelungskomponenten im humanoiden Roboter eine Spezialentwicklung notwendig. Bei der Entwicklung dieser Komponente liegt der Schwerpunkt auf der hardwaretechnischen Eignung für den humanoiden Roboter. Das heißt, die Komponente muss sich für den verteilten, aktornahen Einsatz eignen, kompakte Abmessungen besitzen und die Leistungsanforderungen der Aktoren erfüllen. Weiterhin sollen auf dieser Komponente flexibel unterschiedliche Reglertypen implementiert und über ein Bussystem von den Recheneinheiten aus parametrisiert werden können. Über diese hardwaretechnischen Anforderungen hinaus, wird

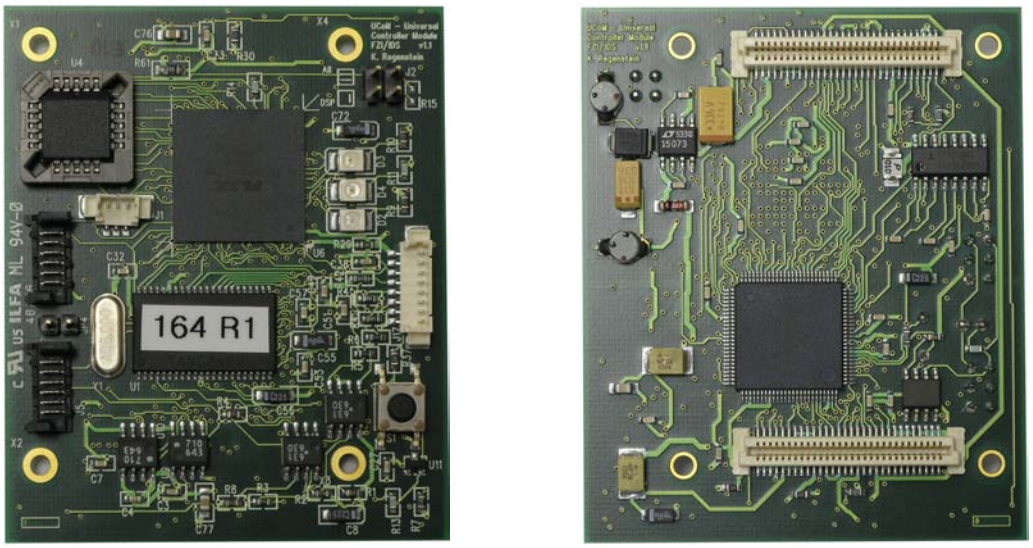


Abb. 4.13.: Das „Universal Controller Module“ (UCoM) (Sicht auf Ober- und Unterseite)

die Komponente so ausgelegt, dass sie sich nahtlos in die vorgeschlagene Steuerungs- und Softwarearchitektur integriert. Aus der Sicht der Software soll sich diese Komponente im Sinne der Modularisierung wie ein MCA-Modul präsentieren (vergleiche Abb. 4.8(a)). Die Regelungskomponente muss eine echtzeitfähige Regelung der Aktoren mit kurzen Zykluszeiten in der Größenordnung von 1 ms ermöglichen und eine echtzeitfähige Busverbindung zu den Industrie-PCs anbieten. Um im inneren Regelkreis für die Aktoransteuerung Latenzen durch Datenübertragung über Bussysteme zu vermeiden, soll zumindest die für diesen Regelkreis direkt relevanten Sensorinformationen (Abschnitt 4.3.3) von der Regelungskomponente evaluiert werden.

Die Modularisierung der zu entwickelnden Komponente soll sich nicht nur auf die Einbindung als Modul in die Softwarearchitektur beziehen. Auch in Bezug auf die Realisierung der Echtzeitkomponente soll sich der modulare Aufbau auch in der Hardware widerspiegeln [129]. Aus diesem Grund wird eine Aufteilung der Echtzeitkomponente in zwei Teilkomponenten gewählt. Grundsätzlich sind von der Echtzeitkomponente zwei Hauptaufgaben zu erfüllen, einerseits die Ausführung der Regleralgorithmen in einer Recheneinheit und andererseits die elektrisch, physikalische Anbindung an die Sensoren und Aktoren. Diese Aufteilung wird auch zur Unterscheidung der Teilkomponenten herangezogen. Es wird also eine Controllerkomponente entwickelt, die für die Ausführung der Regleralgorithmen zuständig ist und universell mit verschiedenen Ansteckboards zur Ansteuerung der Aktorik und Sensorik verknüpft werden kann. Aufgrund dieser Zuordnung einer universellen Controllerkomponenten zu verschiedenen Ansteckboards für verschiedene Aktortypen wird diese Komponente im Folgenden als „Universal Controller Module“ (UCoM) bezeichnet (Abb. 4.13). Auch auf dem UCoM wird die Modularisierung durch die Verwendung eines DSPs und eines FPGAs aufgegriffen. Der DSP ist für

die Ausführung des Regleralgorithmus und die Kommunikation mit den weiteren Recheneinheiten zuständig [130]. Der FPGA dient zur hardwarenahen Sensordatenvorverarbeitung, zur Implementierung weiterer Schnittstellen und zur Ansteuerung von Sensoren und Aktoren. Weiterhin lässt sich die elektrische Verbindung zu den Ansteckboards mit Hilfe des FPGAs flexibel gestalten. Auf dem UCoM wird ein DSP von Freescale verwendet – der *Hybrid Controller* DSP56F803. Das hybrid in der Namensgebung soll auf die nahe Verwandschaft zu einem Mikrocontroller hindeuten, so verfügt der DSP56F803 über eine Vielzahl üblicherweise bei Mikrocontrollern bekannten Schnittstellen wie: PWM-Einheiten, Quadraturzähler, mehrere Timer, serielle Schnittstelle, CAN-Bus, SPI und JTAG. Der mit 80 MHz getaktete DSP ist in der Lage für mehrere Achsen Regleralgorithmen mit Zykluszeiten von unter 1 ms auszuführen. Durch die vorhandene CAN-Busschnittstelle lassen sich die UCoMs echtzeitfähig in die Rechnerumgebung integrieren. Die technischen Spezifikationen des UCoMs sind in Tab. 4.2 zusammengefasst. Beim Entwurf des UCoMs wurde auf eine kompakte Bauform, geringen Stromverbrauch und ein geringes Gewicht geachtet. Da die Leistungsfähigkeit des UCoMs ausreicht, um mehrere Achsen anzusteuern und auch aufgrund der erzielten Abmessungen der Einsatz eines UCoMs je Bewegungsfreiheitsgrad nicht zweckmäßig ist, werden benachbarte Bewegungsfreiheitsgrade durch eine Echtzeitkomponente angesteuert. Dabei wurde eine Abwägung zwischen steigender Größe des Ansteckboards und Sparpotenzial durch die Zusammenfassung durchgeführt. Auch vor dem Hintergrund, dass das komplexeste Gelenk im menschlichen Vorbild das Kugelgelenk ist, und dies mit einer Kombination aus drei Achsen technisch umgesetzt wird, wurde das Ansteckboard so ausgelegt, dass drei Achsen gemeinsam angesteuert werden können.

Durch diese Entscheidung lässt es sich auch jederzeit realisieren, die Echtzeitkomponenten – also die Kombination aus UCoM und Ansteckboard – direkt bei den Aktoren und Sensoren zu platzieren. Durch diese verteilte Anordnung nah beim Aktor lassen sich kurze Verkabelungswege und somit eine geringe Störanfälligkeit realisieren. Da die Regelung direkt auf der Echtzeitkomponente stattfindet und Sensorinformationen nicht im Regelungstakt über den Bus übertragen werden müssen, wird an dieser Stelle kein breitbandiges Bussystem benötigt.

Für den Einsatz im humanoiden Roboter wurde für die Ansteuerung von bis zu drei bürstenbehafteten Elektromotoren das Ansteckboard Dreier-Motorboard entwickelt (Abb. 4.14). Mit Hilfe dieses Boards können drei Elektromotoren bei 24 V Betriebsspannung mit bis zu 5 A Strom versorgt werden. Über das Dreier-Motorboard werden die elektrischen Schnittstellen zu den Aktoren und Sensoren bereitgestellt. Die weiteren Eigenschaften des Dreier-Motorboards sind in Tab. 4.3 zusammengefasst und werden im Kapitel 5 im Rahmen der Beschreibung des Robotersystems *ARMAR-III* näher beschrieben. Weitere technische Details der Realisierung des UCoMs sind im Anhang B beschrieben.

Komponente	Eigenschaften
DSP	80 MHz DSP56F803
FPGA	Altera EPF10k30A
Speicher	64 KB Program Flash, 1 KB Program RAM, 8 KB Data Flash, 4 KB Data RAM, 4 KB Bootflash, bis 128 KB externes RAM
Konfiguration	EPC2-ROM
Schnittstellen	CAN, SCI, JTAG, SPI
Motorcontrol	6 PWM-Kanäle
AD-Wandler	8 Kanäle 12 bit AD-Wandler
Programmierung	Programmierbar über JTAG oder mit DSP-Bootloader über CAN-Bus
Leistungsaufnahme	1 W
Abmessungen	57 mm × 70 mm × 15 mm
Masse	26 g

Tab. 4.2.: Übersicht über die Eigenschaften des UCoM

Komponente	Eigenschaften
Leistung	Drei 24 V Motoren mit bis zu 5 A
Encoder	Eingänge für 6 quadraturcodierte Signale
Strommessung	Differenzielle Strommessung für jeden Motorkanal
Peripherie	2 Analogeingänge, SPI-Schnittstelle, 3 Ein-/Ausgänge
Abmessungen	80 mm × 70 mm × 20 mm
Masse	33 g

Tab. 4.3.: Übersicht über die Eigenschaften des Dreier-Motorboards

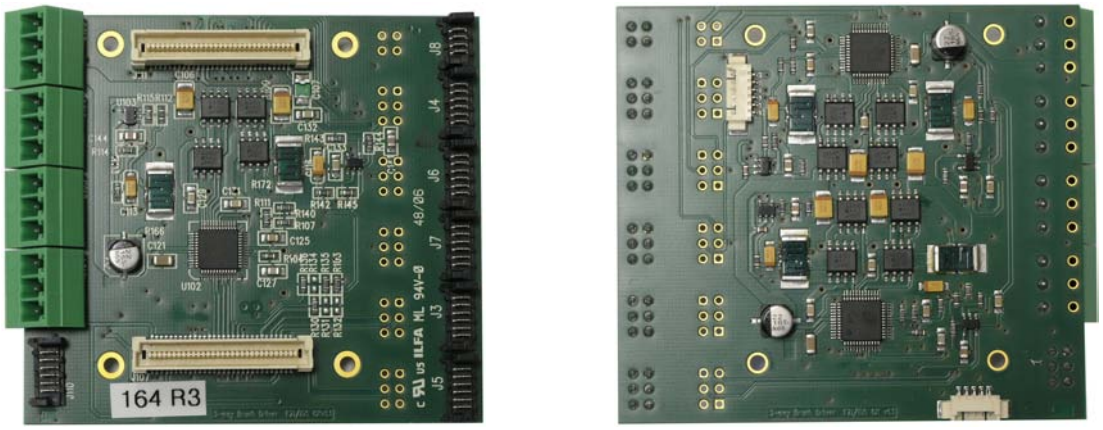


Abb. 4.14.: Das Dreier-Motorboard (Sicht auf Ober- und Unterseite)

4.3.3. Sensoren

Zur Erfassung von Messgrößen, die zur Wahrnehmung der Umwelt und zur Zustandsüberwachung des Roboters dienen, werden Sensoren eingesetzt. In dieser Arbeit wird einerseits zwischen, in Bezug auf die Sensorwerterfassung, digitalen und analogen Sensoren unterschieden. Andererseits wird noch nach dem Übertragungsprinzip zwischen Sensor und Recheneinheit unterschieden. Hier gibt es wiederum die Möglichkeit zur analogen Übertragung und diverse Varianten der digitalen Übertragung mittels unterschiedlicher Protokolle. In den folgenden Absätzen wird dargelegt, auf welche Weise, die in Abschnitt 3.2 erwähnten Messgrößen ermittelt werden können und welche Übertragungswege zu den Recheneinheiten in Frage kommen.

Bildverarbeitung

Kameras Als bildgebende Sensoren zur Aufnahme von Schwarzweiß- oder Farbbildern kommen Kameras mit digitalem Ausgangssignal zum Einsatz. Diese liefern die Bilddaten beispielsweise über FireWire bereits in digitaler Form an die Recheneinheiten.

Bildgebende Abstandssensoren Mittels eines in einer Ebene aufgefächerten Laserstrahls lässt sich in dieser Ebene eine Punktwolke aus Abstandswerten aufnehmen. Diese so genannten Laser-Scanner haben einen Öffnungswinkel von bis zu 270 ° und liefern zusätzlich zur Laufzeitinformation noch einen Reflektanzwert. Laser-Scanner liefern direkt ein digital vorliegendes Signal, welches auch über eine digitale Schnittstelle an die Recheneinheiten ausgegeben wird.

Akustik

Mikrofon Zur akustischen Umwelterfassung werden Mikrofone eingesetzt. Diese wandeln Schall in elektrische Spannung um. Auch hier wird, aufgrund der schwachen Spannungs-

signale, der Analog-Digital-Wandlung direkt am Mikrofon der Vorzug gegeben. Soll das analoge Signal über eine größere Strecke übertragen werden ist eine Verstärkung unerlässlich.

Mikrofon-Arrays Um über die reine Erkennung von Sprache oder Geräuschen hinaus, eine Ortung der Geräuschquelle vornehmen zu können, werden mehrere Mikrofone zu einem Mikrofon-Array verschaltet.

Regelungsbezogene Sensorik

Positionsbestimmung Zur Bestimmung der Gelenkwinkel lassen sich Zähler einsetzen, die beispielsweise einen Strichcode auswerten. Diese Auswertung kann sowohl optisch über Durchlichtschranken oder Reflexlichtschranken als auch über die Auswertung magnetischer Pulse erfolgen. In beiden Fällen kann die Messung als digital betrachtet werden. Die Nutzung des Halleffekts zur Ermittlung der Orientierung eines Magneten in Relation zum Hall-Sensor ergibt ein analoges Ergebnis. Gleiches gilt für ein Potentiometer, dessen Abgriff durch die Gelenkbewegung verschoben wird. Im Falle der Zähler können die Zählerpulse direkt in digitaler Form zu den Recheneinheiten übertragen werden und dort ausgewertet werden. Im Falle der Sensoren, die ein analoges Ausgangssignal haben, ist zu entscheiden, ob die Übertragung bis zu den Recheneinheiten analog erfolgen kann, oder ob eine Analog-Digital-Wandlung direkt am Sensor sinnvoll ist. Diese Entscheidung hängt vom Abstand des Sensors von der auswertenden Recheneinheit und der Anforderung an die Signalgüte ab.

Geschwindigkeitsbestimmung Zur Geschwindigkeitsbestimmung können prinzipiell, die gleichen Verfahren zum Einsatz kommen wie bei der Gelenkwinkelbestimmung, da üblicherweise nicht die Geschwindigkeit direkt, sondern die Gelenkwinkeländerung in einem festen Zeitintervall bzw. die Zeit für eine festgelegte Gelenkwinkeländerung gemessen wird. Da für alle Gelenke zwischen Motor und Gelenk eine Untersetzung durch Getriebe geschaltet ist, ist die Antriebsdrehzahl am Motor um Größenordnungen höher als die Abtriebsdrehzahl am Gelenk. Aus diesem Grund eignet sich zur Geschwindigkeitsbestimmung vor allem die Auswertung der Encoder am Motor. Motorencoder funktionieren mit Lichtschranken oder Magnetpulsen.

Drehmomentbestimmung Zur Messung des Drehmoments kommen Dehnmessstreifen in Frage, die sowohl antriebsseitig auf der Motorwelle als auch in Gelenknähe angebracht werden. Bei Kraftübertragung durch Seilzüge ist zusätzlich eine Messung der Längskräfte im Seilzug mit Hilfe von kleinen Kraftsensoren möglich. Alle erwähnten Verfahren beruhen darauf, eine Widerstandsänderung der Dehnmessstreifen bei Dehnung auszuwerten.

Dies geschieht in der Regel über Widerstandsnetzwerke und Messung von Spannungsdifferenzen. Dadurch fällt dieses Verfahren in die Gruppe der analogen Messprinzipien. Da die auftretenden Spannungsdifferenzen sehr gering sind und somit eine analoge Übermittlung des Messwerts fehleranfällig ist, ist eine Analog-Digital-Wandlung nahe beim Sensor ratsam.

Kraftbestimmung Die Messung von Kräften beruht auf den selben Verfahren, wie die Drehmomentmessung, also auch auf dem Einsatz von Dehnmessstreifen. Zur gleichzeitigen Bestimmung der drei Kraft- und Drehmomentkomponenten im Handgelenk, kann ein speziell geformter Kraft-Momenten-Sensor zum Einsatz kommen. Dieser ermittelt über Verrechnung der einzelnen Dehnungen in der Struktur Kräfte und Momente in den drei Raumrichtungen. Auch hier sind die auftretenden Spannungsdifferenzen gering und somit eine Analog-Digital-Wandlung nahe beim Sensor sinnvoll.

Druckmessung Die Messung des Drucks in einem fluidischen Medium erfolgt in der Regel mit piezoelektrischen oder piezoresistiven, aber auch mit kapazitiven Sensoren. Diese geben abhängig vom vorherrschenden Druck ein elektrisches Ausgangssignal aus, welches über einen Analog-Digital-Wandler dem Rechnersystem zugänglich gemacht werden kann.

Kontakterkennung Zur Kontakterkennung kommen einerseits Sensoren zum Einsatz, die eine binäre Ausgabe haben, also erkennen, ob ein Kontakt zu Stande gekommen ist oder nicht. Andere Sensoren können über die Änderung der Leitfähigkeit des Sensormaterials bei Druck auch die Kontaktkraft angeben. Durch Aufbringen einer Matrix aus diesem Sensormaterial lässt sich ein ortsauflösender Sensor aufbauen, der einer künstlichen Haut entspricht. Da auch dieses Verfahren auf dem Messen von kleinen Spannungen beruht, sollte eine sensornahe Analog-Digital-Wandlung durchgeführt werden.

Abstandsbestimmung Verfahren, die zur Bestimmung von Abständen dienen, beruhen meist auf Laufzeitmessung eines Signals. In Frage kommt Laufzeitmessung von Licht oder Ultraschallsignalen. Verbreitet sind Ultraschallsensoren und Infrarotsensoren, die sich in ihrem Abdeckungsbereich und der Entfernungsauflösung unterscheiden.

Motorstrommessung Um die Motoren und die Endstufen-Platinen zu schützen, muss der Stromfluss zu den Motoren überwacht und begrenzt werden. Dies kann durch Spannungsmessung über einen Shunt (Nebenschlusswiderstand) erfolgen. Überschreitet der Spannungsabfall über den Shunt einen definierten Pegel, wird die Spannungsversorgung für den Motor getrennt und somit ein zu hoher Stromdurchfluss verhindert. Neben dieser Schutzschaltung kann der Spannungsabfall auch über einen Analog-Digital-Wandler ausgewertet werden und zu Rückschlüssen auf das Motordrehmoment ausgenutzt werden.

4.3.4. Aktoren

In der Planungsebene werden aus den ermittelten Sensorwerten und dem Handlungsplan Sollwerte für die Aktoren generiert. Diese Sollwerte liegen vorerst als digitales Signal vor und müssen dann über eine für den jeweiligen Aktortyp angepasste Leistungselektronik an die Aktoren weitergeleitet werden. Hier werden kurz die in der Robotik verbreiteten Aktortypen vorgestellt und die zum Betrieb dieser Aktoren notwendige Ansteuerungselektronik besprochen.

Elektromotoren Elektromotoren sind die in der Robotik am weitesten verbreiteten Antriebseinheiten. Sie eignen sich sehr gut für den Einsatz in humanoiden Robotern, da sie über eine hohe Leistungsdichte und kleine Bauform verfügen. Die üblicherweise verwendeten bürstenbehafteten Motoren sind zudem sehr gut anzusteuern und verfügen über eine gute Regelungscharakteristik. Die Ansteuerung erfolgt über eine Leistungselektronik, die an den Motor eine veränderliche Spannung ausgibt und darüber die Drehgeschwindigkeit und das Drehmoment beeinflusst. Die veränderliche Spannung kann einerseits tatsächlich ein analog variierender Spannungspegel sein oder andererseits durch Pulsweitenmodulation einer festen Spannung erreicht werden. Die Ansteuerung bürstenloser Motoren gestaltet sich etwas komplexer, da bei diesen die Kommutierung der Motorspulen durch die Ansteuerelektronik gesteuert werden muss. Die Vorteile bürstenloser Motoren sind eine noch höhere Leistungsdichte und die verminderte elektromagnetische Störung durch das Fehlen der Bürsten.

Hydraulik Hydraulische Antriebe beruhen darauf, dass eine Flüssigkeit unter Druck in den Aktor gepumpt wird und daraus eine mechanische Bewegung resultiert. Üblicherweise wird der Druck durch eine Pumpe erzeugt, die im Regelfall durch einen Elektromotor angetrieben wird. Seitens der Schnittstellen und der Leistungselektronik stellt sich der hydraulische Antrieb im Bezug zum Rechnersystem also vergleichbar zu dem Elektroantrieb dar. Drehmoment- oder Motorstrommessungen werden in diesem Fall dann durch Druckmessungen ersetzt. Allerdings sind hydraulische Antriebe für kompakte Systeme wie humanoide Roboter nur bedingt geeignet, da ihr Gewicht und ihr Bauraum zu groß sind. Zudem stellt das Verlegen der Druckschläuche von der Pumpe zum Aktor und die kaum zu vermeidenden Undichtigkeiten in den Zuleitungen einen Nachteil beim Einsatz im humanoiden Roboter dar.

Pneumatik Im Gegensatz zum hydraulischen Antrieb wird beim pneumatische Antrieb Luft als Medium benutzt. Grundsätzlich ist das Funktionsprinzip gleich. Die Luft wird durch einen Kompressor komprimiert und in einem Drucktank zwischengespeichert. Der Kompressor kann mit einem Elektromotor betrieben werden, wodurch sich die pneumatischen Antriebe gegenüber dem Rechnersystem wiederum, wie der Elektroantrieb darstellen.

Nachteilig für den Einsatz im humanoiden Roboter sind der benötigte Bauraum des Kompressors und des Drucktanks und deren Gewicht. Zudem ist die Regelungscharakteristik pneumatischer Antriebe aufgrund der Kompressibilität von Luft schwer beherrschbar.

Audioausgabe Die Ausgabe von Sprache bzw. Audiosignalen im allgemeinen stellt zwar keinen Aktor im engeren Sinne dar, wird aber in dieser Kategorie aufgeführt, da es sich um eine vom System initiierte Aktion handelt, die sich dem Bereich Aktion zurechnen lässt. Die Audioausgabe erfolgt über Lautsprecher, die im Roboter integriert werden müssen und den eigentlichen Aktor darstellen. Seitens des Rechnersystems muss eine Soundkarte vorhanden sein und ein Audioverstärker, so dass eine ausreichende Lautstärke erreicht wird.

4.3.5. Schnittstellen

Nachdem die Messgrößen nach den in Abschnitt 4.3.3 beschriebenen Verfahren ermittelt wurden, müssen die Messwerte noch zur Weiterverarbeitung zu den Recheneinheiten übertragen und ausgewertet werden. Wie eingangs erwähnt, kommen bei den analogen Signalen für die Übertragung zu den Recheneinheiten die direkte Übertragung des analogen Signals oder die Analog-Digital-Wandlung beim Sensor und Übertragung eines digitalen Signals in Frage. Bei bereits digital vorliegenden Messwerten liegt die digitale Übertragung nahe. Je nach erforderlicher Bandbreite und Länge des Übertragungswegs stehen unterschiedliche Übertragungsformate und Protokolle zur Verfügung. Als besonders geeignet für den Einsatz im humanoiden Roboter haben sich die in den folgenden Absätzen näher beschriebenen Kommunikationswege herausgestellt. Folglich wird es eine Anforderung an die aufzubauende Rechnerarchitektur sein, mindestens diese Übertragungswege als Schnittstellen vorzusehen. Über das Bereitstellen dieser Schnittstellen hinaus, soll die Integration weiterer Verbindungstechniken prinzipiell von der Rechnerarchitektur unterstützt werden.

Gigabit-Ethernet Ethernet ist eine paketbasiertes Verfahren zur kabelgebundenen Datenübertragung. Für Ethernet sind Übertragungsraten von 10 Megabit/s bis 10 Gigabit/s spezifiziert. Aus Sicht des OSI-Modells werden sowohl die physikalische Schicht als auch die Data-Link-Schicht spezifiziert. Der Ethernetstandard ist weitestgehend über die IEEE-Norm 802.3[4] spezifiziert. Der Buszugriff bei Ethernet erfolgt durch „CSMA/CD“ (Carrier Sense Multiple Access/Collision Detection), dadurch kann es bei hoher Busauslastung und ungünstigen Konstellationen zu nicht kalkulierbaren Latenzen für zu übertragende Daten kommen.

FireWire (IEE1394) FireWire ist ein schneller serieller Bus, der auf einer 6-Drahtverbindung beruht. Unterstützt werden Datenübertragungsraten bis zu 400 Mbit/s. An einem FireWire-

[4] `http://standards.ieee.org/getieee802/portfolio.html`

Bus können bis zu 63 Teilnehmer angeschlossen sein. Es wird sowohl isochrone als auch asynchrone Datenübertragung unterstützt. Das Gerät mit der höchsten Knoten-ID fungiert als Master und sorgt für die Arbitrierung. FireWire ist unter der IEEE-Norm 1394a[5] spezifiziert.

CAN-Bus Das „Controller Area Network“ (CAN-Bus) fällt in die Gruppe der Feldbusse [65]. CAN ist ein asynchrones, serielles Bussystem, welches von der Firma Bosch für die Verwendung in der Automobilbranche entwickelt wurde [133]. CAN zeichnet sich durch eine robuste differenzielle Übertragung mit Übertragungsraten bis zu 1 Mbit/s aus. Der Buszugriff erfolgt mittels „CSMA/CA“ (Carrier Sense Multiple Access/Collision Avoidance). Dies wird mit einer verlustfreien, bitweisen Arbitrierung erreicht, bei der die Nachricht mit der höchsten Priorität den Buszugriff erhält. Somit lässt sich für diese Nachricht die Latenz von einem Sendezyklus garantieren. Weiterhin verfügt das CAN-Protokoll über mehrere Verfahren zur Übertragungssicherung wie Bitstuffing und der Bildung einer CRC-Checksumme. Der CAN-Bus wird im Anhang B.2 eingehender beschrieben.

Serielle Schnittstelle (RS-232) Die serielle Schnittstelle ist ein digitale Schnittstelle, die bei Computern, aber auch bei Controllern sehr verbreitet ist. Wenn hier von serieller Schnittstelle gesprochen wird, ist die Implementierung nach der Norm RS-232 bzw. EIA-232 gemeint. Der Standard definiert eine Wort-basierte, bit-serielle Datenübertragung. Die serielle Schnittstelle soll auf dem Roboter hauptsächlich wegen ihrer einfachen Handhabung und praktischen Verwendbarkeit zur Ausgabe von Debug-Informationen integriert werden.

Synchron-Serielle Schnittstelle (SSI) Die von der Firma SICK-STEGMANN entwickelte Synchron-Serielle Schnittstelle zeichnet sich durch elektrisch einfachen Aufbau aus. In der einfachsten Verbindungsvariante reicht zur Datenübertragung eine Verbindung über eine verdrillte Zweidrahtleitung. Die Synchron-Serielle Schnittstelle wurde ursprünglich für die Übertragung von Sensorwerten absoluter Winkelcodierer entworfen. Inzwischen ist der Einsatz dieser Schnittstelle jedoch nicht mehr auf dieses Gebiet eingeschränkt. Sie findet außerdem Verwendung bei inkrementellen Winkelgebern und Analog-Digital-Wandlern.

Serial Peripheral Interface (SPI) Das „Serial Peripheral Interface“ (SPI) ist ein von Motorola entwickeltes Bus-System. Die Übertragung findet mit Hilfe eines synchronen seriellen Datenbusses statt. Nach dem Master-Slave-Prinzip werden die digitalen Schaltkreise miteinander verbunden. Bei Verwendung nur eines Slaves reichen drei Verbindungsleitungen aus. Dies sind eine serielle Clock (SCK), „Master In, Slave Out“ (MISO) sowie

[5] `http://standards.ieee.org/reading/ieee/std_public/description/busarch/1394-1995_desc.html`

„Master Out, Slave In“ (MOSI). Bei Sternförmiger Verkabelung mehrerer Slaves wird je Slave noch ein so genanntes „Slave Select“ (SS) benötigt, um einen Slave als aktiv auszuwählen. Es besteht auch die Möglichkeit mehrere Slaves kaskadiert anzusteuern, in diesem Fall ist nur ein „Slave Select“, das an alle Slaves angeschlossen wird, notwendig.

Quadraturkodierung Der für Quadraturkodierung typische Signalverlauf besteht aus zwei phasenversetzten Rechtecksignalen. Diese Signalform kommt dadurch zustande, dass Licht durch eine Codescheibe fällt und an zwei versetzten Photodioden ausgewertet wird. Die Photodioden sind so versetzt, dass eine Phasenverschiebung von $\pm 90°$ zwischen Kanal A und Kanal B auftritt. Anhand des Vorzeichens der Phasenverschiebung kann erkannt werden in welche Richtung die Codescheibe gedreht wird. Eilt das Signal auf Kanal A dem auf Kanal B 90° voraus, erfolgt die Drehung in die eine Richtung. Eilt das Signal auf Kanal B dem auf Kanal A um 90° voraus so findet die Drehung in die entgegengesetzte Richtung statt. Die Frequenz der Pulse bzw. die Anzahl der Pulse ergibt den zurückgelegten Weg. Zur Auswertung dieser bei größeren Drehgeschwindigkeiten hochfrequenten Signale werden seitens der Rechnerarchitektur spezielle Quadraturzähler benötigt.

Analoge Übertragung Werden Signale analog übertragen, müssen diese spätestens dort, wo sie den Recheneinheiten zur Verfügung gestellt werden sollen, durch Analog-Digital-Wandlung in digitale Signale überführt werden. Unterschiede bei den Analog-Digital-Wandlern gibt es vor allem bei der Samplezeit und in der Auflösung. Typische Samplezeiten bewegen sich im Bereich von einigen Mikrosekunden. Die Auflösung eines Analog-Digital-Wandlers wird in bit angegeben, wobei beispielsweise die Angabe einer Auflösung von 10 bit bedeutet, dass der Wandler 2^{10} also 1024 Spannungspegel unterscheiden kann. Übliche Werte sind 10 bit, 12 bit und 16 bit.

4.4. Zusammenfassung

In diesem Kapitel wurde das Konzept zur Hardware-Software-Steuerung humanoider Roboter vorgestellt. Die Hardware-Software-Architektur untergliedert sich in die drei Aspekte funktional-logische Steuerung, Softwarearchitektur und Hardwarearchitektur. Als Steuerungsarchitektur wird eine in drei Hierarchiestufen organisierte Architektur beschrieben. Diese behandelt die Bereiche Perzeption, Planung und Aktion und diskutiert die Datenflüsse zwischen diesen Bereichen und die Nutzung von Datenbanken zum Aufbau von Umwelt- und Handlungswissen. Die Abstraktion von der tatsächlichen Roboterhardware wird über die Schnittstellen logischer Sensor und logischer Aktor realisiert. Diese stellen den darüber liegenden Softwareschichten einen vereinheitlichten Zugriff auf die eingesetzten Sensoren und Aktoren zur Verfügung. Die Softwarearchitektur basiert auf dem Softwarerahmenwerk MCA2, welches eine durchgehende Modularisierung der Software ermöglicht. Die Modularisierung erfolgt durch

die MCA-Module, die über klar definierte Schnittstellen verfügen und zur weiteren Abstraktion ihrerseits wieder zu so genannten MCA-Gruppen zusammengefasst werden können. Zur formalen Darstellung der MCA-Module und der Datenflüsse zwischen diesen Modulen werden UML-Komponentendiagramme verwendet. Mit den beiden Programmen *mcabrowser* und *mcagui* existieren zwei mächtige Werkzeuge zur Kontrolle der Robotersteuerung. Die Hardwarearchitektur besteht aus einem Verbund verteilter Rechenkomponenten. Als Merkmal zur Aufteilung auf die verschiedenen Komponenten wurde die Zuordnung zu den Domänen Bildverarbeitung, Akustik, Planung und Regelung herangezogen. Weitere Entscheidungskriterien sind die erforderliche Rechenleistung, geforderte Zykluszeiten, der Speicherbedarf und benötigte Verbindungsbandbreite in diesen Domänen. Insbesondere wird zwischen echtzeitfähigen und nicht echtzeitfähigen Komponenten unterschieden. Die Domänen Bildverarbeitung, Akustik und Planung lassen sich dem nicht echtzeitfähigen Bereich zuordnen. Für diese kann die Hardwarerealisierung mit Standardkomponenten, so genannten Industrie-PCs, und Standardbussystemen wie Gigabit-Ethernet erfolgen. Für die Regelungsdomäne wurde eine dezidierte Komponente, das UCoM, entwickelt. Das UCoM übernimmt als verteilte Recheneinheit nahe bei den Sensoren bzw. Aktoren die Sensor-Motor-Regelung auf dem Roboter. In die Softwarearchitektur wird es wie ein MCA-Modul eingebunden, wobei der CAN-Bus zur Kommunikation und als Abstraktionsschnittstelle genutzt wird.

5. Umsetzung und Evaluierung der Architektur auf ARMAR-III

In diesem Kapitel wird die Realisierung und Evaluierung der vorgeschlagenen Hardware-Software-Architektur anhand des humanoiden Roboters *ARMAR-III* beschrieben. *ARMAR-III* ist ein humanoider Roboter, der im Rahmen des Sonderforschungsbereichs „SFB 588 Humanoide Roboter - Lernende und kooperierende multimodale Roboter[1]“ entwickelt wurde (Abb. 5.1). Er soll als humanoider Assistenzroboter, vornehmlich in der Küchenumgebung, zum Einsatz kommen [8]. In den folgenden Abschnitten wird erst der Aufbau des Roboters aus Sicht der Mechatronik geschildert und die Ausstattung mit Sensorik und Aktorik beschrieben. Im Anschluss wird der konkrete Aufbau der Hardwarearchitektur auf *ARMAR-III* und die Umsetzung der Steuerungs- und Softwarearchitektur auf dieser Basis erläutert. Schließlich wird die Hardware-Software-Architektur evaluiert.

5.1. Die Entwicklung des humanoiden Robotersystems ARMAR-III

ARMAR-III ist der dritte humanoide Roboter in der ARMAR-Serie (Abb. 5.2). Der humanoide Roboter *ARMAR-I* wurde zur Untersuchung von Manipulationsaufgaben mit Zweiarmsystemen aufgebaut [13, 12]. Durch inkrementelle Verbesserungen an der Mechanik und am Rechnersystem wurde das System *ARMAR-II* entwickelt. In *ARMAR-II* basierte die Hardware-Software-Architektur auf einem Industrie-PC, der mit C167-Minimodulen der Firma Phytec[2] zur Gelenksteuerung verbunden war. Zur rechenintensiven Bildverarbeitung wurde ein Laptop, welches auf der Plattform montiert war, verwendet. Die Nutzung des Laptops kann jedoch nicht als Lösung im Sinne einer integrierten Rechnerarchitektur gesehen werden. Weiterhin konnten mit den verwendeten C167-Minimodulen, die jeweils für die Steuerung von vier Achsen eingesetzt wurden, nicht die erforderlichen Zykluszeiten erreicht werden, was die Regelgüte stark beeinträchtigt.

Sowohl aus mechatronischer, als auch aus Sicht der Hardware-Software-Architektur wurden bei *ARMAR-III* neue Wege beschritten. Es wurde eine wesentlich höhere Integration erreicht und auch die Anforderungen an das Rechnersystem konnten erfüllt werden.

5.1.1. Das mechanische System

ARMAR-III ist ein humanoider Assistenzroboter. Er bewegt sich nicht mit Hilfe von zwei Beinen, sondern der Oberkörper befindet sich auf einer holonomen Plattform [74]. Der Oberkör-

[1] `http://www.sfb588.uni-karlsruhe.de`

[2] `http://www.phytec.de/de/produkte/module-nach-architektur/16-bit.html`

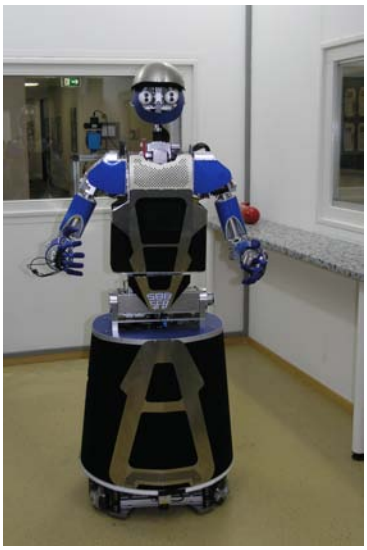


Abb. 5.1.: Der humanoide Roboter *ARMAR-III*

Abb. 5.2.: Die Vorläufer von *ARMAR-III*: *ARMAR-I* (links), *ARMAR-I* mit modifizierter Tragstruktur im Oberkörper und Verkleidung (in der Mitte), *ARMAR-II* mit verbessertem Rechnersystem und Verkleidung (rechts)

per kann in der Hüfte in drei Bewegungsfreiheitsgraden bewegt werden. Auf dem Oberkörper befindet sich ein Hals mit vier Bewegungsfreiheitsgraden, der den Kopf mit den zwei unabhängig voneinander schwenk- und gemeinsam neigbaren Augen trägt. Zur Manipulation verfügt *ARMAR-III* über zwei Arme mit jeweils sieben Bewegungsfreiheitsgraden und einer fluidbetriebenen Fünf-Finger-Hand als Endeffektor. Die Hände sind ihrerseits mit jeweils acht Bewegungsfreiheitsgraden aufgebaut. Ohne die Hände verfügt das Robotersystem über 27 Bewegungsfreiheitsgrade. Die Kinematik des Roboteroberkörpers ist in Abb. 5.3 dargestellt. In den folgenden Abschnitten wird die Mechanik und das Sensor- und Aktorsystem des humanoiden Roboters näher beschrieben.

5.1.2. Aktor- und Sensorsystem

Aus mechanischer Sicht besteht *ARMAR-III* aus den folgenden Teilsystemen:

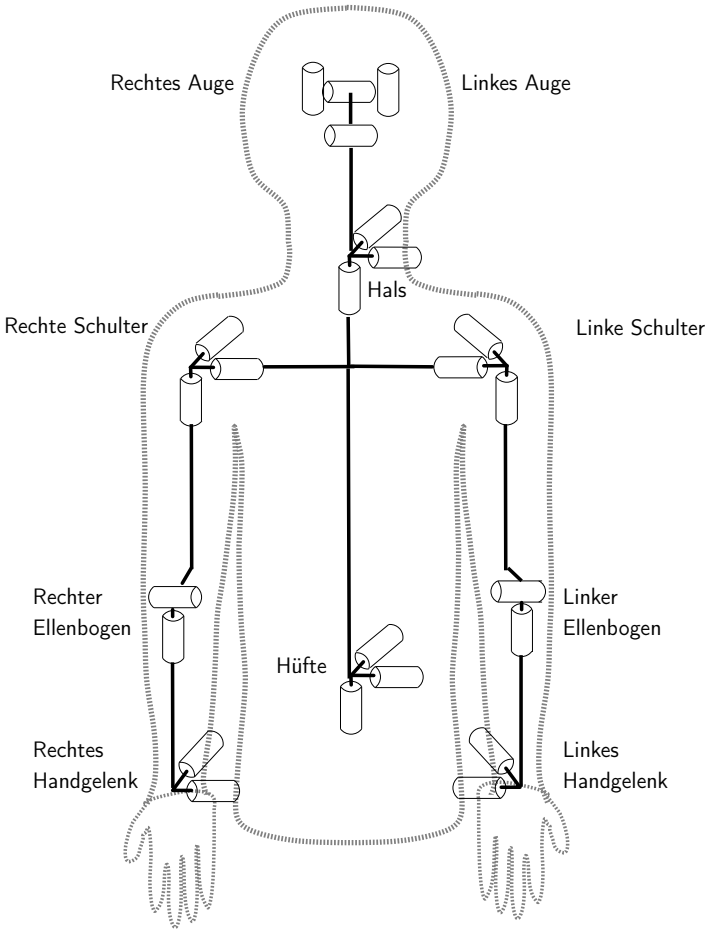


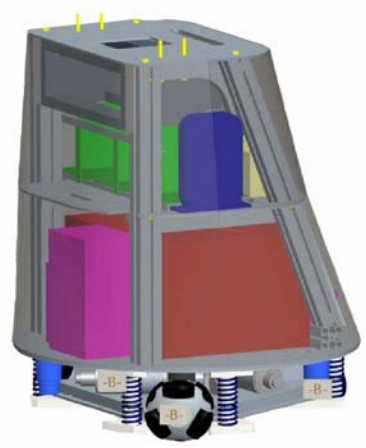
Abb. 5.3.: Kinematik des Roboteroberkörpers und Anordnung der Bewegungsfreiheitsgrade. Die Positionierung der Antriebe stimmt mit Ausnahme der Bewegungsfreiheitsgrade im Ellenbogen mit den eingezeichneten Positionen überein. Die Ellenbogenfreiheitsgrade werden über Seilzüge von Motoren in der Torsobasis angetrieben.

