

Konzeption und Implementierung von sensorbasierten Bewegungsalgorithmen für einen Laufroboter

Diplomarbeit im Studiengang Informatik
im Sommer-Semester 2005

Hochschule Mannheim –
Hochschule für Technik und Gestaltung
Institut für Robotik

Aufgabensteller: Prof. Dr. Thomas Ihme
Zweitkorrektor: Prof. Dr. Miriam Föller

vorgelegt von
Kai Wetzelsberger
Matrikelnummer: 0010070

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit hat in dieser oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegen.

Ludwigshafen, den 12. September 2005

Kai Wetzelsberger

Danksagung

Dank gebührt Prof. Dr. Thomas Ihme für seine Unterstützung und Betreuung im Rahmen dieser Diplomarbeit. Dank gebührt auch meinem Kommilitonen Mariusz Ellwardt, der mich während meiner Diplomarbeit tatkräftig unterstützte.

Bedanken möchte ich mich an dieser Stelle auch bei meinen Eltern Helga Wetzelsberger und vor allem bei meinem Vater Dr. Klaus Wetzelsberger, für das mehrfache Korrekturlesen dieser Diplomarbeit. Dank gebührt auch meinem Freund Kerem Gülensoy, für seine Hilfe bei einigen Abbildungen dieser Arbeit.

Kai Wetzelsberger

Ludwigshafen, den 12. September 2005

Inhaltsverzeichnis

Eidesstattliche Erklärung	i
Danksagung	ii
1 Einleitung	1
1.1 Motivation	1
1.2 Ziel und Aufbau der Arbeit	2
2 Steuerung von Laufmaschinen	4
2.1 Statische Verfahren	5
2.1.1 Laufmuster	5
2.1.2 Zelluläre Automaten	5
2.2 Reaktive Verfahren	5
2.3 Planende Laufsteuerungen	6
3 Methoden zur Bewegungssteuerung	8
3.1 Stabheuschrecke als natürliches Vorbild	8
3.1.1 Natürlicher Laufvorgang der Stabheuschrecke	9
3.1.2 Steuerung des einzelnen Beines	10
3.1.3 Koordination der Beine	11
3.2 Koordinatensysteme am Laufroboter	13
3.2.1 Definition der Koordinatensysteme	13
3.2.2 Beziehungen der Koordinatensysteme zueinander	14
3.3 Direkte Kinematik	16
3.3.1 Denavit und Hartenberg Notation	16
3.4 Krafttransformationen	20

3.5 Kraftgeführte Bewegungen	23
3.5.1 Prinzipien der Kraftregelung	23
3.6 Kraftverteilungen	24
3.6.1 Statische Gleichgewichtsbedingungen	24
3.6.2 Kraftverteilung für drei Stützfüße	25
3.6.3 Kraftverteilung für mehr als drei Stützfüße	30
3.7 Regelungstechnische Grundlagen	31
3.7.1 Laplace–Transformation	31
3.7.2 Grundbegriffe	32
3.7.3 Prinzip des kontinuierlichen PID–Regler	33
3.8 Modular Controller Architecture (MCA)	35
3.8.1 Architektur	36
3.8.1.1 Module	37
3.8.1.2 Gruppen	37
3.8.1.3 Kanten	38
3.8.1.4 Part	39
3.8.1.5 Echtzeit– und Netzwerktransparenz	40
3.8.2 Tools	40
3.8.2.1 MCAGUI	40
3.8.2.2 MCAbrowser	41
3.9 Metrowerks Codewarrior	42
4 Aufbau des sechsbeinigen Roboters	44
4.1 Mechanischer Aufbau des Laufroboters Lauron IVb	45
4.1.1 Der Körper des Laufroboters Lauron IVb	45
4.1.2 Die Beine des Laufroboters Lauron IVb	45
4.1.2.1 Kinematischer Aufbau der Beine	47
4.1.2.2 Arbeitsbereich eines Beines	47
4.1.2.3 Aktoren und Sensoren	47
5 Aufbau des Steuerungssystems	52
5.1 Mikrocontrollerplatinen	52
5.1.1 Mikrocontroller	53

5.1.2	Beincontroller	54
5.2	Industrie-PC/104	55
5.2.1	Hauptcontroller PC/104	56
5.3	Messwerterfassung	57
5.4	Kommunikation	57
5.4.1	Das TCP/IP Referenzmodell	58
5.4.2	Controller Area Network	60
6	Betriebssystem	62
6.1	Echtzeitsystem	62
6.1.1	Linux	64
6.1.2	Real Time Linux	66
6.1.3	RTLinux und RTAI	68
7	Analysse der vorhandenen Steuerung	71
7.1	MCA-Steuerung	71
7.1.1	MCA-Part	71
7.1.2	Group-“BehaviorGroup“	72
7.1.3	Group-“SignalProcessingGroup“	74
7.1.4	Group-“DSP-Remote-Part“	75
7.2	Beincontroller	76
7.2.1	Puls-Weiten-Modulation (PWM) Theorie	76
7.2.2	Digitaler PID-Stellungsalgorithmus	77
8	Lösungskonzept	81
8.1	Zusammenstellung der Anforderungen	81
8.2	Top-Down-Entwurf	81
8.2.1	Implementierung kraftgeführter Bewegungen	82
8.2.1.1	MCA-Group: ActiveCompliance	83
8.2.2	Bohroperation	85
8.2.2.1	MCA-Group: DrillControl	86
8.3	Beincontroller Anforderungen	87
8.3.1	PWM-Kennlinienlinearisierung	87
8.3.1.1	Beincontroller PWM-Offset	89

INHALTSVERZEICHNIS	vi
9 Entstandene Module	91
9.1 ActiveCompliance	91
9.2 DrillControl	94
10 Zusammenfassung	98
Literaturverzeichnis	100

Abbildungsverzeichnis

3.1	Stabheuschrecke[17]	9
3.2	Beinsegmente und Hauptbewegungssachsen im vertikalen Schnitt [33]	10
3.3	Schema zur Generierung der Beinbewegung [28]	11
3.4	Koordinatensysteme Lauron IVb mit Ansicht von oben	13
3.5	Koordinatensystem Lauron IVb – 3D Ansicht	15
3.6	Roboterkinematik [3]	16
3.7	Koordinatensystem eines Beines	17
3.8	Kette von starren Körpern	17
3.9	Geometrische Parameter eines Beines	18
3.10	Fußpunktkoordinatensystem	21
3.11	Geometrische Parameter eines Beines	27
3.12	Ebene Koordinatensysteme zur Masseschwerpunktberechnung . .	29
3.13	Grundstruktur eines Regelungskreises	33
3.14	Blockschaltbild des geschlossenen Regelungssystems	34
3.15	Blockschaltbild PID–Regler im Bildbereich	35
3.16	Sprungantwort und Symbol des PID–Reglers	36
3.17	Modulaufbau Inputs/Outputs	38
3.18	Beispiel einer MCA Gruppe	38
3.19	<i>MCAGUI</i> Steuerungssystem für LAURON IVb	40
3.20	<i>MCAbrowser</i> untersucht Kanten, <i>Control input</i> einer Gruppe .	41
3.21	Codewarrior Entwicklungsumgebung	43
4.1	Die Laufmaschine Lauron IVb	44
4.2	Der Körper der Laufmaschine Lauron IVb	47

4.3	Beinaufnahme	48
4.4	Arbeitsbereich eines Beines	48
4.5	<i>Faulhaber</i> –Kleinstmotor	49
4.6	Verformungskörper des Kraftsensors	50
4.7	Fußsensor	50
4.8	Kalibrierung des Kraftsensors	51
5.1	DSPF803 Blockdiagramm [30]	53
5.2	Physikalischer Aufbau des Beincontrollers	54
5.3	<i>Hardware Abstraction Layer</i>	55
5.4	PC/104 [7]	55
5.5	PC/104 Blockdiagramm [42]	56
5.6	PC/104 mit Erweiterungsplatinen	57
5.7	Kommunikation der Hardwarekomponenten	58
6.1	Kommunikation	63
6.2	Echtzeit-Kernel mit Linux-Kernel als Idle-Task [41]	66
6.3	RT-Linux Schalenmodell [41]	67
7.1	„Virtual MainGroup“ der MCA–Steuerung	72
7.2	„BehaviourGroup“ der MCA–Steuerung	73
7.3	„LegControl“ der MCA–Steuerung	74
7.4	„SignalProcessingGroup“ der MCA–Steuerung	75
7.5	„DSP-Remote-Part“ der MCA–Steuerung	75
7.6	PWM–Erzeugung	77
7.7	PWM–Drehzahl–Kennlinie	78
7.8	Digitaler PID–Gelenkregelkreis	78
7.9	Integration durch Bildung der Rechtecksumme	79
8.1	Mögliche Krafteinwirkungen auf den Roboterkörper und die entsprechenden Ausweichbewegungen	84
8.2	„BehaviourGroup“ mit Steuerungserweiterung	84
8.3	Linearisierte PWM–Drehzahl–Kennlinie	88
8.4	Digitaler PID–Gelenkregelkreis mit Linearisierung der Regelstrecke	88
8.5	Kennlinienlinearisierung	89

ABBILDUNGSVERZEICHNIS

ix

9.1 Aufbau der „ActiveCompliance–Group“	91
9.2 Aufbau der „DrillControl–Group“	95

Kapitel 1

Einleitung

Seit Generationen stellen sich Ingenieure und Wissenschaftler der Herausforderung, künstliche Laufmaschinen zu konstruieren. Der Fortschritt in der Computertechnologie hat die Forschung auf diesem Gebiet in den letzten Jahrzehnten enorm angetrieben. Durch den enormen Leistungszuwachs und die Kompaktheit der Rechensysteme ist es möglich geworden, komplexe Steuerungsaufgaben zu realisieren.

Mit Hilfe der Lauftechnologie von Laufmaschinen lässt sich ein großer Bereich unseres Planeten erreichen. Ferner können technische Operationen an Orten ausgeführt werden die für den Menschen nicht erreichbar oder die Arbeit zu gefährlich erscheint.

1.1 Motivation

In der Gruppe Interaktive Diagnose- und Servicesysteme (IDS) [6] am Forschungszentrum für Informatik (FZI) [4] an der Universität Karlsruhe wird seit Beginn der 90er Jahre das sechsbeinige Laufen untersucht. Im Zuge dieser Untersuchungen wurden Modelle von sechsbeinigen Laufmaschinen entworfen, welche sich an der Geometrie der Stabheuschrecke orientierten. Die bis dato letzte Entwicklungsstufe wird durch die vierte Generation der Laufmaschine Lauron IVb (**LAUfender ROboter, Neuronal gesteuert**) bestimmt. Diese Laufmaschine beherrscht das reaktive Laufen. Das Steuerungssystem baut auf dem ebenfalls am FZI entwickelten Modularen Steuerungssystem MCA (Modular Controller Archtecture) auf.

Zur Erforschung des Verhaltens von Laufmaschinen in komplexer Umgebung und deren Einsatz im Bereich technischer Operationen wurde für das Institut für Robotik der Hochschule Mannheim der Prototyp der sechsbeinigen Laufmaschine Lauron IVb aus Karlsruhe angeschafft.

Die Anwendungsmöglichkeit des Laufroboters Lauron IVb ist nicht allein auf das reaktive Laufverhalten begrenzt. Ferner bietet die Laufmaschine die technische Voraussetzung mit der Umgebung zu interagieren und technische Operationen

auszuführen. Diese technischen Anwendungen unterliegen der Forderung, neben Transport bzw. Selbsttransport am Zielort bestimmte Arbeiten auszuführen. Darunter fallen beispielsweise Manipulation von Objekten oder die Positionierung von Instrumenten bzw. Sensoren. Diese Arbeiten können durch Körperbewegungen wirkungsvoll unterstützt werden.

Die Laufmachine Katharina [38] beherrscht ein statisches drei-Fuß Laufmuster und kraftgeführte Bewegungen. Hingegen benutzt die Laufmaschine Lauron IVb Algorithmen zur Realisierung reaktiven Laufverhaltens. Eine Synthese aus den positiven Eigenschaften der Steuerungen der Laufmaschinen führt zu einer Verbesserung des Gesamtsystems. Auf Grundlage dieser Überlegung wird die Steuerung der Laufmachine Lauron IVb um kraftgeführte Bewegungen erweitert die unter anderem technische Operationen ermöglichen.

1.2 Ziel und Aufbau der Arbeit

Ziel dieser Arbeit ist es, sensorbasierte Steuerungsalgorithmen zu entwickeln, die den Einsatz des Laufroboters Lauron IVb unter dem Gesichtspunkt technischer Anwendungen erlauben. Hierbei spielt die Analyse der vorhandenen Steuerung eine wichtige Rolle. Sie dient dazu, die Steuerung durch neue notwendige Module zu erweitern bzw. vorhandene Module an die neue Steuerung anzupassen. Die sensorbasierten Bewegungsalgorithmen sollen nach einer analytischen Beschreibung auf der Laufmaschine Lauron IVb implementiert werden. Ferner werden die Bewegungsalgorithmen unter der Voraussetzung entwickelt, dass sich der Roboterkörper parallel zur Ebene befindet, welche senkrecht zum Gravitationsvektor steht.

In Kapitel 2 werden die bisherigen Arbeiten im Bereich der Steuerung sechsbewegter Laufroboter vorgestellt.

Kapitel 3.1.1 befasst sich mit dem natürlichen Laufvorgang von (Stab-)Heuschrecken. Hierbei wird das Insekt kurz vorgestellt und die Steuerung des einzelnen Beines sowie die Mechanismen zur Koordination erörtert. In Kapitel 3.2 erfolgt die Definition der Koordinatensysteme der Laufmaschine Lauron IVb und ihre Beziehungen zueinander. Die direkte Kinematik nach Denavit und Hartenberg [52] wird in Kapitel 3.3 vorgestellt. Die Krafttransformationen der mit den Fußsensoren gemessenen Kräfte wird in Kapitel 3.4 erörtert. In Kapitel 3.5 werden die Grundlagen für das Verständnis kraftgeführter Bewegungen geschaffen. Kapitel 3.5.1 geht auf die Kraftverteilung der Gewichtskraft auf die einzelnen Stützfüße der Laufmaschine ein. In Kapitel 3.7 werden die Regelungstechnischen Grundlagen geschaffen die für das Verständnis des Positionsreglers auf den Mikrocontrollern erforderlich ist. Kapitel 3.8 stellt das zur Programmierung verwendete Softwareframework MCA (Modular Controller Architecture) vor. In Kapitel 3.9 wird die zur Entwicklung der Beincontrollersoftware eingesetzte Entwicklungsumgebung Codewarrior vorgestellt.

Kapitel 4 beschreibt die mechanische Konstruktion des Laufroboters Lauron IVb, der für die Entwicklung der Steuerungserweiterung eingesetzt wurde.

Mechatronische Systeme bilden eine Einheit aus Mechanik, Elektronik und Informationsverarbeitung. Kapitel 5 befasst sich aus diesem Grund mit dem Aspekt der Hardware, der Messwerterfassung mit Hilfe der Sensoren und der Kommunikation der Mikrocontrollern mit dem Hauptprozessor PC/104.

In Kapitel 6 wird das verwendete Betriebssystem des Industrie-PCs PC/104 vorgestellt.

Da es für die Erweiterung der vorhanden Steuerung essentiell ist diese zuvor zu Analysieren erfolgt in Kapitel 7 eine Analyse der vorhandenen Steuerung.

Kapitel 8 beschreibt die dieser Arbeit zugrunde liegenden Steuerungsansätze zur realisierung Kraftgeführter Bewegungen, mit deren Hilfe Körperbewegungen erzeugt bzw. überlagert werden.

Die auf Basis des Kapitels 8 entstandenen Softwaremodule werden in Kapitel 9 beschrieben.

In Kapitel 10 erfolgt eine Zusammenfassung der Arbeit.

Kapitel 2

Steuerung von Laufmaschinen

In den letzten Jahrzehnten wurden viele Laufmaschinen meist mit unterschiedlichen Zielsetzungen entwickelt. Die Bandbreite erstreckt sich von einbeinigen über zwei-, vier-, sechs und achtbeinige bis zu anderen mehrbeinigen Maschinen. So vielfältig die Maschinen und deren Zielsetzungen sind, so unterschiedlich stellen sich die verwendeten Steuerungsansätze dar. Manche wählen den Zugang über die klassische Robotik und steuern die Maschine ausschließlich mit analytischen algorithmischen Mitteln, andere versuchen, das Verhalten biologischer neuronaler Netze von Insekten direkt mit Hilfe künstlicher neuronaler Netze nachzubilden; die meisten Steuerungen sind irgendwo dazwischen anzusiedeln. Um eine Laufmaschine zu steuern, sind in jedem Fall die folgenden Aufgaben zu lösen:

- Erfassen der Umwelt mit Hilfe der Sensoren
- Reaktion auf äußere Störeinflüsse
- Bestimmung der Trajektorien der einzelnen Beine
- Koordination der Beine
- Ansteuerung der Aktoren zur Bewegung der Beine

Im Folgenden werden bisherige Arbeiten im Bereich der Steuerung sechsbeiniger Laufroboter vorgestellt. Dabei werden die Verfahren in drei Klassen gegliedert:

- statische Verfahren
- reaktive Verfahren
- planende Laufsteuerungen

2.1 Statische Verfahren

Statische Verfahren erzeugen ein Laufmuster nach einem fest vorgegebenen Schema. Äußere Einflüsse haben dabei keinen Einfluss auf die Laufbewegung. Kollisionen, Hindernisse oder unebenes Gelände können meist nur unzureichend behandelt werden. Für flaches Gelände sind diese Verfahren jedoch gut geeignet. Dank ihrer Einfachheit lassen sie sich mit wenig Aufwand umsetzen.

2.1.1 Laufmuster

Ein sehr einfaches Verfahren sind die Laufmuster (engl. gait pattern). Ein Laufmuster wird durch die Abfolge, nach der die einzelnen Füße angehoben und abgesetzt werden definiert. Die Endstellung der Füße entspricht dabei der am Anfang. Durch die zyklische Ausführung der Laufmuster wird eine kontinuierliche Bewegung erzeugt.

Laufmuster werden in mehreren Arbeiten verwendet. Eine direkte Umsetzung wurde bei der Laufmaschine Katharina [38], TUM [25][31] und [28] angewandt. In manchen Arbeiten wurden Laufmuster aber auch auf Neuronale Netze übertragen. Dabei werden die Berechnungen des Laufmusters und die Bestimmung der Gelenkwinkel zusammengefasst. Als Ergebnis ist eine Optimierung der Rechengeschwindigkeit zu verzeichnen jedoch auf Lasten der Genauigkeit. Die Umsetzung via Neuronale Netze wurden beispielsweise bei den Laufmaschinen Lauftron I und II [48][49] angewandt.

2.1.2 Zelluläre Automaten

Das Laufverhalten von Insekten wie der indischen Stabheuschrecke (*Casausius morosus*) wurde bereits ausgiebig untersucht [22]. Dabei wurden sechs einfache Regeln gefunden [24], die ein einfaches Laufverhalten ermöglichen. Diese Regeln werden meist in Form eines zellulären Automaten beschrieben. Dabei entspricht die Struktur des Automaten der des strickleiterförmigen Nervensystems von Insekten. Einzelne Beine sind nur mit ihren direkten Nachbarn verbunden. Über diese Verbindungen werden Reize ausgelöst, die ein Abheben oder Senken der Beine unterdrücken bzw. aktivieren.

Der Vorteil dieses Verfahrens gegenüber Laufmustern liegt darin, dass Unregelmäßigkeiten in der Synchronität der Beine durch die Regeln automatisch ausgeglichen werden können.

2.2 Reaktive Verfahren

Der Nachteil aller statischen Laufsteuerungen liegt darin, dass sie nur für ebenes Gelände geeignet sind. Hindernisse können zu Störungen der Laufbewegung führen. Um dies zu vermeiden werden reaktive Komponenten in die statische Laufsteuerung integriert. Durch eine Art Reflex werden zusätzliche Verhaltensweisen erreicht, um die bestehenden Nachteile zu kompensieren.

Eine Möglichkeit liegt in der Verwendung des *Elevator-Reflex* [33][35]. Dieser dient der Überwindung kleinerer Hindernisse. Stößt ein Bein bei einer Bewegung gegen ein Hindernis, wird der Fuß automatisch angehoben. Durch mögliche mehrmalige Anwendung kann die entsprechende Hindernis überwunden werden. Somit ist auch das Laufen über Stufen möglich.

Für Vertiefungen im Untergrund wird ein *Suchreflex* [33] angewendet. Findet das Bein beim Aufsetzen keine festen Untergrund, so wird innerhalb des Arbeitsbereiches des entsprechenden Beins nach einem geeigneten Aufsetzpunkt gesucht. Wird dieser gefunden so wird der Laufvorgang vortgesetzt.

Ein weiterer Reflex ist der *Aufstandreflex* [35]. Er wird zum Verhindern zeitlicher Störungen eingesetzt, indem sichergestellt wird, dass sich zu jedem Zeitpunkt die richtige Anzahl an Füßen auf dem Untergrund befindet. Dies kann aber unter bestimmten Bedingungen zu Störungen im Bewegungsablauf führen. Fordert beispielsweise das Laufmuster ein gleichzeitiges Aufsetzen mehrerer Beine so kann das teilweise bedingt durch die technische Umsetzung nicht synchron erfolgen. Dies führt zur permanenten Auslösung des Reflexes. Somit dominiert die Ausnahmebehandlung das normale Verhalten, welches zur Störung der Laufbewegung führt.

Die Reflexe realisieren einen einfachen Ansatz, indem bestehende statische Laufsteuerungen eine Anwendung auf unebenem Untergrund mit Hindernissen ermöglichen. Für die technische Realisierung werden Kraftsensoren verwendet.

Die Laufmaschinen Lauron III, IV und IVb [33] arbeiten auf dem Prinzip reaktiver Laufsteuerungen.

2.3 Planende Laufsteuerungen

Der Nachteil reaktiver Laufsteuerungen liegt darin, dass sie auf ein Hindernis erst dann reagieren wenn eine Störung des Laufvorgangs auftritt. Dieser Nachteil wird mit planerischen Laufsteuerungen umgangen. Diese Laufsteuerungen versuchen vor Eintritt einer Störung diese zu erkennen und zu umgehen. Somit wird eine Störung der Laufbewegung verhindert. Hierzu werden Sensoren eingesetzt, mit denen die Umgebung analysiert wird. Unter Einsatz dieser Daten erfolgt eine Planung in der eine Kollision mit Hindernissen vermieden wird.

Die Planung kann auf unterschiedlichen Ebenen erfolgen. In [35] dienen die Sensorinformationen lediglich der Vermeidung des Elevator-Reflexes, indem das Bein vorausschauend auf die erforderliche Höhe gehoben wird. Die restliche Steuerung erfolgt wie bei reaktiven Laufmustern.