- Holonome Plattform (3 DOF)
- Torsogrundgelenk (3 DOF)
- Torso
- Hals (4 DOF)
- Kopf (3 DOF)
- 2 Armen (je 7 DOF)
- 2 Händen (je 8 DOF)

Anhand dieser Teilsysteme wird das Sensor- und Aktorsystem des Roboters detailliert beschrieben.

(a) Omniwheel

(b) Hokuyo URG-Laserscanner



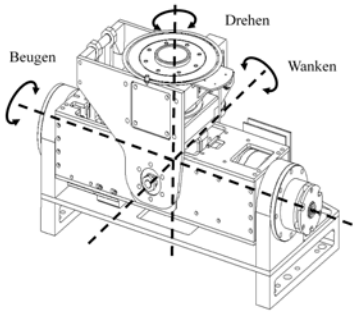
(c) Holonome Plattform Omnibot II (Quelle [74])

Abb. 5.4.: Das für den Antrieb der Plattform verwendete Omniwheel, der Hokuyo URG-Laserscanner zur Erfassung der Position der Plattform anhand einer gespeicherten Karte und die Konstruktionszeichnung der holonomen Plattform

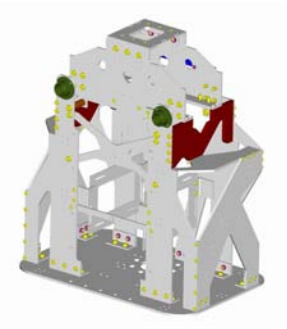
Plattform

Die Plattform dient zur Fortbewegung des Roboters in seiner Arbeitsumgebung. Da *ARMAR-III* als Assistenzroboter in einer strukturierten Umgebung mit ebenem Boden, wie einer Küche oder einer Büroumgebung, eingesetzt werden soll, wurde an dieser Stelle bewusst auf zweibeiniges Laufen verzichtet. Zum Antrieb der Plattform werden drei so genannte Omniwheels[3] verwendet, die jeweils im Winkel von 120° zueinander angeordnet sind. Mit dieser Konstruktion lässt sich ein holonomer Antrieb realisieren (Abb. 5.4). Das heißt, die Plattform kann aus dem Stand in jede Bewegungsrichtung fahren und sich dabei gleichzeitig um die Hochachse drehen. Somit verfügt die Plattform über drei Bewegungsfreiheitsgrade. Als Antrieb werden die Dunkermotoren GR63x55[4] verwendet. Als Sensorik zur odometrischen Regelung der Plattformposition bzw. Plattformgeschwindigkeit kommen die Motorencoder der Antriebsmotoren zum Einsatz, die ein quadraturkodiertes Signal ausgeben. Zur Erstellung von Karten bzw. zur Positionsbestimmung der Plattform werden drei um 120° versetzte Laserscanner eingesetzt. Die URG-04LX-Laserscanner der Firma Hokuyo[5] haben einen Sichtbereich von 240 ° und eine maximale Sichtweite von 4 m. Aufgrund des vorhandenen Platzangebots in der Plattform eignet sich diese zur Aufnahme der Bleigelbatterien zur Stromversorgung des Roboters. Darüber hinaus bleibt noch Bauraum für Elektronik- und Rechnerkomponenten.

[3] `http://www.omniwheel.com/omniwheel/omniwheel.htm`
[4] `http://www.dunkermotoren.de/data/technical_data/motors/pdf/GR63x55.pdf`
[5] `http://www.hokuyo-aut.jp/02sensor/07scanner/urg_04lx.html`



(a) Torsogrundgelenk



(b) Torso als FEM-Modell

(c) Torso

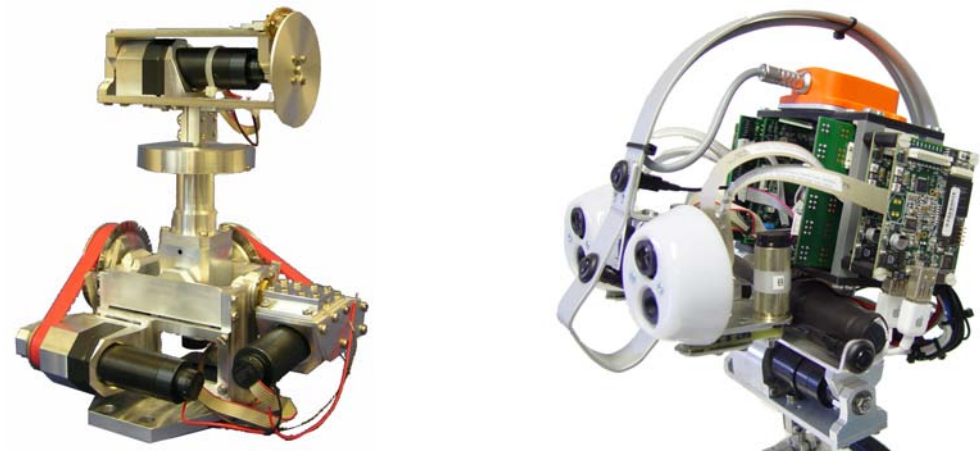
Abb. 5.5.: Das Torsogrundgelenk als Schnittstelle zwischen Plattform und Oberkörper und die Stützstruktur des Roboteroberkörpers (Quelle: [2])

Torsogrundgelenk

Das Torsogrundgelenk stellt die Schnittstelle zwischen der Plattform und dem Oberkörper dar. Mit Hilfe des Torsogrundgelenks werden drei Bewegungsfreiheitsgrade realisiert. Die Oberkörperdrehung erlaubt ein Drehen des Oberkörpers um die Hochachse in einem Winkelbereich von -170° bis +180°. Die seitliche Bewegung des Oberkörpers, das Wanken, ist auf einen Bewegungsbereich von -20° bis +20° ausgelegt. Der Oberkörper kann von -10° nach hinten bis zu 60° nach vorne abgewinkelt werden. Für die jeweiligen Achsen kommen bürstenbehaftete Motoren zum Einsatz, wobei das Wanken und das Beugen noch über eine mechanische Bremse zur Arretierung der momentanen Stellung verfügt. Zur Regelung der drei Achsen werden die quadraturkodierten Werte der Motorencoder benutzt, für Wanken und Beugen ist eine absolute Bestimmung der Position über abtriebsseitig eingebaute Potentiometer möglich.

Torso

Der Torso dient im wesentlichen als Stützstruktur für die Montage der Arme und des Kopfes. Zur Gewichtsersparnis wurde mittels FEM-Analyse in Leichtbauweise eine Struktur aus Aluminiumstreben aufgebaut (Abb. 5.5). Aus Gründen der Reduktion bewegter Massen und der Gewichtsersparnis in den Armen wurden die Antriebe für die Bewegungsfreiheitsgrade Ellenbogen beugen, Unterarm drehen und Arm schwenken in den Torso ausgelagert. Die Antriebe für die beiden Bewegungsfreiheitsgrade im Ellenbogen befinden sich in der Basis des Torsos und treiben die Achsen über Seilzüge an. Die Antriebe für Arm schwenken befinden sich im „Brustkorb“. Für die Regelung der Motoren im Torso werden die quadraturkodierten Signale der Motoren herangezogen. Der Torso bietet darüber hinaus Bauraum für Elektronik- und Rechnerkomponenten.



(a) Hals (Quelle: [2])

(b) Kopf

Abb. 5.6.: Aufnahme des Halses mit vier Bewegungsfreiheitsgraden. Aufnahme des Kopfes mit den paarweise angeordneten Kameras für den Fern- und Nahbereich zur Stereobilderfassung.

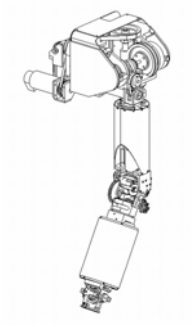
Hals

Der Hals verfügt über vier Bewegungsfreiheitsgrade. Um die Kopfbasis beliebig im Raum auszurichten, würden die drei Achsen Beugen, Wanken und Drehen ausreichen. Um einen menschenähnlicheren Bewegungsablauf erreichen zu können, wurde noch die vierte Achse Nicken in den Hals integriert (Abb. 5.6). Die Achsen Beugen und Wanken werden über Zahnriemen angetrieben und bieten jeweils einen Bewegungsbereich von -45° bis +45°. Die beiden anderen Achsen werden direkt über Motoren mit Harmonic Drive Getriebe[6] betrieben. Die Kopfdrehung verfügt über einen Winkelbereich von -180° bis +180°, das obere Nicken reicht von -60° bis +60°. Zur Positions- und Geschwindigkeitsregelung dieser Achsen werden die quadraturkodierten Signale der Motorencoder verwendet.

Kopf

Der Kopf dient als maßgeblicher Sensorträger für externe Sensorik. Er verfügt für die Augen über eine gemeinsame Nick-Achse und die Augen lassen sich unabhängig voneinander schwenken. Zur Positions- und Geschwindigkeitsregelung der Augen werden die quadraturkodierten Signale der Motorencoder verwendet. Zur Stereobilderfassung kommen je Auge zwei FireWire-Kameras zum Einsatz. Um sowohl im Nahbereich als auch in der Entfernung Bilder in ausreichender Auflösung zu erhalten, sind die Kameras mit Linsen für den Nah- und Fernbereich ausgestattet (Abb. 5.6). Für die akustische Erfassung der Umwelt sind sechs Mikrofone über den Kopf verteilt. Diese sollen einerseits zur Erkennung von Sprache, andererseits zur dreidimensionalen Ortung von Geräuschen genutzt werden. Die Mikrofone werden über einen

[6] `http://www.harmonicdrive.de`



(a) Konstruktionszeichnung

(b) Arm

(c) Fluidhand

Abb. 5.7.: Konstruktionszeichnung des Arms. Aufnahme des Arms mit den haptischen Sensoren auf der Schulter und am Ober- und Unterarm. Aufnahme der Fluidhand in der mit Druckluft betriebenen Variante.

Vorverstärker im Kopf mit einer Sechskanal-Soundkarte verbunden. Zur Bestimmung der Orientierung des Kopfes im Raum dient ein Gyroskop, welches über USB Orientierungsdaten in digitaler Form ausgibt.

Arm

ARMAR-III verfügt über zwei Leichtbauarme mit je sieben Bewegungsfreiheitsgraden. Jeder Arm besitzt in der Schulter drei Bewegungsfreiheitsgrade, im Ellenbogen zwei und im Handgelenk zwei, wobei die Antriebe für die erste Schulterachse und die beiden Achsen im Ellenbogen in den Torso ausgelagert sind (Abb. 5.7). Die Antriebe für Arm heben, Oberarm drehen und Handgelenk beugen und wanken sind in den Arm integriert. Zur Positions- und Geschwindigkeitsregelung werden die Motorencoder verwendet. Im Handgelenk befinden sich zur absoluten Bestimmung der Position Potentiometer, die über einen Analog-Digital-Wandler ausgewertet werden können. Zur Bestimmung des Drehmoments in den drei Achsen der Schulter sind abtriebsseitig auf der Welle bzw. in der Aufnahme der Schneckengetriebe Dehnmessstreifen verbaut. Für die beiden Achsen im Ellenbogen sind nah bei der angetriebenen Achse kleine Kraftsensoren in die Seilzüge eingebaut. Dabei befindet sich je ein Kraftsensor im belasteten und einer im unbelasteten Seilstrang. Durch Differenzbildung der beiden ermittelten Werte lässt sich das Drehmoment in den beiden Achsen bestimmen. Im Handgelenk wird ein 6D-Kraftmomentensensor eingesetzt.

Zur haptischen Interaktion mit dem Roboter ist an der Vorder- und Rückseite der Schulter und auf den Zylindern zur Verkleidung der Ober- und Unterarme eine künstliche Haut aufgebracht [91, 63]. Die künstliche Haut erfasst ortsaufgelöst die Kontaktkräfte. Für den planaren Sensortyp wird über CAN-Bus die resultierende Kraft und der Angriffspunkt ausgegeben, für den zylindrischen Sensortyp werden die resultierenden Kraft- und Drehmomentwerte in X-,

Y- und Z-Richtung übermittelt. Für beide Sensortypen können über USB die ortsaufgelösten Rohdaten ausgelesen werden.

Die Bewegungsbereiche der Achsen sind wie folgt ausgelegt: Arm schwenken von -45° bis +180°, Arm heben von -10° bis +180°, Oberarm drehen von -180° bis +180°, Ellenbogen beugen von -10° bis +130°, Unterarm drehen von -90° bis +90°, Handgelenk wanken von -30° bis +30° und Handgelenk beugen von -60° bis +60°.

Hand

Als Manipulator kommt auf *ARMAR-III* eine Fluidhand zum Einsatz (Abb. 5.7). Es handelt sich um eine feingliedrige Fünf-Finger-Hand mit acht Bewegungsfreiheitsgraden [88, 89]. Als Aktoren in den Fingergelenken kommen spezielle Fluidaktoren zum Einsatz [57]. In der verwendeten Variante wird die Hand mit Druckluft von 6 bar betrieben. Die Ventile und Ansteuerungselektronik sind komplett auf der Hand integriert. Bewegungsbefehle an die Hand werden über eine serielle Schnittstelle übermittelt und lokal von der Hand ausgeführt. Zur handlokalen Regelung werden die integrierten Druck- und Positionssensoren verwendet.

Teilsystem / Achse	Aktoren	Sensoren	Datenübertragung
Plattform			
	3 × Dunkermotoren GR63x55	RE 30	Quadraturkodiert
		3 × Laserscanner	USB
Torsogrundgelenk			
Torso drehen	Faulhaber 3257	HEDS-5540	Quadraturkodiert
Torso beugen	maxon RE40	HEDS-5040	Quadraturkodiert
		Potentiometer	AD-Wandlung
Torso wanken	Faulhaber 3257	HEDS-5540	Quadraturkodiert
		Potentiometer	AD-Wandlung
Torso			
Ellenbogen beugen	Faulhaber 3863	HEDL-5540	Quadraturkodiert
Unterarm drehen	Faulhaber 3863	HEDL-5540	Quadraturkodiert
Arm schwenken	maxon RE40	HEDS-5540	Quadraturkodiert

Fortsetzung auf nächster Seite

Tab. 5.1 – Fortsetzung von voriger Seite

Teilsystem / Achse	Aktoren	Sensoren	Datenübertragung
	Sprachausgabe		Soundkarte
	Visuelle Ausgabe		VGA
Hals			
Hals drehen	Faulhaber 2342	IE-512	Quadraturkodiert
Hals beugen	Faulhaber 2342	IE-512	Quadraturkodiert
Hals wanken	Faulhaber 2342	IE-512	Quadraturkodiert
Hals nicken	Faulhaber 2342	IE-512	Quadraturkodiert
Kopf			
Augen beugen	Harmonic Drive PMA-5A-50	MR-512ML	Quadraturkodiert
Auge drehen	Faulhaber 1524	IE-512	Quadraturkodiert
		Gyroskop	USB
		2 × Kameras nah	FireWire
		2 × Kameras fern	FireWire
		6 × Mikrofon	Soundkarte
Arme			
Heben	maxon RE40	HEDS-5540	Quadraturkodiert
Oberarm drehen	Faulhaber 3257	IE-512	Quadraturkodiert
Handgelenk beugen	maxon RE25	MR-512ML	Quadraturkodiert
		Potentiometer	ADC → SSI
Handgelenk wanken	maxon RE25	MR-512ML	Quadraturkodiert
		Potentiometer	ADC → SSI
		2 × 6-DOF Kraft-Momenten-Sensoren	AD-Wandlung
		8 × Kraftsensor im Seilzug	ADC → CAN

Fortsetzung auf nächster Seite

Tab. 5.1 – Fortsetzung von voriger Seite

Teilsystem / Achse	Aktoren	Sensoren	Datenübertragung
		4 × Momentenmessung mit Dehnmessstreifen	ADC → CAN
		Haut an Schulter, Ober- und Unterarm	CAN, USB
Hand			
Fingerfreiheitsgrade	Fluidaktor	Integrierte Steuerung	RS232

Tab. 5.1.: Übersicht über die Verteilung der Sensoren und Aktoren über die topologischen Teilsysteme des humanoiden Roboters *ARMAR-III*

5.2. Hardware-Software-Architektur auf ARMAR-III

In diesem Abschnitt wird das in Kapitel 4 beschriebene Konzept für eine Hardware-Software-Architektur auf das System *ARMAR-III* angewendet. Für den Bereich der Hardwarearchitektur wird beschrieben, mittels welcher Rechnerkomponenten und Verbindungsstrukturen die Architektur realisiert wird. Für die Steuerungsarchitektur und die Softwarearchitektur wird ebenfalls die Struktur der Steuerung auf *ARMAR-III* und die Modularisierung der Roboterfunktionalitäten mittels MCA2 beschrieben.

5.2.1. Hardwarearchitektur

Für den Aspekt der Hardwarearchitektur wird für das Robotersystem *ARMAR-III* beschrieben, welche Rechnerkomponenten für den Echtzeit- und Nichtechtzeitteil ausgewählt wurden, um einen autonomen Roboterbetrieb mit den geforderten Roboterfähigkeiten zu gewährleisten. Hierbei wird detailliert auf die ausgewählten Komponenten und die Schnittstellen zu deren Verbindung eingegangen. Zur Anpassung an das mechatronische Robotersystem sind insbesondere die Integration der Aktorik und Sensorik zu berücksichtigen. Die in Tab. 5.1 aufgeführte Übersicht über die in *ARMAR-III* benötigten Schnittstellen stellt dabei die Minimalanforderungen dar.

Das Konzept der Hardwarearchitektur sieht die Realisierung mit verteilten Komponenten vor. Zum Einsatz im humanoiden Roboter für den Nichtechtzeitbereich eignen sich die in Abschnitt 4.3.1 vorgestellten Recheneinheiten [128]. Für den Echtzeitteil soll das UCoM und das zugehörige Motortreiberboard zum Einsatz kommen. Bei der Realisierung der verteilten Rechnerarchitektur auf dem humanoiden Roboter waren folgende Fragestellungen relevant: wie viele Komponenten werden benötigt, wie werden die Anwendungsprogramme zur Realisierung der Fähigkeiten auf die einzelnen Recheneinheiten verteilt und wie können die Recheneinheiten topologisch im Roboter verteilt werden. Da die Rechenleistung der vorgestellten Industrie-PCs im Bereich der in den einzelnen Domänen geforderten Rechenleistung liegt (Tab. 3.4), wurde entschieden für jede Domäne einen dezidierten Industrie-PC einzusetzen. Bezüglich der topologischen Verteilung sollen die Industrie-PC nach Möglichkeit so verteilt werden, dass sie nah am Ort der Datenentstehung bzw. bei den zu steuernden Aktoren sind. Im Abschnitt 5.1.2 und in Tab. 5.1 wurden die Teilsysteme des Roboters aufgelistet. Für die Platzierung der Industrie-PCs eignen sich aufgrund des Platzangebots nur die Plattform und der Oberkörper. In der Plattform lassen sich alle drei der in 4.3.1 beschriebenen Rechnertypen unterbringen, im Oberkörper kommt nur der Einsatz des PC/104-Typs in Frage. Von den Platzressourcen lassen sich im Oberkörper zwei PC/104-Systeme verbauen, die Plattform erlaubt den Einbau von bis zu drei Backplane-Systemen. Von der Zuordnung der Domänen zu Rechnereinheiten erfolgte die folgende Aufteilung.

Hardwarerealisierung für Bildverarbeitung

Der Bildverarbeitungsrechner wird im Oberkörper des Roboters platziert, um eine modulare Trennung des Roboters in Oberkörper und Fortbewegungseinheit zu gewährleisten. Somit muss die Bildverarbeitung auf einem PC/104-System durchgeführt werden. Als Anforderung an die Rechenleistung wurde ermittelt, dass diese mit einem 2 GHz Pentium-M Prozessor erfüllt werden. Als Speicherausstattung wurde 1 GB RAM ausgewählt. Das konkret verwendete Rechnersystem ist der Cool RoadRunner 4 der Firma Lippert[7] bestückt mit einem Pentium M 755 Prozessor mit einer Taktung von 2 GHz. Es wurde der maximale Speicherausbau mit 1 GB DDR-333 SODIMM RAM gewählt. Als Schnittstelle zu den Kameras wurde eine PC/104-Lage mit zwei Firewireschnittstellen eingesetzt. Dabei ist je eine Firewireschnittstelle für ein Kamerapaar des rechten bzw. linken Auges im Einsatz. Die Kamerapaare der Augen sind zu diesem Zweck je über einen Firewirehub im Kopf miteinander verbunden. Als weitere Standardschnittstellen bietet der Cool RoadRunner 4 VGA, Gigabit-Ethernet und USB. Dieser Rechner wird als *armar3-4* bezeichnet.

Hardwarerealisierung für Akustik

Der Rechner zur Bearbeitung der Akustiksignale wird in die Plattform integriert. Auch wenn eine Platzierung des Rechners im Oberkörper aufgrund der Nähe zu den Mikrofonen auf den ersten Blick sinnvoller erscheint, ist die Platzierung in der Plattform vorteilhaft. Einerseits bietet der Oberkörper nur noch Platz für einen weiteren PC/104 und dieser wurde für die Regelung reserviert, um zumindest die Kernfähigkeiten des Roboters im Oberkörper zu integrieren. Andererseits lässt sich die Auswertung der sechs Mikrofone, die synchron erfolgen muss, nur mit einer speziellen Soundkarte realisieren, die für den PC/104-Formfaktor nicht verfügbar ist. Mit einem Pentium M 1,6 GHz Prozessor lassen sich die Anforderungen der Akustikverarbeitung erfüllen.

Als konkretes System wird ein backplanebasierter Industrie-PC mit Steckkarten-CPU verwendet. Es kommt die passive Busplatine PCA6105P5[8] mit einem PCISA-Steckplatz für die Steckkarten-CPU und vier PCI-Steckplätzen zum Einsatz. Als Steckkarten-CPU dient die PCI-6881F[9] ausgestattet mit einem Pentium M 1,6 GHz Prozessor. Als Speicherausbau wurde für diesen Rechner 512 MB SODIMM Speicher verwendet. Die Steckkarten-CPU bietet die üblichen Standardschnittstellen, wie das zur Verbindung mit den anderen Recheneinheiten eingesetzte Gigabit-Ethernet. Zur Auswertung der Signale der sechs Mikrofone kommt eine spezielle Soundkarte zum Einsatz, die die zeitsynchrone Aufnahme von über sechs Kanälen erlaubt. Es

[7] `http://www.lippertembedded.com/cool-roadrunner-4.html`
[8] `http://www.advantech.com/products/PCA-6105P5/mod_1-2JKG0V.aspx`
[9] `http://www.advantech.com/products/PCI-6881/mod_1-2JKGI2.aspx`

kommt die Hammerfall DSP 9652-Soundkarte[10] zum Einsatz. Dieser Rechner wird als *armar3-2* bezeichnet.

Hardwarerealisierung für Regelung

Um auch bezüglich der Regelung eine Trennung zwischen Oberkörper und Fortbewegungseinheit zu erzielen, werden für den Bereich der Regelung in *ARMAR-III* zwei unterschiedliche Rechner zur Regelung verwendet. Einer davon ist im Oberkörper, der andere in der Plattform untergebracht. Im Oberkörper wird, neben dem Bildverarbeitungsrechner, ein zweites PC/104-System für die Regelung eingesetzt. Da die Regelung hauptsächlich durch die verteilten UCoM-Komponenten ausgeführt wird, sind die Anforderungen an die PC-Komponente der Regelung geringer als beispielsweise für die Bildverarbeitung. Als PC/104 kommt im Oberkörper der Cool RoadRunner 3 mit einem Pentium III-Prozessor mit 933 MHz und einer Speicherausstattung von 512 MB SDRAM zum Einsatz. Zur Verbindung mit den für die Servo-Motor-Regelung zuständigen UCoMs ist dieser PC/104 mit einer CAN-Karte ausgestattet, die über vier CAN-Schnittstellen verfügt. Zur Auswertung der 6D-Kraftmomentensensoren des Handgelenks kommt eine spezielle Analog-Digital-Wandler-Karte zum Einsatz. Mit der DMM-32-AT[11] können bis zu 32 Analogkanäle ausgewertet werden. Dieser Rechner wird als *armar3-3* bezeichnet.

Für die Regelung der in Tab. 5.1 aufgeschlüsselten Aktoren zur Ansteuerung des Roboters kommt die Kombination aus UCoM und Dreier-Motorboard zum Einsatz. Diese dezentralen Regelungsknoten können aufgrund ihres Formfaktors nah bei den zu bewegenden Achsen angebracht werden. Bewegungsvorgaben werden vom Regelungs-PC/104 über vier CAN-Busse an die UCoMs übermittelt (Tab. 5.2). In der Gegenrichtung übermitteln die UCoMs die aktuellen Sensorinformationen der zugehörigen Achse in frei definierbaren Zeitintervallen. Die Auswertung der achsbezogenen Sensorinformationen, wie Motordrehzahl, Achsposition oder Motorstrom, erfolgt komplett dezentral auf dem UCoM. Abhängig von den durch den Regelungs-PC/104 vorgegebenen Sollgrößen für Geschwindigkeit und Position wird die Regelung der Achsen komplett auf den dezentralen Einheiten durchgeführt. Die Verteilung der UCoMs über den Roboter erfolgte zusammengefasst nach Gelenkkomplexen, so dass die dezentralen Knoten nach Möglichkeit mit drei Achsen ausgelastet sind. Insgesamt sind in *ARMAR-III* elf UCoM-Motorboard-Kombinationen im Einsatz. Die Aufteilung erfolgte nach topologischen Gesichtspunkten, so dass Verbindungswege zwischen Sensoren, Aktoren und UCoM möglichst kurz ausfallen und nicht über Gelenke hinweg gehen. Die zusammengefassten Gelenkkomplexe und die Positionierung der UCoM-Motorboard-Kombinationen im Roboter sind in Tab. 5.2 aufgeführt. Neben der Information, welche Achsen angesteuert werden, ist beschrieben, über welchen

[10] `http://www.rme-audio.de/products_hdsp_9652.php`
[11] `http://www.diamondsystems.com/products/diamondmm32at`

Platzierung	Funktion	Bus
Nacken	Hals beugen Hals wanken Hals drehen	CAN0
Kopf links	Hals oberes Nicken Augen beugen	
Kopf rechts	Auge rechts drehen Auge links drehen	
Schulter rechts, vorne	Handgelenk rechts beugen Handgelenk rechts wanken	CAN1
Schulter rechts, hinten	Arm schwenken rechts Oberarm rechts heben Oberarm rechts drehen	
Bauch rechts	Ellenbogen rechts beugen Unterarm rechts drehen	
Schulter links, vorne	Arm schwenken links Oberarm links heben Oberarm links drehen	CAN2
Schulter links, hinten	Handgelenk links beugen Handgelenk links wanken	
Bauch links	Ellenbogen links beugen Unterarm links drehen	
Plattform	Ansteuerung der drei Plattformmotoren	CAN3
Torsogrundgelenk	Oberkörper drehen Oberkörper beugen Oberkörper wanken	

Tab. 5.2.: Topologische Verteilung der UCoMs über den Roboter. Es wird dargestellt, wo die UCoMs platziert sind, welche Achsen sie ansteuern und über welchen CAN-Bus sie mit dem Regelungs-PC verbunden sind

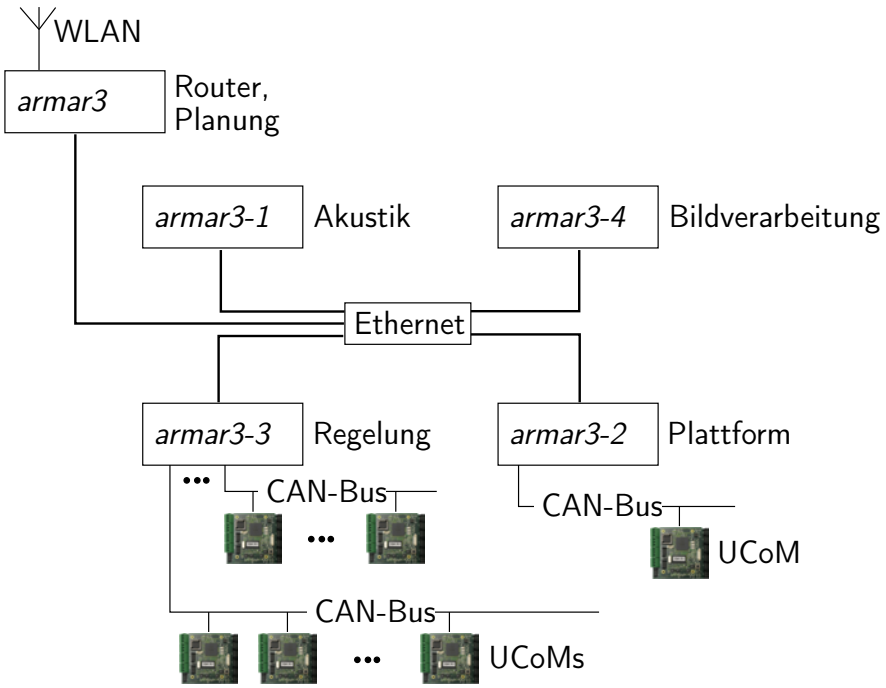


Abb. 5.8.: Struktur der verteilten Systemarchitektur auf *ARMAR-III*. Die Verteilung auf die Recheneinheiten erfolgt anforderungsbezogen und nach Domänen gruppiert. *armar3-3* und die UCoMs stellen den echtzeitfähigen Teil bereit.

CAN-Bus die Controller-Knoten mit dem Regelungs-PC/104 verbunden sind.

Durch die Kombination von DSP und FPGA auf dem UCoM lassen sich für jede Achse die quadraturkodierten Signale des Motors und der Achsencoder im FPGA auswerten, ohne dass der DSP damit belastet wird. Der DSP erhält vom FPGA auf Anfrage im Regelungstakt jeweils den aktuellen Zählerstand. Durch diese Abstraktion lassen sich auf dem FPGA auch andere Protokolle zur Positionsbestimmung implementieren. Die Analogwandlung der Potentiometer wird von einem Analog-Digitalwandler direkt am Potentiometer vorgenommen und über ein „Serial Synchronous Interface“ (SSI) an den FPGA übertragen. Auch bei diesem Sensortyp präsentiert der FPGA dem DSP im Regelungstakt jeweils den aktuellen Zählerstand. Somit ist die tatsächlich verwendete Sensorik ohne Änderung an der DSP-Software austauschbar.

Zur Regelung der Plattform wird das gleiche Grundsystem wie für die Akustik verwendet. Also die passive Busplatine PCA6105P5 mit der Steckkarten-CPU PCI-6881F ausgestattet mit einem Pentium M 1,6 GHz Prozessor und 512 MB SODIMM-Speicher. Zusätzlich verfügt dieser Rechner über eine CAN-Karte zur Verbindung mit einem UCoM, welches für die Servoregelung der drei Motoren der Plattform zuständig ist. Dieser Rechner wird als *armar3-1* bezeichnet.

Hardwarerealisierung für Planung

Die Recheneinheit, die für die Planung und für die Verbindung zur Laborumgebung zuständig ist, wird in der Plattform des Roboters verbaut. Es wird hier das gleiche Grundsystem wie für

Rechnername / (Rechnertyp)	Aufgabe	Prozessor	Speicher	HDD	Besonderheiten
armar3 (Backplane)	Planung, Router	1,6 GHz Pentium M	512 MB	60 GB	WLAN
armar3-1 (Backplane)	Akustik	1,6 GHz Pentium M	512 MB	keine	6-Kanal-Soundkarte
armar3-2 (Backplane)	Plattform-steuerung	1,6 GHz Pentium M	512 MB	keine	2 × CAN
armar3-3 (PC/104)	Regelung	933 MHz	512 MB	keine	4 × CAN AD-Karte DMM-32-AT
armar3-4 (PC/104)	Bild-verarbeitung	2 GHz Pentium M	1 GB	keine	2 × Firewire

Tab. 5.3.: Eigenschaften der in *ARMAR-III* verwendeten Recheneinheiten

die Akustik und für die Plattformsteuerung verwendet. Also die passive Busplatine PCA6105P5 mit der Steckkarten-CPU PCI-6881F ausgestattet mit einem Pentium M 1,6 GHz Prozessor und 512 MB SODIMM-Speicher. Zusätzlich verfügt dieser Rechner über eine WLAN-Steckkarte zur Kommunikation mit weiteren Rechnern in der Laborumgebung und über eine Festplatte zum Ablegen von Modellinformationen und Umweltwissen. Auf diesen Rechner wird im Folgenden als *armar3* Bezug genommen.

Auf allen PC-basierten Recheneinheiten kommt GNU/Linux zum Einsatz. Einerseits stellt der Linux-Kernel bereits eine breite Hardwareunterstützung bereit, andererseits erlaubt es die quelloffene Verfügbarkeit eigene Treibersoftware für Spezialhardware zu implementieren. Auf dem Roboter wird die Debian Distribution mit dem Releasestand Sarge verwendet. Für die auf dem Roboter verbauten Rechner erlaubt diese Distribution eine kleine Grundinstallation und bietet andererseits für die Laborrechner in der Laborumgebung einen großen Softwareumfang zur Softwareentwicklung [101]. Für die echtzeitkritischen Applikationen kommt auf den Roboterrechnern RTAI zum Einsatz. In Abb. 5.8 ist die Verbindungsstruktur der Recheneinheiten dargestellt, die Spezifikationen der *armar*-Rechner sind in Tab. 5.3 zusammengefasst. In Abb. 5.9 wird ein Gesamtüberblick über die Hardware-Software-Architektur gegeben. Hier wird kompakt dargestellt, wie sich die Architektur in die drei Bereiche Perzeption, Planung auf dem Rechnersystem und Aktion aufgliedert und über welche Kommunikationskanäle Informationen ausgetauscht werden. Für die Perzeptions- und Aktionsseite sind die Zuordnungen zu den Domänen Bildverarbeitung, Akustik und Regelung dargestellt. Die Hierarchisierung in Task-Planungs-, Task-Koordinations- und Task-Ausführungs-Ebene mit den verwendeten Recheneinheiten und der zugrundeliegenden Softwarerealisierung ist im Block „Rechnersystem“ dargestellt.

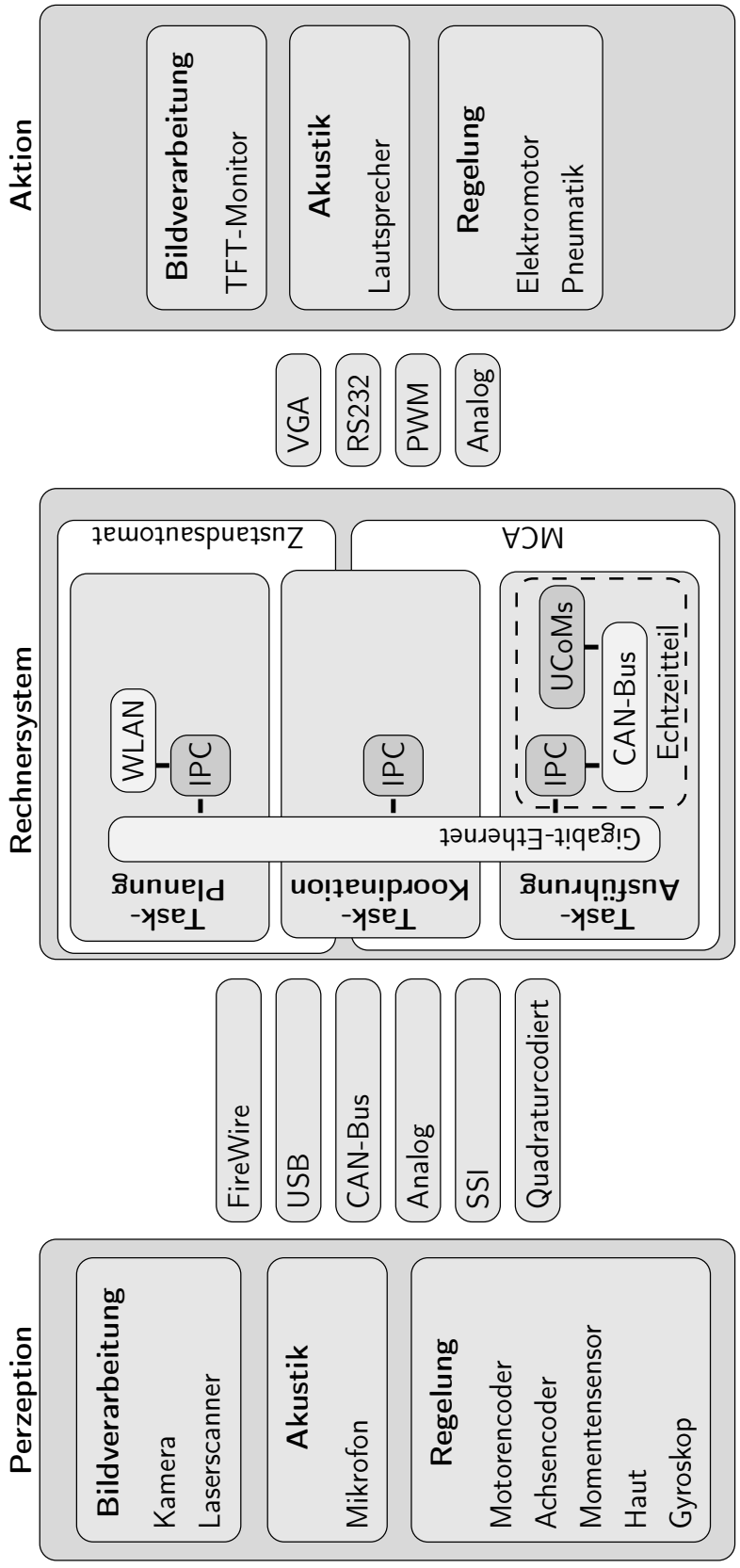


Abb. 5.9.: Übersicht über die Hardware-Software-Architektur auf *ARMAR-III*

5.2.2. Umsetzung der Steuerungsarchitektur und des Softwarerahmenwerks

Die Steuerungsarchitektur gliedert sich in die drei Ebenen Task-Planung, Task-Koordination und Task-Ausführung. Die Umsetzung der Funktionalitäten in der Task-Ausführungs-Ebene erfolgt mittels MCA2. Für die Task-Koordinations-Ebene kommt sowohl MCA2 als auch die Koordination über einen Zustandsautomaten zum Einsatz [9]. Da der Aufbau einer übergeordneten Entscheidungskomponente in dieser Arbeit nicht behandelt werden soll, wurde auf der Task-Planungs-Ebene die Nutzersteuerung zur Initiierung der Ausführung von Handlungen ausgewählt. Auch auf der Task-Planungs-Ebene wird ein Zustandsautomat zur Ausführung der Handlungen und deren Zerlegung in Tasks und Skills verwendet. Da die Handlungsplanung im engeren Sinne und die Ableitung von geeigneten Teilhandlungen ein eigenes komplexes Forschungsfeld darstellt, wurde diese im Rahmen der hier vorgestellten Hardware-Software-Architektur durch fest programmierte Szenarien ersetzt. Diese Szenarien werden durch Interaktion mit dem Nutzer initiiert. Ein Szenario ist ein komplexer Handlungsablauf, der aus einer Vielzahl einzelner Aktionen besteht. Szenarien bilden somit den Rahmen für die Ausführung unterschiedlich komplexer Aufgaben und eignen sich dazu die Fähigkeiten des Roboters in verschiedenen Kontexten zu präsentieren [10]. Die Ausführung der Szenarien beruht auf einer festgelegten Unterteilung in Tasks, die in diesem Szenario abgearbeitet werden. Diese Tasks greifen wiederum auf die elementaren Roboterfähigkeiten, die Skills, zu. Die Abarbeitung der Tasks und Skills wird mit einem Zustandsautomaten kontrolliert. Die Skills werden zyklisch aufgerufen und melden dabei ihren Zustand. Mögliche Zustände sind: *operating*, *success*, *failure* oder *waiting*. Skills greifen ihrerseits über das MCA-Rahmenwerk auf die tatsächlichen Roboterressourcen zu.