Konzepte zur Steuerung von Robotern finden auch in anderen Bereichen Verwendung. So existieren ähnliche Verfahren für die Steuerung von Charakteren in Computerspielen. Dort werden für realistische Laufbewegungen Gelenkmodelle betrachtet. Der Charakter muss sich wie ein gelenkbasierter Roboter durch eine virtuelle Landschaft bewegen. Ein entsprechendes Planungsverfahren wird in [40] vorgestellt. Hier werden optimale Bewegungen durch Backtracking berechnet.

Ein Laufplaner basierend auf dem heuristischen Optimierungsverfahren Ordinal Optimization wird in [37][21] vorgestellt.

Kapitel 3

Zugrunde liegende Konzepte und Methoden zur Bewegungssteuerung

In diesem Kapitel werden die Grundlagen, die zur Konzeption der sensorbasierten Bewegungsalgorithmen heran gezogen wurden vorgestellt. Da diese Arbeit auf einer bereits auf dem Laufroboter vorhandenen Bewegungssteuerung aufbaut, wird im folgenden auf die Grundlagen des Laufvorganges und der Steuerung einzelnen Beines eingegangen. Da die Bewegungssteuerung auf der Theorie der Fortbewegung einer Stabheuschrecke beruht, wird einleitend der natürliche Laufvorgang am Beispiel der Stabheuschrecke beleuchtet. Ferner werden die verwendeten Koordinatensysteme, ihre Beziehung zueinander, die direkte Kinematik nach Denavit und Hartenberg, Krafttransformation, kraftgeführte Bewegungen und Kraftverteilungen erläutert. Des Weiteren wird auf die Theorie der Regelungstechnik eingegangen. Abschließend wird das zur Implementierung verwendete Framework MCA2.3 (Modular Controller Architecture) vorgestellt.

3.1 Stabheuschrecke als natürliches Vorbild zur Bewegungssteuerung

Während der letzten Jahrmillionen hat die Natur eine Vielzahl an Fortbewegungsarten hervorgebracht. Einige Lebewesen spezialisierten sich auf die Fortbewegung auf dem Land, andere in der Luft oder im Wasser. Bemerkenswert ist die perfekte Adaption der Mobilität an ihre angestammte Umgebung. Bis zum heutigen Evolutionsstand erstreckt sich die beingebundene Fortbewegung vom Zweibeiner über Vier-, Sechs und Achtbeiner usw.. Die Schwierigkeit besteht nun darin, Roboter zu entwickeln, die in ihrer Bewegungsart und Mobilität an ihre Vorbilder aus der Natur anknüpfen und diese so perfekt wie möglich nachbilden. Für den Roboter Lauron standen Insekten Modell, die sechs Beine für ihre Fortbewegung zur Verfügung haben. Mit sechs Beinen ist es möglich, während des

Laufvorgangs immer in einem statisch stabilen Zustand zu bleiben ohne dass die Komplexität aufgrund allzu vieler Beine übermäßig ansteigt.

Wie bereits erwähnt hat sich das Laufverhalten der Insekten über die Jahrtausende entwickelt und stets perfektioniert. Daher liegt es nahe, Bewegungsabfolgen und Beinsteuuerungen entsprechend der Natur nachzubilden und diese auf die künstliche Laufmaschine zu übertragen. Der dieser Arbeit zugrundeliegende Laufroboter Laron IVb ist einer Stabheuschrecke nachempfunden. Daher wird im Folgenden zunächst auf das natürliche Laufverhalten dieses Insekts eingegangen.

3.1.1 Untersuchung des natürlichen Laufvorgangs der Stabheuschrecke

Die folgenden Ausführungen über den natürlichen Laufvorgang von Stabheuschrecken sind den Arbeiten von [33][36][28][24][22][23] entnommen. Die Stabheuschrecke (Abbildung 3.1) ist wohl das bezüglich des Laufverhaltens am intensivsten erforschte Insekt.



Abbildung 3.1: Stabheuschrecke[17]

Dieser langbeinige Sechsbeiner ist auf die Fortbewegung auf komplexem Untergrund bestehend aus Zweigen und Blättern ausgerichtet. Wie die meisten Insekten kann die Stabheuschrecke mit allen unterschiedlichen Orientierungen zur Schwerkraft laufen. Der Körper des Tieres ist in drei Teile Kopf (Caput), Brust (Thorax) und Hinterleib (Abdomen) gegliedert. Die Brust ist wiederum in drei Segmente unterteilt, die jeweils ein Beinpaar tragen. Der Kopf trägt zwei lange Fühler. Jedes Bein hat drei Hauptsegmente (Coxa [Hüfte], Femur [Oberschenkel], Tibia [Unterschenkel]) und drei Hauptgelenke (Subcoxal [α], Coxa-Trochanter [β], Femur-Tibia [γ]), die ungefähr in einer Ebene liegen (Abbildung 3.2). Die körperfernen Gelenke (β und γ) sind Scharniergelenke mit jeweils einem Freiheitsgrad; diese sind für die Beuge- und Streckbewegung des Beins verantwortlich. Das körpernahe (α) Gelenk, welches das Bein mit dem Körper verbindet, ist komplexer und gehorcht einem Bewegungsablauf mit zwei Freiheitsgraden. Seine Hauptbewegung entspricht der eines Scharniergelenks mit einer aus der Senkrechten geneigten Achse. Dieser Freiheitsgrad ist hauptsäch-

lich für die Vorwärts- und Rückwärtsbewegung des Beins verantwortlich. Somit ermöglichen die Hauptachsen der drei Hauptgelenke die minimal erforderliche Anzahl an Freiheitsgraden, um den Fuß (Tarsus) beliebig im dreidimensionalen Raum plazieren zu können. Jedes Bein verfügt über einen weiteren Freiheitsgrad, da die Neigung der Achse des α -Gelenks mit Hilfe des zweiten Freiheitsgrades dieses Gelenkes verändert werden kann. Dieser zusätzliche Freiheitsgrad wird zwar gelegentlich zum Laufen verwendet, hat jedoch in erster Linie die Aufgabe, die Beine zum Schutz nahe an den Körper heranzuziehen. Daher kann dieser vernachlässigt und als Beitrag zur β -Gelenkbewegung angesehen werden.

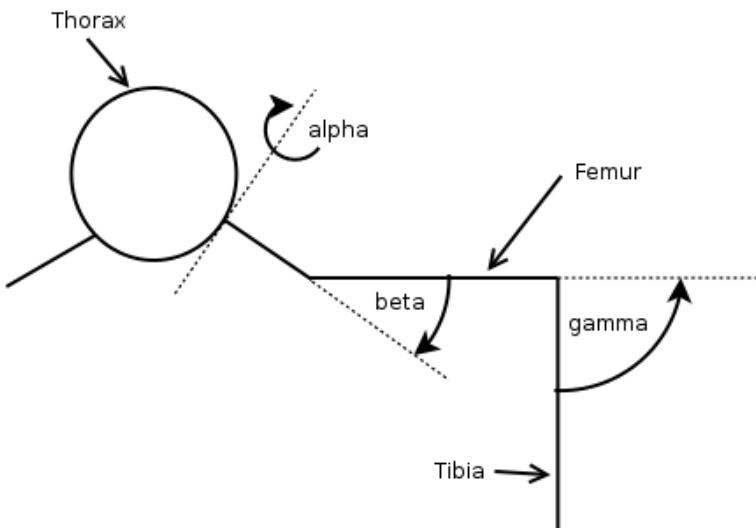


Abbildung 3.2: Beinsegmente und Hauptbewegungsachsen im vertikalen Schnitt [33]

3.1.2 Steuerung des einzelnen Beines

Die Bewegung eines einzelnen Beins wird durch ein mechano-neuronales System erzeugt, das als Schrittmustergenerator bezeichnet wird. Ein laufendes Bein führt eine ständige zyklische Bewegung aus, die grundsätzlich aus zwei Teilen besteht: In der *Stemphase* steht das Bein auf dem Untergrund, stützt den Körper und schiebt ihn in die gewünschte Bewegungsrichtung. In der nachfolgenden *Schwingphase* hebt das Bein vom Boden ab, um in der Luft zum Anfangspunkt der nächsten *Stemphase* zu schwingen. Bei der Stabheuschrecke wird die Beinbewegung in beiden Phasen von mindestens einem Servo-System gesteuert, das über negative Rückkopplung die Beingeschwindigkeit reguliert. Der Mustergenerator entscheidet über den Wechsel zwischen den zwei Phasen und legt somit die Bewegungsrichtung fest. Dabei ist vor allem der Wechsel zwischen *Stemphase* zur *Schwingphase* kritisch, da die Unterstützungsfunction des Beins beendet wird; tritt der Wechsel im falschen Moment ein, hat das Tier keinen Halt mehr und kippt um. Drei Parameter beeinflussen einen Wechsel: die Position des Beins, die Last unter der das Bein gerade steht und die Phase

im Schritzyklus der anderen Beine. Die ersten beiden Parameter hängen vom Zustand des Beins selbst ab. Sie sichern, dass das Bein nur dann vom Boden abhebt, wenn die Position weit genug hinter dem Körper liegt und so die Belastung des Beins klein genug ist. Dem dritten Parameter kommt eine größere Bedeutung zu, da er von der Koordination der Beine untereinander abhängt. Dieser Einfluss entspringt nicht der Umgebung des Beins selbst, sondern wird durch Informationen terminiert, die von den Mustergeneratoren der anderen Beine über neuronale Pfade zum Bein übermittelt werden. Darüberhinaus könnten direkte Umgebungseinflüsse der sensitiven Organe von Nachbarbeinen existieren, die nichts mit den Mustergeneratoren der anderen Beine zu tun haben. Auf die Koordination wird im nächsten Abschnitt ausführlicher eingegangen. Abbildung 3.3 zeigt hierzu ein Schaltungsmodell zur Generierung der Beinbewegung. Der linke Teil im Modell stellt die hierarchisch höhergestellte Schaltung dar, die zwischen den beiden Zuständen *Stemm-* und *Schwingphase* entscheidet. Die Rückkopplung erzeugt die beiden alternativen Zielpositionen *AEP* (anterior extreme position – vordere Bewegungsgrenze) und *PEP* (posterior extreme position – hintere Bewegungsgrenze). Diese werden mit der aktuellen Position verglichen und das Ergebnis noch von der aktuellen Beinlast beeinflusst. Die Ausgabe dieser Schaltung stellt dann die Eingabe der zweiten untergeordneten Schaltung dar, ein über die Geschwindigkeit rückgekoppelter Regler. Je nach Entscheidung der übergeordneten Schaltung zwischen *Schwing-* (positive Ausgabe) und *Stemmphase* (negative Ausgabe) wird das Bein nach vorne (positive Geschwindigkeit) oder hinten (negative Geschwindigkeit) bewegt.

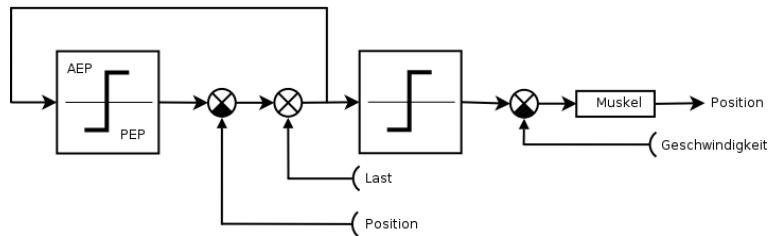


Abbildung 3.3: Schema zur Generierung der Beinbewegung [28]

3.1.3 Koordination der Beine

Wie oben schon beschrieben hängt der Übergang von *Stemmphase* zu *Schwingphase* eines Beins von der Phase der anderen Beine ab. Koordinationsmechanismen beeinflussen aber nicht nur den Stemm-Schwing-Übergang im *PEP* sondern auch den Schwing-Stemmübergang im *AEP* und darüber hinaus die Stärke der Antriebskraft während der *Stemm-* und *Schwingphase*. Diese Mechanismen treten immer nur zwischen direkten Nachbarbeinen auf, wobei die Kopplung zwischen Nachbarbeinen der gleichen Körperseite generell stärker ausgeprägt ist als die Kopplung zwischen gegenüberliegenden Nachbarbeinen. Es können insgesamt sechs Mechanismen identifiziert werden [24]:

1. Ein Bein wird am Start der *Schwingphase* gehindert, sofern das hinte-

re Bein seine Schwingphase noch nicht beendet hat. Dies wird erreicht, indem die Grenze für den Stemmphasenstart (*PEP*) weiter nach hinten verschoben und somit die Stemmpause des Beins verlängert wird. Dieser Mechanismus hat einen hemmenden Einfluss und wirkt von hinten nach vorne.

2. Beginnt ein Bein seine Stemmpause, so drängt es das vordere und das gegenüberliegende Bein, in die Schwingphase überzugehen. Somit wird die Stemmpause dieser Beine verkürzt. Dieser Mechanismus hat einen reizenden Einfluss und wirkt von hinten nach vorne sowie seitwärts.
3. Je weiter ein Bein während der Stemmpause nach hinten bewegt wird, desto früher wechselt das hintere und auch das seitliche Bein in die Stemmpause. Somit wird erreicht, dass die Nachbarbeine in die Stemmpause übergehen bevor das Bein abhebt. Dieser Mechanismus hat ebenfalls einen reizenden Einfluss und wirkt von vorne nach hinten sowie teilweise seitwärts.
4. Beim Laufen wird der Fuß des Hinterbeins direkt hinter den des Mittelbeins, der des Mittelbeins direkt hinter den des Vorderbeins und der des Vorderbeins direkt hinter den Fühler gesetzt. Somit wird das nachfolgende Bein beim Finden von Fußpunkten unterstützt (*targeting*). Für das Klettern auf Ästen ist dieser Mechanismus essentiell, auf durchgängigem Untergrund hingegen trivial, weshalb er anscheinend nur bei Kletterern ausgebildet ist. Dieser Mechanismus wirkt von vorne nach hinten.
5. Erhöht sich die Antriebskraft eines Beins aufgrund eines Widerstands während der Stemmpause, so erhöht sich die Antriebskraft auch bei allen benachbarten Beinen (*coactivation*). Dieser Mechanismus wirkt in alle Richtungen außer zwischen den Hinterbeinen.
6. Ist der targeting-Mechanismus erfolglos, kann es vorkommen, dass das hintere Bein auf den Fuß des vorderen tritt. In einem solchen Fall ist der treading-on-tarsus (*TOT*) Reflex zu beobachten. Das hintere Bein wird nochmal kurz angehoben und leicht nach hinten gesetzt; offensichtlich vermeidet dieser Mechanismus ein Stolpern und wirkt von vorne nach hinten.

Punkt 1,2 und 3 sind redundant, weil sie alle den gleichen Effekt haben: Die fast sofortige Wiederherstellung der Beinkoordination im Fall einer Störung. Im Schaltungsmodell aus Abbildung 3.3 entsprechen sie der Änderung des *AEP/PEP*-Werts der Relais-Charakteristik. Die nächsten drei Mechanismen 4,5 und 6 haben einen geringeren Einfluss.

Das Gangmuster der Stabheuschrecke auf ebenem Gelände bildet eine mechatronische Welle der Schwingbeine von hinten nach vorne. Beim ungestörten Laufen benutzt sie zwei Gangarten. Der Tripod wird beim schnellen Laufen unter wenig Last eingenommen, wobei abwechselnd die drei Beine 0, 3, 4 und 1, 2, 5 am Boden sind. Ansonsten ist der Tetrapod zu beobachten, bei dem ständig vier Beine am Boden sind. Die Stabheuschrecke beginnt zu laufen, indem sie mit allen auf dem Boden stehenden Beinen nach hinten driftet. Der Kreislauf jedes Beins startet daher immer in der Stemmpause. Beim Laufen in weiten Kurven wird die Koordination der Beine beibehalten und nur die Schrittlänge - also

AEP und *PEP* - angepasst; bei den äußereren Beinen erhöht und bei den inneren erniedrigt. In engen Kurven werden hingegen rechte und linke Beine entkoppelt, so dass die äußereren mit einer höheren Frequenz laufen.

3.2 Koordinatensysteme am Laufroboter

Zur Bewegungssteuerung werden verschiedene Koordinatensysteme als Bezugssysteme verwendet. Auf diese beziehen sich die Koordination der Bewegung (Kinematik) und die Sensorinformationen. Bei Lauron IVb werden kartesische Koordinatensysteme verwendet, somit stehen die Koordinatenachsen paarweise senkrecht aufeinander. Diese Koordinatensysteme stehen alle in klar definiertem Bezug zueinander. Das Modell eines Beins ist dem vereinfachten Modell eines Beins der Stabheuschrecke (siehe Abbildung 3.2) nachempfunden.

3.2.1 Definition der Koordinatensysteme

Im Folgenden werden die Koordinatensysteme genauer erläutert, da diese für die Koordination der Bewegung und die Darstellung der Sensorinformation essenziell sind. In Abbildung 3.4 ist eine schematische Zeichnung des Zentralkörpers mit eingezeichneten Koordinatensystemen zu sehen.

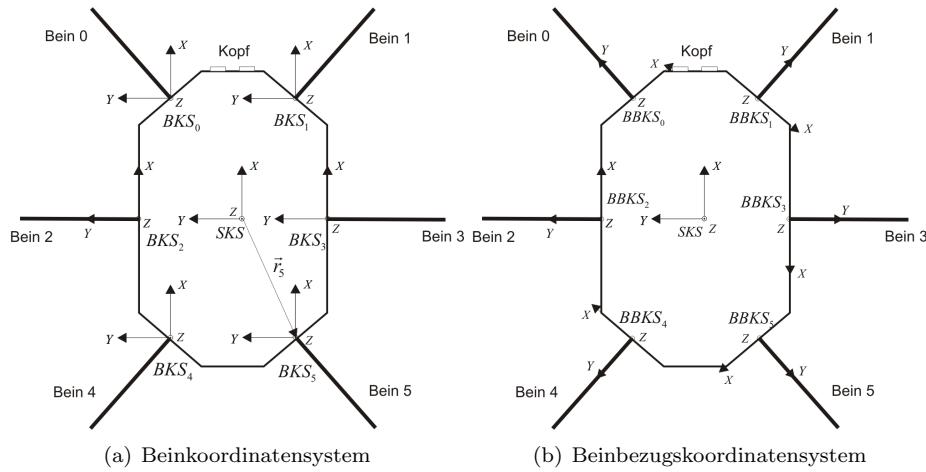


Abbildung 3.4: Koordinatensysteme Lauron IVb mit Ansicht von oben

Das Beinkoordinatensystem (BKS) $\{B^i; X, Y, Z\}$

Alle Beine des Roboters werden mit Indizes versehen. Dabei werden die Beine von vorne links (B^0) nach hinten rechts (B^5) durchnummieriert. Der Koordinatensprung (B^i) liegt im Beinbefestigungspunkt am Zentralkörper. Die *x-Achse* zeigt entlang der Zentralkörperlängsachse nach vorne in die Hauptbewegungsrichtung bzw. dem Kopf, die *z-Achse* zeigt in Bezug auf den Roboterkörper senkrecht nach oben und die *y-Achse* bildet das Rechtssystem (Abbildung 3.4a).

Das Beinbezugskoordinatensystem (BBKS) $\{BB;X,Y,Z\}$

Das *BBKS* liegt wie auch das BKS_i , im Mittelpunkt der sechs Bohrungen des Zentralkörpers, an denen das α -Gelenk befestigt ist (Abbildung 3.4b). Die z -Achse zeigt in Bezug auf den Roboterkörper senkrecht nach oben, die y -Achse hat Ihren Ursprung im Mittelpunkt der sechs Bohrungen und zeigt parallel zur Ebene des Roboterkörpers nach außen vom Zentralkörper weg. Die x -Achse bildet das Rechtssystem. Somit zeigt die x -Achse bei den Beinen B^1, B^3, B^5 nach hinten und bei den linken Beinen B^0, B^2, B^4 nach vorne. Daraus ist ersichtlich, dass das *BBKS* identisch zum BKS_2 ist.

Das Fußpunktkoordinatensystem (FPKS) $\{F;X,Y,Z\}$

Der Ursprung des *FPKS* liegt im Endpunkt der Beinspitze, die z -Achse zeigt in das D-Segment (Abbildung 3.9), die x -Achse ist beim senkrecht vom Körper weg ausgestreckten Mittelbein parallel zur Zentralkörperlängsachse. Somit identisch mit der x -Achse des *BBKS*. Die y -Achse zeigt beim senkrecht auf dem Boden stehenden Bein entlang der Waagerechten vom Zentralkörper weg (Abbildung 3.5).

Das Gelenkkoordinatensystem (GKS) $\{q^i; \theta^i, \alpha^i, \beta^i, \gamma^i\}$

Jedes Bein besitzt Gelenkkoordinaten, die die Freiheitsgrade beschreiben. Diese werden mit α^i für das vom Körper aus gesehene erste Gelenk, β^i für das zweite und γ^i für das dritte Gelenk bezeichnet. Ferner wird der Winkel θ^i für die konstruktionsbedingte Neigung der α^i -Gelenke aus der Waagerechten nach unten eingeführt (Abbildung 3.7). Der Vektor \vec{q} fasst den Neigungswinkel und die drei beschriebenen Freiheitsgrade zusammen. [Wloka 92][52]

Das Symmetriekoordinatensystem (SKS) $\{S;X,Y,Z\}$

Im Symmetriepunkt des Zentralkörpers ist das Symmetriekoordinatensystem zu finden. Es besitzt die gleiche Orientierung wie die Beinkoordinatensysteme (Abbildung 3.4).

Das Untergrundkoordinatensystem (UKS) $\{W;X,Y,Z\}$

Der Ursprung des Untergrundkoordinatensystems stimmt mit den Koordinatensystemen BKS_i und *SKS* überein, jedoch richtet sich die Orientierung nach der Erde. Die x - y -Ebene des Untergrundkoordinatensystems liegt parallel zur Erdoberfläche, die z -Achse zeigt entgegen der Erdgravitation. Die x -Achse des *UKS* entsteht durch Projektion der x -Achse des *SKS* entlang der Erdgravitation. Die y -Achse bildet wie auch beim *SKS* das Rechtssystem.

3.2.2 Beziehungen der Koordinatensysteme zueinander

Die verschiedenen Koordinatensysteme aus dem vorhergehenden Kapitel werden zur mathematischen Beschreibung bestimmter Bewegungen bzw. Auswertung von Sensorinformationen benötigt. Um einen Bezug zwischen den einzelnen Koordinatensystemen herzustellen, sind Transformationen bzw. Rotationen

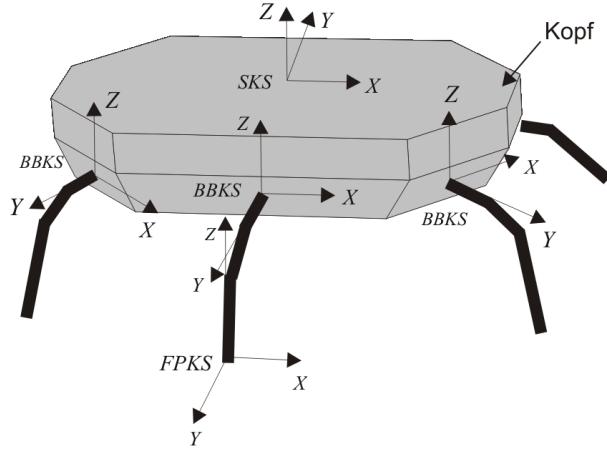


Abbildung 3.5: Koordinatensystem Lauron IVb – 3D Ansicht

notwendig.