Auf Sensorikseite wird zwischen Sensorwerten, die für die Echtzeitregelung der Bewegungsfreiheitsgrade benötigt werden und Sensorwerten, für die keine Echtzeitanforderungen gelten unterschieden. In die erste Gruppe fallen zum Beispiel Positions-, Geschwindigkeits- oder Drehmomentsensoren. Diese werden direkt auf der Task-Ausführungs-Ebene von den UCoMs ausgewertet und sowohl direkt für die Sensor-Motor-Regelung verwendet als auch an die Task-Koordinations- und Task-Planungs-Ebene übermittelt. In die andere Gruppe fallen Sensoren, wie zum Beispiel Kameras oder Mikrofone. Zur Auswertung der Kamerabilder wird das „Integrating Vision Toolkit“ (IVT[12]) verwendet. Das IVT ist eine plattformunabhängige Bildverarbeitungsbibliothek mit einer objektorientierten Architektur. Sie bietet eine klar definierte Kameraschnittstelle, ein Kameramodell sowie eine Vielzahl von schnellen Bildbearbeitungsalgorithmen und mathematischen Funktionen. Darüber hinaus stellt es für viele Plattformen eine grafische Benutzerschnittstelle bereit [15].

Durch die modulare Untergliederung der Szenarien in Tasks und Skills, lassen sich auf einfache Weise verschiedene Szenarien erzeugen. Aktuell wurden für *ARMAR-III* 25 Skills imple-

[12] `http://ivt.sourceforge.net`

mentiert, die bei der Ausführung der 15 bisher möglichen Tasks benötigt werden. Mit diesen Tasks und Skills wurde bereits eine Vielzahl von Szenarien auf dem Roboter durchgeführt. In Abschnitt 5.3 wird ein komplexes Szenario herausgegriffen und näher beschrieben. An dieser Stelle wird anhand der Auflistung einiger repräsentativer Tasks und Skills ein Überblick über die Fähigkeiten des humanoiden Roboters *ARMAR-III* gegeben:

Skill „Objekt Suchen“ Dieser Skill stellt die Fähigkeit zur Verfügung mit dem aktiven Kopf Objekte in der Umwelt zu suchen. Dazu wird durch Bewegen des Kopfes das Blickfeld vor dem Roboter nach bekannten Objekten durchsucht. Dabei kann die Suche sowohl alle Objekte zurückmelden, die entdeckt werden als auch auf einen bestimmten Objekttyp eingeschränkt werden [14].

Skill „Bildbasiertes Greifen“ Mit Hilfe von *visual servoing* ermöglicht dieser Skill das Greifen von Objekten, deren Position im Raum bekannt ist. Für die Ausführung des Griffs wird die Plattform in eine günstige Position bewegt und die eigentliche Greifbewegung unter zu Hilfenahme der Hüfte mit einem Arm ausgeführt. Während der Greifbewegung wird über einen Marker am Handgelenk die Annäherung an das Objekt im Regelkreis überwacht [153].

Skill „Objekte platzieren“ Dieser Skill dient dazu gegriffene Objekte auf ebenen Flächen abzustellen. Dazu wird einerseits die Position der Abstellfläche visuell analysiert und andererseits durch einen 6D-Kraft-Momenten-Sensor im Handgelenk erkannt, wann das Objekt die Abstellfläche berührt.

Skill „Objekte übergeben“ Mit diesem Skill wird der Roboter in die Lage versetzt, Objekte an einen Menschen oder anderen Roboter zu überreichen. Wie der Skill zum Platzieren von Objekten stützt sich dieser Skill auf visuelle Rückmeldung und Informationen aus dem 6D-Kraft-Momenten-Sensor im Handgelenk.

Skill „Tür Öffnen“ Dieser Skill versetzt den Roboter in die Lage, unterschiedliche Türen, wie zum Beispiel eine Kühlschrank- oder Schranktür, zu öffnen. Das Öffnen der Türen erfolgt positions- und kraftgeregelt.

Skill „Tür Schließen“ Dieser Skill stellt die Fähigkeit bereit, geöffnete Türen zu schließen. Zum Schließen der Tür wird die Tür an einer vordefinierten Stelle berührt und positions- und kraftgeregelt geschlossen.

Mit diesen Skills wurden beispielsweise die folgenden Tasks realisiert:

Task „Tisch abräumen“ Hier soll der Roboter alle Objekte, die sich auf dem Tisch befinden, entfernen. Dazu muss der Roboter die Objekte bildbasiert greifen, transportieren und an einer anderen Abstellfläche abstellen.

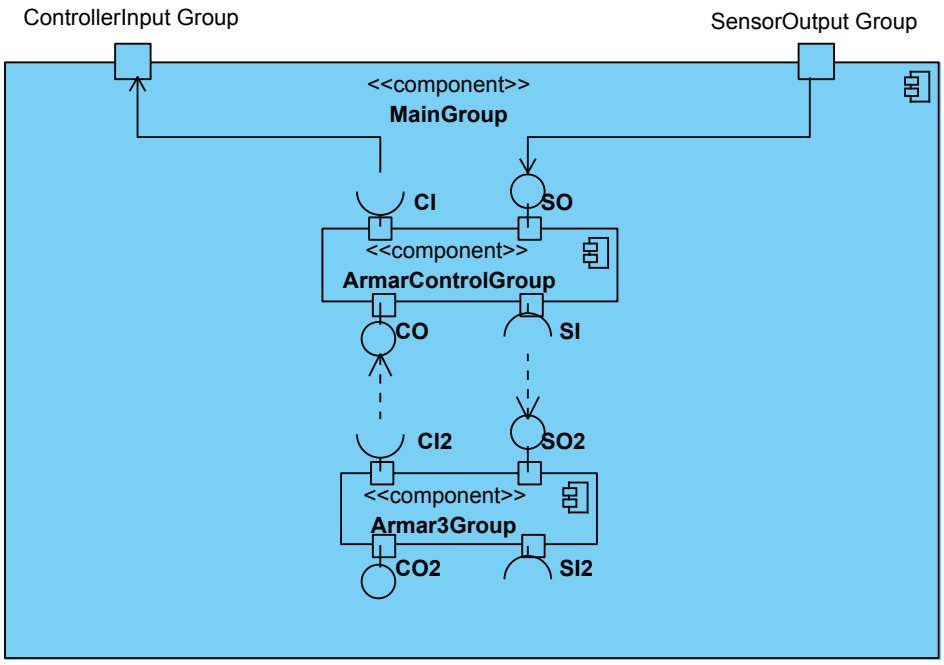


Abb. 5.10.: Darstellung der `MainGroup` in UML

Task „Objekt bringen“ Dieser Task dient dazu dem Benutzer ein Objekt zu überbringen. Dabei muss der Roboter das geforderte Objekt lokalisieren, bildbasiert greifen und im Anschluss den Benutzer lokalisieren und das Objekt übergeben.

Task „Objekt in Kühlschrank stellen“ Bei diesem Task geht es darum ein bestimmtes Objekt in den Kühlschrank zu stellen. Hierfür muss der Roboter das Objekt lokalisieren und bildbasiert greifen, die Kühlschranktür positions- und kraftgeregelt öffnen, das Objekt im Kühlschrank platzieren und schließlich die Kühlschranktür schließen.

Die eigentliche Funktionalität ist mit Hilfe des modularen Softwarerahmenwerks MCA2 aufgebaut. Anhand der Aufgabe der Zwei-Arm-Manipulation wird im Folgenden erläutert, wie die Aufteilung in Module vorgenommen wurde. Die oberste MCA-Ebene, die `MainGroup`, beinhaltet in der `ArmarControlGroup` die Anbindung an den Zustandsautomaten zur Abarbeitung des Szenarios (Abb. 5.10). Die Gruppe `Armar3Group` (Abb. 5.11), die in der Hierarchie unterhalb der `ArmarControlGroup` angeordnet ist, beinhaltet die tatsächliche Steuerung des Roboters. Diese lässt sich durch das von MCA2 unterstützte in einander Verschachteln von Modulen und Gruppen strukturiert aufbauen. In dieser Gruppe sind die folgenden MCA-Gruppen hierarchisch angeordnet: die `DualForceGroup` zur kraftgeregelten Manipulation mit zwei Armen, die `KinematicsGroup` zur Kinematikberechnung, die `CollisionGroup` zur Erkennung und Vermeidung von Kollisionen und die `ExecutionGroup` zur Kommunikation mit der Regelungskomponente UCoM. Von diesen MCA-Gruppen wird die Untergliederung der `KinematicsGroup` am Beispiel des rechten Arms näher beschrieben. Die `KinematicsGroup`

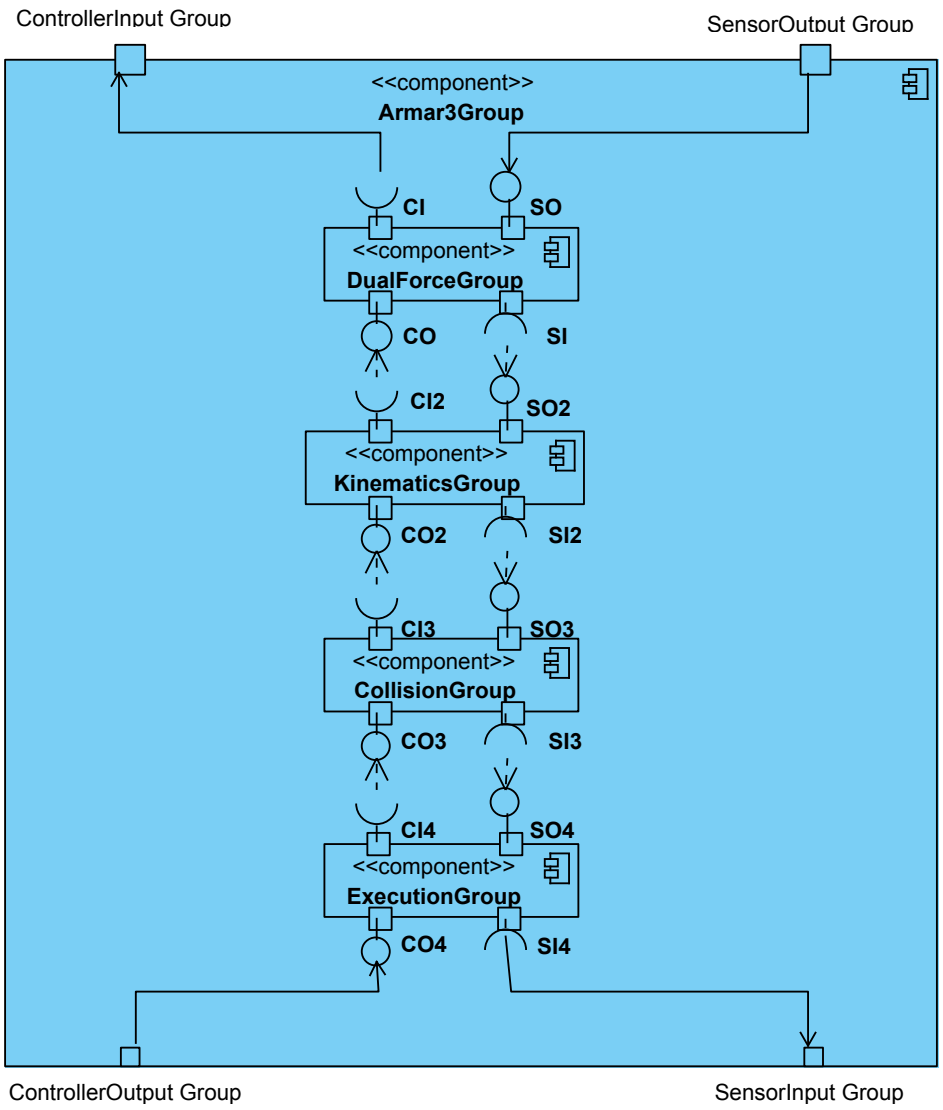


Abb. 5.11.: Darstellung der `Armar3Group` in UML

teilt sich in Gruppen für die Teilsysteme des Roboters auf (Abb. 5.12). So gibt es die Gruppen `DualArmGroup` für die beiden Arme, die `HipGroup` für das Hüftgelenk und die `HeadGroup` für den Hals und den Kopf des Roboters. An dieser Stelle wird der Vorteil der Modularisierung am Beispiel der `DualArmGroup` (Abb. 5.13) deutlich. Die `DualArmGroup` setzt sich aus zwei fast identischen Gruppen für den rechten und linken Arm zusammen. Viele Funktionalitäten zur Ansteuerung eines Armes sind für rechten und linken Arm identisch und können somit in beiden Gruppen wiederverwendet werden. Die MCA-Gruppe `Right Arm` (Abb. 5.14) setzt sich aus der `ArmKinematicsGroup` zur Berechnung der Armkinematik und der `HandGroup` zur Ansteuerung der Fünf-Finger-Hand zusammen. In der `ArmKinematicsGroup` (Abb. 5.15) werden schließlich sowohl die Module zur Vorwärts- als auch zur inversen Kinematik implementiert. Darüber hinaus werden Gelenkwinkel- und kartesische Interpolation vorgenommen. In der Sicht aus Abb. 5.11 werden die in dieser Gruppe generierten Bewegungen an die `CollisionGroup`

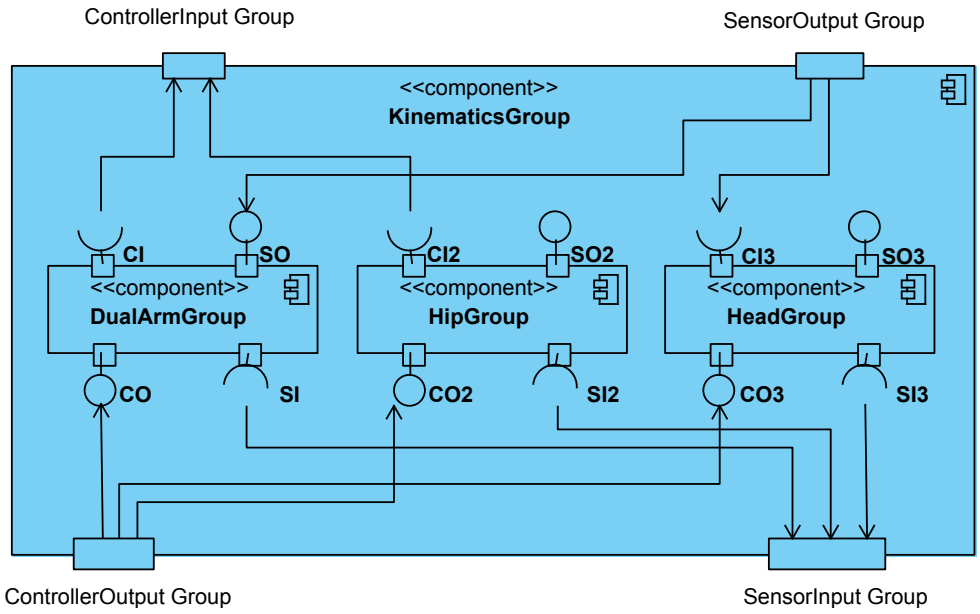


Abb. 5.12.: Darstellung der `KinematicsGroup` in UML

weitergeleitet und dort auf Kollisionsfreiheit überprüft und dann an die `ExecutionGroup` übermittelt. Diese ist für die Kommunikation mit den Echtzeitregelungskomponenten über CAN-Bus zuständig. Die UCoMs binden sich am CAN-Bus unterhalb der `ExecutionGroup` wie in Abb. 4.8(a) dargestellt wie ein MCA-Modul ein.

Motorregelung auf dem UCoM

Auf der Servo-Motor-Regelungsebene bindet sich das UCoM, genau wie ein MCA-Modul in die Softwarestruktur ein. Dabei werden vom UCoM über die CAN-Schnittstelle bereits abstrahierte Sensordaten an den Regelungs-PC übertragen. Die eigentliche Gelenkregelung und auch die Erfassung der dafür nötigen Sensorwerte erfolgt lokal auf dem UCoM mit einer Frequenz von 1 kHz. Die Vorgaben für die lokale Gelenkregelung erhält das UCoM vergleichbar zur Kommunikation der Sensorwerte in abstrahierter Form über den CAN-Bus.

Auf dem UCoM wird das Programm *armar3.fl.S* ausgeführt. Dieses Programm implementiert einen kaskadierten Geschwindigkeits-Positionsregler für den sowohl für den Geschwindigkeits- als auch den Positionsteil der Regelung die PID-Parameter eingestellt werden können. Um auf allen UCoMs das gleiche Anwendungsprogramm einsetzen zu können, muss dieses beim Programmstart über weite Bereiche konfiguriert werden. Diese Konfiguration erfolgt durch den in Anhang B.5 beschriebenen *Attribute-Tree*. In diesem werden einerseits für den Bewegungsfreiheitsgrad spezifische Einstellungen wie zum Beispiel die PID-Parameter, Maximalgeschwindigkeiten oder Bewegungsbereiche festgelegt. Andererseits erfolgt anhand einer Übersetzungstabelle eine Anpassung der Kantennamen, die zur Kommunikation mit dem UCoM verwendet

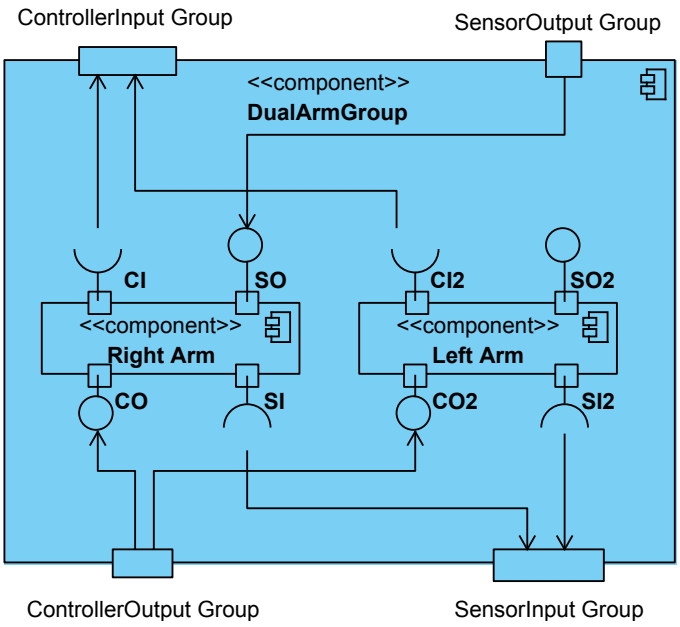


Abb. 5.13.: Darstellung der `DualArmGroup` in UML

werden. Da das Programm *armar3.fl.S* komplett generisch ist, werden auf dem UCoM intern die drei Bewegungsfreiheitsgrade mit `Motor A`, `Motor B` und `Motor C` benannt. Mit Hilfe des *Attribute-Tree* wird so beispielsweise die Zielposition der ersten Achse, `Motor A Target`, in `Right Eye Target` übersetzt, so dass auf Seiten des Regelungs-PCs eine klare Zuordnung möglich wird.

5.3. Evaluation anhand der Küchen-Demo

Die vorgestellte und auf *ARMAR-III* umgesetzte Hardware-Software-Architektur soll anhand eines realistischen Einsatzszenarios des humanoiden Roboters evaluiert werden. Als Szenario wurde die so genannte Küchendemo ausgewählt. In der Küchenumgebung lassen sich die geforderten Fähigkeiten des Roboters sehr gut in einer geschlossenen Demonstration zeigen. Die Anwendung in der Küche stellt hohe Anforderungen an das Gesamtsystem. Die Aufgaben wurden so gewählt, dass zur Ausführung sämtliche Teilsysteme benötigt werden. Unter anderem beinhaltet die Demonstration folgende Teilaspekte

- Regelung des komplexen Gesamtsystems in Echtzeit – Koordinierte Bewegungen der Teilsysteme Plattform, Torso, Arme, Hand, Kopf und Auge
- Interaktion und Kommandierung über Sprache
- Verstehen einer gestellten Aufgabe und Zerlegung in Teilaufgaben, die dann sequenziell oder parallel ausgeführt werden

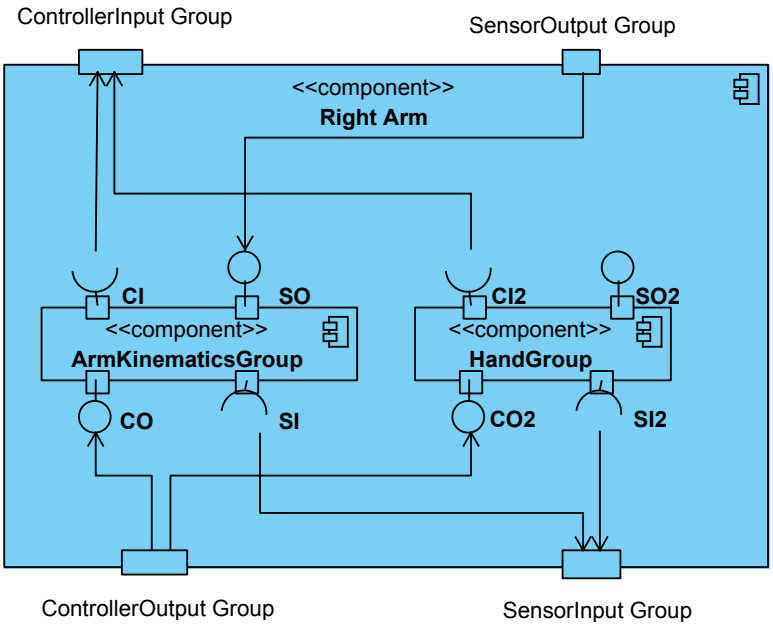


Abb. 5.14.: Darstellung der `RightArmGroup` in UML - die entsprechende Gruppe existiert ebenfalls für den linken Arm

- Visuelle Objekterkennung und Greifen über Hand-Auge-Koordination
- Visuelles Lernen von unbekannten Objekten
- Visuelle Erkennung und Verfolgung des Nutzers
- Ortung von Sprecher und Geräuschquellen
- Navigation und Kollisionsvermeidung in der Küchenumgebung

In der Evaluationsdemonstration werden typische Mensch-Roboter-Interaktionen in Alltagsumgebungen sowie die Fähigkeiten des Roboters im Umgang mit üblichen Geräten in der Küche wie Geschirrspülmaschine oder Kühlschrank demonstriert. Im Evaluationsszenario soll die Interaktion mit dem Menschen nicht über Tastatureingriffe erfolgen, sondern ausschließlich über sprachliche Kommunikation. Im Rahmen dieser Demonstration wird die Eignung der Hardware-Software-Architektur zur Ausführung komplexer Handhabungsaufgaben in Alltagsumgebungen, zur Perzeption, zur Interaktion und zur Integration der Teilaspekte zu einem autonomen Gesamtsystem deutlich.

5.3.1. Ablauf der Küchen-Demo

Im Rahmen der Demonstration soll der Roboter für Benutzer sowohl einfarbige als auch texturierte Objekte von einer entfernten Arbeitsfläche herbeiholen und diese anschließend dem Benutzer überreichen. Zur Initiation dieser Handlung führt der Benutzer mit dem Roboter einen

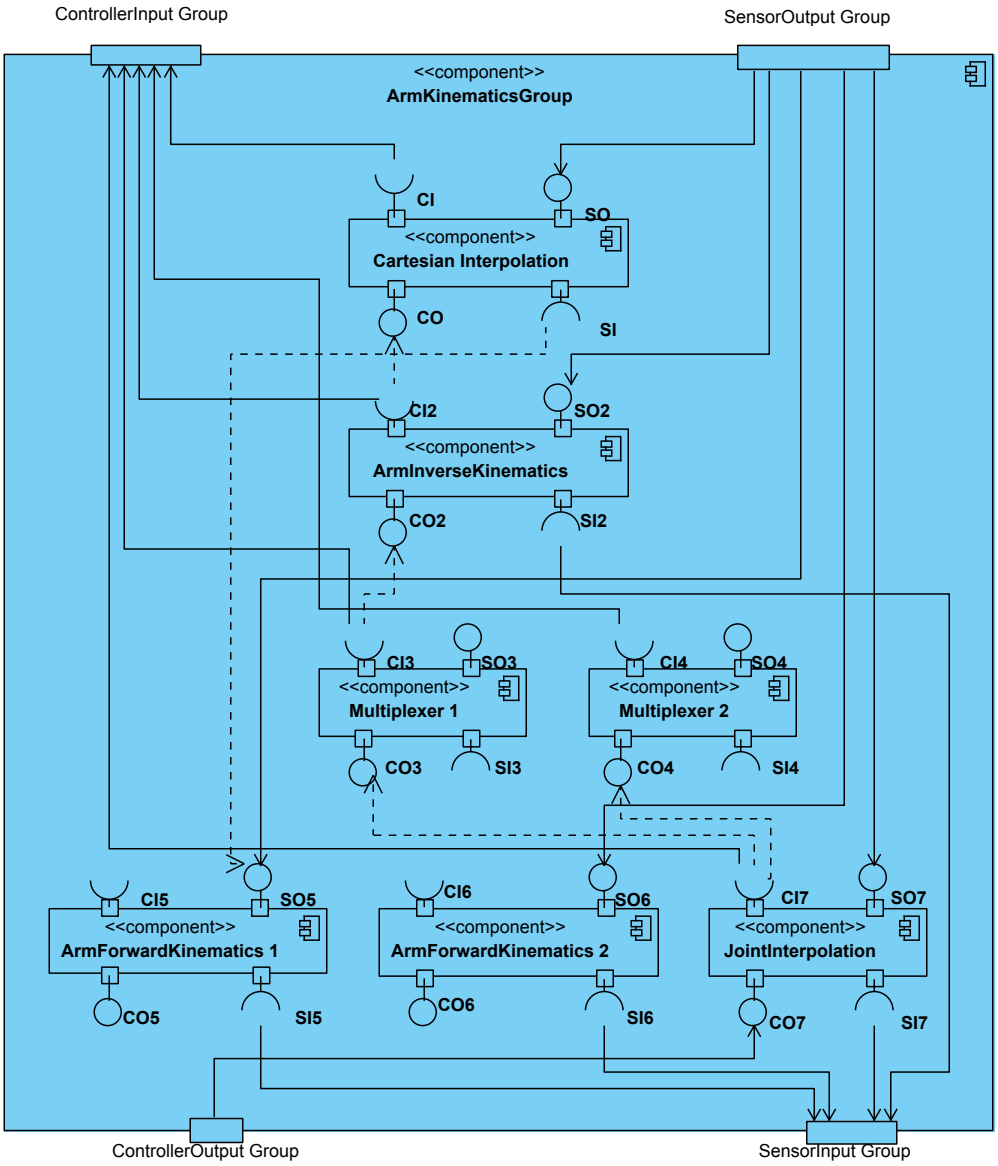


Abb. 5.15.: Darstellung der `ArmKinematicsGroup` in UML

Dialog, in dem er dem Roboter die Handlungsanweisung gibt und in einem Klärungsdialog weitere Informationen zum zu bringenden Objekt übermittelt. So kann der Benutzer dem Roboter den Objekttyp, die Objektposition und die Farbe über Sprachinteraktion mitteilen. Uneindeutigkeiten können über weitere Dialoge oder auch durch Zeigegesten aufgelöst werden

Anschließend fährt der Roboter zur bezeichneten Stelle. Dazu nutzt er das Wissen über die Küchenabmessungen und Hindernisse bzw. bekannte Objekte, die über das Küchenmodell und den Szenengraph modelliert sind (Abb. 5.16). Der Roboter ist dabei in der Lage, die genaue Position der Objekte an der angegebenen Stelle zu lokalisieren und die Objekte zu erkennen. Mit den so gewonnenen Umweltinformationen wird das Szenenmodell aktualisiert. Nach der Navigation zur Objektposition ermittelt der Roboter eine günstige Position für das Greifen des

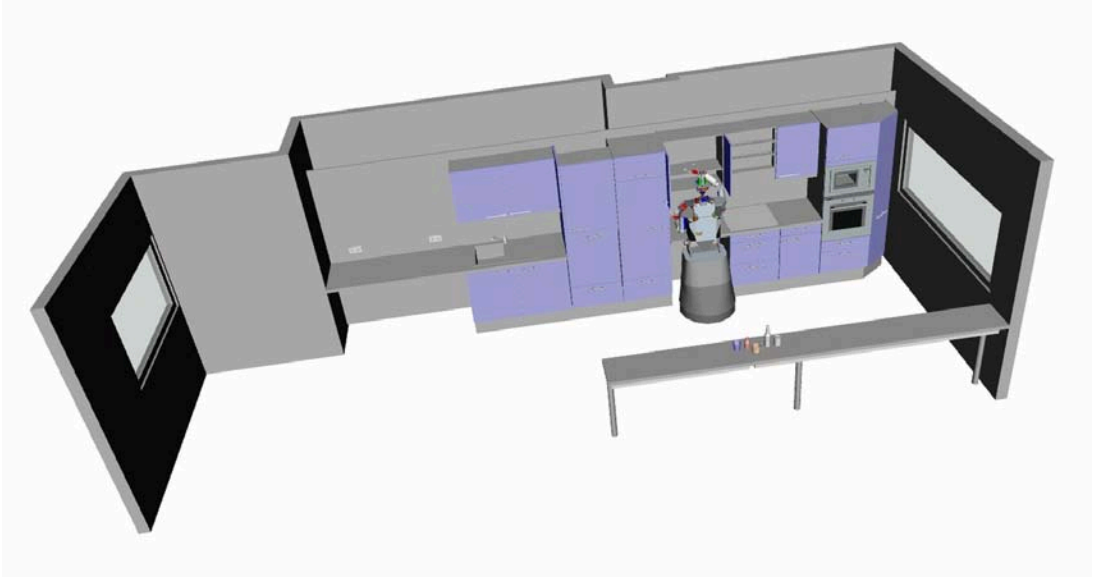


Abb. 5.16.: Modell der Küchenumgebung und Szenengraph, die für die Evaluierung der Roboterfähigkeiten im Rahmen der Küchendemo verwendet werden

Objektes und nimmt diese ein. Das eigentliche Greifen erfolgt mittels *visual servoing*. Nach erfolgtem Griff fährt der Roboter zum Benutzer, dessen letzte Position auch im Szenenmodell abgelegt wurde. Befindet sich der Nutzer nicht mehr an seiner ursprünglichen Position, kann der Roboter ihn visuell bzw. akustisch orten und die neue Position ansteuern.

Die zweite prototypische Handlung in der Küchenumgebung ist das Öffnen des Kühlschranks und das Greifen eines Gegenstandes im Kühlschrank. Auch diese Handlung kann über sprachliche Interaktion angestoßen werden. Nach dem Handlungsbefehl fährt der Roboter eigenständig zum Kühlschrank, dessen Position er aus dem Szenenmodell kennt. Der Roboter stellt fest, ob die Kühlschranktür bereits geöffnet ist und öffnet diese für den Fall, dass sie geschlossen ist. Das Öffnen der Kühlschranktür stellt im Vergleich zum Griff des Bechers von der Arbeitsfläche eine komplexere Aufgabe dar, da die Bewegung zum Öffnen Kraft-Momenten-geregelt auf einer Zwangsbahn erfolgt. Nach dem Öffnen der Kühlschranktür ist der Roboter in der Lage, die Objekte im Kühlschrank zu erkennen und dem Benutzer den Inhalt des Kühlschranks mitzuteilen. Auf Anforderungen kann er ein Objekt aus dem Kühlfach greifen und dem Benutzer reichen.

Als weitere typische Aufgabe für einen Assistenzroboter in der Küchenumgebung wurde das Einräumen von Gegenständen in die Geschirrspülmaschine ausgewählt. Auch für diese Aufgabe wird das Kommando über Sprache gegeben und der Roboter fährt daraufhin die Geschirrspülmaschine an, öffnet die Tür und zieht den Korb nach vorne. Die Objekte, die in die Geschirrspülmaschine geräumt werden sollen, lässt sich der Roboter vom Benutzer anreichen, sucht einen freien Platz im Spülkorb und stellt die Objekte dort ab. Nach Abschluss des Einräumens schließt der Roboter die Spülmaschine.

Bei Dialogen behält der Roboter den Kopf des Benutzers mit seinen Augen im Blickfeld und ist in der Lage den Nutzer mittels Gesichtsidentifikation zu erkennen. Unbekannte Personen können dem Roboter vorgestellt werden und werden im Anschluss auch vom Roboter erkannt. Objekte die der Roboter noch nicht in seinem Szenenmodell abgelegt hat, können ansichtsbasiert und sprachlich kommentiert eingelernt werden.

Im Rahmen der Küchendemo konnte gezeigt werden, dass durch die vorgeschlagene und umgesetzte Hardware-Software-Architektur eine robuste und reproduzierbare Ausführung dieses komplexen Demonstrationsszenarios ermöglicht wird. Insbesondere konnten die im Abschnitt 5.3 benannten Teilaspekte erfolgreich durchgeführt werden.

5.4. Leistungsbewertung der Regelungskomponente

Durch die Entwicklung des UCoMs konnte gegenüber den in *ARMAR-I* verwendeten C167-Minimodulen eine Leistungssteigerung auf der Motor-Servoregelungs-Ebene um einen Faktor von etwa zehn erreicht werden. Durch das UCoM sind auf dieser Ebene nun Regelungszyklen von unter 1 ms möglich. Dank dieser Verkürzung der Zykluszeiten wird eine Verbesserung der Regelungsgüte erzielt und somit die Realisierung menschenähnlicher Bewegungsabläufe ermöglicht. Im Folgenden wird anhand von Benchmarks und konkreten Regelungsalgorithmen die Leistungsfähigkeit der beiden Komponenten bewertet. Im Anhang C werden weitere Einzelheiten zu den beiden Regelungskomponenten beschrieben.

5.4.1. Benchmarks zur Leistungsbestimmung

Der augenfälligste Unterschied beim Leistungsvergleich zwischen DSP56F803 und C167 ist die Taktzykluszeit. Während der C167 für einen Takt 80 ns benötigt, braucht der DSP nur 25 ns. Allein durch diesen Geschwindigkeitsvorteil, sollte der DSP Aufgaben circa dreimal so schnell abarbeiten können wie der C167. Dabei unbeachtet bleiben vorerst noch die Vorteile, die der DSP durch seine effizientere Architektur hat. Bei reinen Additionsaufgaben kommt dieser Vorteil noch nicht zum Tragen, da der C167 diese auch in einem Takt abarbeiten kann. Bei Multiplikationen benötigt der C167 allerdings 5 Takte, wo der DSP nur einen Takt benötigt. Bei der Division ist dieser Unterschied noch deutlicher, hier liegt das Verhältnis der benötigten Taktzyklen bei zehn zu eins. Bei den häufig auftretenden Sprüngen bzw. Move-Befehlen ist der DSP um den Faktor zwei schneller, solange die Daten bereits in Registern vorliegen. Ist dies nicht der Fall kann der DSP durch eine höhere Geschwindigkeit bei der Adressberechnung und den Speicherzugriffen seinen Geschwindigkeitsvorteil weiter ausbauen. Wie an diesen Zahlenbeispielen zu sehen ist, hängt der tatsächliche Geschwindigkeitsvorteil durch die Verwendung des DSPs stark von den im Anwendungsprogramm benötigten Operationen ab. Um eine realistische Abschätzung zu erhalten, wurden verschiedene Standardbenchmarks durchgeführt und au-

ßerdem die Ausführung eines tatsächlich bisher verwendeten einfachen Regelungsprogramms verglichen. Als Standardbenchmarks und Regelungsalgorithmen wurden folgende Programme ausgewählt:

- Whetstone
- Proportional-Integral-Differential-Regler (PID-Regler)
- Ackermann
- Sieb des Eratosthenes
- Fibonacci

Die ersten drei werden in diesem Abschnitt behandelt, die letzten drei werden zusammen mit einer detaillierten Beschreibung der beiden Regelungskomponenten im Anhang C aufgeführt.

Die Standardbenchmarks dienen einer besseren Vergleichbarkeit mit anderen Controller-Boards. Anhand des Vergleichs bei Regelungsaufgaben, kann gezeigt werden, welchen Vorteil der DSP für die Anwendung im Roboter bietet und ob dieser sogar den Vorteil bei den Standardbenchmarks übersteigt. Für den DSP wurde für jeden der hier aufgezählten Benchmarktypen jeweils anhand des Assemblercodes die theoretische Ausführungsdauer ermittelt, für den C167 konnte die theoretische Betrachtung nicht durchgeführt werden. Danach wurde der reale Leistungsvergleich der beiden Komponenten mittels Zeitmessungen bei der Erledigung der Benchmarkaufgaben durchgeführt. Hierzu musste beim C167 ein separater Zähler verwendet werden, beim DSP konnte der Zähler, der die Controller über die MCA-Schleife kontrolliert, genutzt werden. Das genaue Verfahren ist in den nächsten beiden Absätzen beschrieben.

Vorgehen zur Zeitmessung beim C167

Beim C167 wurde die Zeitmessung durch einen separaten Timer gestartet, dessen Zählerstand nach Abarbeitung der Aufgabe ausgewertet wurde. Der Zusammenhang zwischen Timerticks und vergangener Zeit ist beim C167 nicht fest vorgegeben, sondern kann über das Register „Bits Input Selection“ (T3I) festgelegt werden. Nach folgender Formel errechnet sich der Zusammenhang zwischen Zählerstand und verstrichener Zeit:

$$t_{Tick} = \frac{8 \cdot 2^{\mathrm{T3I}}}{f_{\mathrm{CPU}}}$$

Abhängig von den zu erwartenden Zeiten wurde die Auflösung festgelegt, um zu vermeiden, dass Zählerüberläufe auftreten. Zusätzlich wurde versucht, um möglichst eine Eins-zu-Eins-Vergleichbarkeit zu gewährleisten, beim C167 und beim DSP die gleiche Zeitauflösung zu verwenden. Beim C167 konnte die theoretische Laufzeitbestimmung nicht durchgeführt werden.

Dies liegt daran, dass die Entwicklungsumgebung des C167 nicht die Möglichkeit bietet, das Kompilat des C-Codes in Assembler zu betrachten. Eine Abschätzung, wie der C-Code in Assembler durch den Compiler realisiert wird, birgt zuviele Unwägbarkeiten, wie zum Beispiel fehlendes Wissen darüber, wo genau die Daten im Prozessor abgelegt werden und wie viele NOP-Operationen zur Vermeidung von Pipelinekonflikten eingefügt werden mussten. Da somit eine theoretische Behandlung trotz großen Aufwandes nur zu ungenauen Ergebnissen führen würde, wurde diese Abschätzung beim C167 unterlassen.

Vorgehen zur Zeitmessung beim DSP

Beim DSP konnte zur Bestimmung der Durchlaufzeit des Benchmarks der für den MCA-Zyklus zuständige Zähler verwendet werden. Um die Dauer der Abarbeitung des Benchmarks bzw. des Regelungsprogramms zu ermitteln, wurde der Zählerstand des MCA-Zyklus bei Eintritt in das Benchmark ausgelesen und vom Zählerstand am Ende des Benchmarks subtrahiert. Da die Entwicklungsumgebung des DSPs unterschiedliche Einstellungen für Code-Optimierung bietet wurden die Benchmarkprogramme mit jeder der möglichen Einstellungen compiliert und dann die Messung durchgeführt. Die möglichen Einstellungen für diese „Global Optimization“ genannte Option sind:

1. Keine Optimierung
2. Level 1: *Common Subexpression Elimination*, *Arithmetic Transformations*, *Dead Code Elimination*, *Constant Folding*, *Removal of Unreachable Code*, *Jump Chaining*
3. Level 2: Wie Level 1, zusätzlich *Copy Propagation*, *Constant Propagation*
4. Level 3: Wie Level 2, zusätzlich *Loop-Invariant Code Motion*, *Strength Reduction*, *Loop Unrolling*, *Lifetime Analysis*
5. Level 4: Wiederholte Level 3 Optimierungen

Bei den Ergebnissen wird jeweils das genutzte Optimierungslevel mit angegeben; Ergebnisse ohne Angabe des Optimierungslevels wurde mit „Keine Optimierung“ erzeugt.

Die Ermittlung der Zeit aus den gemessenen Zählerschritten ergibt sich beim DSP aus folgenden Punkten. Die verwendete Taktfrequenz beträgt 80 MHz und somit errechnet sich die Taktzykluszeit zu 12,5 ns. Diese Taktung kann jedoch nur im Kern des DSP realisiert werden und der IP-Bus, der für die Zähler zuständig ist läuft maximal mit einer Frequenz von 40 MHz, somit ist der minimale Takt des Zählers 25 ns. Zusätzlich musste beim Zähler für den MCA-Zyklus noch ein so genannter *Prescaler*, der nur jedes 2^n te Ereignis zählt, eingesetzt werden, um die gewünschten Zykluszeiten der MCA-Schleife zu realisieren. Mit dem für den *Prescaler* eingestellten Wert von 128 liegt die Zeit für einen Zählertick bei $3,2\,\mu s$.

Im Gegensatz zum C167 war beim DSP dank der Möglichkeiten der Entwicklungsumgebung eine Zuordnung des C-Codes zum erzeugten Assembler-Code möglich (Anhang C.4). Somit wurde auch für die Standard-Benchmarks, das heißt für die Ackermann-Funktion, das Sieb des Eratosthenes und für die Berechnung der Fibonaccizahlen, im Vorfeld die theoretische Laufzeit bestimmt (Anhang C.3). Anhand des Assembler-Codes konnte der Beginn und das Ende des Benchmark-Codes gut identifiziert und auch sämtliche Sprünge und Schleifen erkannt werden. Damit konnte die Ausführungshäufigkeit der einzelnen Operationen bestimmt werden und anhand des DSP-Family-Manuals [54] die Ausführungsdauer ermittelt werden. Hierbei wurde zusätzlich beachtet, dass die Ausführungsdauer eines Befehles von den Operanden abhängig ist. Bei unbekannten Operanden wurde die niedrigere Zeit angenommen. Die errechneten Laufzeiten werden dann in den Abschnitten zu den jeweiligen Benchmarks aufgeführt.

Beurteilung mittels Whetstone-Benchmark

Der Whetstone-Benchmark vereint mehrere Module, die jeweils unterschiedliche Aspekte des Prozessors auslasten. Die Ausführungsanzahl, der einzelnen Module ist dabei so gewählt, dass die Verteilung der Aufgaben einer realistischen Nutzung möglichst nahe kommt. Beispielsweise gibt es Module, die auf Integer-Arithmetik, Fließkomma-Operationen oder Feldzugriffe testen. Folgende Module wurden ausgeführt:

Modul 2 Berechnung mit Feldelementen

Modul 3 Übergabe eines Arrays als Parameter

Modul 4 Ausführung bedingter Sprünge

Modul 6 Arithmetik mit Integerzahlen

Modul 7 Ausführung trigonometrischer Funktionen

Modul 8 Prozeduraufrufe

Modul 9 Feldreferenzen und parameterlose Prozeduraufrufe

Gemessene Laufzeit beim DSP

Die in Tab. 5.4 aufgelisteten Module wurden in der angegebenen Verteilung zehnmal hintereinander ausgeführt, dabei ergaben sich die ebenfalls mit aufgeführten Taktzyklen und Laufzeiten.

Um einen Zählerüberlauf zu vermeiden wurden die Taktzyklen nach jedem Modul ermittelt und zum Schluss aufsummiert. Die Teilergebnisse für die einzelnen Module können Tab. 5.4 entnommen werden. Insgesamt benötigt die Ausführung 243 981 Taktzyklen, also 780,739 ms.

Modulnummer	Wiederholungen	Taktzyklen	Laufzeit
Modul 2	12	346	1,1072 ms
Modul 3	14	2 496	7,9872 ms
Modul 4	345	119	0,3808 ms
Modul 6	210	674	2,1568 ms
Modul 7	32	10 553	33,7696 ms
Modul 8	899	9 825	31,44 ms
Modul 9	616	385	1,232 ms

Tab. 5.4.: Ausführungshäufigkeit der Module und ermittelte Laufzeiten beim DSP

Modulnummer	Taktzyklen in Optimierungsstufe				
	Ohne	Level 1	Level 2	Level 3	Level 4
Modul 2	346	346	346	346	346
Modul 3	2 496	2 496	2 496	2496	2496
Modul 4	119	118	119	119	119
Modul 6	674	678	678	678	678
Modul 7	10 553	10 553	10 552	10 552	10 552
Modul 8	9 825	9 825	9 825	9 825	9 825
Modul 9	385	385	357	357	357

Tab. 5.5.: Laufzeiten des Whetstone-Benchmarks in verschiedenen Optimierungsstufen

Die Differenz zur Aufsummierung der Zeiten für die Einzelmodule resultiert aus der zehnfachen Ausführung und dem Overhead der Schleife.

Wie in Tab. 5.5 zu erkennen ist, bringen die Optimierungsstufen bei den meisten Modulen keine Verbesserung. Eine erkennbare Veränderung gab es nur bei Modul 9, was sich dann auch durch die mehrfache Ausführung in der Gesamtausführungszeit auswirkte.

Gemessene Laufzeit beim C167

Beim C167 wurde bei dieser Messung ebenfalls wieder die Zeitauflösung 3,2 μs gewählt. Dabei ergaben sich die in Tab. 5.6 zusammengefassten Werte.

Für die im Gesamten zehn durchgeführten Durchläufe benötigte der C167 inklusive des Schleifen-Overheads 2 138 492 Taktzyklen, also 6,84317 s.

Modulnummer	Wiederholungen	Taktzyklen	Laufzeit
Modul 2	12	4 953	15,8496 ms
Modul 3	14	42 172	134,9504 ms
Modul 4	345	119	4,704 ms
Modul 6	210	7 540	24,128 ms
Modul 7	32	46 689	149,4048 ms
Modul 8	899	9 825	339,856 ms
Modul 9	616	4 818	15,4176 ms

Tab. 5.6.: Ausführungshäufigkeit der Module und ermittelte Laufzeiten beim C167

Leistungsbestimmung bei Regelungsprozessen

Für den konkreten Anwendungsfall im Roboter, also für die Ausführung von Regelungsprozessen, wurden prototypisch die Ausführungszeiten eines PID-Reglerprogramms auf den beiden Komponenten untersucht. Anhand dieser Prozesse lässt sich bewerten, welchen Vorteil die Verwendung des DSPs bei den im Roboter auftretenden Regelungsproblemen bietet. Dabei wurden die folgenden drei Fälle bei der Betrachtung unterschieden:

- Regler inaktiv (PID $= 0$, Sollwert $=$ Istwert)
- Regler aktiv ohne Regeldifferenz (PID $\neq 0$, Sollwert $=$ Istwert)
- Regler aktiv mit Regeldifferenz (PID $\neq 0$, Sollwert $\neq$ Istwert)

In der Abb. 5.18 sind die Laufzeiten in den drei Fällen für einen PID-Regler auf dem C167 abgebildet. In der Abb. 5.19 sind für den DSP die Laufzeiten für einen PID-Regler dargestellt. Auch in dieser konkreten Anwendung ist ersichtlich, dass mit dem UCoM eine Verminderung der Zykluszeit um etwa den Faktor zehn erreicht werden konnte.

Gegenüberstellung und Bewertung der Messungen

Beim Whetstone-Benchmark lässt sich durch die klare Aufteilung in einzelne Module, die bestimmte Aspekte testen, erkennen, wo die Stärken und Schwächen der beiden Controller-Boards liegen. In Abb. 5.17 werden die Laufzeiten in den jeweiligen Modulen bei C167 und DSP grafisch gegeneinander aufgetragen.

Bei der Berechnung von Array-Elementen in Modul 2 ist der DSP um mehr als 14 mal schneller als der C167. In Modul 3 werden Arrays als Parameter übergeben, dabei hat der DSP fast die 17fache Geschwindigkeit. Bei der Abarbeitung bedingter Sprünge ist der DSP 12,35 mal schneller. Für Modul 6, in dem arithmetische Berechnungen mit Integerzahlen ausgeführt werden liegt der Geschwindigkeitsvorteil noch bei 11,2. Einzig in Modul 7 bei der Berechnung

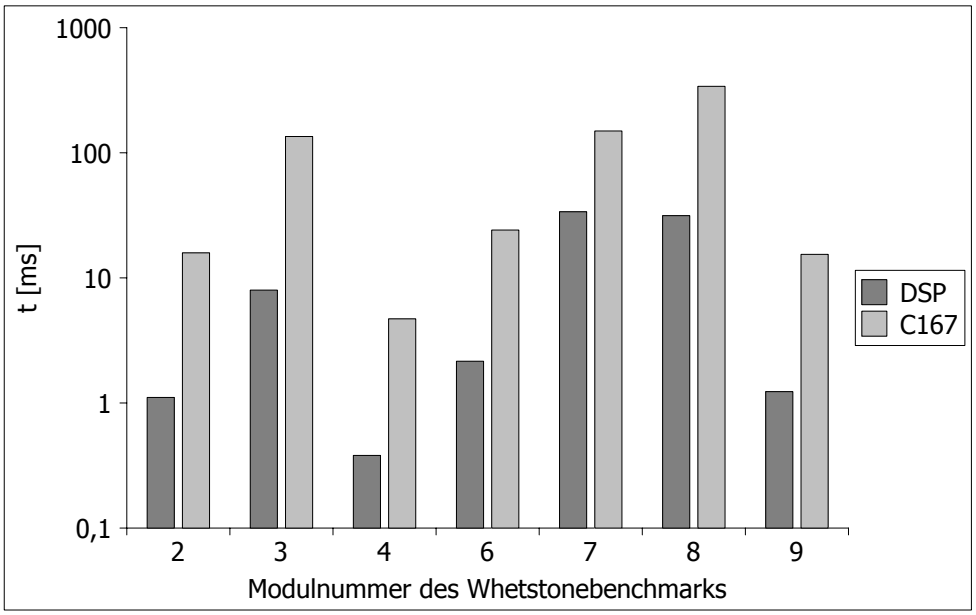


Abb. 5.17.: Gegenüberstellung der Laufzeiten von DSP und C167 beim Whetstone-Benchmark. Es wurde eine logarithmische Auftragung gewählt, so dass die Laufzeiten für den DSP noch deutlich zu erkennen sind.

trigonometrischer Funktionen fällt der Vorsprung mit etwa 4,4 relativ gering aus. Für Prozeduraufrufe mit einfachen Bezeichnern in Modul 8 liegt der Faktor schon wieder bei 10,8 zu Gunsten des DSPs, in Modul 9 bei parameterlosen Prozeduraufrufen und Array-Bezügen sogar wieder bei 12,5. Über die für das Whetstone-Benchmark typische Gewichtung gemittelt ergibt sich für den DSP ein Geschwindigkeitsvorteil um den Faktor 8,8.

Bei der Ausführung der für den Einsatz im Roboter besonders relevanten Regelungsprozesse ergab sich eine Verkürzung der Zykluszeit um etwa den Faktor zehn. Somit lassen sich auch bei Ansteuerung von bis zu drei Motoren, die das Dreier-Motorboard antreiben kann, Zykluszeiten von unter 1 ms und damit eine deutliche Verbesserung der Regelungsgüte erreichen.

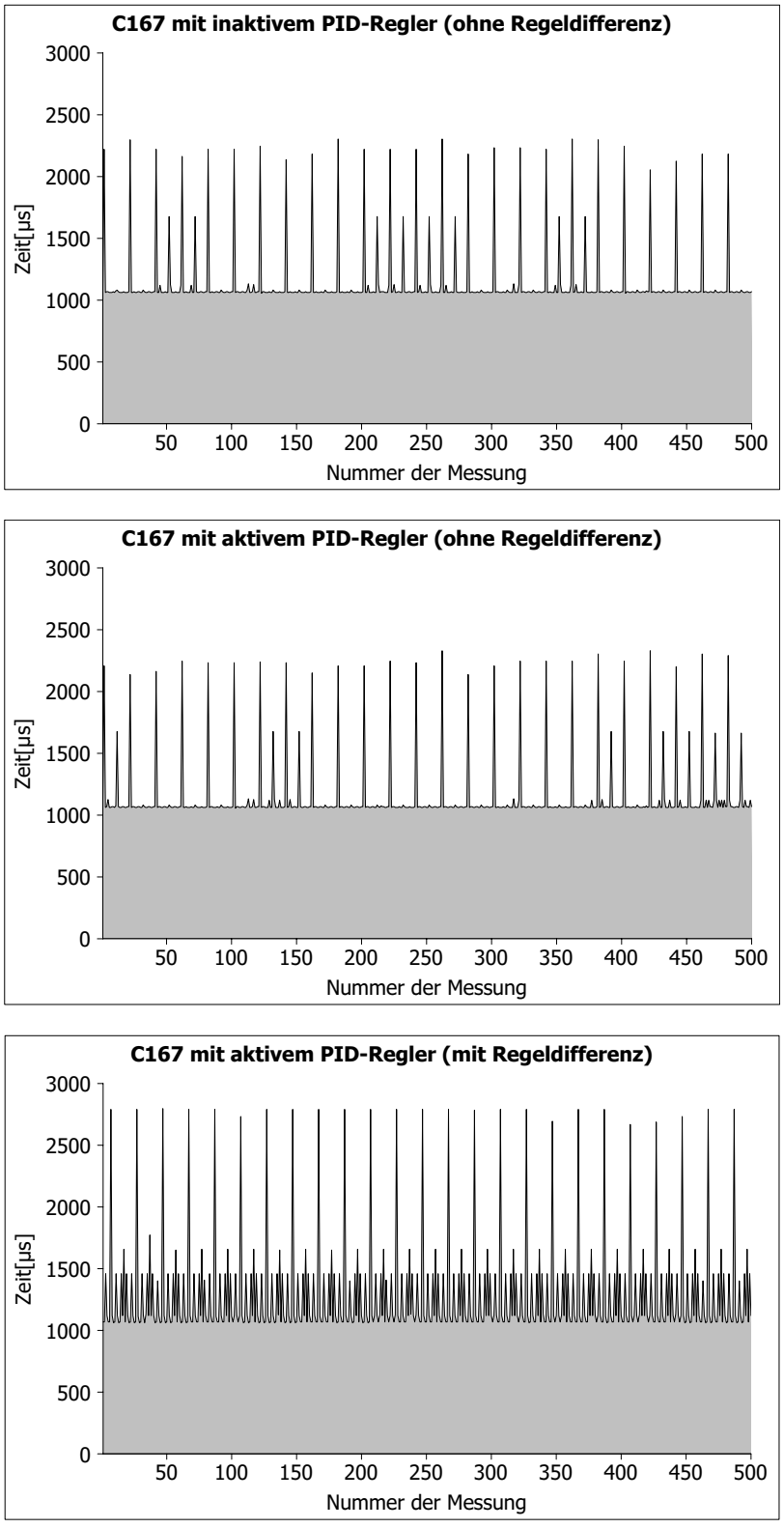


Abb. 5.18.: Darstellung der Laufzeit eines PID-Reglers auf dem C167 im Ruhezustand, mit aktivem Regler ohne Regeldifferenz und mit aktivem Regler mit Regeldifferenz. Die regelmäßigen Peaks lassen sich durch Interrupts zur Analog-Digital-Wandlung und die Kommunikation über CAN-Bus erklären.

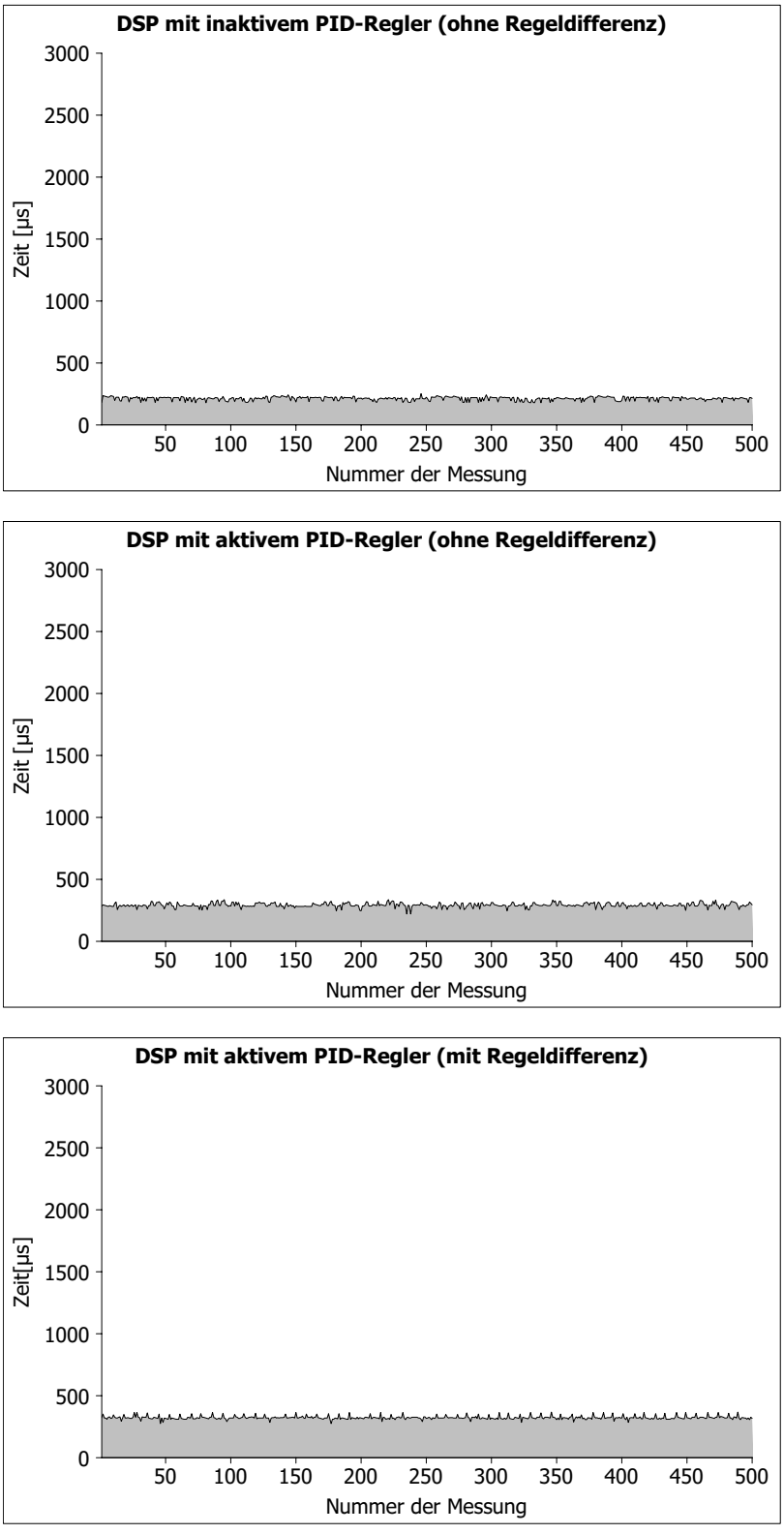
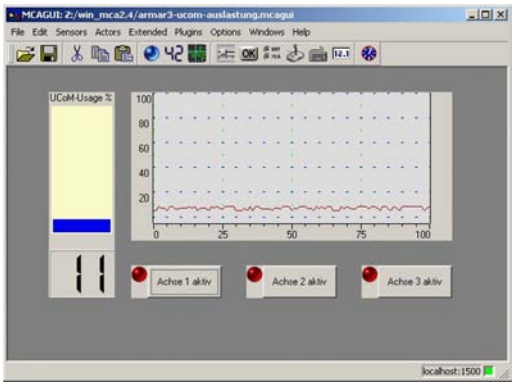
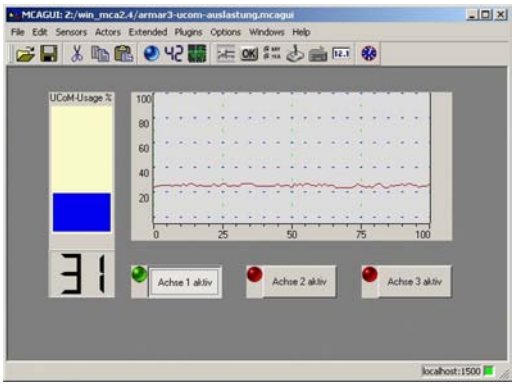


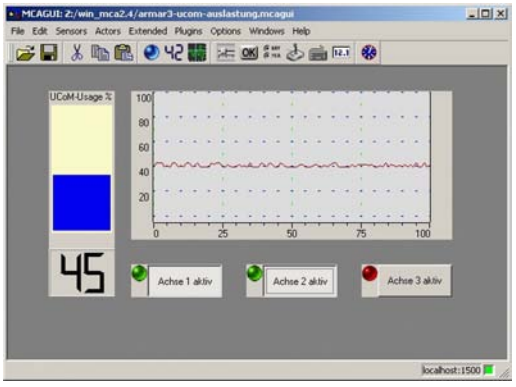
Abb. 5.19.: Darstellung der Laufzeit eines PID-Reglers auf dem DSP im Ruhezustand, mit aktivem Regler ohne Regeldifferenz und mit aktivem Regler mit Regeldifferenz. Zur besseren Vergleichbarkeit wurde die gleiche Skalierung wie in Abb. 5.18 verwendet.



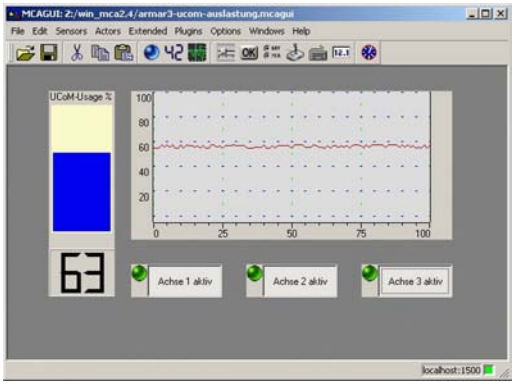
(a) Die Grundauslastung des UCoMs liegt für 1 ms Zykluszeit bei circa 10%



(b) Die Auslastung des UCoMs bei aktivem Regler für eine Achse



(c) Die Auslastung des UCoMs bei aktivem Regler für zwei Achsen



(d) Die Auslastung des UCoMs bei aktivem Regler für drei Achsen

Abb. 5.20.: Dargestellt ist die prozentuale Auslastung eines UCoMs bei Ausführung der auf *ARMAR-III* eingesetzten kaskadierten Positions-Geschwindigkeitsregler. Die Zykluszeit betrug 1 ms.

5.4.2. UCoM-Auslastung in ARMAR-III

Zur Bewertung der Leistungsfähigkeit der Regelungskomponenten, wurde die Auslastung der UCoMs beim Einsatz in *ARMAR-III* bestimmt. Auf dem UCoM wird das Programm *armar3.fl.S* ausgeführt. Diese Programm implementiert auf dem UCoM drei kaskadierte Geschwindigkeits-Positions-Regler. Sowohl für den Geschwindigkeits- als auch für den Positionsteil des Reglers lassen sich die PID-Parameter im Betrieb einstellen. Zur Auslastungsbestimmung wurde die Zeit zur Ausführung der Regleralgorithmen herangezogen. Somit ergibt sich die prozentuale Auslastung als:

$$\text{UCoM-Auslastung} = \frac{t_{\text{Lauf}}}{t_{\text{Taktzyklus}}}$$

In Abb. 5.20 sind für eine Zykluszeit von 1 ms vier Fälle dargestellt: in Abb. 5.20(a) ist die Grundauslastung für ein UCoM mit abgeschalteten Reglern dargestellt. Für die Ausführung der MCA-Schleife und Abarbeitung von Verwaltungs- und Kommunikationsaufgaben benötigt das

UCoM circa 100 µs, was bei der eingestellten Zykluszeit von 1 ms eine Auslastung von ungefähr 10% ergibt. In den Abb. 5.20(b) - Abb. 5.20(d) wird die Auslastung des UCoMs bei aktivem Regler für ein bis drei Achsen dargestellt. Gut zu erkennen ist, dass selbst bei Regelung der Maximalanzahl von drei Bewegungsfreiheitsgraden die Zykluszeit von 1 ms eingehalten werden kann. Es könnten also noch komplexere Regelalgorithmen auf dem UCoM implementiert werden. Sollte für bestimmte Bewegungsfreiheitsgrade eine kürzere Zykluszeit benötigt werden, könnte dies realisiert werden, indem diese Achse separat von einem UCoM angesteuert wird. In diesem Fall lassen sich für die Regelung Frequenzen von bis zu 3 kHz realisieren.

5.5. Eignung für weitere Robotersysteme

Die vorgestellte Hardware-Software-Architektur ist durch ihren modularen Aufbau und ihre gute Skalierbarkeit nicht nur für den Einsatz in humanoiden Robotersystemen geeignet. Humanoide Roboter stellen im Vergleich zu anderen Robotersystemen höhere Anforderungen an die Hardware-Software-Architektur. Eine flexible Anpassung der Architektur auf andere Roboter lässt sich somit durch geringe Änderungen an der Hardwareumsetzung realisieren. Die vorgestellte Rechnerarchitektur wird nicht nur auf dem humanoiden Roboter *ARMAR-III* eingesetzt, sondern auch auf einer Reihe weiterer Roboter, die am FZI entwickelt wurden. Die weiteren Anwendungsgebiete reichen dabei von einfachen Steuerungsanwendungen wie beim Betrieb des rotierenden Sick-Laserscanners (RoSi) bis zu wiederum komplexeren Aufgaben wie der Steuerung der sechsbeinigen Laufmaschine *LAURON-IV*. Im Folgenden sind diese Systeme aufgelistet:

5.5.1. LAURON-IV

Die biologisch motivierte sechsbeinige Laufmaschine *LAURON-IV* (Abb. 5.21(a)) wurde am FZI entwickelt, um statisch stabiles Laufen in unebenem Gelände zu untersuchen [58]. Der mechanische Aufbau der Laufmaschine orientiert sich an dem einer Stabheuschrecke. Wie dieses Vorbild besitzt sie sechs Beine an einem länglichen Zentralkörper, in dem die notwendige Steuerungselektronik untergebracht ist. *LAURON-IV* verfügt über insgesamt 20 Bewegungsfreiheitsgrade, wovon 18 auf die Beine entfallen und zwei auf den Kamerakopf. In der Laufmaschine kommen neben den Encodern zur Bestimmung der Gelenkpositionen zahlreiche weitere Sensorsysteme zum Einsatz. Dies sind zum Beispiel ein GPS-System, ein Stereokamerasystem, 3D-Kraftsensoren in den Füßen und eine 360°-Kamera.

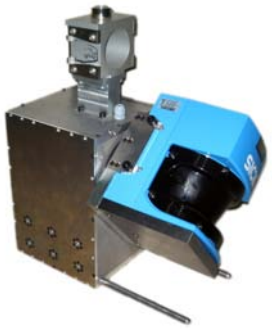
Die Steuerungssoftware von *LAURON-IV* ist mittels MCA2 realisiert und gliedert sich in mehrere hierarchische Ebenen. Die hardwarenahe Ebene wird mit Hilfe der UCoMs aufgebaut. Auf dieser Ebene ist jeweils ein UCoM für die Geschwindigkeits-Kaskaden-Regelung jedes Beines und des Kopfes zuständig. Über den CAN-Bus wird die Verbindung zur darüber-

(a) *LAURON-IV*

(b) *Kairo-II*

(c) *Odete*

(d) Puma200



(e) RoSi

Abb. 5.21.: Weitere Roboter die auf der vorgestellten Hardware-Software-Architektur basieren

liegenden Abstraktionsebene hergestellt. Diese Ebene generiert aus Bewegungstrajektorien die Sollwinkel für die verwendete Roboterkinematik und ist zuständig für die Sensordatenvorverarbeitung. Oberhalb dieser Abstraktionsebene befindet sich die eigentliche, als verhaltensbasierte Steuerung realisierte, Robotersteuerung. Als Rechnerkomponenten für die Abstraktionsebene und Robotersteuerungsebene kommt ein Verbund aus zwei PC/104-Systemen zum Einsatz.

5.5.2. KAIRO-II

Kairo-II ist ein mehrsegmentiger Roboter (Abb. 5.21(b)). Dieser wurde am FZI, aufbauend auf den Erfahrungen mit dem Kanalroboter *MakroPlus* [22], entwickelt. *Kairo-II* besteht aus sechs Segmenten die über fünf Knickelemente mit jeweils drei Bewegungsfreiheitsgraden verbunden sind. Zusammen mit den jeweils zwei Bewegungsfreiheitsgraden für den Antrieb der Segmente ergeben sich für das Gesamtsystem 27 Bewegungsfreiheitsgrade. Durch diese große Redundanz ist der Roboter in der Lage während einer Transportfahrt einzelne Segmente auf einer frei definierten Trajektorie zu bewegen. Somit können gezielt Inspektionsszenarien ausgeführt werden. Mit Hilfe von Sensoren zur Krafterfassung ist es *Kairo-II* möglich sich flexibel an die

Umgebungsbedingungen anzupassen.

Die Steuerung des Roboters ist mit MCA2 aufgebaut und hierarchisch organisiert [23, 21]. Zur Ansteuerung der Aktorik kommen für den Kanalroboter angepasste UCoM-Kombinationen zum Einsatz. Für den Antrieb der Segmente wird das so genannte Alpha-Modul (Abb. B.10(a)) und für die Ansteuerung der Knickelemente das so genannte Gamma-Modul (Abb. B.10(b)) eingesetzt. Über den CAN-Bus sind diese Module mit einem PC/104-System verbunden auf dem die Koordination der Gelenkregelung und die Missionssteuerung implementiert werden.

5.5.3. Odete

Die mobile Plattform *Odete* (Abb. 5.21(c)) ist ein autonomer Roboter für die Erforschung des autonomen Fahrens in Gebäuden. Dabei sind die Forschungsschwerpunkte die Lokalisation, die Navigation und die simultane Lokalisation und Kartierung. Hierfür verfügt die *Odete*-Plattform über einen Differenzialantrieb und einen oder zwei Sick Laserscanner. Forschungsergebnisse können für den Einsatz auf fahrerlosen Transportsystemen transferiert werden.

Im Gegensatz zu anderen Versionen der *Odete*-Plattform in denen eine Architektur basierend auf C167, PC/104 und einem weiteren Industrie-PC zum Einsatz kommt [138], wird in diesem Fall eine auf UCoMs und nur einem PC/104 basierende Architektur verwendet. Für den Antrieb des Roboters und die Auswertung der Odometriedaten kommen insgesamt drei UCoMs zum Einsatz. Für die in MCA2 implementierten höheren Aufgaben wie zum Beispiel die Auswertung der Laserscannerdaten, die Kollisionsvermeidung, die Umweltmodellierung und die Missionssteuerung wird ein PC/104-System verwendet.

5.5.4. Puma200

Der Puma200-Roboterarm ist ein von der Firma Stäubli vertriebener sechsachsiger Roboterarm (Abb. 5.21(d)). Ursprünglich ist dieser Roboterarm für den industriellen Einsatz und den Betrieb mit einem Steuerschrank ausgelegt. Die sechs Achsen des Roboters werden elektrisch angetrieben und sind mit Inkrementalgebern für die Positionsbestimmung ausgestattet.

Für die Erforschung von Manipulationsaufgaben mit einem Roboterarm auf einer mobilen Plattform, wurde dieses System so umgerüstet, dass es mit PC-basierten Rechnerkomponenten angesteuert werden kann. Hierzu wurde die Steuerelektronik durch drei UCoMs ersetzt, die für die Gelenkregelung und die Ansteuerung der Gelenkarretierung zuständig sind. Über CAN-Bus wird die Verbindung zu einem PC/104-System hergestellt, auf dem die inverse Kinematik und die höheren Regelungsebenen mit MCA2 realisiert werden.

5.5.5. RoSi - Rotierender Sick-Laserscanner

Die Aufnahme von dichten 3D-Punktwolken ist für viele Forschungs- und Anwendungszwecke eine wichtige Voraussetzung. Insbesondere für die Kollisionsvermeidung und die simultane Lokalisation und Kartierung werden diese Daten benötigt. Für den Einsatz auf mobilen Systemen wurde aus einem SICK LMS 2D-Laserscanner ein endlos rotierender 3D-Laserscanner aufgebaut [97, 146]. In Abb. 5.21(e) ist der Aufbau, wie er für die „Kerntechnische Hilfsdienst GmbH“ (KHG) aufgebaut wurde, dargestellt.

Die präzise Regelung der Drehgeschwindigkeit erfolgt mit einem UCoM. Als Steuerrechner zur Auswertung der Laserscannerdaten kommt ein Industrie-PC zum Einsatz. Auf diesem wird mit in MCA2 implementierten Modulen die Verknüpfung des 2D-Linienscans mit den vom UCoM übermittelten Positionsdaten durchgeführt und zu einer 3D-Punktwolke zusammengesetzt.

6. Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Hardware-Software-Architektur zur Integration von Sensorik, Datenverarbeitung und Aktorik bei humanoiden Robotern entworfen. Teil dieses Entwurfs war die Betrachtung der funktional-logischen Aspekte der Steuerung, die Unterstützung der Implementierung von Roboterfunktionalitäten durch ein Softwarerahmenwerk und die Berücksichtigung spezifischer Randbedingungen bei humanoiden Robotern, wie zum Beispiel der Platzverhältnisse oder geringer Energieressourcen. Es wurde sowohl die Vorgehensweise beim Entwurf als auch die exemplarische Umsetzung für das Robotersystem *ARMAR-III* vorgestellt. Es wurde gezeigt, dass die entworfene und realisierte Rechnerarchitektur Anforderungen für humanoide Roboter erfüllt und sich darüber hinaus auch für den Einsatz in anderen Robotersystemen eignet. Durch den modularen Aufbau ist eine Erweiterbarkeit durch weitere Sensoren und Aktoren und auch die Anpassung an neu verfügbare Recheneinheiten möglich. Im Folgenden werden die dabei behandelten Fragestellungen und die erarbeiteten Lösungen zusammengefasst.

6.1. Ergebnisse der Arbeit

Steuerungsarchitektur Zur Steuerung humanoider Roboter müssen Aufgaben auf unterschiedlichen Abstraktionsebenen bearbeitet werden. Zur Lösung dieser Aufgaben wurde eine hierarchische 3-Ebenen-Steuerungsarchitektur entwickelt. Sie gliedert sich in die Ebenen Planung, Koordination und Ausführung. Auf der Planungsebene werden komplexe Handlungsaufgaben geplant und in einzelne Teilaufgaben zerlegt. Diese werden wiederum auf der Koordinationsebene mit Hilfe eines Zustandsautomaten koordiniert und synchronisiert. Die Teilaufgaben werden auf Basisfähigkeiten bzw. auf so genannte Motorbefehle abgebildet und als entsprechende Sollwertvorgaben übermittelt. Auf der Ausführungsebene erfolgt dann mittels dezidierter Controller die Umsetzung der angeforderten Basisfähigkeiten.

Softwarerahmenwerk Im Sinne eines Hardware-Software-Codesigns wurden bei der Entwicklung der Hardware-Software-Architektur Strukturen des Softwarerahmenwerks MCA2 in den Systementwurf einbezogen. MCA2 ist ein hierarchisch und modular aufgebautes Softwarerahmenwerk, das auf Softwaremodulen mit den klar definierten Schnittstellen – *Controller Input*, *Controller Output*, *Sensor Input*, *Sensor Output* und *Parameter* – basiert. Diese Module lassen sich zu Gruppen zusammenfassen, die nach außen hin wiederum die gleichen Schnittstellen zur Verfügung stellen wie ein einzelnes Modul. Durch dieses Vorgehen lassen sich bereits

erstellte Module gut wiederverwenden und durch Kombination dieser Module komplexe Funktionalitäten realisieren.

Abbildung auf Hardware-Software-Architektur Da sowohl die Sensoren als auch die Aktoren räumlich über den Roboter verteilt angeordnet sind, ist es notwendig, auch die Recheneinheiten der Architektur dezentral und verteilt zu platzieren. Die Anforderungen an die Recheneinheiten für die Planungs- und Ausführungsebene werden von PC-basierten Komponenten erfüllt. Durch die Verwendung standardisierter Komponenten und Schnittstellen können diese mit dem technischen Fortschritt durch leistungsfähigere Recheneinheiten ersetzt werden. Zur Erfüllung der hohen Anforderungen der Regelung auf der Ausführungsebene wurde eine dezidierte Komponente entwickelt, die durch ihre aktornahe Platzierung die Regelung der Gelenke in Echtzeit ermöglicht. Die Kommunikation zwischen den PC-basierten Komponenten findet über Gigabit-Ethernet statt. Die Kommunikation zu und unter den dezidierten Komponenten erfolgt über CAN-Bus.

Elektrischer und elektronischer Aufbau des humanoiden Roboters ARMAR-III Zur Evaluierung der vorgestellten Architektur wurde *ARMAR-III* mit den ausgewählten Recheneinheiten und vielfältiger Sensorik ausgestattet. Für jeden Bewegungsfreiheitsgrad werden die Motordrehwinkel und Motorströme mit Hilfe der entwickelten dezidierten Komponente erfasst, für ausgewählte Achsen zusätzlich Drehmomente und absolute Achsposition. Weiterhin werden für die Robotersteuerung die Informationen von Kameras, Mikrofonen, Laserscannern, Kraft-Momenten-Sensoren und einer künstlichen Haut ausgewertet.

Im Rahmen einer komplexen Demonstration, die sämtliche Anforderungsaspekte an den humanoiden Roboter abdeckte, konnte die Leistungsfähigkeit der vorgestellten Hardware-Software-Architektur gezeigt werden. In dieser so genannten Küchen-Demo führt der Roboter komplexe Handhabungsaufgaben aus und interagiert ausschließlich über Sprache mit dem Benutzer. Dabei liefen alle Prozesse zur Planung der Handlungsausführung und zur Roboterregelung autonom auf dem Roboter. Auf der Seite der Echtzeitregelung wurde die Leistungsfähigkeit der entwickelten Komponente, dem UCoM, in verschiedenen Benchmarks und im konkreten Einsatzfall untersucht. Mit Hilfe dieser Komponente konnten die Regelzyklen für die Servo-Motor-Regelung des Roboters auf unter 1 ms reduziert und somit eine menschenähnliche Bewegungsausführung ermöglicht werden.