UKS und SKS

Der Übergang von *UKS* und *SKS* wird mit Hilfe der Neigungssensoren durchgeführt [33].

GKS und BBKS

Die Gelenkkoordinaten *GKS* sind über inverse Kinematikbeziehungen mit den *BBKS* verbunden, welche die Position der Fußpunkte beschreibt (Abbildung 3.6 und 3.7). Die direkte Kinematik liefert, ausgehend vom *BBKS* die Position der Fußpunkte.

BBKS und BKS_i

Der Übergang vom *BBKS* ins *BKS_i* wird mittels einer Drehung um die z-Achse realisiert, wobei der konstruktionsbedingte Winkel ϕ den Betrag des Drehwinkels der vorderen und hinteren Beine in der z-Achse bezüglich des mittleren Beins (B^2) darstellt (Abbildung 3.4). Somit ergibt sich für den Koordinatenübergang $BBKS \Rightarrow BKS_i$:

$$\begin{array}{ll} BBKS \Rightarrow BKS_0 : \text{Rotation}_z(-\phi) & BBKS \Rightarrow BKS_1 : \text{Rotation}_z(180 + \phi) \\ BBKS \Rightarrow BKS_2 : \text{Rotation}_z(0) & BBKS \Rightarrow BKS_3 : \text{Rotation}_z(180) \\ BBKS \Rightarrow BKS_4 : \text{Rotation}_z(\phi) & BBKS \Rightarrow BKS_5 : \text{Rotation}_z(180 - \phi) \end{array}$$

BKS_i und SKS

Für die Transformation vom *BKS_i* ins *SKS* müssen die Koordinaten um den Abstand r_i vom Beinkoordinatensystem zum Symmetriekoordinatensystem verschoben werden (Abbildung 3.4a $\vec{r_5}$ für das 5. Bein).

$${}^{SKS}\vec{R}_i = {}^{BKS}\vec{r}_i + \vec{r}_i \quad (3.1)$$

FPKS und BBKS

Der Übergang der Orientierung vom *FPKS*, in dem die Kraftsensormesswerte dargestellt werden, zur Orientierung bezüglich des *BBKS* wird Sensordatentransformation genannt. In Kapitel 3.4 wird dieser Übergang erläutert.

3.3 Direkte Kinematik

Die direkte Kinematik [3] beschäftigt sich mit der Frage, wie aus den Gelenkwinkeln der Armelemente die Position und Orientierung des Endeffektors (engl.: Tool Center Point, *TCP*) in Bezug auf das Basiskoordinatensystem (*BK*) bestimmt werden kann.

Zur Berechnung von Position und Orientierung werden einfach die über Sensoren gemessenen Gelenkwinkelwerte q_i in die Denavit-Hartenberg Matrizen (siehe: Denavit-Hartenberg Transformation 3.3.1) eingetragen und diese Matrizen anschließend in der Reihenfolge der Armelemente multipliziert:

$${}^{BK}S_{TCP}(q) = {}^{BK}T_{OKS_1}(q_1) \cdot {}^{OKS_1}T_{OKS_2}(q_2) \cdot \dots \cdot {}^{OKS_{n-1}}T_{TCP}(q_n) \quad (3.2)$$

Lage und Orientierung des TCP-Koordinatensystems TCP können mit Hilfe dieser Matrix nun relativ zum Basiskoordinatensystem ausgedrückt werden, was gleichbedeutend mit der Lösung des direkten kinematischen Problems ist. Die Zusammenhänge versucht folgendes Bild 3.6 zu verdeutlichen:

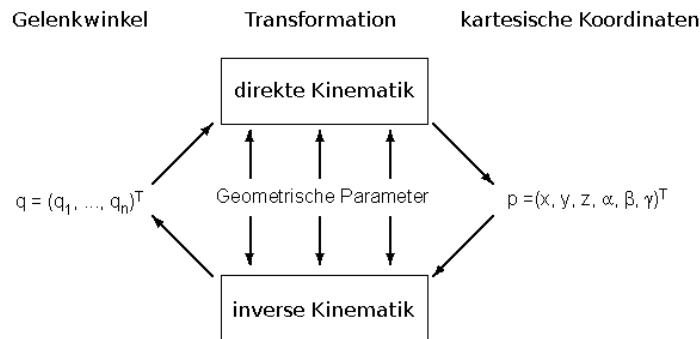


Abbildung 3.6: Roboterkinematik [3]

3.3.1 Denavit und Hartenberg Notation

Um eine kinematische Analyse durchführen zu können, soll ein Bein als ein System aus starren Körpern aufgefasst werden. Denavit und Hartenberg [52]

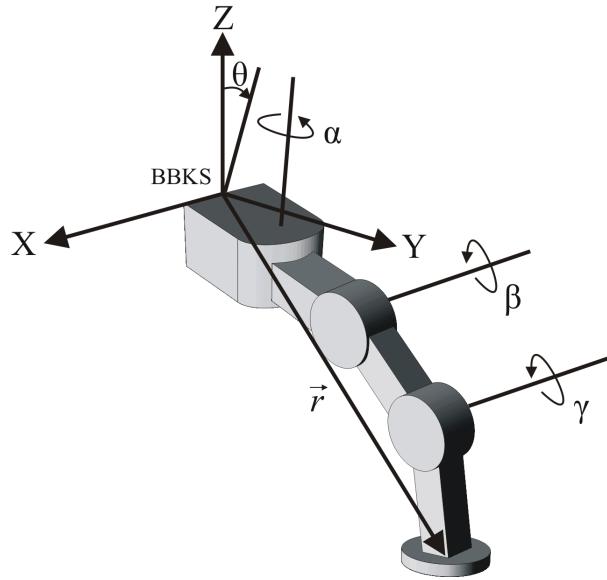


Abbildung 3.7: Koordinatensystem eines Beines

entwickelten hierzu eine Methode, um in Starrkörper Koordinatensysteme zu legen. Die Gelenkwinkel stellen hierbei die Freiheitsgrade dar und dienen als Eingangsstellgrößen. Abbildung 3.7 zeigt das Koordinatensystem und die Gelenkwinkel. Das Koordinatensystem $\{BB;X,Y,Z\}$ stellt das *BBKS* dar, welches direkt im Beinbefestigungspunkt liegt (Abbildung 3.5). Die Gelenkwinkel α, β, γ bilden die zugehörigen Freiheitsgrade. Ferner beschreibt der Winkel θ^i die konstruktionsbedingte Neigung der α -Gelenke aus der Waagerechten nach unten. Wird $\alpha, \beta, \gamma, \theta$ auf null gesetzt, ist das Bein senkrecht ausgestreckt, zeigt entlang der *y-Achse* und verläuft parallel zur *x-y-Achse* des *BBKS*.

Der Vektor \vec{r} ist der Vektor vom Ursprung des *BBKS* zum Fußendpunkt.

Durch Zuordnung von vier Parametern pro Segment S_n lässt sich eine beliebig lange Segmentkette beschreiben [52]. Bei diesen vier variablen vom Gelenktyp

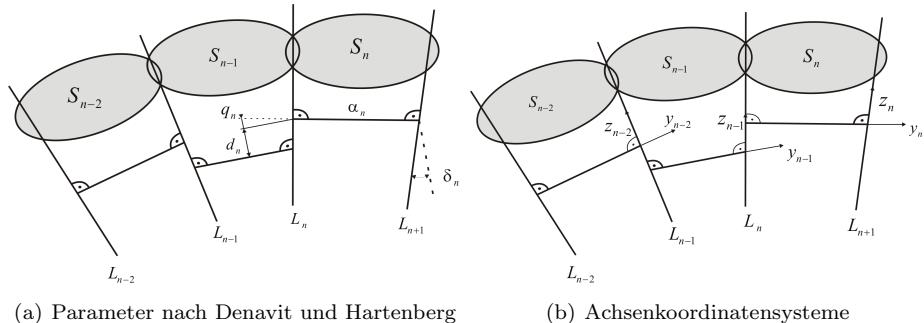


Abbildung 3.8: Kette von starren Körpern

Segment n	a_n	δ_n	d_n	q_n
1	A	$-\frac{\pi}{2}$	0	θ
2	B	$\frac{\pi}{2}$	0	α
3	C	0	0	β
4	D	0	0	γ

Tabelle 3.1: Denavit und Hartenberg Parameter für ein Bein

abhängigen Größen handelt es sich um drei Geometrieparameter und eine Achsvariablen. Abbildung 3.8a zeigt den Aufbau einer Kette von starren Körpern nach Denavit und Hartenberg. Jeder Freiheitsgrad wird durch die Rotation q_n um die Achse L_n dargestellt. Die Distanz zwischen den Achsen wird mit Hilfe der Normalen bestimmt, welche sich zwischen den beiden Achsen konstruieren lassen. Der zugehörige Parameter wird a_n genannt. Der Parameter δ_n beschreibt die Verdrehung der beiden Achsen zueinander. Ferner beschreibt der Parameter d_n den Achsenversatzabstand zwischen den Normalen. Die Abbildung 3.9 zeigt die Anwendung der Parameter auf ein Bein. Tabelle 3.1 ist die Zuordnung der einzelnen Denavit und Hartenberg Parameter (*D-H-Parameter*) pro Segment zu entnehmen.

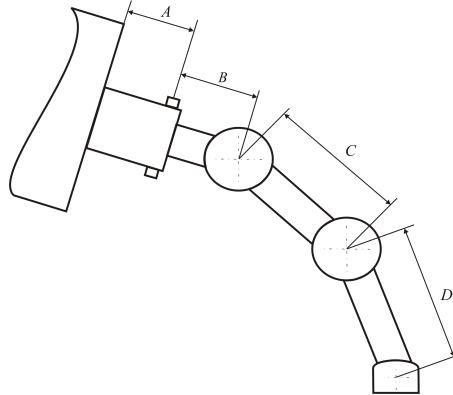


Abbildung 3.9: Geometrische Parameter eines Beines

Mit Hilfe der *D-H-Parameter* lässt sich eine Transformationsmatrix angeben, die die Achsenkoordinatensysteme $\{(n-1)\}$ in $\{(n)\}$ überführen. Hierzu werden Einzeltransformationen, welche aus Rotationen und Translationen bestehen benötigt. Diese Transformation wird mit ${}_{n-1}^n T$ bezeichnet.

$${}_{n-1}^n T = \text{Rot}(x_{n-1}, q_n) \cdot \text{Trans}(0, 0, d_n) \cdot \text{Trans}(0, a_n, 0) \cdot \text{Rot}(y_n, \theta_n) \quad (3.3)$$

Die in Gleichung (3.3) angeführten Elementartransformationen entsprechen folgender Vorschrift:

1. Rotation um die x_{n-1} -Achse um den Winkel q_n .

2. Transformation entlang der *z-Achse* um den Wert d_n .
3. Transformation entlang der *y-Achse* um den Wert a_n .
4. Rotation um die y_n -*Achse* um den Winkel θ_n .