6.2. Ausblick

Die in dieser Arbeit vorgestellte Hardware-Software-Architektur gibt vor allem die Struktur vor und lässt bei der Auswahl der konkreten Rechenkomponenten und Verbindungseinrichtungen

weiteren Entwicklungsspielraum. Mit Hinblick auf den schnellen technischen Fortschritt bei Rechenkomponenten werden in absehbarer Zeit deutlich leistungsfähigere Einzelkomponenten verfügbar werden, so dass zum Beispiel der Verbund aus Industrie-PCs durch einen einzigen *Multicore*-Rechner ersetzt werden kann. Eine solche Tendenz zur Reduktion der Komponentenanzahl und zur Zentralisierung in Bezug auf die PC-basierten Komponenten birgt besonders mit Blick auf die Entwicklung eines zweibeinigen Roboters Vorteile. Im Gegensatz zu einem Roboter auf einer mobilen Plattform sind dort die Platzressourcen für Recheneinheiten weiter eingeschränkt. Entgegen dieser Zentralisierungstendenz bei den PC-basierten Komponenten wird für den echtzeitfähigen Teil der Architektur, also bei der Roboterregelung, eine dezentrale Ansteuerung auch in Zukunft bevorzugt. Das UCoM stell eine Vorstufe zu einem noch weiter spezialisierten „System on Chip“ (SoC) bzw. „Application Specific Integrated Circuit“ (ASIC) dar. Ziel ist es, in diesem Bereich eine noch feinere Granularität zu erreichen und für jeden Bewegungsfreiheitsgrad einen SoC oder ASIC zur Ansteuerung zu verwenden. Dieser Tendenz wurde in *ARMAR-IIIb*, dem Nachfolger von *ARMAR-III*, bereits Rechnung getragen. Für den Antrieb der mobilen Plattform werden Motoren mit integriertem Motorregler verwendet.

Auch in den anderen Domänen lässt sich in Zukunft dezidierte Hardware einsetzen. In den Bereichen Bildverarbeitung und Akustik sind ASICs und SoCs gewinnbringend einsetzbar. Sowohl die Bildverarbeitung als auch die Akustik beinhalten sehr rechenaufwendige Prozesse, die sich, sobald die Algorithmen hierzu hinreichend ausgereift sind, in dezidierter Hardware sehr effizient realisieren lassen. Im Hinblick auf die Energieeffizienz und auch auf die Auslastung der PC-Komponenten stellt eine Reduktion der Anzahl einen Fortschritt dar. Somit müssen Rechnerressourcen wie Speicher und verfügbare CPU-Leistung nicht mehrfach vorgehalten werden. Wenn die Rechenleistung, die jetzt durch einen Verbund mehrerer PCs erreicht wird, mit beispielsweise einem *Multicore*-Rechner erbracht werden kann, lassen sich dadurch sowohl Energieverbrauch, Gewicht als auch Kosten verringern.

Durch die nicht auf konkrete Komponenten und Verbindungseinrichtungen festgelegte Struktur der Hardware-Software-Architektur können Komponenten und Verbindungseinrichtungen entsprechend dem technischen Fortschritt angepasst werden. Dabei bleiben die Aspekte funktional-logische Steuerungsarchitektur und Softwarearchitektur weiterhin gültig. Somit ist einerseits eine Anpassung an den technischen Fortschritt problemlos möglich, andererseits lässt sich die Architektur gut skalieren und an andere Robotersysteme anpassen. Eine Übertragung der Hardware-Software-Architektur, beispielsweise auf zweibeinige Roboter, zum Beispiel den geplanten *ARMAR-V* mit veränderten Randbedingungen, ist problemlos möglich. Auch die Anpassung auf andere Robotersysteme, wie Laufmaschinen, mobile Plattformen oder als eingebettete Steuerung in autonomen Fahrzeugen oder Sensorikanwendungen wurde bereits erfolgreich durchgeführt.

A. Quantitative Abschätzung der Anforderungen an die Hardware-Software-Architektur

Besonders bei der Übertragung auf eine Hardware-Architektur zeigen sich die unterschiedlichen Ansprüche der einzelnen Domänen an das Rechnersystem. Die Bildverarbeitung sorgt für ein hohes Datenaufkommen und stellt somit hohe Anforderungen an die zur Datenübertragung. Beispielsweise ergeben sich bei unkomprimierter Datenübertragung von einer Kamera mit der Auflösung von 640×480 Pixeln bei einer Farbtiefe von 24 bit und einer Framerate von 25 fps (*frames per second*) folgende Datenraten

$$(640 \cdot 480)\,{}^{\text{Pixel}}/_{\text{Frame}} \cdot 24\,{}^{\text{bit}}/_{\text{Pixel}} \cdot 25\,{}^{\text{Frame}}/_{\text{s}} = 23\,040\,000\,{}^{\text{bit}}/_{\text{s}} = 23,04\,{}^{\text{Mbit}}/_{\text{s}}$$

Da für Tiefensehen zwei oder mehr Kameras eingesetzt werden, muss diese Datenrate mit der Anzahl der Kameras multipliziert werden. Neben der Übertragung stellt die Verarbeitung der Kameradaten, insbesondere die Analyse der Bilder bezüglich des Vorkommens gemeinsamer Merkmale zur Berechnung von Tiefeninformation und die Erkennung von Objekten im Bild, hohe Anforderungen an die Rechenleistung des Rechnersystems. Seitens der Zykluszeiten sind die Anforderungen durch die von den meisten Kamerasystemen vorgegebenen 25 fps mit 40 ms eher gering.

Im Bereich der Audiowahrnehmung bewegen sich diese Werte in einem niedrigeren Bereich. Für ein Mikrofon ist das Datenaufkommen bei einer Abtastrate von 40 kHz und einer Genauigkeit von 8 bit

$$40\,000\,{}^{\text{Abtastungen}}/_{\text{s}} \cdot 8\,{}^{\text{bit}}/_{\text{Abtastung}} = 320\,{}^{\text{kbit}}/_{\text{s}}$$

Zur Spracherkennung würde ein Mikrofon reichen, wenn jedoch die Position des Sprechers oder anderer Geräuschquellen erkannt werden soll, werden mindestens zwei Mikrofone benötigt. Um Abschattungseffekte bei der Schallausbreitung zu vermeiden und um sowohl Azimut als auch Elevation bestimmen zu können, werden teilweise sogar vier oder sechs Mikrofone eingesetzt. Für sechs Mikrofone ergibt sich dann ein Datenaufkommen von 1,92 MB/s. Die benötigte Rechenleistung für die Erkennung von Sprache sind hoch. Ebenso sind die Anforderungen zur Berechnung der Korrelationen zwischen den einzelnen Kanälen zur Bestimmung der Richtung, aus der das Geräusch kommt, hoch. Für die Anforderung bezüglich der Zykluszeit ist nicht die hohe Abtastrate von 40 kHz relevant, da nicht auf einzelnen Abtastungen gearbeitet wird. Zur Erkennung von Sprache und zur Bestimmung der Korrelation der Kanäle, werden jeweils längere Abschnitte im Sekundenbereich herangezogen. Somit ergeben sich für das Rechnersystem

relevante Zykluszeiten im Bereich von 1 s.

Für den regelungsrelevanten Teil der Hardware-Software-Architektur ergeben sich gänzlich andere Anforderungen an das Rechnersystem. Für einen einzelnen Sensor bzw. Aktor ergibt sich folgende Datenrate:

$$16\,\mathrm{bit}/\mathrm{Nachricht} \cdot 1000\,\mathrm{Nachrichten}/\mathrm{s} = 16\,\mathrm{kbit}/\mathrm{s}$$

Geht man von einem Roboter mit ca. 20 Bewegungsfreiheitsgraden aus, für den im Schnitt je Bewegungsfreiheitsgrad zwei Sensorwerte erfasst werden, ergibt sich damit eine benötigte Übertragungsrate für die Sensorwerte und die Aktorstellwerte von insgesamt

$$16\,\mathrm{kbit}/\mathrm{s} \cdot 60 = 960\,\mathrm{kbit}/\mathrm{s}$$

Mit insgesamt unter einem Mbit/s für sämtliche Aktoren und Sensoren ist die geforderte Datenrate relativ gering, allerdings sind die benötigten Echtzeitbedingungen bei den Zykluszeiten in der Größenordnung von 1 ms eine Herausforderung. Die Echtzeitanforderung ist nicht nur für die Datenübertragung relevant, sondern das Rechnersystem muss diese auch bei der Berechnung der neuen Sollwertvorgaben für die Aktoren erfüllen können.

Auf Seiten der Planung ist es ausreichend, wenn circa jede Sekunde ein aktualisierter komplexer Handlungsplan generiert wird. Die Anforderungen an das Rechnersystem bezüglich Datenübertragungsraten und Rechenleistung liegen in der gleichen Größenordnung wie bei der Bildverarbeitung.

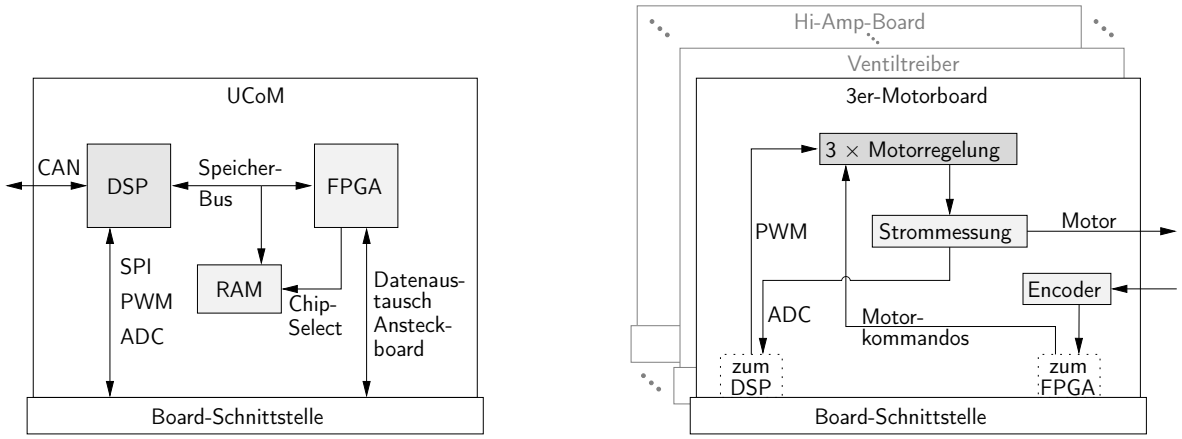
B. UCoM

B.1. Einbindung von DSP und FPGA

Um Daten zwischen DSP und FPGA auszutauschen wird der externe Datenbus des DSP verwendet. Der FPGA wird transparent in den externen Speicherbereich des DSP eingeblendet. Da sich der FPGA den externen Adressbereich mit dem externen RAM, welches auch auf dem UCoM verwendet werden soll, teilen muss, wurde die Registerzahl für den Datenaustausch zwischen DSP und FPGA auf 64 festgelegt. Um den externen Speicherplatz so wenig wie möglich einzuschränken, wurde zur Unterscheidung zwischen Speicherzugriff und FPGA-Zugriff nicht ein einzelnes Bit des 16 Bit breiten Adressbusses herangezogen, sondern ein bestimmtes Bitmuster der oberen 10 Bit des Adressbusses. Konkret gehen alle Speicherzugriffe, deren Bitmuster wie folgt aussieht, an den FPGA: 0100 0000 00XX XXXX. Die Unterscheidung welche Komponente die Daten übernimmt, wird im FPGA gefällt. Grundsätzlich liegen Daten- und Adressbus parallel am FPGA und am externen Speicher (Abb. B.1). Zusätzlich ist die Chipselect-Leitung des externen Speichers, die den Speicher aktiv oder inaktiv schalten kann, mit dem FPGA verbunden. Im Standardfall ist der externe Speicher inaktiv und wird nur nach der Adressauswertung durch den FPGA aktiv geschaltet. Nachdem der DSP einen externen Speicherzugriff initiiert und somit Daten- und Adressbus beschreibt, analysiert der FPGA das Bitmuster der Adresse. Für den Fall, dass die oberen 10 Bit dem oben angegebenen Muster 0100 0000 00XX XXXX entsprechen, erkennt der FPGA, dass die Daten für ihn bestimmt sind und entscheidet anhand der unteren 6 Bit für welches Register die anstehenden Daten sind. Im gleichen Zug lässt er die Chipselect-Leitung für den externen Speicher deselektiert und verhindert damit, dass der Schreib- oder Lesezugriff auf den Speicher stattfindet. Für alle anderen Bitmuster in den oberen 10 Bit entscheidet der FPGA, dass die Daten für den externen Speicher angelegt wurden und schaltet das Chipselect-Signal des Speichers entsprechend auf aktiv, wodurch die Daten in diesen geschrieben bzw. aus diesem gelesen werden können.

B.2. Kommunikation über CAN-Bus

Der CAN-Bus ist ein Feldbus [65], der als asynchroner, serieller Bus realisiert ist. CAN zeichnet sich durch eine robuste differenzielle Übertragung mit Übertragungsraten bis zu 1 Mbit/s aus. Der Buszugriff erfolgt mittels CSMA/CA. Mit einer verlustfreien, bitweisen Arbitrierung erhält die Nachricht mit der höchsten Priorität den Buszugriff. Damit kann für diese Nachricht die



(a) Seite des UCoMs (b) Seite des Ansteckboards

Abb. B.1.: Anschlussschema der Ansteckboards an das UCoM

Telegrammfeld	Start	Identifier	RTR	IDE	r0	DLC	DATA	CRC	ACK	EOF+IFS
Feldlänge in Bit	1	11	1	1	1	4	0 - 64	15	2	10

Tab. B.1.: Aufbau der CAN-Nachricht

Latenz von einem Sendezyklus garantiert werden.

B.2.1. Aufbau der CAN-Nachrichten

Eine CAN-Nachricht, ein so genannter CAN-Frame, besteht aus den in Tab. B.1 dargestellten Elementen. Die Bedeutung der einzelnen Elemente des CAN-Frames werden nachfolgend erläutert:

Start dominantes Bit, signalisiert den Start des CAN-Frames

Identifier Prioritätsinformation für die Busarbitrierung und Zuordnung der Nachricht zu einem Empfänger

RTR (Remote Transmission Request) Unterscheidung zwischen Datentelegramm (dominant) und Datenanforderungstelegramm (rezessiv)

IDE (Identifier Extension) Unterscheidung zwischen Standardframe und Extended Frame

r0 Reserviert

DLC (Data Length Code) Gibt die Länge des nachfolgenden Datenfeldes an

Anzahl an Datenbytes	Übertragunsrate mit Standard Identifier kbit/s	Extended Identifier kbit/s
0	0	0
1	145	107
2	254	193
3	338	264
4	405	323
5	460	374
6	505	417
7	544	455
8	577	489

Tab. B.2.: Effektive Datenrate der CAN-Nachrichten bei 1 Mbit/s

DATA Enthält die Daten des Telegramms. Dies können 0 - 8 Byte sein.

CRC (Cyclic Redundancy Check) CRC-Prüfsumme über den vorausgegangenen Teil des Frames

ACK Rückmeldung anderer Bus-Knoten über korrekten Empfang des Frames

EOF (End of Frame) Kennzeichnung des Nachrichtenendes durch Senden von sieben rezessiven Bits

IFS (Inter Frame Space) Senden von drei weiteren rezessiven Bits, um den Knoten Zeit zu geben, die Nachricht zu verarbeiten

Die Nachrichteneffizienz der CAN-Nachrichten steigt mit einer größeren Anzahl an Datenbytes (Tab. B.2)

B.2.2. Vergabe der Identifier

Eine grundlegende Aufgabe bei der Vernetzung von Komponenten über CAN-Bus ist die Auslegung, wie Identifier zu vergeben sind. Der Identifier des CAN-Protokolls ist 11 Bit lang und es muss sichergestellt werden, dass niemals zwei Knoten gleichzeitig eine Nachricht mit demselben Identifier aussenden. Um dies zu erreichen, wurde der Identifier in drei Bereiche unterteilt. Die Aufteilung erfolgte in Untergruppen 5 Bit / 1 Bit / 5 Bit (Tab. B.3). Die niederwertigen 5 Bit stellen die Nummer des Controller-Boards dar. Das mittlere Bit im Identifier unterscheidet zwischen Controller-Boards, die eine fest einprogrammierte Nummer haben und jenen

10	9	8	7	6	5	4	3	2	1	0
Funktion[5:0]					Typ	Nummer[5:0]				

Tab. B.3.: Aufbau des CAN-Identifiers

Controller-Boards, die sich diese Nummer automatisch anhand ihrer Position in der CAN-Bus-Topologie selbst zuweisen. Die hochwertigen 5 Bit stellen schließlich den Nachrichten-Typ dar und beschreiben, welche Bedeutung die Daten im Datenfeld haben. Durch die Übermittlung des Nachrichten-Typs im Identifier kann das Datenfeld komplett zur Übermittlung der Nutzdaten verwendet werden und somit der Protokoll-Overhead reduziert werden.

Zusätzlich zu den beiden CAN-Daten-Leitungen CAN-Hi und CAN-Lo wird noch eine Spannungsversorgung und eine so genannte Init-Leitung beim Aufbau der CAN-Bus-Topologie verwendet. Die Spannungsversorgung dient zum Betrieb der Optokoppler, die den CAN-Bus galvanisch von den Controller-Boards trennen. Über die Init-Leitung können die Controller-Boards ihre Position im CAN-Bus ermitteln. Die Init-Leitung ist keine durchgehende Verbindung, sondern sie wird durch die am CAN-Bus angeschlossenen Controller-Boards unterbrochen, sie geht immer vom Init-Ausgang des einen Boards zum Init-Eingang des nächsten Boards. Der Pegel der Init-Leitung wird am Init-Eingang ausgewertet. Über den Pegel der Init-Leitung zum nächsten Board kann Information ausgetauscht werden, an welcher Stelle im CAN-Bus sich das entsprechende Board befindet. Die Init-Anschlüsse sind Ein-/Ausgänge, das heißt es kann sowohl ein Pegel gesetzt werden, als auch der Pegel ausgelesen werden. Eine detailliertere Skizze zur Verschaltung der Initsignale findet sich in Abb. B.2 .

Am Init-Eingang und am Init-Ausgang liegt elektrisch im Standardfall ein Hi-Pegel an (Abb. B.3(a)). In der Initialisierungsphase soll die Position der Boards im CAN-Bus ermittelt werden: hierzu legen alle Boards ihren Init-Ausgang auf den elektrischen Lo-Pegel (Abb. B.3(b)). Außer dem ersten Board in der Kette empfängt somit jeder Controller an seinem Init-Eingang einen Lo-Pegel. Das Board, welches in dieser Phase einen Hi-Pegel an seinem Init-Eingang misst, ist also das erste Board in der Kette. Nach diesem Schritt wird über den CAN-Bus kommuniziert, dass nun das zweite Board in der Kette gesucht wird (Abb. B.3(c)). Dann legt das erste Board seinen Init-Ausgang auf Hi und signalisiert damit dem Board nach ihm in der Kette, dass es sich die Nummer zwei zuweisen soll. Dieser Vorgang wird solange wiederholt bis sich auch das letzte Board in der Kette seine Nummer zugewiesen hat (Abb. B.3(d) - Abb. B.3(g)). Die elektrische Auslegung dieses Initialisierungsverfahrens sieht sogar vor, dass dieser Prozess auch in umgekehrter Richtung durchgeführt werden kann und somit das letzte UCoM in der CAN-Kette eindeutig identifiziert werden kann. In diesem Falle würden alle UCoMs ihren Init-Eingang einen Lo-Pegel ausgeben und nur das letzte würde an seinem Init-Ausgang einen Hi-Pegel messen. Da das Auffinden des letzten UCoMs auch einfacher über einen Time-Out funktioniert, wird an dieser Stelle jedoch auf dieses Verfahren bei der Initialisierung verzichtet.

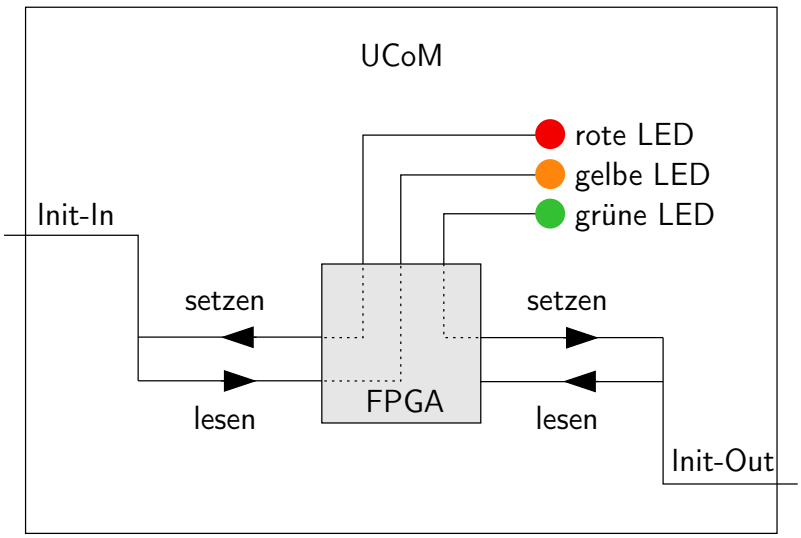


Abb. B.2.: Detaillierte Darstellung der Verschaltung der Init-Signale auf dem UCoM

B.2.3. Signalisierung des Init-Zustandes

Mit Hilfe dreier LEDs in den Farben rot, gelb und grün wird während der Initialisierungsphase der Zustand der Init-Leitungen visualisiert. Wie in Abb. B.2 ersichtlich, sind Init-Eingang und Init-Ausgang doppelt am FPGA angeschlossen. Dies dient dazu, beide sowohl setzen als auch auslesen zu können. In den Leitungen befinden sich noch Serien- und Parallel-Widerstände, die in der Abbildung der Übersichtlichkeit halber nicht dargestellt wurden. Im Standardfall werden auf jedem UCoM Init-Eingang und Init-Ausgang elektrisch über einen so genannten Pull-Up auf Hi gezogen. Das heißt, die elektrischen Pegel sind den oben genannten logischen genau entgegengesetzt. Ein elektrische Hi-Pegel entspricht dem Zustand „Nicht gesetzt“, also logisch Lo und der elektrische Lo-Pegel entspricht dem logischen „Gesetzt“, also logisch Hi. Die jeweiligen Widerstände sind so ausgelegt, dass der FPGA in der Lage ist, gegen diesen Pull-Up seine Ausgänge zuverlässig auf Lo zu ziehen. Weiterhin musste berücksichtigt werden, dass für den Fall, dass beispielsweise Init-Ausgang von UCoM-1 und Init-Eingang von UCoM-2 entgegengesetzt angesteuert werden, die zulässigen Ströme an den Pins des FPGAs nicht überschritten werden. Der Zustand der Init-Leitungen wird während der Initialisierungsphase an den LEDs ausgegeben. Die Zuordnung der Farben wurde in folgender Form vorgenommen:

Rot: Leuchtet, wenn über den FPGA der Init-Eingang aktiv auf Lo gezogen wird, signalisiert also: Init-Eingang gesetzt

Gelb: Leuchtet, wenn über den FPGA der Init-Ausgang aktiv auf Lo gezogen wird, signalisiert also: Init-Ausgang gesetzt

Grün: Leuchtet, wenn der FPGA am Init-Eingang einen Lo-Pegel misst, signalisiert also: Gesetzter Init-Ausgang des vorherigen UCoMs empfangen

Direkt nach dem Einschalten einer Kette von UCoMs die nach dem Muster in Abb. B.3 verbunden sind, leuchtet also beim ersten UCoM in der Kette nur die gelbe LED und bei allen folgenden sowohl die gelbe als auch die grüne LED. Anhand dieser Signalisierung lässt sich sehr einfach die richtige Verkabelung überprüfen. FPGA-intern sind die LEDs über einen Multiplexer mit den LEDs verbunden. Nach dem Abschluss der Initialisierungsphase ist der Zustand der Init-Leitungen nicht mehr von Interesse und muss demzufolge auch nicht mehr angezeigt werden. Sobald das Programm gestartet ist, werden die LEDs zur Anzeige des Programmstatus genutzt.

Rot: Signalisiert einen Fehler auf dem CAN-Bus

Gelb: Leuchtet kurz auf, wenn die dem UCoM vorgegebene Zykluszeit nicht eingehalten wurde. Abhängig davon, um wie viel die angegebene Zeit überschritten wurde, wird die Helligkeit der LED gesteuert.

Grün: Zeigt durch Blinken die ordnungsgemäße Ausführung des geladenen Programms an. Die Frequenz des Blinkens lässt die eingestellte Zykluszeit erkennen.

B.2.4. Zuordnung der Nachrichten-Funktion

Es gibt verschiedene Nachrichtentypen, deren Wichtigkeit sich unterscheidet. Unter Berücksichtigung der Priorisierung nach dem CAN-Bus-Protokoll wurden die Funktions-IDs den einzelnen Funktionen zugewiesen. Da am CAN-Bus Nachrichten mit der kleinsten ID die höchste Priorität besitzen, wurde zum Beispiel der Reset-Nachricht die Funktions-ID „0“ zugewiesen. Ein Überblick über die Verteilung der Funktions-IDs befindet sich in Tab. B.4.

B.3. Software auf dem UCoM

Als Entwicklungsumgebung für die Anwendungsprogramme auf dem DSP56F803 kommt Metrowerks Codewarrior[1] in der Version 5.5 zum Einsatz (Abb. B.4). Auf dem UCoM sind verschiedene Funktionalitäten implementiert. Dazu zählen einerseits die Basisfunktionalitäten zum Betrieb des UCoM wie beispielsweise der Bootloader, der Datenaustausch zwischen DSP und FPGA und die Bereitstellung von Schnittstellen zur Kommunikation und Erfassung von Sensorwerten. Andererseits wurden mehrere Anwendungsprogramme für das UCoM wie P- oder PID-Regler und das in *ARMAR-III* eingesetzte Programm *armar3.fl.S* (Abschnitt 5.2.2) entwickelt. Hier soll etwas näher auf die initiale Programmierung und die Kommunikation zwischen PC und UCoM in der Startphase eingegangen werden.

[1] `http://www.metrowerks.com`

Funktion	Initiator	Bit 10 - Bit 5						Bit 4 - Bit 0
Reset	PC	0	0	0	0	0	0	sequenzielle ID
XReset	PC	0	0	0	0	1	0	sequenzielle ID
Error	DSP	0	0	0	1	0	0	sequenzielle ID
Init Ack	DSP	0	0	0	1	1	0	sequenzielle ID
Program Finished	DSP	0	0	1	0	0	0	sequenzielle ID
Info	DSP	0	0	1	0	1	0	sequenzielle ID
Ready	DSP	0	0	1	1	0	0	sequenzielle ID
SetStatus	PC	0	0	1	1	1	0	sequenzielle ID
Info Request	PC	0	1	0	0	0	0	sequenzielle ID
Report Request	PC	0	1	0	0	1	0	sequenzielle ID
Init Request	PC	0	1	0	1	0	0	sequenzielle ID
Program	PC	0	1	0	1	1	0	sequenzielle ID
Start	PC	0	1	1	0	0	0	sequenzielle ID
MCA Controller Input	PC	0	1	1	0	1	0	sequenzielle ID
MCA Sensor Output	DSP	0	1	1	1	0	0	sequenzielle ID
MCA Parameter	PC	0	1	1	1	1	0	sequenzielle ID
Report	DSP	1	0	0	X	X	0	sequenzielle ID
MCA Ready	DSP	1	0	1	0	0	0	sequenzielle ID
MCA Start	PC	1	0	1	0	1	0	sequenzielle ID
MCA DumpIO Descr Request	PC	1	0	1	1	0	0	sequenzielle ID
MCA DumpIO Descr	DSP	1	0	1	1	1	0	sequenzielle ID
MCA Dump CI	PC	1	1	0	0	0	0	sequenzielle ID
MCA Admin Data	PC	1	1	0	0	1	0	sequenzielle ID
MCA Alive	PC	1	1	0	1	0	0	sequenzielle ID
MCA DSP CI	DSP	1	1	0	1	1	0	sequenzielle ID

Tab. B.4.: Übersicht über die Funktions-IDs

B.3.1. Der Bootloader

Der Bootloader auf dem DSP des UCoM stellt die Firmware des UCoMs dar. Er muss mittels der JTAG-Schnittstelle einmalig programmiert werden. Die weitere Kommunikation findet über den CAN-Bus statt. Mit Hilfe des Bootloaders wird die initiale Kommunikation mit den UCoMs ermöglicht. Er stellt dem Anwender folgende Funktionen zur Verfügung

- Kommunikation über CAN-Bus
- Setzen der Init-Pegel für Debugzwecke
- Initialisierung der UCoMs und Vergabe der Nummer
- Programmierung des Anwenderprogramms über CAN-Bus
- Anzeige des gespeicherten Programms mit zugehöriger Check-Summe
- Starten des Anwenderprogramms

Er ist für die im Anhang B.2.2 beschriebene Initialisierungsphase zuständig. Üblicherweise kommuniziert auf PC-Seite das Linuxprogramm *dsp_console2can* oder *dsp_remote_part* mit der UCoM-Firmware. Das Konsolenprogramm *dsp_console2can* dient hauptsächlich zur initialen Programmieren eines Anwendungsprogramms in das UCoM und zu Debug-Zwecken (Abb. B.5). Das Programm bietet folgende Optionen, die im Folgenden kurz beschrieben werden sollen:

1: Reset All Resettet alle UCoMs, die am CAN-Bus angeschlossen sind. Die Antwort der UCoMs auf das Resetkommando wird ausgegeben, somit kann hierdurch beobachtet werden, welche UCoMs auf Nachrichten über den CAN-Bus reagieren.

2: Init Startet die Initialisierung der UCoMs. Gibt aus, welche UCoMs erfolgreich initialisiert wurden und gibt somit Aufschluss darüber, ob die Initialisierungsleitung korrekt verbunden ist.

3: GetInfo Gibt Informationen zu einem bestimmten UCoM an. Es werden Informationen wie in Abb. B.5 dargestellt ausgegeben, zum Beispiel der Programmname, das Programmierdatum oder die Checksumme. (Diese Funktion steht erst nach erfolgreich durchgeführter Initialisierung zur Verfügung).

4: SetProgEnable Mittels dieser Funktion kann man die Programmierung eines bestimmten oder aller UCoMs freischalten. (Diese Funktion steht erst nach erfolgreich durchgeführter Initialisierung zur Verfügung).

5: Program Startet die Programmierung der ausgewählten UCoMs mit dem geladenen Anwendungsprogramm. Es besteht die Möglichkeit ein einzelnes UCoM zu programmieren oder unter Angabe von „Alle“, die UCoMs die mittels Program Enable ausgewählt wurden. (Diese Funktion steht erst nach erfolgreich durchgeführter Initialisierung zur Verfügung).

6: ExtRAMLoad Mittels dieser Option kann das Anwendungsprogramm in das externe RAM auf dem UCoM geladen werden, anstatt ins Flash programmiert zu werden

7: Start Startet das ins Flash programmierte Anwendungsprogramm

8: SetAutoStart Setzt ein Flag, das dafür sorgt, dass das Anwendungsprogramm direkt nach einem Reset gestartet wird und die Bootloaderfunktion nicht ausgeführt wird

9: XReset Führt einen Reset aus bei dem das AutoStart-Flag gelöscht wird

10: SetInitIn Setzt für ein bestimmtes UCoM den Init-In Pin auf einen ausgewählten Pegel. Der gesetzte Pegel lässt sich an den Status-LEDs der UCoMs ablesen und dient somit zur Fehleranalyse. (Diese Funktion steht erst nach erfolgreich durchgeführter Initialisierung zur Verfügung).

11: SetInitOut Setzt den Init-Out Pin auf einen definierten Pegel. (Diese Funktion steht erst nach erfolgreich durchgeführter Initialisierung zur Verfügung).

12: SetSeqID Bietet die Möglichkeit, die durch die Initialisierung zugewiesene sequenzielle ID zu ändern. (Diese Funktion steht erst nach erfolgreich durchgeführter Initialisierung zur Verfügung).

13: Count Führt ein „GetInfo“ für alle UCoMs durch

14: Get First Fordert eine Info-Nachricht von dem UCoM, das als erstes in der CAN-Kette ist. Dient zur Überprüfung der korrekten Verkabelung der Initialisierungsleitung. Antworten beispielsweise zwei UCoMs auf dieses Kommando, lässt sich auf eine Unterbrechung der Initialisierungsleitung schließen.

15: Get Last Im Gegensatz zu Get First wird hier vom letzten UCoM in der CAN-Kette eine Info-Nachricht angefordert

16: DSPListUpdate Erstellt eine Liste aller am CAN-Bus aktiven UCoMs

17: Print DspList Gibt die Liste der am CAN-Bus aktiven UCoMs aus

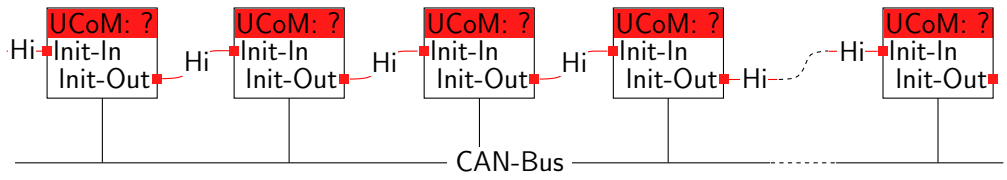
18: Load S-Record File Alternativ zur Angabe in der Kommandozeile beim Start des Programms, kann mittels dieser Option das Anwendungsprogramm zur Programmierung der UCoMs geladen werden

19: Calc ProgID Berechnet die Checksumme der geladenen S-Record Datei. Dieser Wert kann mit der Checksumme des im UCoM geladenen Programms verglichen werden.

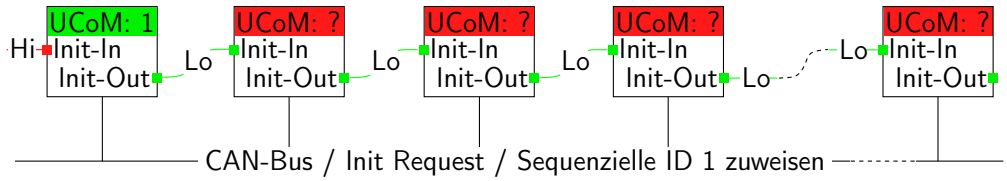
20: ProgAll Setzt automatisch für alle UCoMs das ProgEnable-Flag und führt dann eine Programmierung aller UCoMs aus

21: Reset Specific Resettet ein bestimmtes UCoM in der CAN-Kette. Nach dem Reset-Kommando muss noch die Nummer des zu resettenden UCoMs angegeben werden. (Diese Funktion steht erst nach erfolgreich durchgeführter Initialisierung zur Verfügung)

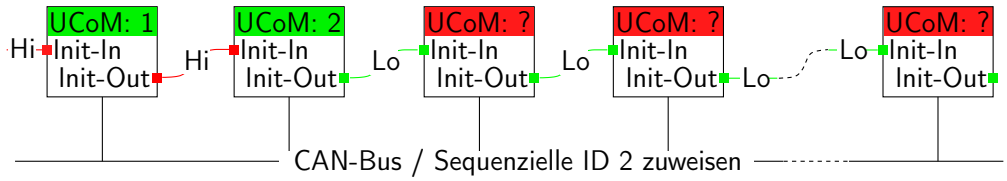
q: Quit Beendet das Programm



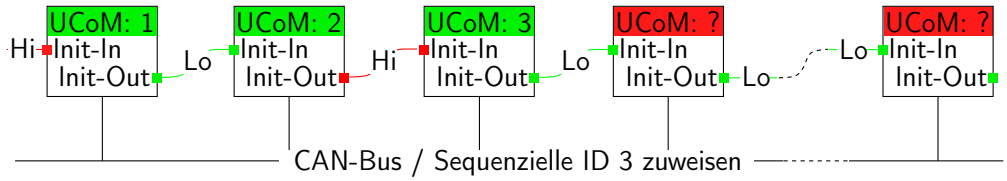
(a) Elektrischer Standardzustand bei Einschalten der UCoMs



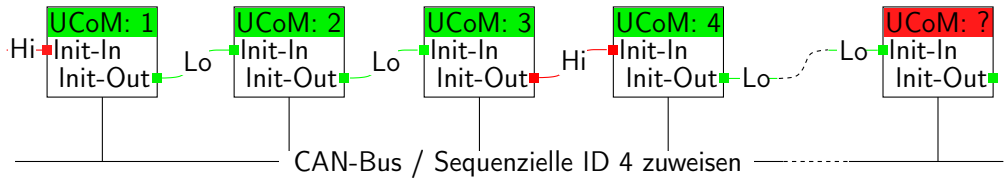
(b) Bootloader setzt alle Init-Ausgänge auf Lo, Nummer 1 wird zugewiesen



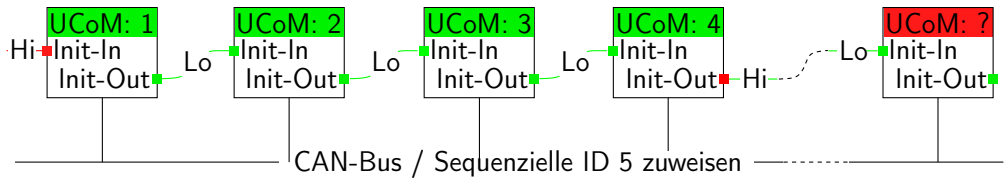
(c) UCoM 1 setzt seinen Init-Ausgang auf Hi, Nummer 2 wird zugewiesen



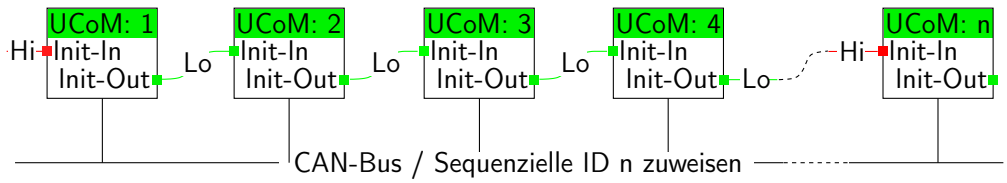
(d) UCoM 2 setzt seinen Init-Ausgang auf Hi, Nummer 3 wird zugewiesen



(e) UCoM 3 setzt seinen Init-Ausgang auf Hi, Nummer 4 wird zugewiesen



(f) UCoM 4 setzt seinen Init-Ausgang auf Hi, Nummer 5 wird zugewiesen



(g) UCoM n-1 setzt seinen Init-Ausgang auf Hi, Nummer n wird zugewiesen

Abb. B.3.: Schematischer Ablauf des Initialisierungsprozesses

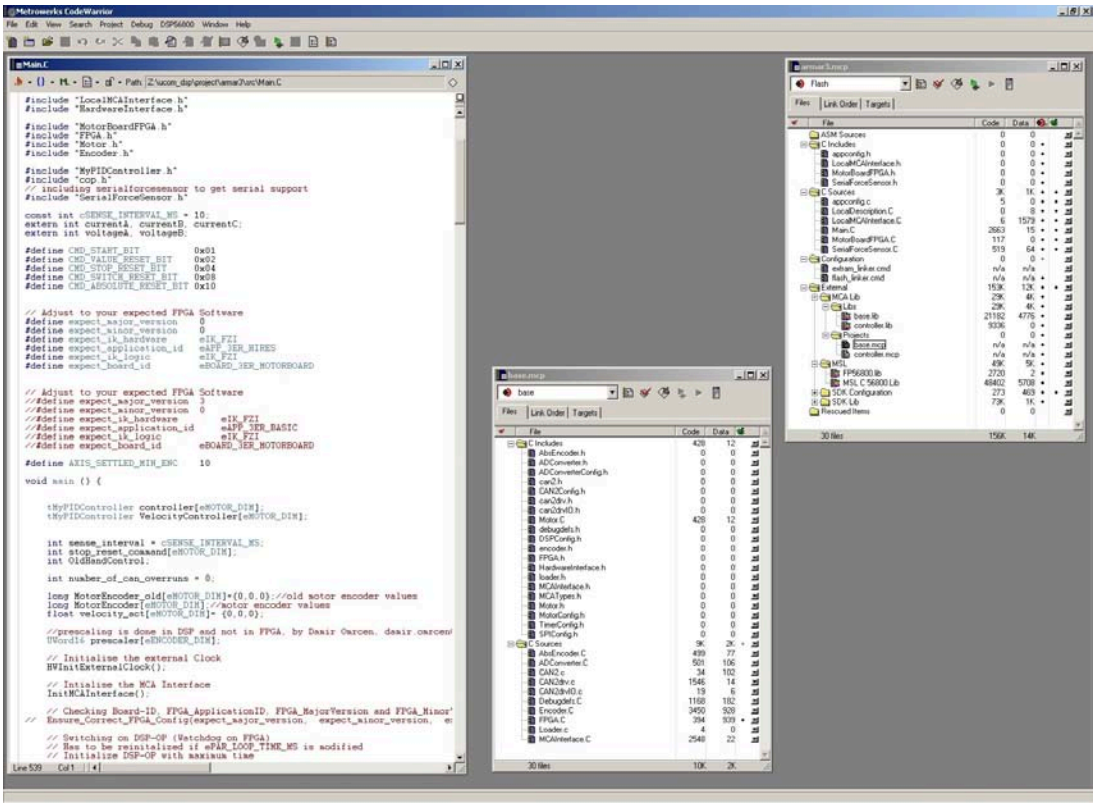


Abb. B.4.: Metrowerks Codewarrior 5.5: Screenshot der für die Entwicklung der Anwendungsprogramme auf dem DSP56F803 eingesetzten IDE

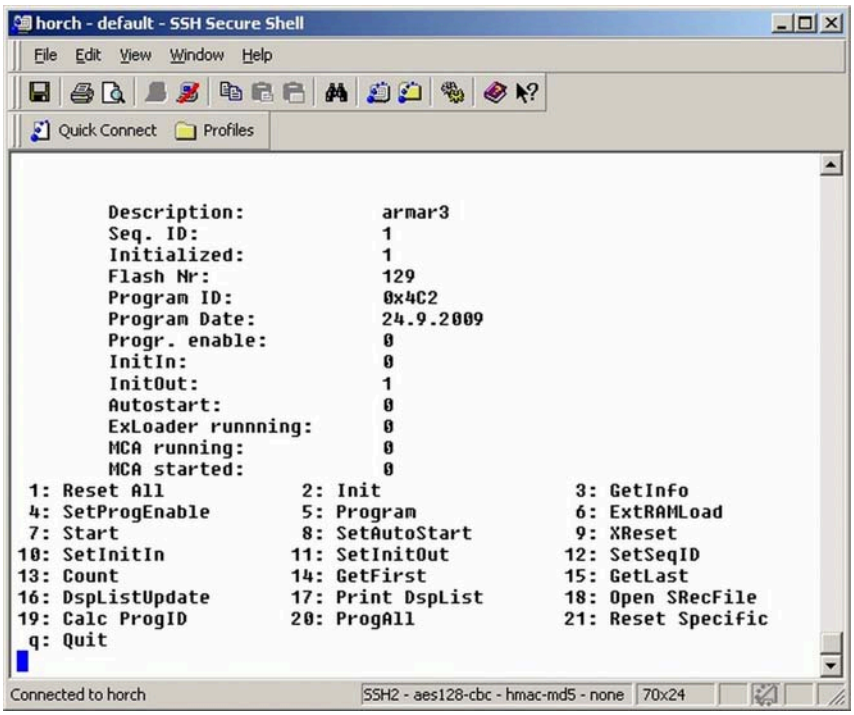


Abb. B.5.: Screenshot zu der Konsolenanwendung *dsp_console2can*, die zur Programmierung der UCoMs und zu Debugzwecken eingesetzt wird. Hier ist die Ausgabe des Aufrufs von GetInfo dargestellt.

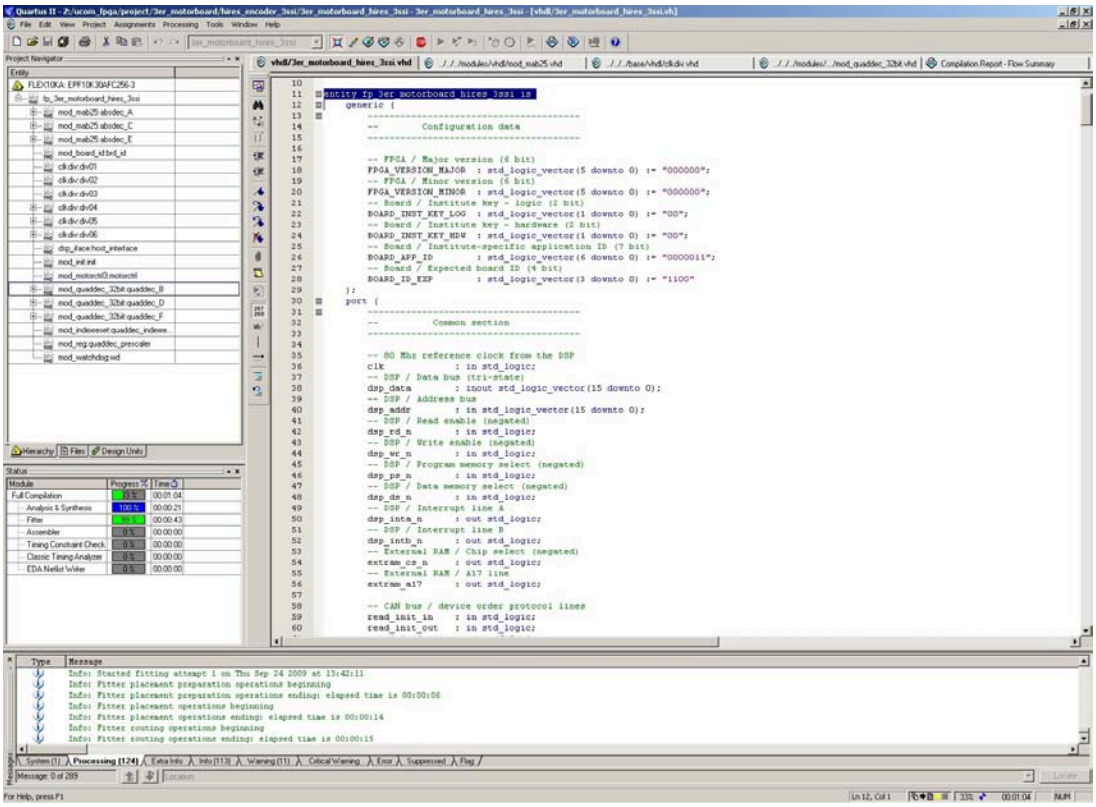


Abb. B.6.: Altera Quartus 7.2: Screenshot der Entwicklungsumgebung für die FPGA-Programme

B.3.2. FPGA

Als Entwicklungsumgebung für die Programmierung des FPGA kommt Quartus 7.2 von Altera[2] zum Einsatz (Abb. B.6). Im Zusammenspiel mit dem DSP sorgt der FPGA für sehr flexible Konfigurationsmöglichkeiten des UCoMs. Für das Zusammenspiel von DSP und FPGA bzw. zur Entlastung des DSPs sind die im Folgenden beschriebenen Funktionalitäten im FPGA realisiert.

Adressdecodierung

Die Grundlage für den Datenaustausch zwischen DSP und FPGA stellt die in Anhang B.1 beschriebene Adressdecodierung dar. Diese wird auf dem FPGA realisiert. Der FPGA analysiert das Bitmuster das am Adressbus anliegt und entscheidet anhand dieser Information, ob der Zugriff auf den externen Speicher oder den FPGA erfolgen soll. Aufgrund der hohen Taktrate der Speicherzugriffe, stellt diese Auswertung hohe Anforderungen an den FPGA. Zum Datenaustausch stellt der FPGA 64 Schreib-/Leseregister zur Verfügung.

[2] `http://www.altera.com/support/software/sof-quartus.html`

Quadraturdecodierung

Für die Auswertung der hochfrequenten, quadraturcodierten Signale der Positionsencoder der Motoren sind im FPGA Module zur Quadraturdecodierung implementiert. Diese Quadraturdecodierer liefern die aktuelle Position des Motors mit einer Auflösung von 16 bit zurück. Über einen *Prescaler* besteht die Möglichkeit für große Bewegungsbereiche ein Überlaufen des Zählers zu verhindern. Auf den Zählerstand kann nicht nur lesend zugegriffen werden, sondern er kann auch bei der Initialisierung in einer bekannten Position auf einen beliebigen Wert gesetzt werden.

Schnittstellenimplementierung

Neben den Quadraturzählern werden noch weitere Sensoren direkt an den FPGA angeschlossen. Für diese Sensoren sind verschiedene Schnittstellenprotokolle im FPGA realisiert. Beispielsweise unterstützt der DSP zwar die SPI-Schnittstelle, kann aber nur einen Knoten selektieren. Mittels des FPGAs wird diese Einschränkung aufgehoben und die Möglichkeit gegeben mehrere SPI-Slaves anzusteuern. Eine weitere verbreitete Schnittstelle bei Sensoren ist die SSI-Schnittstelle. Sie kommt beispielsweise auf den in *ARMAR-III* verwendeten Hall-Sensoren und dezentral platzierten Analog-Digital-Wandlern zum Einsatz. Auf dem FPGA ist diese Schnittstelle als Modul integriert und kann flexibel anstelle der Quadraturzähler an die Encoderschnittstelle herausgeführt werden.

Schutzeinrichtungen für Zusammenarbeit mit anderen Ansteckboards

Da das UCoM universell auch mit anderen Ansteckboards, die vielfältige Funktionalitäten ermöglichen, betrieben werden soll, muss über den FPGA sichergestellt werden, dass die richtige Anwendungssoftware geladen ist. Da auf dem UCoM diese beiden Komponenten unabhängig voneinander programmiert werden können, muss sichergestellt werden, dass diese Programme zueinander kompatibel sind. Bedingt durch das modulare Konzept des UCoM, welches unterschiedliche Ansteckboards zulässt, muss darüber hinaus noch überprüft werden, welches Ansteckboard gerade verwendet wird. Dies ist wichtig, da die Belegung des Verbindungssteckers zu großen Teilen durch das FPGA-Programm festgelegt wird und für unterschiedliche Ansteckboards variiert. Wenn beispielsweise das FPGA für die Verwendung eines Dreier-Motorboards konfiguriert ist, aber ein Ventiltreiberboard angesteckt wird, kann dies zu Fehlfunktionen führen. Im schlimmsten Fall kann dies sogar zu Schäden sowohl am Ansteckboard als auch am UCoM selbst führen. Zur Abstimmung und Kontrolle der richtigen Kombination von DSP- und FPGA-Programm werden folgende Aspekte herangezogen

- Board-ID
- Application-ID (App-ID)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IKL[1:0]		App-ID[6:0]							IKH[1:0]		0	Board-ID[3:0]			

Tab. B.5.: Versionsregister 1 für das FPGA-Programms

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Major-Version[5:0]						Minor-Verion[5:0]						Unused[3:0]			

Tab. B.6.: Versionsregister 2 für das FPGA-Programms

- Major Version
- Minor Version
- Institutskey-Logic (IKL)
- Institutskey-Hardware (IKH)

Das Konzept der Board-ID dient dazu, das angesteckte Board zu identifizieren. Sie wird durch ein Widerstandsnetzwerk auf dem Ansteckboard eingestellt. Mittels dieses Widerstandsnetzwerks können vier Bit eingestellt werden, somit besteht die Möglichkeit, insgesamt 16 Ansteckboards zu unterscheiden. Diese 4 Bit Zahl wird sowohl im FPGA als auch im DSP als zu erwartende Board-ID abgelegt. Das heißt, es gibt die Board-ID einerseits als Identifikation auf dem Ansteckboard und andererseits als Zuordnung eines FPGA-Programmes bzw. eines DSP-Programmes für ein bestimmtes Ansteckboard. Im Folgenden wird die über das Widerstandsnetzwerk auf dem Ansteckboard ausgelesene Kombination als Board-ID bezeichnet, die im FPGA-Programm abgelegte Zahl als FPGA-Board-ID und die im DSP-Programm hinterlegte Zahl als DSP-Board-ID.

Die Application-ID ist eine 7 Bit Nummer mit der die Anwendung des jeweiligen Boards näher beschrieben wird. Sie wird immer in Bezug auf eine Board-ID vergeben. Das heißt für jedes Board können 128 verschiedene Anwendungen beschrieben werden. Beispielsweise sind für ein spezifisches Board, abhängig vom jeweiligen Einsatzzweck, unterschiedliche Steckerbelegungen denkbar. So sind in der Standardanwendung für das Dreier-Motorboard alle Steckverbinder als Encodereingänge ausgelegt. In anderen Anwendungen werden weniger Encoder-Steckplätze benötigt, aber dafür ein digitaler Schaltausgang. Dieser Unterschied spiegelt sich dann in der Application-ID wider. Die Felder Major Version und Minor Version dienen zur Revisionskontrolle der entsprechenden Anwendung, beziehen sich also immer auf eine feste Kombination Board-ID und Application-ID. Die Felder Institutskey-Logic und Institutskey-Hardware dienen Verwaltungszwecken. In Tab. B.5 und Tab. B.6 sind die beiden Register zur Versionierung des FPGA-Programms beschrieben. Zur Vermeidung von Fehlfunktionen und Schäden an der Hardware wurde zur Vergabe der angesprochenen Felder folgendes Vorgehen festgelegt.

Die Board-ID wird schon FPGA-intern mit der FPGA-Board-ID verglichen. Für den Fall, dass diese beiden nicht übereinstimmen, geht das FPGA in einen neutralen Zustand über. Dieser neutrale Zustand zeichnet sich dadurch aus, dass alle Verbindungen zum Ansteckboard in einen sicheren Zustand versetzt werden. Für Eingänge bedeutet dies, dass die anliegenden Signale im FPGA nicht weiterverarbeitet werden und somit auch nicht zu einem Fehlverhalten führen können. Ausgänge werden in den hochohmigen Zustand versetzt. Die Kommunikation zwischen DSP und FPGA ist davon nicht betroffen, da diese komplett auf dem UCoM stattfindet. Da sämtliche Ausgänge zum Ansteckboard im hochohmigen Zustand sind, kann verhindert werden, dass an einem Ausgang ein Pegel ausgegeben wird, der das Ansteckboard beschädigt. Zusätzlich kann es nicht vorkommen, dass ein ausgegebener Pegel an dem tatsächlich verwendeten Ansteckboard eine andere Bedeutung hat und somit zu einer Fehlfunktion führt. Beispielsweise könnte ein Signal, das bei einem Sensor-Board die Datenwandlung anfordert, an einem Motortreiber-Board dazu führen, dass der Motor anläuft. Da das FPGA zum Ansteckboard keine Pegel ausgibt, kann es auch nicht zu dem für das FPGA schädlichen Zustand kommen, dass am Ansteckboard aktiv ein anderer Pegel anliegt als das FPGA auszugeben versucht. In diesem Zustand würden an dem entsprechenden Ausgang Ströme fließen, die unter Umständen zu einem Defekt des FPGA führen könnten. Durch das oben beschriebene Vorgehen, wird also sowohl das FPGA geschützt als auch das angeschlossene Ansteckboard.

Nachdem die FPGA-interne Überprüfung der Board-ID abgeschlossen ist, werden die Felder Application-ID, Major-Version, Minor-Version und die Institutsschlüssel vom DSP ausgelesen. Im DSP wird die Board-ID nochmals auf Übereinstimmung mit der DSP-Board-ID überprüft. Für den Fall, dass der DSP für eine andere Board-ID programmiert wurde, wird eine Fehlermeldung an der Konsole am PC ausgegeben und das DSP-Programm beendet sich. Nach der Kontrolle der Board-ID werden nacheinander noch die Application-ID, die Major und Minor Version und die Institutsschlüssel überprüft. Das Verhalten bei nicht übereinstimmender Application-ID ist genauso wie bei falscher Board-ID. Für die Vergabe der Versionsnummern gilt als Richtlinie, dass die Major Version bei einer Änderung, die eine Anpassung des DSP-Programms fordert, erhöht werden muss. Eine Erhöhung der Minor Version muss nicht zwangsläufig eine Änderung im DSP-Programm zur Folge haben. Aufgrund dieser Vorgabe wird bei einem Versionsunterschied in der Major Version auch das DSP-Programm abgebrochen und ein Reset durchgeführt. Bei einer Differenz in der Minor Version wird nur ein Warnhinweis auf der Konsole ausgegeben, das DSP-Programm aber normal ausgeführt.

B.4. Flexibilität des UCoMs

Das UCoM ist so gestaltet, dass eine flexible Erweiterbarkeit durch unterschiedliche Module gegeben ist. Dies kann sowohl durch entsprechende Ansteckboards geschehen als auch durch die Kombination des UCoM-Schaltungsteils und einer Erweiterungsplatine auf einer gemeinsa-

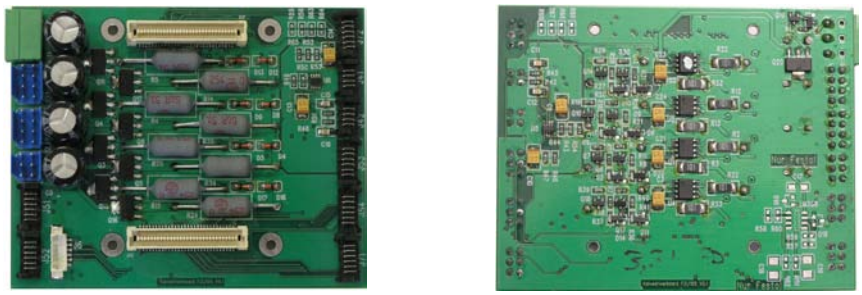


Abb. B.7.: Das Ventiltreiberboard (Sicht auf Ober- und Unterseite)

men Platine. Gesteckte Platinen werden dabei als UCoM@Ansteckboardname bezeichnet, die Nomenklatur für die auf einer Platine kombinierte Variante lautet UCoM-Ansteckboardname. Es wurden bereits die folgenden Module realisiert:

- UCoM@3M (Abb. 4.13 und 4.14): UCoM mit ansteckbarem Dreier-Motorboard
- UCoM@Ventiltreiberboard (Abb. B.7): UCoM mit angestecktem Ventiltreiberboard
- UCoM@Sensorboard (Abb. B.8): UCoM mit angestecktem Sensorboard
- UCoM@Hochstromboard: UCoM mit angestecktem Motortreiberboard für Ströme bis 30 A
- UCoM@Testboard: UCoM mit angestecktem Testboard zur Funktionsüberprüfung des UCoM
- UCoM-3M (Abb. B.9): Bauraumoptimierte Kombination von UCoM und Dreier-Motorboard für den Unterarm des humanoiden Roboters *ARMAR-III*
- UCoM-KairoAlpha (Abb. B.10(a)): UCoM mit Motortreiber für die Antriebselemente des Kanalroboters *Kairo-II*
- UCoM-KairoGamma (Abb. B.10(b)): UCoM mit Motortreiber für die Knickelemente des Kanalroboters *Kairo-II*

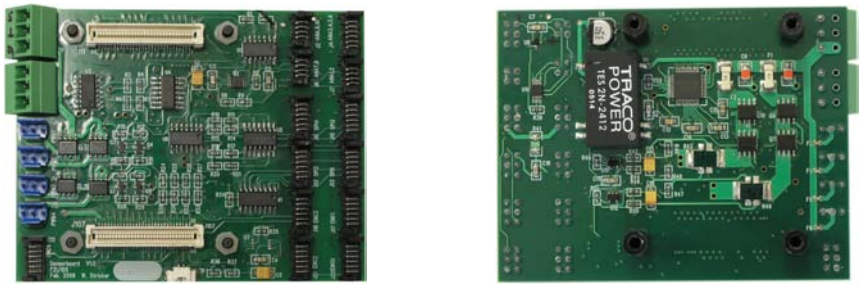


Abb. B.8.: Das Sensorboard (Sicht auf Ober- und Unterseite)

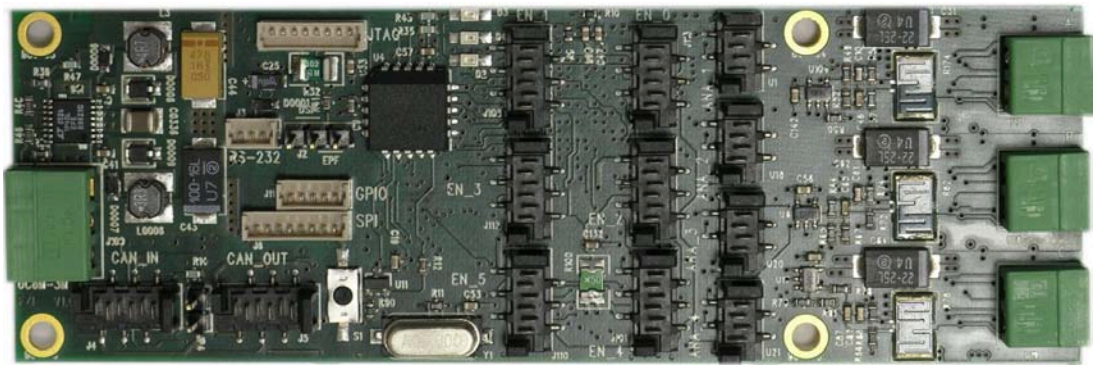
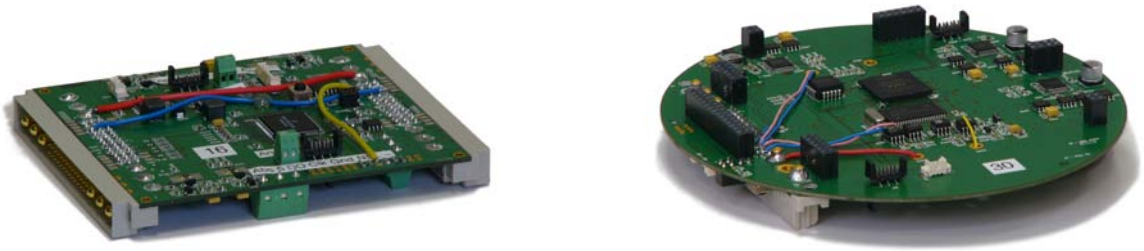


Abb. B.9.: Das UCoM-3M als Kombination aus UCoM und Dreier-Motorboard auf einer gemeinsamen Platine



(a) Alpha-Modul (b) Gamma-Modul

Abb. B.10.: Das *Kairo-II* Alpha-Modul als Kombination aus UCoM und zwei Motortreiberstufen und das Gamma-Modul als Kombination aus UCoM und drei Motortreiberstufen auf für den Kanalroboter angepassten Platinen

B.5. Konfiguration der UCoMs

Das Ausführungsprogramm auf dem UCoM lässt sich wie von MCA2 bekannt über vielfältige Parameter konfigurieren. Im Fall von *ARMAR-III* sind das beispielsweise die PID-Parameter zur Anpassung der Reglercharakteristik, Gelenkendanschläge, maximale Gelenkgeschwindigkeiten oder Einstellungen für die Quadraturdecoder. Neben den Parametern lassen sich auch Standardwerte für die *Controller Input* anpassen. Grundsätzlich können diese Werte mittels *mcabrowser* auch zur Laufzeit geändert werden. Mittels einer Konfigurationsdatei, dem so genannten *Attribute-Tree*, können diese Einstellungen auch automatisiert beim Start der MCA-Anwendung auf PC-Seite geladen werden.

Mittels des *Attribute-Tree* können beim Start der MCA-Anwendung auf dem PC die Standardwerte für Parameter und *Controller Input* auf dem UCoM gesetzt werden. Neben der Einstellung dieser Werte übernimmt der *Attribute-Tree* auch die Übersetzung der Kanten auf systemweit eindeutige Namen. Im UCoM sind die Kanten im Programm *armar3.fl.S* jeweils gleich benannt, damit dieses Programm generisch verwendet werden kann. Die Zuordnung der generischen Namen zu den eindeutigen Kantennamen wird im Abschnitt Mapping Table vorgenommen. Im Anschluss findet sich ein beispielhafter *Attribute-Tree* für das für die Ansteuerung der Augen zuständigen UCoMs.

```
DSP{
    0{
    Prefix:Eyes
        MappingTable{
          # CI
            0:MotorA Command ==> Right Eye Command
            1:MotorB Command ==> Left Eye Command
            2:MotorC Command ==> MotorC Command
            3:MotorA Target ==> Right Eye Target
            4:MotorB Target ==> Left Eye Target
            5:MotorC Target ==> MotorC Target
            6:MotorA Reset Offset ==> Right Eye Reset Offset
            7:MotorB Reset Offset ==> Left Eye Reset Offset
            8:MotorC Reset Offset ==> MotorC Reset Offset
            9:Send Command via serial ==> Send Command via serial
            10:MotorA TargetVel ==> Right Eye Target Vel
            11:MotorB TargetVel ==> Left Eye Target Vel
            12:MotorC TargetVel ==> MotorC Target Vel
          # SO
            13:Encoder 0 ==> Right Eye Joint Sensor
            14:Encoder 1 ==> Right Eye Motor Encoder
            15:Encoder 2 ==> Left Eye Joint Sensor
            16:Encoder 3 ==> Left Eye Motor Encoder
            17:Encoder 4 ==> Encoder 4
            18:Encoder 5 ==> Encoder 5
            19:Voltage A ==> Right Eye Voltage
            20:Voltage B ==> Left Eye Voltage
            21:Voltage C ==> Voltage C
            22:AD Channel 6 ==> AD Channel 6
            23:AD Channel 7 ==> AD Channel 7
            24:MotorA Reset Answer ==> Right Eye Reset Answer
            25:MotorB Reset Answer ==> Left Eye Reset Answer
```

```
    26:MotorC Reset Answer ==> MotorC Reset Answer
    27:Velocity A ==> Right Eye Velocity
    28:Velocity B ==> Left Eye Velocity
    29:Velocity C ==> MotorC Velocity
    30:Motor A Settled ==> Right Eye Axis Settled
    31:Motor B Settled ==> Left Eye Axis Settled
    32:Motor C Settled ==> MotorC Axis Settled
  # PAR
    33:LoopTime(ms) ==> LoopTime(ms)
    34:Max PWM ==> Max PWM
    35:Revert Encoder CountDir(BitMask) ==> Revert Encoder CountDir(BitMask)
    36:Set Joint Moving Direction(BitMask) ==> Set Joint Moving Direction(BitMask)
    37:Encoder Prescaler(BitMask) ==> Encoder Prescaler(BitMask)
    38:MotorA P ==> RightEye P
    39:MotorA I ==> RightEye I
    40:MotorA D ==> RightEye D
    41:MotorB P ==> LeftEye P
    42:MotorB I ==> LeftEye I
    43:MotorB D ==> LeftEye D
    44:MotorC P ==> MotorC P
    45:MotorC I ==> MotorC I
    46:MotorC D ==> MotorC D
    47:Reset A PWM ==> Reset RightEye PWM
    48:Reset B PWM ==> Reset LeftEye PWM
    49:Reset C PWM ==> Reset MotorC PWM
    50:MotorA feedforward gain ==> RightEye feedforward gain
    51:MotorB feedforward gain ==> LeftEye feedforward gain
    52:MotorC feedforward gain ==> MotorC feedforward gain
    53:MotorA Velocity P ==> RightEye Velocity P
    54:MotorA Velocity I ==> RightEye Velocity I
    55:MotorA Velocity D ==> RightEye Velocity D
    56:MotorA Max Velocity (x1000)==> RightEye Max Velocity (x1000)
    57:MotorB Velocity P ==> LeftEye Velocity P
    58:MotorB Velocity I ==> LeftEye Velocity I
    59:MotorB Velocity D ==> LeftEye Velocity D
    60:MotorB Max Velocity (x1000) ==> LeftEye Max Velocity (x1000)
    61:MotorC Velocity P ==> MotorC Velocity P
    62:MotorC Velocity I ==> MotorC Velocity I
    63:MotorC Velocity D ==> MotorC Velocity D
    64:MotorC Max Velocity (x1000) ==> MotorC Max Velocity (x1000)
    65:MotorA Max Position ==> RightEye Max Position
    66:MotorA Min Position ==> RightEye Min Position
    67:MotorB Max Position ==> LeftEye Max Position
    68:MotorB Min Position ==> LeftEye Min Position
    69:MotorC Max Position ==> MotorC Max Position
    70:MotorC Min Position ==> MotorC Min Position
}MappingTable

Parameter{
    LoopTime(ms):1
    Max PWM:14000
    Revert Encoder CountDir(BitMask):0x0
    Set Joint Moving Direction(BitMask):0x3
    Encoder Prescaler(BitMask):0x444
    Reset RightEye PWM:-7000
    Reset LeftEye PWM:-7000
    Reset MotorC PWM:7000
    RightEye P:50
    RightEye I:0.0005
    RightEye D:0
```

```
            LeftEye P:50
            LeftEye I:0.0005
            LeftEye D:0
            RightEye feedforward gain:1
            LeftEye feedforward gain:1
            MotorC feedforward gain:1
            RightEye Velocity P:1
            RightEye Velocity I:0.01
            RightEye Velocity D:0
            RightEye Max Velocity (x1000):500
            LeftEye Velocity P:1
            LeftEye Velocity I:0.01
            LeftEye Velocity D:0
            LeftEye Max Velocity (x1000):500
            MotorC Velocity P:0
            MotorC Velocity I:0
            MotorC Velocity D:0
            MotorC Max Velocity (x1000):1
            RightEye Max Position:10000
            RightEye Min Position:-10000
            LeftEye Max Position:10000
            LeftEye Min Position:-10000
            MotorC Max Position:30000
            MotorC Min Position:-30000
        }Parameter

        ControllerInput{
            Right Eye Command:0
            Left Eye Command:0
            Right Eye Target:0
            Left Eye Target:0
            Right Eye Reset Offset:-12725
            Left Eye Reset Offset:-11457
        }ControllerInput
    }0
}DSP
```

C. Vergleich UCoM - C167

Die in ARMAR-I verwendete Rechnerarchitektur basierte auf C167 Mikrocontrollern von Infineon. Es wurde ein Leistungsvergleich des C167 und des nun verwendeten DSP56F803 von Freescale durchgeführt [99]. Hierzu wurde die Architektur der beiden Bausteine einander gegenübergestellt und sowohl theoretische Betrachtungen der Leistungsfähigkeit als auch Tests mit realen Fragestellungen aus der Regelungstechnik durchgeführt.

C.1. Architektur des Mikrocontrollers

Als Mikrocontroller bezeichnet man die Kombination von zumindest Mikrorechner und Peripheriefunktionen auf einem Chip. Oft werden auch mehrere Zähler und Analog-Digital-Wandler auf dem Chip integriert. Ziel eines Mikrocontrollers ist es, mit möglichst wenigen externen Komponenten Rechenaufgaben und Kommunikation zu erledigen [26]. Sein Einsatz bietet sich somit vor allem bei einfachen Steuer- und Regelungsaufgaben an. Im Folgenden wird näher auf die Eigenschaften und die Architektur des C167 von Infineon eingegangen. Aus dem Blockschaltbild des C167 im User's Manual [77] lässt sich der Aufbau des C167 erkennen. Die wesentlichen Bestandteile der 16-Bit-CPU des C167 sind die „Arithmetisch Logische Einheit" (ALU), eine vierstufige Pipeline, General Purpose Register und mehrere Special Function Register. Der Mikrocontroller ist so aufgebaut, dass die meisten Befehle in einem Takt abgearbeitet werden können. Folgende Funktionen bilden hier eine Ausnahme: Sprünge benötigen zwei, 16×16-Bit-Multiplikationen benötigen fünf und 32-/16-Bit-Divisionen benötigen zehn Takte. Der C167 kann mit 25 MHz bzw. 33 MHz betrieben werden. Daraus ergibt sich für die Länge eines Taktes 40 ns bzw. 30 ns. Da in diesem Fall ein Maschinenzyklus jedoch aus zwei Takten besteht, ergibt sich für den Takt, in dem Befehle abgearbeitet werden, 80 ns bzw. 60 ns. Die verwendete Speicherarchitektur ist eine Von-Neumann-Architektur, das heißt, es gibt keinen separaten Programm- bzw. Datenspeicher, sondern einen gemeinsamen Adressraum. Die Gesamtgröße des adressierbaren Speichers beträgt 16 Megabyte und ist byte- und wortweise ansprechbar. Der weitere Speicherausbau sieht folgendermaßen aus: auf dem Chip stehen für Programm-Code und unveränderliche Daten 128 Kilobyte maskenprogrammierbares Read-Only Memory zur Verfügung. Das interne RAM (IRAM) bietet 2 Kilobyte Platz für benutzerdefinierte Daten und den System-Stack. Weitere 2 Kilobyte bietet das externe RAM (XRAM). Zusätzlich können extern jeweils bis zu 16 Megabyte ROM bzw. RAM adressiert werden. An Peripherie verfügt der C167 über einen Analog-Digital-Wandler, zwei Capture-Compare-Einheiten, eine

PWM-Einheit mit vier Ausgängen, zwei Zeitgebereinheiten mit jeweils fünf Zeitgebern und eine Watchdogeinheit. Der Analog-Digital-Wandler arbeitet mit einer Auflösung von 10 Bit und einer minimalen Samplezeit von $7,8\,\mu s$. Untereinander sind die Speichereinheiten über Busse verbunden. Der C167 verfügt im Wesentlichen über vier Speicherbusse:

1. Instruktions-/Daten-Bus: Verbindet den Prozessorkern mit den Speichereinheiten

2. On-Chip XBUS: Verbindet das XRAM mit dem CAN-Modul und dem „externen Bus Controller“ (EBC)

3. Externer Instruktions-/Daten-Bus: Stellt die Verbindung zwischen Kern und externem Speicherbus-Controller her

4. Peripherie Datenbus: Dient zum Ansprechen der Peripherie, wie zum Beispiel der Einheiten wie PWM, Zähler oder Analog-Digital-Wandler

Sämtliche Busse sind 16 Bit breit und können somit in einem Taktzyklus eine Instruktion bzw. ein Datum übertragen. Zur Kommunikation mit anderen Komponenten verfügt der C167 über eine CAN-Schnittstelle in der Revision 2.0B [76]. In den beiden folgenden Absätzen wird noch näher auf den Kern des C167 und die Pipelinestruktur eingegangen, um später die Unterschiede zur Architektur des DSP56F803 von Freescale klar herausstellen zu können.

C.1.1. C167-Kern-Komponenten

Die Aufgabe der CPU ist es, Instruktionen und Operanden zu laden, auszuführen und die Ergebnisse zu speichern. Der Zugriff auf den internen Speicher kann direkt erfolgen, der Zugriff auf den externen Speicher und die Peripherie wird jedoch über den EBC geregelt. Der Vorteil dieser Trennung und des Einsatzes des EBC, liegt darin, dass die CPU ungestört arbeiten kann, solange sie nicht auf Daten die vom EBC angefordert wurden, warten muss. Dieser Gedanke wird auch für die Peripherie aufgegriffen und so kann die Peripherie dank eigener Zeitgeber größtenteils unabhängig von der CPU arbeiten. Der reguläre Datenaustausch erfolgt über so genannte Spezialregister, nur für die Signalisierung unerwarteter Ereignisse werden Interrupts verwendet.

C.1.2. C167-Instruction-Pipelining

Mittels seiner vierstufigen Pipeline erreicht der C167 im Idealfall eine Geschwindigkeitssteigerung gegenüber sequenzieller Ausführung um den Faktor 4. Allgemein erhält man bei einer

k-stufigen Pipeline und der Abarbeitung von n Befehlen eine Beschleunigung um den Faktor

$$S = \frac{n \cdot k}{k + (n-1)}$$

$$\lim_{n \to \infty} S = k$$

Daraus lässt sich auch die oben angegebene vierfache Geschwindigkeit für die vierstufige Pipeline ablesen.

C.2. Architektur des digitalen Signalprozessors

Unter einem digitalen Signalprozessor (DSP) versteht man einen Prozessor, der speziell auf die schnelle Verarbeitung digitaler Signale optimiert ist. Damit auch analoge Signale verarbeitet werden können, ist oft direkt im DSP ein Analog-Digital-Wandler integriert. Zur effizienten Verarbeitung digitaler Daten wird einerseits eine spezialisierte Recheneinheit benötigt, andererseits sind schnelle breitbandige Bussysteme notwendig, die für den Datentransfer zwischen Recheneinheit und Speicher sorgen. Der Kern des verwendeten *16-Bit Hybrid Controller 56F803* von Freescale kann mit bis zu 80 MHz getaktet werden. Die drei Hauptkomponenten des DSP56F803 sind der Programm-Controller, die „Adress-Berechnungs-Einheit“ (AGU) und die „Arithmetisch-Logische-Einheit“ (ALU). Diese drei Einheiten arbeiten nach dem Prinzip einer Pipeline parallel zueinander und sorgen damit für eine gute Auslastung der Hardware-Komponenten. Durch die geschickte Anordnung der am häufigsten genutzten Register ergibt sich ein weiterer Geschwindigkeitsvorteil, da durch diese Anordnung der Datentransfer über allgemeine Bussysteme entfällt. Dank dieser Maßnahmen ist der DSP56F803 in der Lage sämtliche arithmetisch-logischen Operationen in einem Takt auszuführen. Durch die dezidierte „Multiply-and-Accumulate-Einheit“ (MAC) ist er in der Lage, sogar eine 16×16 Bit Multiplikation mit anschließender Addition in dieser Zeit auszuführen. Die MAC-Einheit spielt ihre Vorteile vor allem bei den bei Regelungsaufgaben häufig vorkommenden Polynomberechnungen aus und eignet sich somit sehr gut für den Einsatz im Roboter. Seitens der Speicherarchitektur kommt beim DSP eine Harvard-Architektur zum Einsatz, die die schnelle Datenverarbeitung nicht durch Speicherzugriffe verlangsamt. Der Adressbereich des DSPs beträgt jeweils 2^{16} Wörter. Die detaillierte Speicherausstattung des DSPs ist wie folgt: Programm-Speicher von 31,5 K$\times$16 Bit Flash und 512$\times$16 Bit RAM; Datenspeicher von 4 K$\times$16 Bit Flash, 2 K$\times$16 Bit RAM und 2 K$\times$16 Bit Bootflash. Zur Verbindung dieser Speicherstrukturen kommen mehrere Bussysteme zum Einsatz, die im Blockschaltbild des DSP56F803 [55] ebenso wie die Peripherie detailliert beschrieben sind. Die für die Anwendung im Roboter relevante Peripherie soll hier kurz erwähnt werden. Der DSP56F803 verfügt über zwei vierkanalige Analog-Digital-Wandler mit einer Auflösung von 12 Bit, 4 Allzweck-Timer, einen sechskanaligen Pulsweitenmodulator

(PWM-Einheit) sowie weitere 16 Allzweck-I/O-Pins die wahlweise als Ein-/Ausgänge oder als serielle Schnittstelle bzw. als SPI-Schnittstelle beschaltet werden können. Als Kommunikationsschnittstelle verfügt der DSP56F803 über ein CAN 2.0 A/B-Modul.

C.2.1. Aufbau des DSP-Kerns

Die Hauptkomponenten des DSP-Kerns, also die ALU, die AGU und der Programm-Controller verfügen über jeweils eigenständige Register-Sätze und eine unabhängige Kontroll-Logik, um ein paralleles Arbeiten zu gewährleisten. Der DSP-Kern bekommt die für die Berechnungen benötigten Daten aus dem internen Programm- und Datenspeicher, über die externe Speicher-Schnittstelle (EMI) bzw. von den diversen Peripheriekomponenten. Für die DSP56800-Kerne stehen zwei Speicher-Schnittstellen zur Auswahl: Einerseits der „Peripheral Global Data Bus“ (PGDB) und die beim DSP56F803 tatsächlich eingesetzte „Freescale Standard IP-Bus“-Schnittstelle. Bei dieser Schnittstelle können Peripherie-Register in regulär vorhandenen Speicher abgebildet werden und somit kann der Zugriff sehr einfach über standardmäßige Read- und Write-Befehle erfolgen. Da sämtliche Kommunikation zwischen den Komponenten über schnelle Adress- und Datenbusse erfolgt, können die drei folgenden Operationen in einem Instruktionszyklus abgearbeitet werden und bilden damit sozusagen eine dreistufige Pipeline: Holen der Instruktion, zwei Adressberechnungen in der AGU und eine Berechnung in der ALU bzw. bei Shift-Operationen der Bit-Manipulations-Einheit.