Da es sich bei dem Roboterbein um vier Segmente handelt, müssen die vier homogenen Koordinaten $(_1^0T, \frac{1}{2}^1T, \frac{2}{3}^2T, \frac{3}{4}^3T)$, die sich aus Gleichung 3.3 ergeben, durch Multiplikation miteinander verknüpft werden. Das Ergebnis ist dann die Position der Fußspitze bezüglich des BBKS.

$${}_4^0T = {}_1^0T \cdot {}_2^1T \cdot {}_3^2T \cdot {}_4^3T \quad (3.4)$$

Durch Substitution der Teiltransformationen $(_1^0T, \frac{1}{2}^1T, \frac{2}{3}^2T, \frac{3}{4}^3T)$ mit der Berechnungsvorschrift aus Gleichung 3.3 ergibt sich Gleichung 3.5.

Durch die Anordnung der Gelenkkachsen der Beine ergeben sich Vereinfachungen daher, dass Gelenkkachsen parallel sind oder senkrecht aufeinander stehen. Folglich werden manche *D-H Parameter* zu null.

$$\begin{aligned} {}_4^0T &= Rot(x_0, \theta) \cdot Trans(0, A, 0) \cdot Rot(y_1, -90^\circ) \cdot \\ &\quad Rot(x_1, \alpha) \cdot Trans(0, B, 0) \cdot Rot(y_2, +90^\circ) \cdot \\ &\quad Rot(x_2, \beta) \cdot Trans(0, C, 0) \cdot \\ &\quad Rot(x_2, \gamma) \cdot Trans(0, D, 0) \end{aligned} \quad (3.5)$$

Durch Ausmultiplizieren der einzelnen homogenen Transformationsmatrizen erhält man die homogene Transformationsmatrix (3.6) für das gesamte Bein.

$${}_4^0T = {}_1^0T \cdot {}_2^1T \cdot {}_3^2T \cdot {}_4^3T = \begin{pmatrix} {}_4^0R & {}^0\vec{r} \\ 0 & 1 \end{pmatrix} \quad (3.6)$$

Zur Bestimmung des Fußendpunkts ist nur der Vektor ${}^0\vec{r}$ aus Gleichung 3.6 von Bedeutung. Dieser beschreibt die Koordinaten zum Fußendpunkt entsprechend der Abbildung 3.7.

$$\begin{aligned}
r_x &= -\sin(\alpha) \cdot (D \cdot \cos(\beta) \cdot \cos(\gamma) - \\
&\quad D \cdot \sin(\beta) \cdot \sin(\gamma) + \cos(\beta) \cdot C + B) \\
r_y &= ((\cos(\theta) \cdot \cos(\alpha) \cdot \cos(\beta) - \sin(\theta) \cdot \sin(\beta)) \cdot \cos(\gamma) + \\
&\quad (-\cos(\theta) \cdot \cos(\alpha) \cdot \sin(\beta) - \sin(\theta) \cdot \cos(\beta)) \cdot \sin(\gamma)) \cdot D + \\
&\quad (\cos(\theta) \cdot \cos(\alpha) \cdot \cos(\beta) - \sin(\theta) \cdot \sin(\beta)) \cdot C + \\
&\quad \cos(\theta) \cdot \cos(\alpha) \cdot B + \cos(\theta) \cdot A \\
r_z &= ((\sin(\theta) \cdot \cos(\alpha) \cdot \cos(\beta) + \cos(\theta) \cdot \sin(\beta)) \cdot \cos(\gamma) + \\
&\quad (-\sin(\theta) \cdot \cos(\alpha) \cdot \sin(\beta) + \cos(\theta) \cdot \cos(\beta)) \cdot \sin(\gamma)) \cdot D + \\
&\quad (\sin(\theta) \cdot \cos(\alpha) \cdot \cos(\beta) + \cos(\theta) \cdot \sin(\beta)) \cdot C + \\
&\quad \sin(\theta) \cdot \cos(\alpha) \cdot B + \sin(\theta) \cdot A
\end{aligned} \tag{3.7}$$

Somit beschreibt der Ortsvektor (r_x, r_y, r_z) im BBKS den Fußendpunkt. Die Parameter α, β, γ sind die Freiheitsgrade der Segmente, welche die variablen Eingaben (q_n) darstellen. A, B, C, D sind konstante Größen (a_n), die die Länge der einzelnen Segmente angeben. Der Winkel θ gibt die konstruktionsbedingte Neigung des α -Gelenks aus der Waagerechten nach unten an. (Vgl. Abbildung 3.7 und 3.9).

3.4 Krafttransformationen

Die Kraftsensoren der Beine liefern Daten über wirkende Kräfte. Der Kraftsensor registriert drei senkrecht aufeinander stehende Kraftkomponenten. Diese Kraftmesswerte werden auf den Fußendpunkt bezogen. Somit ist der Ort und die Richtung des Kraftvektors abhängig von der Positionierung des Kraftsensors im Raum. Eine sinnvolle Interpretation der Kraftkomponenten ist also nur möglich, wenn diese im Bezug auf ein System betrachtet werden. Daher erfolgt die Interpretation der Sensordaten im Bezug auf das aktuelle Bein.

Die Kraftsensoren der Laufmaschine Lauron IV/b befinden sich im D-Segment (Abbildung 3.9) der Beine. Es handelt sich hier um Drei-Komponenten-Kraftsensoren (siehe Kapitel 4.1.2.3). Die erfassten Kraftkomponenten F_1, F_2 und F_3 stehen orthogonal zueinander und sind in Bezug auf die Position des Kraftsensors im Raum ausgerichtet, also entsprechend der Ausrichtung des D-Segments. Da die Kräfte in Bezug auf das Fußpunktkoordinatensystem (FPKS), somit am Fuß angreifend angenommen werden, kann die räumliche Orientierung des Kraftangriffspunkts mit Hilfe der direkten Kinematik nach Denavit und Hartenberg (Kapitel 3.3.1) bestimmt werden. Der Kraftwert kann parallel verschoben werden. Um die Kraftwerte bezüglich der Orientierung im *Beinbezugskoordinatensystem* darzustellen muss eine Drehung ausgeführt werden.

Der Vektor ${}^0\vec{r}$ aus Gleichung (3.6) beschreibt die Koordinaten zum Fußendpunkt respektive den Kraftangriffspunkt (Abbildung 3.7), 0R beschreibt den Übergang vom *Beinbezugskoordinatensystems* in das *Fußpunktkoordinatensys-*

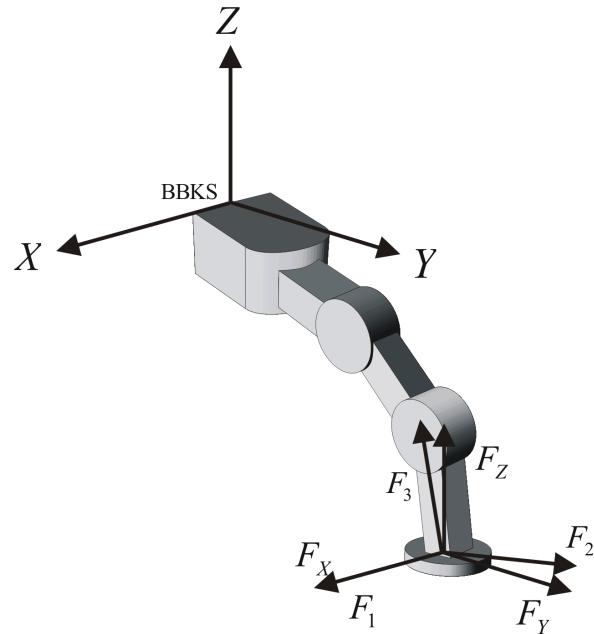


Abbildung 3.10: Fußpunktkoordinatensystem

tem. Da aber der Übergang vom *Fußpunktkoordinatensystem* in das *Beinbezugskoordinatensystem* erforderlich ist, muss die Rotation ${}_0^4R$ bestimmt werden, was aufgrund struktureller Eigenschaften der Rotationsmatrix durch folgende Beziehung möglich ist:

$${}_0^4R = {}_4^0R^{-1} = {}_4^0R^T \quad (3.8)$$

Durch die Rotation ${}_0^4R$ werden die Komponenten der wirkenden Kräfte so rotiert, dass sie sich parallel zu denen des *Beinbezugskoordinatensystems* befinden. Gleichung (3.10) zeigt die Transformation der Signale der Kraftsensorkomponenten in die parallelen Komponenten bezüglich des *Beinbezugskoordinatensystems*. Der Übersichtlichkeit wegen werden Abkürzungen eingeführt, welche der Tabelle 3.2 zu entnehmen sind.

Abkürzung	Ausführliche Schreibweise
$s\alpha$	$\sin(\alpha)$
$c\alpha$	$\cos(\alpha)$
$s\beta$	$\sin(\beta)$
$c\beta$	$\cos(\beta)$
$s\gamma$	$\sin(\gamma)$
$c\gamma$	$\cos(\gamma)$
$s\theta$	$\sin(\theta)$
$c\theta$	$\cos(\theta)$

Tabelle 3.2: Abkürzende Schreibweisen

$$\begin{aligned}
 R_1 &= c\alpha \\
 R_2 &= c\theta \cdot s\alpha \\
 R_3 &= s\delta \cdot s\alpha \\
 R_4 &= -s\alpha \cdot (c\beta \cdot c\gamma - s\beta \cdot s\gamma) \\
 R_5 &= (c\theta \cdot c\alpha \cdot c\beta - s\theta \cdot s\beta) \cdot c\gamma + (-c\theta \cdot c\alpha \cdot s\beta - s\theta \cdot c\beta) \cdot s\gamma \\
 R_6 &= (s\theta \cdot c\alpha \cdot c\beta + c\theta \cdot s\beta) \cdot c\gamma + (-s\theta \cdot c\alpha \cdot s\beta + c\theta \cdot c\beta) \cdot s\gamma \\
 R_7 &= s\alpha \cdot (c\beta \cdot s\gamma + s\beta \cdot c\gamma) \\
 R_8 &= -(c\theta \cdot c\alpha \cdot c\beta - s\theta \cdot s\beta) \cdot s\gamma + (-c\theta \cdot c\alpha \cdot s\beta - s\theta \cdot c\beta) \cdot c\gamma \\
 R_9 &= -(s\theta \cdot c\alpha \cdot c\beta + c\theta \cdot s\beta) \cdot s\gamma + (-s\theta \cdot c\alpha \cdot s\beta + c\theta \cdot c\beta) \cdot c\gamma
 \end{aligned} \tag{3.9}$$

Die endgültige Übertragungsmatrix erhält man, indem die Variablen $R_1 \dots R_9$ aus Gleichung (3.10) mit den trigonometrischen Berechnungsvorschriften aus (3.9) substituiert werden.

$$\begin{pmatrix} F_x^{BBKS} \\ F_y^{BBKS} \\ F_z^{BBKS} \end{pmatrix} = \begin{pmatrix} R_1 & R_2 & R_3 \\ R_4 & R_5 & R_6 \\ R_7 & R_8 & R_9 \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \\ F_3 \end{pmatrix} \tag{3.10}$$

Der Kraftvektor aus Gleichung (3.10) liegt nun bezüglich Orientierung des *Beinbezugskoordinatensystems* vor. Um die Orientierung bezüglich des *Beinkoordinatensystems* respektive des *Symmetriekoordinatensystems* zu erhalten, müssen die Fußkraftvektoren der an den Beinen angreifenden Kräfte aus Gleichung (3.10) um die z-Achse rotiert werden. Gleichung (3.11) zeigt die Rotation der z-Achse um den Winkel λ .

$$\begin{pmatrix} F_x^{BKS} \\ F_y^{BKS} \\ F_z^{BKS} \end{pmatrix} = \begin{pmatrix} \cos(\lambda) & -\sin(\lambda) & 0 & F_x^{BBKS} \\ \sin(\lambda) & \cos(\lambda) & 0 & F_y^{BBKS} \\ 0 & 0 & 1 & F_z^{BBKS} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.11}$$

Die Winkel λ_i für die Koordinatenübergänge von *Beinbezugskoordinatensystem*

in die *Beinkoordinatensysteme* (BKS_i) sind Kapitel 3.2.2 zu entnehmen.