C.2.2. DSP-Kern-Komponenten

Wie im obigen Absatz erwähnt, besitzt die Data-ALU eine eigene Kontroll-Logik und einen separaten Registersatz. Dieser setzt sich aus drei 16 Bit Input-Registern (X0, Y0 und Y1) zusammen. Diese drei Register können auch als ein 16 Bit Register X0 und ein zusammengefasstes 32 Bit Register Y angesprochen werden. Weiterhin gibt es noch die zwei 36 Bit Akkumulator-Register (A und B), welche sich auch in drei Sektionen ansprechen lassen: zwei 4 Bit Extension-Register (A2 und B2) und die 16 höherwertigen (A1 und B1) bzw. niederwertigen Bits (A0 und B0). Weitere Details werden im Family-Manual der DSP56F80x-Reihe [54] dargestellt und sollen hier nicht weiter vertieft werden. Der Vorteil an dieser flexiblen Registerstruktur ist, dass einerseits eine sehr hohe Präzision bei den Datenstrukturen ermöglicht wird und andererseits die Register mit der geringeren Bit-Breite in großer Anzahl für System- und Kontrollaufgaben bereitstehen.

C.2.3. DSP-MAC-Einheit

Die MAC-Einheit ist ein besonderes Merkmal digitaler Signalprozessoren. Mittels der MAC-Einheit ist die Verarbeitung von drei 16 Bit-Operanden in einem Instruktionszyklus möglich.

Die ersten beiden Operanden werden in einer 16×16 Bit Multiplikation miteinander multipliziert und schließlich der dritte Operand zum Ergebnis addiert, welches als 36 Bit Wert gespeichert wird. Die Eingaben werden direkt aus dem Data-ALU-Register bezogen und genau wie das Ausgaberegister im gleichen Takt sowohl gelesen als auch geschrieben. Die drei Operanden für die Berechnung können aus folgenden Quellen stammen:

- X, Y Register (also X0, Y0, Y1)
- Akkumulatoren
- Direkt aus dem Speicher

C.3. Leistungsvergleich

Wie in Abschnitt 5.4.1 beschrieben, wurde die Leistungsfähigkeit von UCoM und C167 in verschiedenen Benchmarks evaluiert. Die ersten drei Benchmarks wurden bereits besprochen, die Ergebnisse der verbleibenden drei – Ackermann, Sieb des Eratosthenes und Fibonacci – werden in den folgenden Absätzen aufgeführt.

C.3.1. Benchmark - Ackermann

Für diesen Test wurde eine vereinfachte Version der Ackermann-Funktion von Rózsa Péter benutzt, die folgende Definition besitzt:

$$\begin{aligned} a(0,j) &= j+1 \\ a(k+1,0) &= a(k,1) \\ a(k+1,j+1) &= a(k,a(k+1,j)) \end{aligned}$$

Diese Funktion ergibt sehr schnell wachsende Ergebnisse. Im konkreten Fall wurde fünfmal hintereinander der Wert von $a(3,3)$ berechnet.

Theoretische Laufzeit des DSP

Die theoretische Laufzeit beim DSP ergab sich aus dem Assemblercode bei ausgeschalteter Optimierung zu:

$$n_{ges} = n_{Funktionsaufruf} + \sum_{i=0}^{3} (n_{Fall_i} \cdot n_{Ticks_i})$$

Die Laufzeit berechnet sich damit zu

$$t_{Lauf} = \frac{n_{ges}}{2} \cdot t_{Taktzyklus}$$

Die Anzahl Taktzyklen berechnet sich zu

$$n_{ges} = 264 + (12\,160 \cdot 44 + 5\,940 \cdot 14 + 285 \cdot 26 + 5\,935 \cdot 36) = 839\,534$$

und ergeben insgesamt eine theoretische Laufzeit von

$$t_{Lauf} = \frac{839\,534}{2} \cdot 25\,\text{ns} = 10,494175\,\text{ms}$$

Gemessene Laufzeit beim DSP

Die durch Laufzeitmessung ermittelte Zeit zur Ausführung lag beim DSP ohne Optimierungen bei 3 280 Takten also $3\,280 \cdot 3,2\,\mu\text{s} = 10,496\,\text{ms}$. Bei Übersetzung des Codes mit den anderen Optimierungsstufen gab es am Assembler-Code keine wesentlichen Änderungen und somit benötigte die Ausführung des Programms in sämtlichen Varianten immer die gleiche Anzahl Taktzyklen.

Gemessene Laufzeit beim C167

Die Ackermann-Funktion benötigte bei der Ausführung auf dem C167 insgesamt 40 299 Takte. Da in diesem Fall die gleiche Zeitauflösung wie beim DSP gewählt wurde, errechnet sich die benötigte Zeit zu $40\,299 \cdot 3,2\,\mu\text{s} = 128,9568$ ms.

C.3.2. Benchmark - Sieb des Eratosthenes

Das Sieb des Eratosthenes ist ein Algorithmus mit dessen Hilfe alle Primzahlen bis zu einer angegebenen maximalen Zahl bestimmt werden. Die Funktionsweise des Algorithmus sieht wie folgt aus: Anfangs sind alle natürlichen Zahlen zwischen 2 und dem Maximum unmarkiert, daraufhin werden sukzessive für alle Zahlen bis zum Maximum sämtliche Vielfache berechnet und markiert. Nach Abschluss dieses Verfahrens sind nur noch Primzahlen unmarkiert und somit alle Nicht-Primzahlen sozusagen „herausgesiebt“. Für die Messung wurde das Maximum auf 250 gesetzt und die Berechnung zehnmal in Folge ausgeführt.

Theoretische Laufzeit beim DSP

Im Programmcode sieht man, dass bei dieser Aufgabe viele Schleifen und If-Anweisungen enthalten sind, was eine korrekte Vorhersage der Laufzeit erschwert. Für die Berechnung kann hier

keine einfache Formel angegeben werden und wird deshalb auch nur exemplarisch aufgeführt:

$$n_{ges} = n_{Funktionsaufruf} + n_{Funktion}$$
$$t_{Lauf} = \frac{n_{ges}}{2} \cdot t_{Taktzyklus}$$

In diesem Fall ergibt sich folglich diese Zeit:

$$n_{ges} = 43 + 340\,338 = 340\,381$$
$$t_{Lauf} = \frac{340\,381}{2} \cdot 25\text{ ns} = 4,2547625\text{ ms}$$

Gemessene Laufzeit beim DSP

Die am DSP gemessene Laufzeit schwankte zwischen 1 338 und 1 339 Taktzyklen, was einer Ausführungsdauer von 4,2816 ms bzw. 4,2848 ms entspricht. Angesichts der schweren Analysierbarkeit des Assemblercodes liegt das gemessene Ergebnis im Rahmen der Vorhersage. Beim Sieb des Eratosthenes ergab der Wechsel von „Keine Optimierung“ zu „Level 1 - Optimierung“ noch keine Änderung. Bei Wechsel von „Level 1“ auf „Level 2“ änderten sich die benötigten Taktzyklen auf 1 146, also 3,6672 ms. Obwohl sich der Code kaum änderte, kommt es durch die häufige Ausführung zu einer nicht unwesentlichen Beschleunigung. Die Optimierungen „Level 3“ und „Level 4“ ergaben eine weitere Verbesserung um circa 0,2 ms auf 3,41144 ms.

Gemessene Laufzeit beim C167

Beim C167 wurden für die Ausführung des Siebs des Eratosthenes 6 355 bzw. 6 356 Taktzyklen benötigt. Mit der auch hier verwendeten Einstellung des Zählers ergibt das eine Zeit von $6\,355 \cdot 3,2\,\mu\text{s} = 20,336$ ms bzw. 20,3392 ms bei 6 356 Takten.

C.3.3. Benchmark - Fibonacci

Die Berechnung der Fibonacci-Zahlen wird auf folgende Weise rekursiv definiert

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}$$

Es werden also die beiden ersten Elemente vorgegeben und alle weiteren berechnen sich aus der Summe der beiden Vorläufer-Elemente. Es ergibt sich die Folge:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1 597, 2 584, 4 181, 6 765, ...
Bei diesem Laufzeittest wurde die Berechnung von f_{20} dreimal durchgeführt, wobei in diesem Fall die beiden Startelemente der Rekursion mit $f_1 = f_2 = 1$ festgelegt wurden.

Theoretische Laufzeit beim DSP

Für das Programm ergibt sich hier folgende Laufzeit:

$$n_{ges} = n_{Funktionsaufruf} + \sum_{i=0}^{2} (n_{Fall_i} \cdot n_{Ticks_i})$$
$$t_{Lauf} = \frac{n_{ges}}{2} \cdot t_{Taktzyklus}$$

In diesem Fall ergibt sich folglich diese Zeit:

$$n_{ges} = 210 + 40\,587 \cdot 40 + 20\,292 \cdot 52 + 20\,295 \cdot 4 = 2\,760\,054$$
$$t_{Lauf} = \frac{2\,760\,054}{2} \cdot 25\text{ ns} = 34,500675\text{ ms}$$

Gemessene Laufzeit beim DSP

Bei der Messung ergaben sich Laufzeiten von 10 781 bzw. 10 782 Taktzyklen, was Ausführungszeiten von 34,4992 ms und 34,5024 ms entspricht. In diesem Fall liegt die kleinere der tatsächlich ermittelten Laufzeiten unter der theoretisch ermittelten, was sich aber durch die in $3,2\,\mu$s gerasterte Zeitmessung erklären lässt. Für die Berechnung der Fibonacci-Zahlen ergab nur der Wechsel von „Keine Optimierung“ zu „Level 1“ eine Änderung auf 10 573 bis 10 575 Taktzyklen, also eine Laufzeit zwischen 33,8336 ms und 33,840 ms. Die weiteren Optimierungsstufen brachten keine Verbesserung der Laufzeit mehr.

Gemessene Laufzeit beim C167

Da bereits beim DSP die Ausführung viel Zeit in Anspruch nahm, stand zu erwarten, dass die Ausführung auf dem C167 noch länger dauern würde. Um zu verhindern, dass der Zähler beim C167 überläuft, wurde die Zeitauflösung halbiert, also auf $t_{Tick} = 6,4\,\mu$s eingestellt. Als Ergebnis kamen 59 454 bzw. 59 455 Taktzyklen heraus. Was wiederum Zeiten von 380,5056 ms und 380,512 ms entspricht.

Gegenüberstellung und Bewertung der Messungen

Beim Ackermann-Benchmark wird nicht so klar wie beim Whetstone-Benchmark nach einzelnen Modulen, die speziell auf eine Befehls- bzw. Zugriffsart abzielen, unterschieden. Deshalb hat das Ergebnis des Ackermann-Benchmarks vor allem für diese spezielle Zusammensetzung

seine Aussagekraft. Der DSP führt das Ackermann-Benchmark um den Faktor 12,286 schneller aus als der C167. Für das Sieb des Eratosthenes und die Fibonacci-Funktion gilt bezüglich der Aussagekraft das gleiche, wie für das Ackermann-Benchmark. Beim Sieb des Eratosthenes ist der DSP nur um den Faktor 4,75 schneller. Die Berechnung der Fibonacci-Zahlen ist mit dem Faktor 11,03 auf dem DSP wiederum deutlich schneller.

Durch die klare Aufteilung in einzelne Module, die bestimmte Aspekte testen, lässt sich beim Whetstone-Benchmark erkennen, wo die Stärken und Schwächen der beiden Controller-Boards liegen. In Abb. 5.17, sind die Laufzeiten in den jeweiligen Modulen bei C167 und DSP grafisch gegeneinander aufgetragen.

C.4. Assemblersourcen zu den Benchmarks

Im Folgenden ist exemplarisch der Assemblercode für das Ackermann-Benchmarks abgedruckt, die verbleibenden Benchmarks werden in [99] behandelt. Die angegebenen Assemblersourcen sind das Ergebnis der Übersetzung ohne Optimierung. TZ gibt dabei an wie viele Taktzyklen zur Ausführung des Befehls benötigt wurden.

```
-- Ackermann ------------------------------------------------------------
;
;  244:
;  245:
;  246:
;  247: long A(int x,int y)
;  248: {
;  249:          if (x == 0)
;

-- Beginn Fall_0 --------------------------------------------------------

0x00000000                    FA:                                    TZ
0x00000000   0xDE0B                    lea      (SP)+                  2
0x00000001   0xBD38                    moves    X:0x0038,N             2
0x00000002   0xDD0B                    move     N,X:(SP)+              2
0x00000003   0xBD39                    moves    X:0x0039,N             2
0x00000004   0xDD1F                    move     N,X:(SP)               2
0x00000005   0x9138                    moves    Y0,X:0x0038            2
0x00000006   0x9339                    moves    Y1,X:0x0039            2
0x00000007   0xBE38                    tstw     X:0x0038               2
0x00000008   0xA206                    bne      *+7       ; 0x00000f   6
;
;  250:                  return y+1;
;

-- Ende Fall_0 ----------------------------------------------------------
-- Beginn Fall_1 --------------------------------------------------------

0x00000009   0xB039                    moves    X:0x0039,X0            2
0x0000000A   0x6431                    incw     X0                     2
0x0000000B   0x6CB0                    clr      B                      2
0x0000000C   0x8180                    move     X0,B0                  2
0x0000000D   0x6C00                    tfr      B,A                    2
```

```
0x0000000E  0xA912              bra     *+19    ; 0x000021      4
;
;  251:                 else if (y == 0)
;

-- Ende Fall_1 ------------------------------------------------------
-- Beginn Fall_2 und Fall_2 -----------------------------------------

0x0000000F  0xBE39              tstw    X:0x0039                2
0x00000010  0xA206              bne     *+7     ; 0x000017      6
;
;  252:                 return A(x-1,1);
;  253:         else
;

-- Ende Fall_3 ------------------------------------------------------

0x00000011  0xB138              moves   X:0x0038,Y0             2
0x00000012  0x6413              decw    Y0                      2
0x00000013  0xC301              movei   #1,Y1                   2
0x00000014  0xE9C80000          jsr     0x000000                8
0x00000016  0xA90A              bra     *+11    ; 0x000021      4
;
;  254:                 return A(x-1,A(x,y-1));
;

-- Ende Fall_2 ------------------------------------------------------
-- Beginn Fall_3 ------------------------------------------------------

0x00000017  0xB339              moves   X:0x0039,Y1             2
0x00000018  0x6493              decw    Y1                      2
0x00000019  0xB138              moves   X:0x0038,Y0             2
0x0000001A  0xE9C80000          jsr     0x000000                8
0x0000001C  0x8308              move    A0,Y1                   2
0x0000001D  0xB138              moves   X:0x0038,Y0             2
0x0000001E  0x6413              decw    Y0                      2
0x0000001F  0xE9C80000          jsr     0x000000                8
;
;  255: }

-- Ende Fall_3 ------------------------------------------------------
-- Beginn Fall_0 ------------------------------------------------------

0x00000021  0xFD1B              move    X:(SP)-,N               2
0x00000022  0xE040              nop                             2
0x00000023  0x9D39              moves   N,X:0x0039              2
0x00000024  0xFD1B              move    X:(SP)-,N               2
0x00000025  0xE040              nop                             2
0x00000026  0x9D38              moves   N,X:0x0038              2
0x00000027  0xEDD8              rts                            10

-- Ende Fall_0 ------------------------------------------------------
```

D. Schaltungsdetails zu UCoM und Motorboard

In diesem Kapitel werden technische Details wie Schaltpläne, Layoutskizzen und Steckerbelegungen für das UCoM und das Dreier-Motorboard wiedergegeben.

D.1. Schaltplan

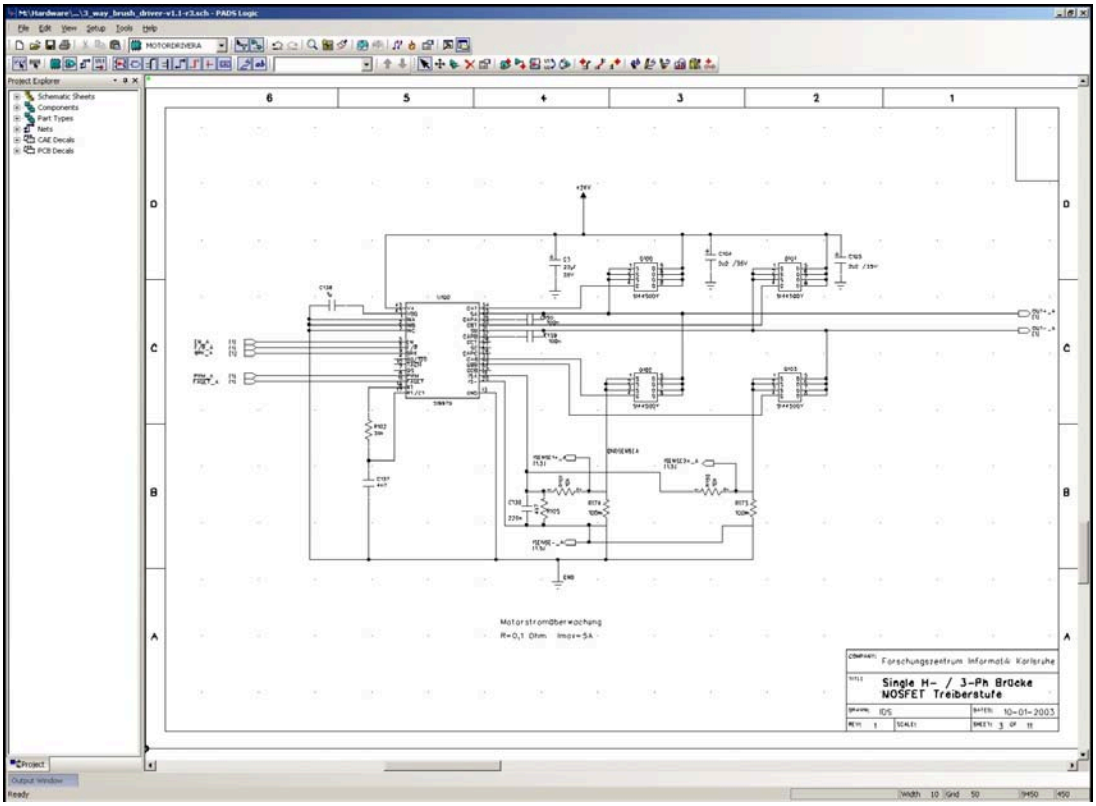
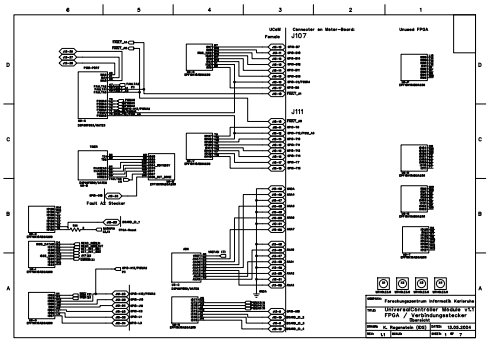
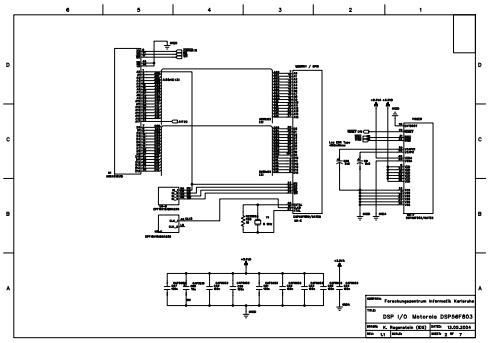


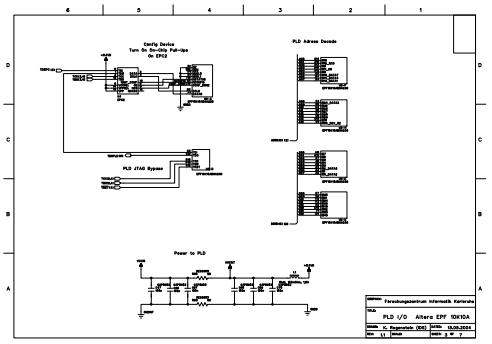
Abb. D.1.: Screenshot der zum Schaltplanentwurf verwendeten Software PADS Logic. Es ist ein Ausschnitt aus dem Schaltplan des Dreier-Motorboards – die Beschaltung eines H-Brückentreibers zur Motoransteuerung – dargestellt.



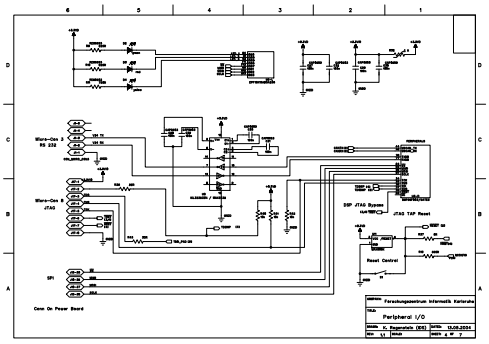
(a) Schnittstelle zum Erweiterungsboard



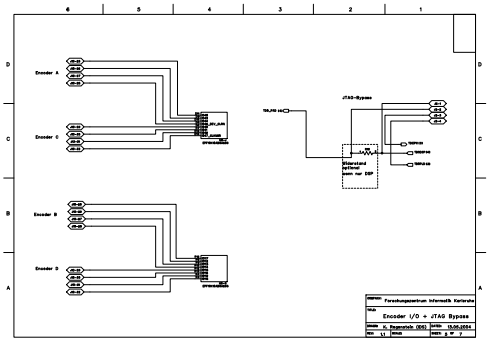
(b) DSP-Einbindung



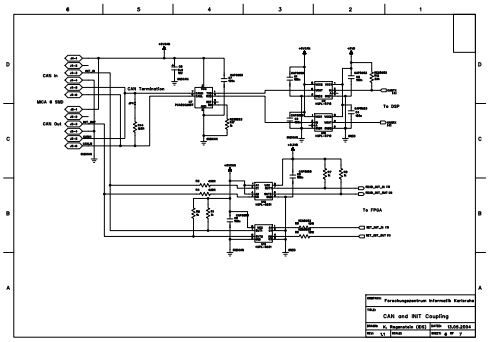
(c) FPGA-Einbindung



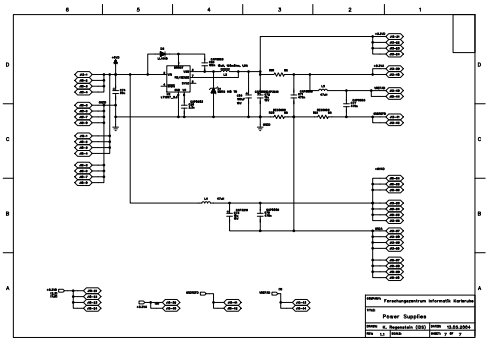
(d) Peripherie



(e) Encoder



(f) CAN-Schnittstelle



(g) Spannungsversorgung

Abb. D.2.: Schaltpläne des UCoMs

D.2. Layout

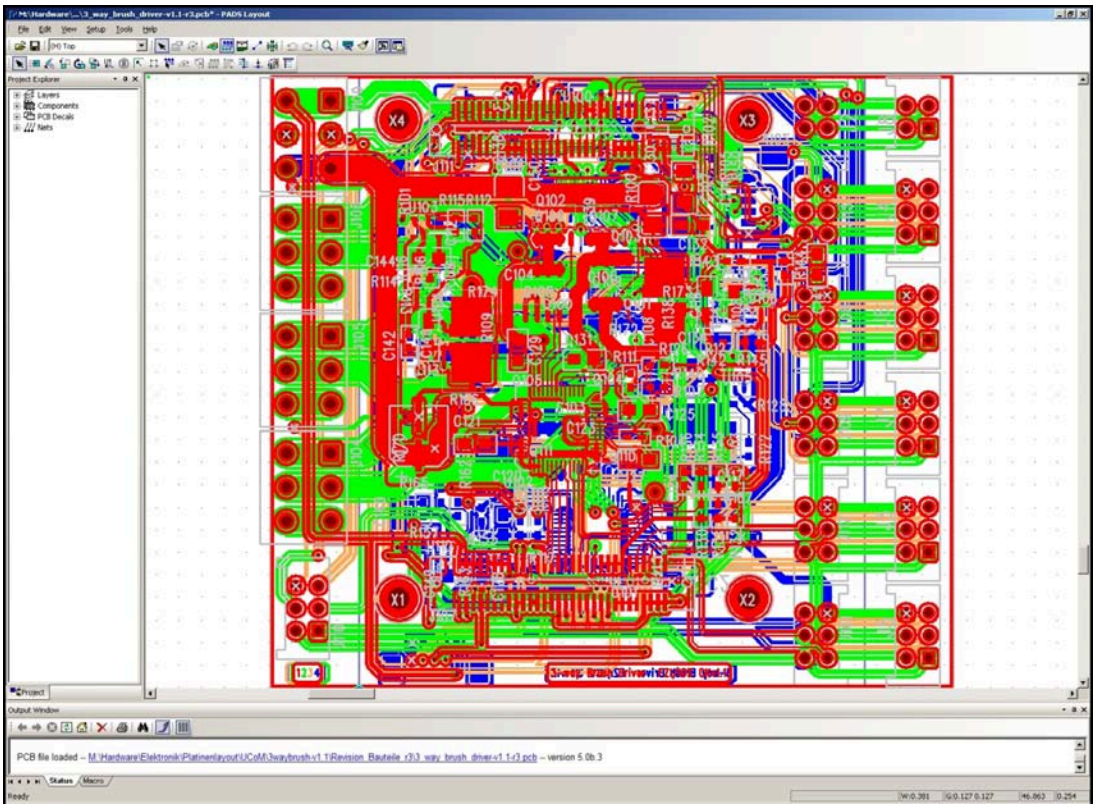


Abb. D.3.: Screenshot der zum Schaltplanentwurf verwendeten Software PADS Layout. Es ist die Layoutansicht zum Routing des Dreier-Motorboards dargestellt.

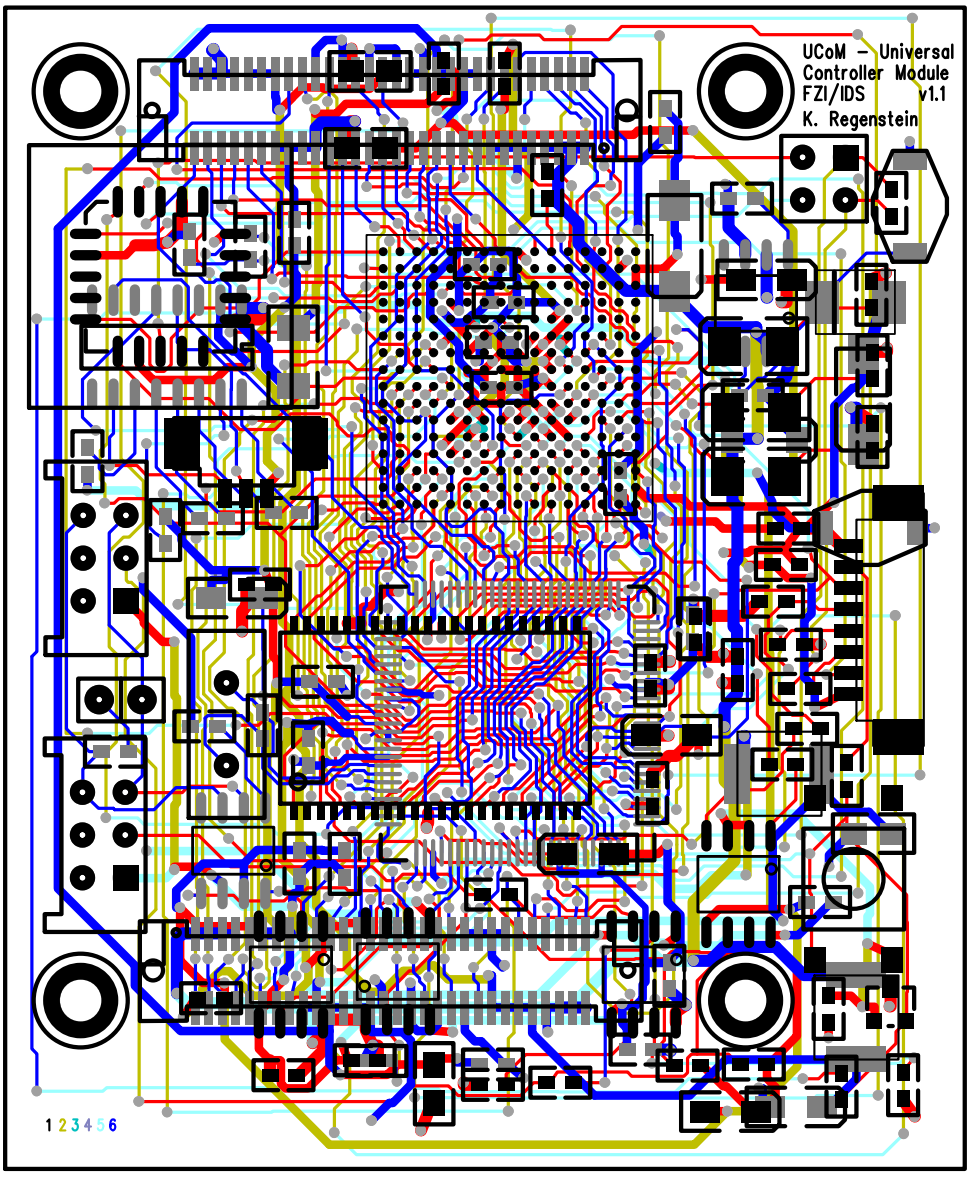


Abb. D.4.: Layout und Routing des UCoMs

D.3. Steckerbelegung

Anschluss	Funktion
Pin 1	5 V CAN (von Boardversorgung galvanisch getrennt)
Pin 2	Not Connected
Pin 3	Init_In
Pin 4	GND CAN (von Boardversorgung galvanisch getrennt)
Pin 5	CAN HI
Pin 6	CAN LO

Tab. D.1.: Steckerbelegung des CAN-Bus

Anschluss	Funktion
Jumper gesteckt	CAN-Bus mit 120 Ω terminiert
Jumper nicht gesteckt	CAN-Bus unterminiert

Tab. D.2.: CAN-Terminierungs-Jumper

Anschluss	Funktion
Pin 1	GND
Pin 2	RX
Pin 3	TX

Tab. D.3.: Steckerbelegung der seriellen Schnittstelle

Anschluss	Funktion
Pin 1	3,3 V
Pin 2	TDI
Pin 3	TDO
Pin 4	TMS
Pin 5	TCK
Pin 6	/TRST
Pin 7	/HRST
Pin 8	GND

Tab. D.4.: Steckerbelegung der JTAG-Schnittstelle

Anschluss	Funktion
Jumper rechts gesetzt	Nur DSP in JTAG-Kette
Jumper oben und unten gesetzt	Alle Bauteile in JTAG-Kette

Tab. D.5.: Konfiguration der JTAG-Kette

E. Abbildungsverzeichnis

F. Tabellenverzeichnis

G. Literaturverzeichnis

[1] AKACHI, Kazuhiko ; KANEKO, Kenji ; KANEHIRA, Noriyuki ; OTA, Shigehiko ; MIYAMORI, Go ; HIRATA, Masaru ; KAJITA, Shuuji ; KANEHIRO, Fumio: Development of Humanoid Robot HRP-3P. In: *Proceedings of 5th IEEE-RAS International Conference on Humanoid Robots*, 2005

[2] ALBERS, A. ; BRUDNIOK, S. ; OTTNAD, J. ; SAUTER, C. ; SEDCHAICHARN, K.: Upper Body of a new Humanoid Robot - the Design of ARMAR III. In: *6th IEEE-RAS International Conference on Humanoid Robots*, 2006, S. 308–313

[3] ANDO, N. ; SUEHIRO, T. ; KITAGAKI, K. ; KOTOKU, T. ; YOON, Woo-Keun: RT-Component Object Model in RT-Middleware – Distributed Component Middleware for RT (Robot Technology). In: *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 2005, S. 457–462

[4] ANDO, N. ; SUEHIRO, T. ; KITAGAKI, K. ; KOTOKU, T. ; YOON, Woo-Keun: RT-middleware: distributed component middleware for RT (robot technology). In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, S. 3933–3938

[5] ARDENCE: *RTX™Frequently Asked Questions*. `http://www.directinsight.co.uk/downloads/evc/035/RTX_FAQ.pdf`. Version: Dezember 2007

[6] ARDENCE: *Ardence RTX*. `http://www.directinsight.co.uk/downloads/evc/031/RTX.pdf`. Version: Februar 2008

[7] ARKIN, Ronald C.: *Behavior-based robotics*. MIT Press, 2000. – ISBN 0–262–01165–4

[8] ASFOUR, T. ; AZAD, P. ; VAHRENKAMP, N. ; REGENSTEIN, K. ; BIERBAUM, A. ; WELKE, K. ; SCHRÖDER, J. ; DILLMANN, R.: Toward humanoid manipulation in human-centred environments. In: *Robotics and Autonomous Systems* 56 (2008), S. 54–65. `http://dx.doi.org/10.1016/j.robot.2007.09.013`. – DOI 10.1016/j.robot.2007.09.013

[9] ASFOUR, T. ; LY, D.N. ; REGENSTEIN, K. ; DILLMANN, R.: Coordinated Task Execution for Humanoid Robots. In: *Experimental Robotics IX* Bd. 21. Springer Berlin / Heidelberg, 2006, S. 259–267

[10] ASFOUR, T. ; REGENSTEIN, K. ; AZAD, P. ; SCHRÖDER, J. ; BIERBAUM, A. ; VAHRENKAMP, N. ; DILLMANN, R.: ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. In: *Proceedings of IEEE-RAS International Conference on Humanoid Robots*, 2006

[11] ASFOUR, T. ; REGENSTEIN, K. ; AZAD, P. ; SCHRÖDER, J. ; DILLMANN, R.: ARMAR-III: A HUMANOID PLATFORM FOR PERCEPTION-ACTION INTEGRATION. In: *Proceedings of second international workshop on Human-Centred Robotic Systems*. München, Deutschland, 6.–7. Oktober 2006

[12] ASFOUR, Tamim: *Sensomotorische Bewegungskoordination zur Handlungsausführung eines humanoiden Roboters*, Universität Karlsruhe (TH), Diss., 2003

[13] ASFOUR, Tamim ; BERNS, Karsten ; DILLMANN, Rüdiger: The Humanoid Robot ARMAR: Design and Control. In: *Proceedings of IEEE/RAS International Conference on Humanoid Robots*, 2000

[14] AZAD, P. ; ASFOUR, T. ; DILLMANN, R.: Stereo-based 6D object localization for grasping with humanoid robot systems. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, S. 919–924

[15] AZAD, Pedram ; GOCKEL, Tilo ; DILLMANN, Rüdiger: *Computer Vision : principles and practice*. Elektor International, 2008. – ISBN 978–0–905705–71–2

[16] BALZERT, Heide: *UML 2 kompakt : mit Checklisten*. Heidelberg : Spektrum Akad. Verl., 2005. – ISBN 3–8274–1389–3

[17] BALZERT, Helmut: *Lehrbuch der Software-Technik: Software-Entwicklung*. Bd. 1. Heidelberg, Germany : Spektrum Akademischer Verlag, 2000. – ISBN 3–8274–0480–0

[18] BASS, L. ; CLEMENTS, P. ; KAZMAN, R.: *Software Architecture in Practice*. Addison-Wesley, 2003

[19] BEKEY, George ; AMBROSE, Robert ; KUMAR, Vijay ; SANDERSON, Art ; WILCOX, Brian ; ZHENG, Yuan: INTERNATIONAL ASSESSMENT OF RESEARCH AND DEVELOPMENT IN ROBOTICS / World Technology Evaluation Center, Inc. Version: 16. Februar 2006. `http://www.wtec.org/robotics/report/screen-robotics-final-report-highres.pdf`. 2006. – Forschungsbericht

[20] BIPED TEAM/TAKANISHI LABORATORY: *Biped Humanoid Robot Group WABIAN-2R*. `http://www.takanishi.mech.waseda.ac.jp/research/wabian/index.htm`. Version: November 2007

[21] BIRKENHOFER, C. ; REGENSTEIN, K. ; ZÖLLLNER, J.-M. ; DILLMANN, R.: Architecture of Multi-Segmented Inspection Robot KAIRO-II. In: *Sixth International Workshop on Robot Motion and Control*. Bukowy Dworek, Poland, Juni 2007

[22] BIRKENHOFER, C. ; SCHOLL, K. U. ; ZÖLLNER, J.M. ; DILLMANN, R.: MakroPLUS - ein modulares Systemkonzept eines mehrsegmentigen, autonomen Kanalroboters. In: *Proceedings of 18. Fachgespräch Autonome Mobile Systeme*. Karlsruhe, 18. Dezember 2003

[23] BIRKENHOFER, C. ; STUDER, S. ; ZÖLLNER, J.-M. ; DILLMANN, R.: Hybrid Impedance Control for Multi-Segmented Inspection Robot Kairo-II. In: *Proceedings of International Conference on Informatics in Control, Automation and Robotics*. Setubal, Portugal, 2006

[24] BORST, C. ; OTT, C. ; WIMBOCK, T. ; BRUNNER, B. ; ZACHARIAS, F. ; BÄUML, B. ; HILLENBRAND, U. ; HADDADIN, S. ; ALBU-SCHÄFFER, A. ; HIRZINGER, G.: A humanoid upper body system for two-handed manipulation. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 2007, S. 2766–2767

[25] BRAITENBERG, Valentin: *Vehikel*. Rowohlt, 1993. – ISBN 3–499–19531–3

[26] BRINKSCHULTE, U. ; UNGERER, T.: *Mikrocontroller und Mikroprozessoren*. Springer-Verlag, 2002

[27] BROOKS, A. ; KAUPP, T. ; MAKARENKO, A. ; WILLIAMS, S. ; OREBÄCK, A.: Towards component-based robotics. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, S. 163–168

[28] BROOKS, A. ; KAUPP, T. ; MAKARENKO, A. ; WILLIAMS, S. B. ; OREBÄCK, A.: Orca: A Component Model and Repository. In: BRUGALI, D. (Hrsg.): *Software Engineering for Experimental Robotics*. Springer Berlin/Heidelberg, 2007, S. 231–251

[29] BROOKS, Rodney A.: A Robust Layered Control System for a Mobile Robot / Massachusetts Institute of Technology. Cambridge, MA, USA : Massachusetts Institute of Technology, 1985. – Forschungsbericht

[30] BROOKS, Rodney A.: Intelligence without representation. In: *Artificial Intelligence* 47 (1991), Januar, Nr. 1-3, S. 139–159. http://dx.doi.org/10.1016/0004-3702(91)90053-M. – DOI 10.1016/0004–3702(91)90053–M

[31] BRUYNINCKX, H.: Open robot control software: the OROCOS project. In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 3, 2001. – ISSN 1050–4729, S. 2523–2528

[32] BRUYNINCKX, H. ; SOETENS, P. ; KONINCKX, B.: The real-time motion control core of the Orocos project. In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 2, 2003. – ISSN 1050–4729, S. 2766–2771

[33] BÄUML, B. ; HIRZINGER, G.: Agile Robot Development (aRD): A Pragmatic Approach to Robotic Software. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, 3741–3748

[34] BÄUML, Berthold: Towards the Evaluation of Software Concepts for Complex Mechatronic Systems. In: PRASSLER, Erwin (Hrsg.) ; NILSSON, Klas (Hrsg.) ; SHAKHIMARDANOV, Azamat (Hrsg.): *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware*, 2007

[35] BÄUML, Berthold ; HIRZINGER, Gerd: When hard realtime matters: Software for complex mechatronic systems. In: *Robotics and Autonomous Systems* 56 (2008), Nr. 1, S. 5–13. http://dx.doi.org/10.1016/j.robot.2007.09.017. – DOI 10.1016/j.robot.2007.09.017. – ISSN 0921–8890