3.5 Kraftgeführte Bewegungen

Bei Laufrobotern entsteht durch die Art der Fortbewegung ein hohes Maß an Interaktion mit dem Untergrund. Ferner werden durch technische Operationen aufgabenbezogene Interaktionen mit der Umgebung hergestellt.

Unter Einsatz binärer taktiler Kraftsensoren lassen sich Bewegungscoordinierungen ableiten [51]. Durch die Auswertung der Kraftsensorinformationen lässt sich beispielsweise ermitteln, welches Bein Bodenkontakt hat bzw. kann das Auftreffen auf ein Hindernis festgestellt werden. Diese Daten haben Einfluss auf die Bewegungsplanung und führen somit zu einer Neuplanung der Bewegung eines Beines oder des ganzen Systems.

Ferner lassen sich unter Einsatz von Kraftsensoren kontinuierliche Bewegungen beeinflussen. Diese Eigenschaft wird im folgenden genauer beleuchtet.

3.5.1 Prinzipien der Kraftregelung

Die Beeinflussung von Bewegung bzw. Bewegungsführung beruht auf der Grundlage, dass ein Zusammenhang zwischen Position und Kraft hergestellt wird [51]. Dieses Prinzip kann dazu genutzt werden, bestimmte Bewegungen während der Ausführung zu verändern oder Bewegungskomponenten in Abhängigkeit von wirkenden Kräften durchzuführen. Das dieser Arbeit zu Grunde liegende Ziel besteht darin, eine Position oder eine Geschwindigkeit unter Einfluss der Umgebung so zu beeinflussen, dass bestimmte Kraftwirkungen entstehen bzw. aus Kraftwirkungen bestimmte Geschwindigkeiten oder Positionen resultieren. Zu diesem Zweck werden Geschwindigkeitsregelung und Kraftregelung miteinander verbunden. Die Regler *Active Compliance* und *Damping Control* [51][38] kombinieren die Größen Position und/oder Geschwindigkeit mit der Kraft und führen zum gewünschten Ergebnis.

Active Compliance

Die *Active Compliance* wird auch als Nachgiebigkeitsregler bezeichnet. Sie erzeugt eine proportionale Positionsänderung entsprechend einer wirkenden Kraft. Die Gleichung (3.12) beschreibt diese Gegebenheit.

$$\Delta x = k \cdot (F_{soll} - F_{ist}) \quad (3.12)$$

Δx enthält die Positionsänderung, k ist der Nachgiebigkeitsfaktor und F_{soll}, F_{ist} entspricht der Sollkraft bzw. der aktuell wirkenden Kraft genannt Istkraft. Diese Gleichungsbeziehung (3.12) ist dem Federkraftgesetz gleichzusetzen und beschreibt somit das Verhalten einer mechanischen Feder. Wird der Nachgiebigkeitsfaktor k größer so hat dies eine größere Nachgiebigkeit zur Folge. Für den mehrdimensionalen Fall wird der Faktor k zu einer Matrix. Gleichung (3.13) zeigt den Fall der aktiven Nachgiebigkeit für drei Dimensionen.

$$\begin{pmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{pmatrix} = \begin{pmatrix} k_X & 0 & 0 \\ 0 & k_Y & 0 \\ 0 & 0 & k_Z \end{pmatrix} \cdot \left(\begin{pmatrix} F_{soll,X} \\ Y_{soll,Y} \\ Z_{soll,Z} \end{pmatrix} - \begin{pmatrix} F_{ist,X} \\ F_{ist,Y} \\ F_{ist,Z} \end{pmatrix} \right) \quad (3.13)$$

Damping Control

Damping Control wird auch als Dämpfungsregler bezeichnet und beschreibt die Erzeugung einer Geschwindigkeit in Abhängigkeit von einer wirkenden Kraft.

$$\Delta \dot{x} = k \cdot (F_{soll} - F_{ist}) \quad (3.14)$$

Die in der Beziehung 3.14 enthaltenen Variablen sind analog zu denen aus Gleichung (3.12) zu verstehen mit dem Unterschied, dass das Resultat keine Positionsänderung sondern eine Geschwindigkeit ist. Der Faktor k wird bei diesem Regler als Dämpfungsfaktor bezeichnet. Gleichung (3.14) beschreibt das Verhalten eines Dämpfers. Höhere wirkende Kraftdifferenzen führen zu größeren Geschwindigkeiten. Die dadurch erzeugte Bewegung wirkt in Richtung der wirkenden Kraft: Die Bewegung hält für die Dauer der wirkenden Kraft an. Dieses Verhalten entspricht einer Ausweichbewegung. Wird der Dämpfungsfaktor k grösser so wird die Dämpfung geringer. Dieser Regler wirkt auf die Position integrierend. Die aktive Dämpfung lässt sich ebenfalls für mehrere Dimensionen formulieren. Die Gleichung (3.15) zeigt den Fall für drei Dimensionen.

$$\begin{pmatrix} \Delta \dot{X} \\ \Delta \dot{Y} \\ \Delta \dot{Z} \end{pmatrix} = \begin{pmatrix} k_X & 0 & 0 \\ 0 & k_Y & 0 \\ 0 & 0 & k_Z \end{pmatrix} \cdot \left(\begin{pmatrix} F_{soll,X} \\ Y_{soll,Y} \\ Z_{soll,Z} \end{pmatrix} - \begin{pmatrix} F_{ist,X} \\ F_{ist,Y} \\ F_{ist,Z} \end{pmatrix} \right) \quad (3.15)$$

3.6 Kraftverteilungen

Während des Laufvorgangs wirkt die Gewichtskraft des Roboters auf den Untergrund. Diese Gewichtskraft verteilt sich auf die einzelnen Stützfüße. Zur Bestimmung der Kraftverteilung auf die in der Stützphase befindlichen Beine können statische Gleichgewichtsbedingungen verwendet werden [38].

3.6.1 Statische Gleichgewichtsbedingungen

Wenn Kräftepaare wirken entstehen Drehwinkel die *Momente* verursachen. Hierbei ist bedeutend, wo und mit welchem Hebel ein Moment angreift. Allgemein gilt

$$\vec{M} = \vec{r} \times \vec{F}. \quad (3.16)$$

Wirken auf einen Körper mehrere Momente und Kräfte, so führt dies zur Entstehung von resultierenden Kräften und Momenten

$$\vec{M} = \sum \vec{M}_i \quad (3.17)$$

$$\vec{F} = \sum \vec{F}_i. \quad (3.18)$$

Man spricht von einem Gleichgewicht der Kräfte und Momente falls sich alle Kräfte und Momente gegenseitig aufheben und es zu keiner Bewegung des Körpers kommt. Folglich lauten die statischen Gleichgewichtsbedingungen

$$\sum \vec{M}_i = 0 \text{ mit } \sum M_{i,X} = 0, \sum M_{i,Y} = 0 \text{ und } \sum M_{i,Z} = 0 \quad (3.19)$$

$$\sum \vec{F}_i = 0 \text{ mit } \sum F_{i,X} = 0, \sum F_{i,Y} = 0 \text{ und } \sum F_{i,Z} = 0. \quad (3.20)$$

Die Anzahl der skalaren Gleichgewichtsbedingungen entsprechen den sechs möglichen Freiheitsgraden. Im folgendem wird der Fall angenommen, indem alle Kräfte, d. h. Gewichtskraft und Stützkräfte, in die selbe Richtung wirken. $F_{i,X} = 0$ und $F_{i,Y} = 0$ führt zu einer Reduzierung der Anzahl an Gleichungen. Folglich gilt

$$\sum F_{i,Z} = 0, \sum M_{i,X} = 0 \text{ und } \sum M_{i,Y} = 0. \quad (3.21)$$

Diese Gleichgewichtsbedingungen bilden ein Gleichungssystem, das sich in einer Matrix darstellen lässt

$$\begin{pmatrix} 1 \\ \vec{X}^{(i)} \\ \vec{Y}^{(i)} \end{pmatrix} \cdot \vec{F}^{(i)} = \begin{pmatrix} 1 \\ X^{(R)} \\ Y^{(R)} \end{pmatrix} \cdot F_g. \quad (3.22)$$

$\vec{X}^{(i)}$ und $\vec{Y}^{(i)}$ sind hierbei die Koordinaten der am jeweiligen Bein i angreifenden Kraft $\vec{F}^{(i)}$. Folglich handelt es sich hier um die Koordinaten der Beine, die über die direkte Kinematik nach Denavit und Hartenberg (vgl. Kapitel 3.3.1) berechnet werden. Da diese Koordinaten im *Beinbezugskoordinatensystem* vorliegen, müssen sie zuvor in das *Symmetriekoordinatensystem* überführt (vgl. Kapitel 3.2.2) werden. $X^{(R)}$ und $Y^{(R)}$ beschreiben die Koordinaten des Masseschwerpunktes der Laufmaschine. Ausgehend von diesem Koordinatenpunkt wirkt die Gewichtskraft F_g . Die Variable F_g ist somit Platzhalter für das Gesamtgewicht der Laufmaschine inkl. Beine. Die Gesamtkraft verteilt sich in einem bestimmten Verhältnis auf die jeweiligen, mit dem Boden in Kontakt stehenden Beine $\vec{F}^{(i)}$.

3.6.2 Kraftverteilung für drei Stützfüße

Werden drei Beine als Stützbeine verwendet, so führt dies zu drei Gleichungen mit drei Unbekannten. Die Kraftverteilung $\vec{F}^{(i)}$ kann daher eindeutig durch die

Lösung des Gleichungssystems bestimmt werden. Das mechanische System ist bei diesem Fall bestimmt.

Beim Dreifußgang sind idealerweise immer drei Beine mit dem Untergrund in Berührung. Um den Wechsel der Stützfüße von den Beinen B^0, B^3, B^4 zu den Beinen B^1, B^2, B^5 (siehe Kapitel 3.2.1) zu berücksichtigen, wird eine Selektionsmatrix eingeführt, bei der nur die Hauptdiagonalelemente besetzt sind. Um ein Element eines Vektors $\vec{X}^{(i)}, \vec{Y}^{(i)}$ zur Berechnung einzubeziehen, wird das entsprechende Element der Hauptdiagonalen der Selektionsmatrix mit 1 besetzt ansonsten mit 0. Somit müssen beim Dreifußgang immer drei Elemente entsprechend der Stützfüße der Hauptdiagonalen der Selektionsmatrix mit 1 besetzt sein, folglich die anderen drei mit 0. Gleichung (3.23) zeigt den Fall für die Stützfüße B^0, B^3, B^4 , die Gleichung (3.24) veranschaulicht den Beinkontakt mit dem Untergrund für die Stützfüße B^1, B^2, B^5 .