[36] BURGHART, C. ; MIKUT, R ; ASFOUR, T. ; SCHMID, A. ; KRAFT, F. ; SCHREMPF, O. ; HOLZAPFEL, H. ; STIEFELHAGEN, R. ; SWERDLOW, A. ; BRETTHAUER, G. ; DILLMANN, R: Kognitive Architekturen für humanoide Roboter: Anforderungen, Überblick und Vergleich. In: *Proceedings of 17th Workshop Computational Intelligence*, 2007

[37] BURGHART, C. ; MIKUT, R. ; STIEFELHAGEN, R. ; ASFOUR, T. ; HOLZAPFEL, H. ; STEINHAUS, P. ; DILLMANN, R.: A Cognitive Architecture for a Humanoid Robot: A First Approach. In: *Proceedings of 5th IEEE-RAS International Conference on Humanoid Robots*, 2005, S. 357–362

[38] BUTTERFASS, J. ; GREBENSTEIN, M. ; LIU, H. ; HIRZINGER, G.: DLR-Hand II: Next Generation of a Dextrous Robot Hand. In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 1, 2001. – ISSN 1050–4729, S. 109–114

[39] BUTTERFASS, J. ; HIRZINGER, G. ; KNOCH, S. ; LIU, H.: DLR's multisensory articulated hand. I. Hard- and software architecture. In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 3, 1998, S. 2081–2086

[40] CALISI, D. ; CENSI, A. ; IOCCHI, L. ; NARDI, D.: OpenRDK: a modular framework for robotic software development. In: *Proceedings of International Conference on Intelligent Robots and Systems*, 2008, S. 1872–1877

[41] CALISI, D. ; CENSI, A. ; IOCCHI, L. ; NARDI, D.: OpenRDK: a modular framework for robotics software development / Dipartimento di Informatica e Sistemica Anonio Ruberti. 2008. – Forschungsbericht

[42] CAPRARI, G. ; ARRAS, K.O. ; SIEGWART, R.: The autonomous miniature robot Alice: from prototypes to applications. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* Bd. 1, 2000, S. 793–798

[43] CAPRARI, G. ; ESTIER, T. ; SIEGWART, R.: Fascination of Down Scaling - Alice the Sugar Cube Robot. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 2000

[44] CAPRARI, G. ; SIEGWART, R.: Design and control of the mobile micro robot alice. In: *Proceedings of the 2nd International Symposium on Autonomous Minirobots for Research and Edutainment*, 2003, S. 23–32

[45] CAPRARI, G. ; SIEGWART, R.: Mobile micro-robots ready to use: Alice. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, S. 3295–3300

[46] CÔTÉ, C. ; BROSSEAU, Y. ; LÉTOURNEAU, D. ; RAÏEVSKY, C. ; MICHAUD, F.: Robotic software integration using MARIE. In: *International Journal of Advanced Robotic Systems* 3 (2006), März, Nr. 1, 55–60. `http://marie.sourceforge.net/`

[47] CÔTÉ, C. ; BROSSEAU, Y ; LÉTOURNEAU, D. ; RAÏEVSKY, C. ; MICHAUD, F.: Using MARIE in software development and integration for autonomous mobile robotics. In: *International Journal of Advanced Robotic Systems, Special Issue on Software Development and Integration in Robotics* 3 (2006), Nr. 1, S. 55–60

[48] CÔTÉ, C. ; LÉTOURNEAU, D. ; MICHAUD, F. ; BROSSEAU, Y.: Software Design Patterns for Robotics : Solving integration problems with MARIE. In: *Proceedings of IEEE International Conference on Robotics and Automation, Workshop on Software Development and Integration in Robotics*, 2005 (SDIR-I)

[49] CÔTÉ, C. ; LÉTOURNEAU, D. ; MICHAUD, F. ; VALIN, J.-M. ; BROSSEAU, Y. ; RAÏEVSKY, C. ; LEMAY, M. ; TRAN, V.: Code reusability tools for programming mobile robots. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* Bd. 2, 2004, S. 1820–1825

[50] DUNKEL, Jürgen ; HOLITSCHKE, Andreas: *Softwarearchitektur für die Praxis*. Berlin [u.a.] : Springer, 2003 (Xpert.press). – ISBN 3–540–00221–9

[51] ENDERLE, Stefan ; UTZ, Hans ; SABLATNÖG, Stefan ; SIMON, Steffen ; KRAETZSCHMAR, Gerhard ; PALM, Günther: MIRO: Middleware for autonomous mobile robots. In: *Telematics Applications in Automation and Robotics*, 2001

[52] FARINELLI, Ro ; GRISETTI, Giorgio ; IOCCHI, Luca: Design and implementation of modular software for programming mobile robots. In: *Journal of Advanced Robotic Systems* 3 (2006), März, Nr. 1, S. 37–43

[53] FEIERTAG, R. J. ; ORGANICK, E. I.: The Multics Input/Output system. In: *Proceedings of the third ACM symposium on Operating systems principles*. New York, NY, USA, 1971, S. 35–41

[54] FREESCALE: *DSP56800 16-Bit Digital Signal Processor Family Manual*. 3.1, November 2005

[55] FREESCALE: *DSP56F803 Data Sheet*, Januar 2007. `http://www.freescale.com/files/dsp/doc/data_sheet/DSP56F803.pdf`

[56] FZI: *Modular Controller Architecture Version 2*. `http://www.mca2.org/`

[57] GAISER, I. ; SCHULZ, S. ; KARGOV, A. ; KLOSEK, H. ; BIERBAUM, A. ; PYLATIUK, C. ; OBERLE, R. ; WERNER, T. ; ASFOUR, T. ; BRETTHAUER, G. ; DILLMANN, R.: A new anthropomorphic robotic hand. In: *Proceedings of 8th IEEE-RAS International Conference on Humanoid Robots*, 2008, S. 418–422

[58] GASSMANN, B. ; SCHOLL, K. U. ; BERNS, K.: Locomotion of LAURON III in Rough Terrain. In: *Proceedings of International Conference on Advanced Mechatronics*, 2001

[59] GAMMA, Erich: *Design patterns*. Addison-Wesley, 2000. – ISBN 0–201–63361–2

[60] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: Design Patterns: Abstraction and Reuse of Object-Oriented Design. In: *Proceedings of European Conference on Object-Oriented Programming*. Kaiserslautern, Juli 1993, S. 406–431

[61] GERKEY, B.P. ; VAUGHAN, R.T. ; STOY, K. ; HOWARD, A. ; SUKHATME, G.S. ; MATARIC, M.J.: Most valuable player: a robot device server for distributed control. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* Bd. 3, 2001, S. 1226–1231

[62] GERKEY, Brian P. ; VAUGHAN, Richard T. ; HOWARD, Andrew: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: *ICAR 2003*. Coimbra, Portugal, Juni 2003, S. 317–323

[63] GÖGER, D. ; WEISS, K. ; BURGHART, C. ; WÖRN, H.: Sensitive skin for a humanoid robot. In: *Proceedings of Human-Centered Robotic Systems*. München, Deutschland, 6.–7. Oktober 2006

[64] GOMBERT, B. ; HIRZINGER, G. ; PLANK, G. ; SCHEDL, M.: Modular concepts for a new generation of light weight robots. In: *Proceedings of 20th International Conference on Industrial Electronics, Control and Instrumentation* Bd. 3, 1994, S. 1507–1514

[65] GRUHLER, Gerhard (Hrsg.): *Feldbusse und Geräte-Kommunikationssysteme*. Franzis, 2001

[66] HAIDACHER, S. ; BUTTERFASS, J. ; FISCHER, M. ; GREBENSTEIN, M. ; JOEHL, K. ; KUNZE, K. ; NICKL, M. ; SEITZ, N. ; HIRZINGER, G.: DLR hand II: Hard- and Software Architecture for Information Processing. In: *IEEE International Conference on Robotics and Automation* Bd. 1. Taipei, Taiwan, 14.–19. September 2003, S. 684–689

[67] HASSLACHER, Brosl ; TILDEN, Mark W.: Living Machines. In: STEELS, L. (Hrsg.): *IEEE Workshop on Bio-Mechatronics*, 1996, S. 143–169

[68] HEINEMAN, G. T. (Hrsg.) ; COUNCIL, W. T. (Hrsg.): *Component-based software engineering*. Addison-Wesley, 2001. – ISBN 0–201–70485–4

[69] HIRAI, K.: Current and future perspective of Honda humamoid robot. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* Bd. 2, 1997, S. 500–508

[70] HIRAI, Kazuo ; HIROSE, Masato ; HAIKAWA, Yuji ; TAKENAKA, Toru: The Development of Honda Humanoid Robot. In: *Proceedings of IEEE International Conference on Robotics and Automation*. Leuven, Belgium, Mai 1998

[71] HIRUKAWA, Hirohisa ; KANEHIRO, Fumio ; KANEKO, Kenji ; KAJITA, Shuuji ; FUJIWARA, Kiyoshi ; KAWAI, Yoshihiro ; TOMITA, Fumiaki ; HIRAI, Shigeoki ; TANIE, Kazuo ; ISOZUMI, Takakatsu ; AKACHI, Kazuhiko ; KAWASAKI, Toshikazu ; OTA, Shigehiko ; YOKOYAMA, Kazuhiko ; HANDA, Hiroyuki ; FUKASE, Yutaro ; MAEDA, Junichiro ; NAKAMURA, Yoshihiko ; TACHI, Susumu ; INOUE, Hirochika: Humanoid robotics platforms developed in HRP. In: *Robotics and Autonomous Systems* 48 (2004), Nr. 4, S. 165–175. `http://dx.doi.org/10.1016/j.robot.2004.07.007`. – DOI 10.1016/j.robot.2004.07.007

[72] HIRZINGER, G. ; ALBU-SCHÄFFER, A. ; HAHNLE, M. ; SCHAEFER, I. ; SPORER, N.: On a new generation of torque controlled light-weight robots. In: *Proceedings of IEEE*

International Conference on Robotics and Automation Bd. 4, 2001. – ISSN 1050–4729, S. 3356–3363

[73] HIRZINGER, G. ; SPORER, N. ; ALBU-SCHÄFFER, A. ; HÄHNLE, M. ; KRENN, R. ; PASCUCCI, A. ; SCHEDL, M.: DLR's torque-controlled light weight robot III - are we reaching the technological limits now? In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 2, 2002, S. 1710–1716

[74] HÖLL, Dominik M.: *Konstruktion der holonomen Antriebsplattform Omnibot II*, Universität Karlsruhe (TH), Diplomarbeit, Mai 2006

[75] HONDA: *Honda Wordlwide | Asimo | History*. `http://world.honda.com/ASIMO/history/`. Version: 2006

[76] INFINEON: *C167CR, C167SR Data Sheet*. 3.2. Infineon Technologies, Juli 2001

[77] INFINEON: *C167CR Derivatives, User's Manual*. 3.2. Infineon Technologies, Mai 2003

[78] KAGAMI, S. ; KUFFNER, Jr. J. J. J. J. ; NISHIWAKI, K. ; SUGIHARA, T. ; MICHIKATA, T. ; AOYAMA, T. ; INAHA, M. ; INOUE, H.: Design and implementation of remotely operation interface for humanoid robot. In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 1, 2001. – ISSN 1050–4729, S. 401–406

[79] KAGAMI, Satoshi ; NISHIWAKI, Koichi ; KUFFNER JR., James J. ; KUNIYOSHI, Yasuo ; INABA, Masayuki ; INOUE, Hirochika: Design and Implementation of Software Research Platform for Humanoid Robotics : H7. In: *Proceedings of IEEE-RAS International Conference on Humanoid Robots*, 2001

[80] KAGAMI, Satoshi ; NISHIWAKI, Koichi ; SUGIHARA, Tomomichi ; KUFFNER, James J. ; INABA, Msayuki ; INOUE, Hirochika: Design and Implementation of Software Research Platform for Humanoid Robotics : H6. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Seoul, Korea, 21.–26. Mai 2001

[81] KAJITA, Shuuji ; KANEKO, Kenji ; KANEIRO, Fumio ; HARADA, Kensuke ; MORISAWA, Mitsuharu ; NAKAOKA, Shin-ichiro ; MIURA, Kanako ; FUJIWARA, Kiyoshi ; NEO, Ee S. ; HARA, Isao ; YOKOI, Kazuhito ; HIRUKAWA, Hirohisa: Cybernetic Human HRP-4C: A humanoid robot with human-like proportions. In: *Proceedings of 14th International Symposium on Robotics Research*. Lucerne, Switzerland, 31. August – 3. September 2009

[82] KANEHIRO, F. ; FUJIWARA, K. ; KAJITA, S. ; YOKOI, K. ; KANEKO, K. ; HIRUKAWA, H. ; NAKAMURA, Y. ; YAMANE, K.: Open architecture humanoid robotics platform. In:

Proceedings of IEEE International Conference on Robotics and Automation Bd. 1, 2002, S. 24–30

[83] KANEHIRO, Fumio ; HIRUKAWA, Hirohisa ; KAJITA, Shuuji: OpenHRP: Open Architecture Humanoid Robotics Platform. In: *International Journal of Robotics Research* 23 (2004), Nr. 2, S. 155–165

[84] KANEHIRO, Fumio ; ISHIWATA, Yoichi ; SAITO, Hajime ; AKACHI, Kazuhiko ; MIYAMORI, Gou ; ISOZUMI, Takakats ; KANEKO, Kenji ; HIRUKAWA, Hirohisa: Distributed Control System of Humanoid Robots based on Real-time Ethernet. In: *Proceedings of International Conference on Intelligent Robots and Systems*, 2006, S. 2471–2477

[85] KANEKO, Kenji ; KANEHIRO, Fumio ; KAJITA, Shuuji ; HIRUKAWA, Hirohisa ; KAWASAKI, Toshikazu ; HIRATA, Masaru ; AKACHI, Kazuhiko ; ISOZUMI, Takakatsu: Humanoid Robot HRP-2. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. New Orleans, LA, USA, April 2004

[86] KANEKO, Kenji ; KANEHIRO, Fumio ; KAJITA, Shuuji ; YOKOYAMA, Kazuhiko ; AKACHI, Kazuhiko ; KAWASAKI, Toshikazu ; OTA, Shigehiko ; ISOZUMI, Takakatsu: Design of Prototype Humanoid Robotics Platform for HRP. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. EPFL, Lausanne, Switzerland, Oktober 2002

[87] KAPLAN, FRÉDÉRIC: WHO IS AFRAID OF THE HUMANOID? INVESTIGATING CULTURAL DIFFERENCES IN THE ACCEPTANCE OF ROBOTS. In: *International Journal of Humanoid Robotics* 1 (2004), Nr. 3, S. 1–16

[88] KARGOV, A. ; ASFOUR, T. ; PYLATIUK, C. ; OBERLE, R. ; KLOSEK, H. ; SCHULZ, S. ; REGENSTEIN, K. ; BRETTHAUER, G. ; DILLMANN, R.: Development of an anthropomorphic hand for a mobile assistive robot. In: *9th International Conference on Rehabilitation Robotics*, 2005, S. 182–186

[89] KARGOV, Artem ; PYLATIUK, Christian ; KLOSEK, Heinrich ; OBERLE, Reinhold ; SCHULZ, Stefan ; BRETTHAUER, Georg: Modularly designed lightweight anthropomorphic robot hand. In: *Proceedings of IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2006, S. 155–159

[90] KEPPLIN, V. ; SCHOLL, K. U. ; BERNS, K.: A Mechatronic Concept for a Sewer Inspection Robot. In: *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 1999

[91] KERPA, O. ; WEISS, K. ; WÖRN, H.: Development of a flexible tactile sensor system for a humanoid robot. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* Bd. 1. Las Vegas, Nevada, USA, Oktober 2003, S. 1–6

[92] KIM, Jung-Hoo ; PARK, Seo-Wook ; PARK, Ill-Woo ; OH, Jun-Ho: Development of a Humanoid Biped Walking Robot Platform KHR-1 - Initial Design and Its Performance Evaluation. In: *Proceedings of 3rd IARP International Workshop on Humanoid and Human Friendly Robotics*. Tsukuba, Japan, 2002, S. 14 – 21

[93] KIM, Jung-Hoon ; OH, Jun-Ho: Torque Feedback Control of the Humanoid Platform KHR-1. In: *Proceedings of 3rd IEEE International Conference on Humanoid Robots*. Karlsruhe und München, Oktober 2003

[94] KIM, Jung-Yup ; PARK, Ill-Woo ; LEE, Jungho ; KIM, Min-Su ; CHO, Baek-kyu ; OH, Jun-Ho: System Design and Dynamic Walking of Humanoid Robot KHR-2. In: *Proceedings of IEEE International Conference on Robotics and Automation*. Barcelona, Spain, 18.–22. April 2005, S. 1431–1436

[95] KIM, Jung-Yup ; PARK, Ill-Woo ; LEE, Jungho ; OH, Jun-Ho: Experiments of Vision Guided Walking of Humanoid Robot, KHR2. In: *Proceedings of 5th IEEE-RAS International Conference on Humanoid Robots*, 2005

[96] KIM, Jung-Yup ; PARK, Ill-Woo ; OH, Jun-Ho: Design and Walking Control of the Humanoid Robot, KHR-2 (KAIST Humanoid Robot-2). In: *Proceedings of International Conference on Control, Automation and Systems*, 2004, S. 1540 – 1543

[97] KOHLHEPP, Peter ; WALTHER, Marcus ; STEINHAUS, Peter: Schritthaltende 3D-Kartierung und Lokalisierung für mobile Inspektionsroboter. In: *Proceedings of 18. Fachgespräch Autonome Mobile Systeme*. Karlsruhe, 18. Dezember 2003

[98] KRAIF, Ursula (Hrsg.): *Duden, das große Fremdwörterbuch*. Dudenverlag, 2007. – ISBN 3–411–04164–1

[99] KREUTZ, Stephanie: *Leistungsvergleich des Motorola DSP56F803 mit Infineons C167-Mikrocontroller*, Universität Karlsruhe (TH), Studienarbeit, 2006

[100] LEHRSTUHL FÜR ANGEWANDTE MECHANIK: *Laufmaschine JOHNNIE*. `http://www.amm.mw.tu-muenchen.de/index.php?id=182`. Version: Oktober 2008

[101] LENK, Micha: *Entwurf und Realisierung der Software- und Hardware-Architektur des humanoiden Roboters ARMAR-IIIb*, Universität Karlsruhe (TH), Diplomarbeit, 2008

[102] LÖFFLER, K. ; GIENGER, M. ; PFEIFFER, F. ; ULBRICH, H.: Sensors and control concept of a biped robot. In: *IEEE Transactions on Industrial Electronics* 51 (2004), Nr. 5, S. 972–980. http://dx.doi.org/10.1109/TIE.2004.834948. – DOI 10.1109/TIE.2004.834948. – ISSN 0278–0046

[103] LOHMEIER, S. ; LÖFFLER, K. ; GIENGER, M. ; ULBRICH, H. ; PFEIFFER, F.: Computer system and control of biped "Johnnie". In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 4, 2004. – ISSN 1050–4729, S. 4222–4227

[104] LOHMEIER, S. ; LÖFFLER, K. ; GIENGER, M. ; ULBRICH, H. ; PFEIFFER, F.: Sensor system and trajectory control of a biped robot. In: *Proceedings of 8th IEEE International Workshop on Advanced Motion Control*, 2004, S. 393–398

[105] LÉTOURNEAU, D. ; CÔTÉ, C. ; RAÏEVSKY, C. ; BROSSEAU, Y. ; MICHAUD, F.: Using MARIE for mobile robot software development and integration. In: BRUGALI, D. (Hrsg.): *Software Engineering for Experimental Robotics*. Springer-Verlag, 2006

[106] LY, D.N. ; REGENSTEIN, K. ; ASFOUR, T. ; DILLMANN, R.: A modular and distributed embedded control architecture for humanoid robots. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* Bd. 3, 2004, S. 2775–2780

[107] MAKARENKO, A. ; BROOKS, A. ; KAUPP, T.: Orca: Components for robotics. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Workshop for Robotic Standardization*, 2006

[108] MALLET, A. ; FLEURY, S. ; BRUYNINCKX, H.: A specification of generic robotics software components: future evolutions of $G^{en}{}_{o}M$ in the Orocos context. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and System* Bd. 3, 2002, S. 2292–2297

[109] MICHAUD, F. ; BROSSEAU, Y. ; CÔTÉ, C. ; LÉTOURNEAU, D. ; MOISAN, P. ; PONCHON, A. ; RAÏEVSKY, C. ; VALIN, J.-M. ; BEAUDRY, E. ; KABANZA, F.: Modularity and integration in the design of a socially interactive robot. In: *Proceedings of IEEE International Workshop on Robot and Human Interactive Communication*, 2005, S. 172–177

[110] MOORE, G. E.: Cramming More Components onto Integrated Circuits. In: *Electronics* 38 (1965), April, Nr. 8, S. 114–117. http://dx.doi.org/10.1109/JPROC.1998.658762. – DOI 10.1109/JPROC.1998.658762

[111] NESNAS, Issa A. D. ; VOLPE, Richard ; ESTLIN, Tara ; DAS, Hari ; PETRAS, Richard ; MUTZ, Darren: Toward Developing Reusable Software Components for Robotic Appli-

cations. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001, S. 2375–2383

[112] NISHIWAKI, K. ; KAGAMI, S. ; KUFFNER, J. J. ; INABA, M. ; INOUE, H.: Humanoid 'JSK-H7' : Research platform for autonomous behavior and whole body motion. In: *Proceedings of 3rd IARP International Workshop on Humanoid and Human Friendly Robotics*, 2002, S. 2–9

[113] NISHIWAKI, K. ; SUGIHARA, T. ; KAGAMI, S. ; KANEHIRO, F. ; INABA, M. ; INOUE, H.: Design and development of research platform for perception-action integration in humanoid robot: H6. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* Bd. 3, 2000, S. 1559–1564 vol.3

[114] OGURA, Yu ; AIKAWA, Hiroyuki ; SHIMOMURA, Kazushi ; KONDO, Hideki ; MORISHIMA, Akitoshi ; LIM, Hun-ok ; TAKANISHI, Atsuo: Development of a New Humanoid Robot WABIAN-2. In: *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*. Orlando, Florida, Mai 2006

[115] OKADA, K. ; OGURA, T. ; HANEDA, A. ; KOUSAKA, D. ; NAKAI, H. ; INABA, M. ; INOUE, H.: Integrated system software for HRP2 humanoid. In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 4, 2004, S. 3207–3212

[116] OTT, C. ; EIBERGER, O. ; FRIEDL, W. ; BAUML, B. ; HILLENBRAND, U. ; BORST, C. ; ALBU-SCHÄFFER, A. ; BRUNNER, B. ; HIRSCHMÜLLER, H. ; KIELHOFER, S. ; KONIETSCHKE, R. ; SUPPA, M. ; WIMBOCK, T. ; ZACHARIAS, F. ; HIRZINGER, G.: A Humanoid Two-Arm System for Dexterous Manipulation. In: *Proceedings of 6th IEEE-RAS International Conference on Humanoid Robots*, 2006, S. 276–283

[117] PARK, Ill-Woo ; KIM, Jung-Yup ; LEE, Jungho ; OH, Jun-Ho: Mechanical Design of Humanoid Robot Platform KHR-3 (KAIST Humanoid Robot - 3: HUBO). In: *Proceedings of 5th IEEE-RAS International Conference on Humanoid Robots*, 2005

[118] PARK, Ill-Woo ; KIM, Jung-Yup ; LEE, Jungho ; OH, Jun-Ho: Online Free Walking Trajectory Generation for Biped Humanoid Robot KHR-3 (HUBO). In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Orlando, Florida, Mai 2006

[119] PARK, Ill-Woo ; KIM, Jung-Yup ; PARK, Seo-Wook ; OH, Jun-Ho: Development of humanoid robot platform KHR-2 (KAIST humanoid robot-2). In: *Proceedings of 4th IEEE/RAS International Conference on Humanoid Robots* Bd. 1, 2004, S. 292–310

[120] PARNAS, D. L.: On the criteria to be used in decomposing systems into modules. In: *Communications of the ACM* 15 (1972), Nr. 12, S. 1053–1058. http://dx.doi.org/10.1145/361598.361623. – DOI 10.1145/361598.361623. – ISSN 0001–0782

[121] PARNAS, D. L. ; CLEMENTS, P. C. ; WEISS, D. M.: The modular structure of complex systems. In: *Proceedings of 7th international conference on Software engineering*. Piscataway, NJ, USA : IEEE Press, 1984. – ISBN 0–8186–0528–6, S. 408–417

[122] PARNAS, David L.: Designing Software for Ease of Extension and Contraction. In: *IEEE Transactions on Software Engineering (TSE)* 5 (1979), Nr. 2, S. 128–138

[123] PFEIFFER, Friedrich: The TUM walking machines. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 365 (2007), Januar, Nr. 1850, S. 109–131. http://dx.doi.org/10.1098/rsta.2006.1922. – DOI 10.1098/rsta.2006.1922

[124] PONT, F. ; KOLSKI, S. ; SIEGWART, R.: Applications of a real-time software framework for complex mechatronic systems. In: *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2005, S. 1324–1329

[125] PONT, F. ; SIEGWART, R.: Towards Improving Robotic Software Reusability Without Losing Real-Time Capabilities. In: *Proceedings of 1st International Conference on Informatics in Control, Automation and Robotics*. Setúbal, Portugal, 25. – 28. August 2004

[126] PONT, F. ; SIEGWART, R.: A real-time software framework for indoor navigation. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, S. 2085–2090

[127] PRINCE, Christopher G. ; MISLIVEC, Eric J.: *Humanoid Theory Grounding*. Poster presentation at Humanoids 2001: The 2nd IEEE-RAS International Conference on Humanoid Robots, Tokyo, Japan. http://cogprints.org/1926/. Version: 22. – 24. November 2001

[128] REGENSTEIN, K. ; ASFOUR, T. ; DILLMANN, R.: Designing a computer architecture for the humanoid robot ARMAR-III. In: *Proceedings of French-German Workshop on Humanoid and Legged Robots*, 2006

[129] REGENSTEIN, K. ; KERSCHER, T. ; BIRKENHOFER, C. ; ASFOUR, T. ; ZÖLLNER, J.M. ; DILLMANN, R.: A modular approach for controlling mobile robots. In: *Proceedings of 10th International Conference on Climbing and Walking Robots*. Singapore, 16.–18. Juli 2007

[130] REGENSTEIN, K. ; KERSCHER, T. ; BIRKENHOFER, C. ; ASFOUR, T. ; ZÖLLNER, J.M. ; DILLMANN, R.: Universal Controller Module (UCoM) - component of a modular concept in robotic systems. In: *Proceedings of IEEE International Symposium on Industrial Electronics*. Centro Cultural and Centro Social Caixanova - Vigo, Spain, 4.–7. Juni 2007

[131] REGENSTEIN, Kristian ; DILLMANN, Rüdiger: Design of an open hardware architecture for the humanoid robot ARMAR. In: *Conference Documentation of Humanoids 2003, International Conference on Humanoid Robots*. Karlsruhe/München, Deutschland, 1.–3. Oktober 2003, S. 3

[132] RÖSSLER, Patrick ; HANEBECK, Uwe D.: Telepresence Techniques for Exception Handling in Household Robots. In: *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* Bd. 1. The Hague, The Netherlands, Oktober 2004, S. 53–58

[133] ROBERT BOSCH GMBH: *CAN Specification Version 2.0*. Stuttgart: Robert Bosch GmbH, 1991

[134] SAKAGAMI, Yshiaki ; WATANABE, Ryujin ; AOYAMA, Chiaki ; MATSUNGA, Shinichi ; HIGAKI, Nobuo ; FUJIMURA, Kikuo: The intelligent ASIMO: System overview and integration. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. EPFL, Lausanne, Switzerland, Oktober 2002

[135] SCHOLL, K. U. ; ALBIEZ, J. ; GASSMANN, B.: MCA - An Expandable Modular Controller Architecture. In: *Proceedings of 3rd Real-Time Linux Workshop*. Milano, Italy, 2001

[136] SCHOLL, K. U. ; KEPPLIN, V. ; ALBIEZ, J. ; DILLMANN, R.: Develloping robot prototypes with an expandable modular controller architecture. In: *Proceedings of the 6th International Conference on Intelligent Autonomous Systems*, 2000, S. 67–74

[137] SCHOLL, K. U. ; KEPPLIN, V. ; BERNS, K. ; DILLMANN, R.: Controlling a Multijoint Robot for Autonomous Sewer Inspection. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 2000

[138] SCHRÖDER, Joachim ; GOCKEL, Tilo ; DILLMANN, Rüdiger ; GLÖCKNER, Siegfried: Entwurf und Aufbau einer holonomen, mobilen Antriebsplattform für einen humanoiden Serviceroboter. In: *Proceedings of 18. Fachgespräch Autonome Mobile Systeme*. Karlsruhe, 18. Dezember 2003

[139] SIEGWART, R. ; ARRAS, K. O. ; JENSEN, B. ; PHILIPPSEN, R. ; TOMATIS, N.: Design, Implementation and Exploitation of a New Fully Autonomous Tour Guide Robot. In: *Proceedings of the 1st International Workshop on Advances in Service Robotics*, 2003

[140] SIEGWART, Roland ; ARRAS, Kai O. ; BOUABDALLAH, Samir ; BURNIER, Daniel ; FROIDEVAUX, Gilles ; GREPPIN, Xavier ; JENSEN, Björn ; LOROTTE, Antoine ; MAYOR, Laetitia ; MEISSER, Mathieu ; PHILIPPSEN, Roland ; PIGUET, Ralph ; RAMEL, Guy ; TERRIEN, Gregorie ; TOMATIS, Nicola: Robox at Expo.02: A large-scale installation of personal robots. In: *Robotics and Autonomous Systems* 42 (2003), 01. März, Nr. 3-4, S. 203–222

[141] SIMON, D. ; GIRAULT, A.: Synchronous programming of automatic control applications using ORCCAD and ESTEREL. In: *Proceedings of 40th IEEE Conference on Decision and Control* Bd. 4, 2001, S. 3290–3295

[142] SIMON, D. ; PISSARD-GIBOLLET, R. ; ARIAS, S.: Orccad, a framework for safe robot control design and implementation. In: *Proceedings of 1st National Workshop on Control Architectures of Robots: software approaches and issues*. Montpellier, 06.–07. April 2006

[143] SOETENS, P. ; BRUYNINCKX, H.: Realtime Hybrid Task-Based Control for Robots and Machine Tools. In: *Proceedings of IEEE International Conference on Robotics and Automation*, 2005, S. 259–264

[144] SOETENS, Peter: *The Orocos Component Builder's Manual : Open RObot COntrol Software : 1.4.1*

[145] SOETENS, Peter: *Orocos Open Robot Control Software*. V Jornades de Programari Lliure. `http://www.orocos.org/documents/orocos-presentation-Barcelona06.pdf`. Version: Juli 2006

[146] STEINHAUS, Peter ; DILLMANN, Rüdiger: Aufbau und Modellierung des RoSi Scanners zur 3D-Tiefenbildakquisition. In: *Proceedings of 18. Fachgespräch Autonome Mobile Systeme*. Karlsruhe, 18. Dezember 2003

[147] TANENBAUM, Andrew S.: *Computernetzwerke*. Pearson, 2003 `http://www.ubka.uni-karlsruhe.de/hylib-bin/suche.cgi?opacdb=UBKA_OPAC&fbt=5442404&nd=10607279`

[148] TOMATIS, N. ; TERRIEN, G. ; PIGUET, R. ; BURNIER, D. ; BOUABDALLAH, S. ; SIEGWART, R.: Design and System Integration for the Expo.02 Robot. In: *Proceedings of 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002

[149] TOMITA, F. ; YOSHIMI, T. ; UESHIBA, T. ; KAWAI, Y. ; SUMI, Y. ; MATSUSHITA, T. ; ICHIMURA, N. ; SUGIMOTO, K. ; ISHIYAMA, Y.: R&D of versatile 3D vision system VVV. In: *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics* Bd. 5, 1998, S. 4510–4516

[150] UHL, K. ; ZIEGENMEYER, M.: MCA2 - An Extensible Modular Framework for Robot Control Applications. In: *Proceedings of 10th International Conference on Climbing and Walking Robots*. Singapore, 16.–18. Juli 2007

[151] ULBRICH, Prof. Dr.-Ing. H.: *Grundlagen und Perspektiven mechatronischer Systeme*. 4. Münchner Wissenschaftstage im "Jahr der Technik". `http://www.muenchner-wissenschaftstage.de/mwt2004/content/e160/e707/e728/e750/filetitle/ulbrich_ger.pdf`. Version: 25. Oktober 2004

[152] UTZ, H. ; SABLATNOG, S. ; ENDERLE, S. ; KRAETZSCHMAR, G.: MIRO: - middleware for mobile robot applications. In: *IEEE Transactions On Robotics and Automation* 18 (2002), August, Nr. 4, S. 493–497. `http://dx.doi.org/10.1109/TRA.2002.802930`. – DOI 10.1109/TRA.2002.802930

[153] VAHRENKAMP, N. ; WIELAND, S. ; AZAD, P. ; GONZALEZ, D. ; ASFOUR, T. ; DILLMANN, R.: Visual servoing for humanoid grasping and manipulation tasks. In: *Proceedings of 8th IEEE-RAS International Conference on Humanoid Robots*. Daejeon, Korea, 1.–3. Dezember 2008, S. 406–412

[154] VAUGHAN, R.T. ; GERKEY, B.P. ; HOWARD, A.: On device abstractions for portable, reusable robot code. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)* Bd. 3, 2003, S. 2421–2427

[155] VDI-GESELLSCHAFT ENTWICKLUNG KONSTRUKTION VERTRIEB: *VDI-Richtlinie 2206 – Entwicklungsmethodik für mechatronische Systeme*. Juni 2004

[156] VDI-GESELLSCHAFT PRODUKTIONSTECHNIK: *Montage- und Handhabungstechnik; Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole*. Mai 1990

[157] VOLPE, R. ; NESNAS, I. ; ESTLIN, T. ; MUTZ, D. ; PETRAS, R. ; DAS, H.: The CLARAty architecture for robotic autonomy. In: *Proceedings of IEEE Aerospace Conference* Bd. 1, 2001, S. 121–132

[158] VOLPE, Richard ; NESNAS, Issa A. D. ; ESTLIN, Tara ; MUTZ, Darren ; PETRAS, Richard ; DAS, Hari: CLARAty: Coupled Layer Architecture for Robotic Autonomy / NASA - Jet Propulsion Laboratory. 2000. – Forschungsbericht

[159] WALTER, William G.: *Das lebende Gehirn*. Knaur, 1963

[160] WEIDEMANN, H. J. ; PFEIFFER, F. ; ELTZE, J.: The six-legged TUM walking robot. In: *Proceedings of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems. Advanced Robotic Systems and the Real World* Bd. 2, 1994, S. 1026–1033

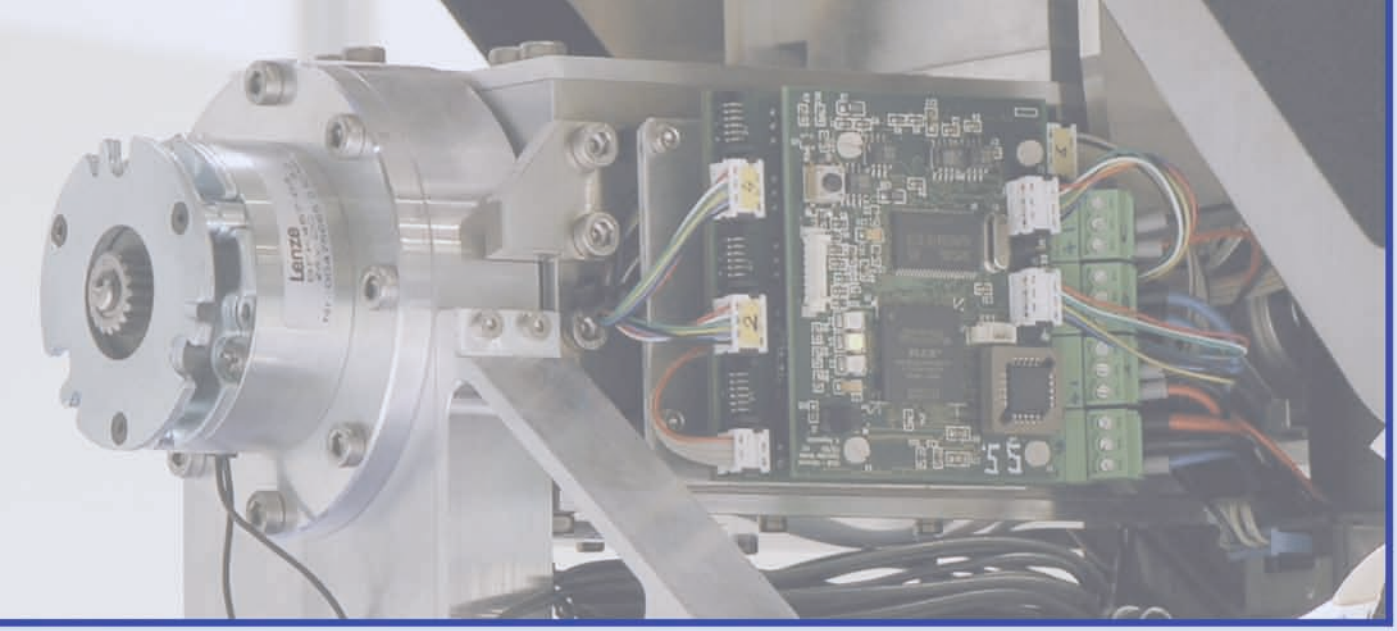
[161] YAMAGUCHI, J. ; INOUE, S. ; NISHINO, D. ; TAKANISHI, A.: Development of a bipedal humanoid robot having antagonistic driven joints and three DOF trunk. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* Bd. 1, 1998, S. 96–101

[162] YAMAGUCHI, J. ; SOGA, E. ; INOUE, S. ; TAKANISHI, A.: Development of a bipedal humanoid robot-control method of whole body cooperative dynamic biped walking. In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 1, 1999, S. 368–374

[163] ZAGLER, A. ; PFEIFFER, F.: „MORITZ“ a pipe crawler for tube junctions. In: *Proceedings of IEEE International Conference on Robotics and Automation* Bd. 3, 2003. – ISSN 1050–4729, S. 2954–2959

[164] ZUFFEREY, J. C. ; BEYELER, A. ; FLOREANO, D.: Vision-based navigation from wheels to wings. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems* Bd. 3, 2003, S. 2968–2973

[165] ZUSER, Wolfgang ; GRECHENIG, Thomas ; KÖHLE, Monika: *Software Engineering mit UML und dem Unified Process*. München : Pearson Studium, 2004 `http://swbplus.bsz-bw.de/bsz112081436cov.htm;http://swbplus.bsz-bw.de/bsz112081436vlg.htm`. – ISBN 3–8273–7090–6



Humanoide Roboter sind hochkomplexe Systeme. Sie zeichnen sich durch ein sehr heterogenes Sensor- und Aktorsystem aus, welches sehr hohe und breit gefächerte Anforderungen an die verwendete Architektur stellt. Damit ein humanoider Roboter mit dem Menschen interagieren und im Servicebereich eingesetzt werden kann, muss er eine Vielzahl von Anforderungen erfüllen. Zum Erreichen der Roboterfähigkeiten, wird einerseits ein Ansatz zur funktional-logischen Steuerung des Roboters benötigt, andererseits muss zur Realisierung der Fähigkeiten in Software ein leistungsfähiges Softwarerahmenwerk bereitgestellt werden.

In der vorliegenden Arbeit wird sowohl der Entwurf einer funktionalen Steuerungsarchitektur, das verwendete Softwarerahmenwerk als auch die Abbildung auf eine dezidierte Hardwarearchitektur unter Berücksichtigung der spezifischen Randbedingungen bei humanoiden Robotern beschrieben. Schließlich wird die Umsetzung der vorgestellten Hardware-Software-Architektur auf dem humanoiden Robotersystem ARMAR-III behandelt und das Gesamtsystem evaluiert.

ISBN: 978-3-86644-527-7

ISBN 978-3-86644-527-7
9 783866 445277 >