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ \vec{X}^{(0)} & \vec{X}^{(1)} & \vec{X}^{(2)} & \vec{X}^{(3)} & \vec{X}^{(4)} & \vec{X}^{(5)} \\ \vec{Y}^{(0)} & \vec{Y}^{(1)} & \vec{Y}^{(2)} & \vec{Y}^{(3)} & \vec{Y}^{(4)} & \vec{Y}^{(5)} \end{pmatrix} \cdot \overbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}}^{\text{Selektionsmatrix}} \cdot \begin{pmatrix} \vec{F}^{(0)} \\ \vec{F}^{(1)} \\ \vec{F}^{(2)} \\ \vec{F}^{(3)} \\ \vec{F}^{(4)} \\ \vec{F}^{(5)} \end{pmatrix} = \begin{pmatrix} 1 \\ X^{(R)} \\ Y^{(R)} \end{pmatrix} \cdot F_g \quad (3.23)$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ \vec{X}^{(0)} & \vec{X}^{(1)} & \vec{X}^{(2)} & \vec{X}^{(3)} & \vec{X}^{(4)} & \vec{X}^{(5)} \\ \vec{Y}^{(0)} & \vec{Y}^{(1)} & \vec{Y}^{(2)} & \vec{Y}^{(3)} & \vec{Y}^{(4)} & \vec{Y}^{(5)} \end{pmatrix} \cdot \overbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}^{\text{Selektionsmatrix}} \cdot \begin{pmatrix} \vec{F}^{(0)} \\ \vec{F}^{(1)} \\ \vec{F}^{(2)} \\ \vec{F}^{(3)} \\ \vec{F}^{(4)} \\ \vec{F}^{(5)} \end{pmatrix} = \begin{pmatrix} 1 \\ X^{(0)} \\ Y^{(0)} \end{pmatrix} \cdot F_g \quad (3.24)$$

Um die Gleichungssysteme (3.23) und (3.24) lösen zu können bzw. die Stützkräfte der Beine zu bestimmen, müssen die verwendeten Stützfüße, die Koordinaten der Fußpunkte der Stützfüße, die Gewichtskraft und die Koordinaten

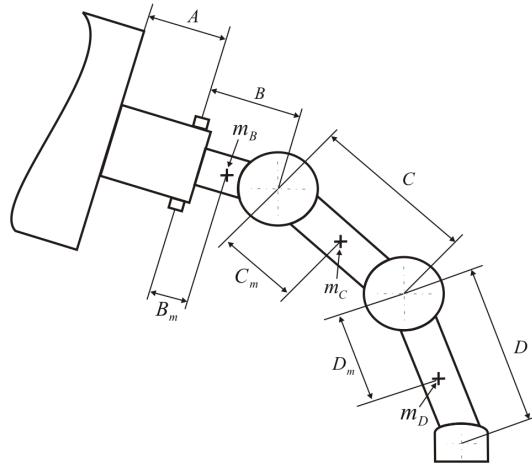


Abbildung 3.11: Geometrische Parameter eines Beines

des Masseschwerpunkts bekannt sein. Der Masseschwerpunkt der Laufmaschine ist berechenbar, wenn die Verteilung der Masse über den gesamten Roboter bekannt ist. Gleichung (3.25) bestimmt die Verteilung der Masse.

$$\vec{V}^{(R)} = \frac{\int \vec{V} m(\vec{V}) d\vec{V}}{\int m(\vec{V}) d\vec{V}} \quad (3.25)$$

$\vec{V}^{(R)}$ beschreibt bei dieser Gleichung den Ortsvektor des Masseschwerpunktes und \vec{V} ist der zugehörige Vektor zum Massepunkt m der Laufmaschine. Da es fern der Realität ist den genauen Masseschwerpunkt bei der Laufmaschine zu bestimmen, wird ein Modell eingeführt, welches jeweils starre Körper zu einem Massepunkt mit den dazu entsprechenden Koordinaten zusammenfasst. Somit wird die Bestimmung des Massenschwerpunktes auf wenige zu betrachtende Massepunkte reduziert. Gleichung (3.26) zeigt diese Vereinfachung.

$$\vec{V}^{(R)} = \frac{\sum \vec{V}_i m_i}{\sum m_i} \quad (3.26)$$

Um die Massepunkte zu bestimmen, wird der Roboterkörper und die Beinsegmente als Massepunkte zusammengefasst. Dies ist möglich, da sie sich als starre Körper auffassen lassen. Die Beine werden wie bei der direkten Kinematik nach Denavit und Hartenberg (siehe Kapitel 3.3.1) in die Segmente A, B, C und D unterteilt (vgl. Abbildung 3.11). Da das Segement A mit dem Körper fest verbunden ist, wird es dem Körper zugeteilt. Für die Segmente B, C und D müssen die Massepunkte m_B , m_C und m_D bestimmt werden. Die Koordinaten zu den Massepunkten werden zum vorhergehenden Gelenk mit B_m , C_m und D_m bestimmt. Abbildung 3.11 zeigt diesen Zusammenhang. Die Bestimmung der Koordinaten des Beinschwerpunkts erfolgt analog zur direkten kinematischen Transformation nach Denavit und Hartenberg (siehe Kapitel 3.3.1) berechnet. Da wir bei der Berechnung des Masseschwerpunkts davon ausgehen, dass die

Laufmaschine senkrecht zur Ebene des Gravitationsvektors steht, folglich parallel zu dieser Ebene angenommen wird, müssen nur die x und y Koordinaten betrachtet werden. Steht die Laufmaschine bezogen auf den Gravitationsvektor nicht parallel zum Untergrund und somit der Gravitationsvektor beliebig gerichtet, müssen alle Koordinaten (x, y, z) betrachtet werden. Bei vorliegender Arbeit werden allerdings nur Transformationen der Koordinaten der Massepunkte im ebenen *Beinbezugskoordinatensystem (BBKS)* betrachtet. Die Gleichungen zur Bestimmung der Massepunkte der einzelnen Beinsegmente im *Beinbezugskoordinatensystem (BBKS)* sind wie folgt:

Der Übersichtlichkeit wegen werden die Abkürzungen aus Tabelle 3.2 verwendet.

$${}^B\vec{V}^{(m_B)} = \begin{pmatrix} {}^BX^{(m_B)} \\ {}^BY^{(m_B)} \end{pmatrix} = \begin{pmatrix} -s\alpha \cdot B_m \\ c\theta \cdot (c\alpha * B_m + A) \end{pmatrix} \quad (3.27)$$

$$\begin{aligned} {}^B\vec{V}^{(m_C)} &= \begin{pmatrix} {}^BX^{(m_C)} \\ {}^BY^{(m_C)} \end{pmatrix} = \\ &\begin{pmatrix} -s\alpha \cdot (c\beta \cdot C_m + B) \\ (c\theta \cdot (c\alpha \cdot c\beta - s\theta \cdot s\beta) \cdot C_m + c\theta \cdot c\alpha \cdot B + c\theta \cdot A) \end{pmatrix} \end{aligned} \quad (3.28)$$

$${}^B\vec{V}^{(m_D)} = \begin{pmatrix} {}^BX^{(m_D)} \\ {}^BY^{(m_D)} \end{pmatrix} \text{ mit}$$

$$\begin{aligned} {}^BX^{(m_D)} &= -s\alpha \cdot (D_m \cdot c\beta \cdot c\gamma - D_m \cdot s\beta \cdot s\gamma + c\beta \cdot C + B) \\ &\quad (3.29) \end{aligned}$$

$$\begin{aligned} {}^BY^{(m_D)} &= ((c\theta \cdot c\alpha \cdot c\beta - s\theta \cdot s\beta) \cdot c\gamma + (-c\theta \cdot c\alpha \cdot s\beta - s\theta \cdot c\beta) \cdot s\gamma) \cdot D_m + (c\theta \cdot c\alpha \cdot c\beta - s\theta \cdot s\beta) \cdot C + c\theta \cdot c\alpha \cdot B + c\theta \cdot A \end{aligned}$$

Die Massepunkte (3.27), (3.28), (3.29) für die einzelnen Beinsegmente sind nun bezüglich des *Beinbezugskoordinatensystems (BBKS)* bekannt. Um den Massenschwerpunkt der Laufmaschine zu bestimmen müssen diese in das Symmetriekoordinatensystem überführt werden. Für diese Transformation sind zwei Schritte erforderlich:

1. Überführung der Koordinaten in das *Beinkoordinatensystem (BKS)* (vgl. Kapitel 3.2.2).
2. Überführung der Koordinaten im BKS (aus 1.) in das *Symmetriekoordinatensystem (SKS)*.

Für die erste Überführung der Koordinaten ist eine Rotation um den Seitenwinkel ϕ_i notwendig für die zweite wird eine Transformation um den Abstand

r_i vom Koordinatenursprung des *Symmetriekoordinatensystems* zum Beinbefestigungspunkt des jeweiligen Beins durchgeführt. Der Abstand r^K beschreibt die Distanz des Massepunkts des Körpers der Laufmaschine zum *Symmetriekoordinatensystem*. Abbildung 3.12 zeigt die verwendeten ebenen Koordinatensysteme.

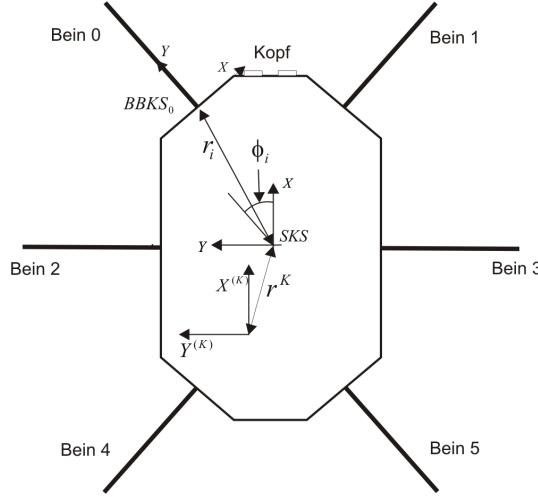


Abbildung 3.12: Ebene Koordinatensysteme zur Masseschwerpunktberechnung

Die folgenden Gleichungen zeigen die Überführung:

$${}^R \vec{V}^{(m)} = {}_B^R T \cdot {}^B \vec{V}^{(m)} = Rot(\phi_i) \cdot Trans(r_i) \cdot {}^B \vec{V}^{(m)} \quad (3.30)$$

$${}^R \vec{V}^{(m)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & r_i \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} {}^B X^{(m)} \\ {}^B Y^{(m)} \\ 1 \end{pmatrix} \quad (3.31)$$

$${}^R \vec{V}^{(m)} = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 \\ \sin(\phi) & \cos(\phi) & r_i \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} {}^B X^{(m)} \\ {}^B Y^{(m)} \\ 1 \end{pmatrix} \quad (3.32)$$

Somit ergeben sich alle Masseschwerpunkte der Beine und den Roboterkörper im *Symmetriekoordinatensystem*. Der Masseschwerpunkt des gesamten Roboters lässt sich nun auf Grundlage der Gleichung (3.26) bestimmen:

$$X^{(R)} = \frac{m_B \sum^R X^{(i,m_B)} + m_C \sum^R X^{(i,m_C)} + m_D \sum^R X^{(i,m_D)} + X^{(K)} \cdot m_K}{6 \cdot (m_B + m_C + m_D) + m_K} \quad (3.33)$$

$$Y^{(R)} = \frac{m_B \sum^R Y^{(i, m_B)} + m_C \sum^R Y^{(i, m_C)} + m_D \sum^R Y^{(i, m_D)} + Y^{(K)} \cdot m_K}{6 \cdot (m_B + m_C + m_D) + m_K} \quad (3.34)$$

Durch Auflösung der Gleichung (3.22) lässt sich die Kraftverteilung für die jeweils drei Stützfüße folgendermaßen berechnen:

$$\begin{pmatrix} \vec{F}^{(0)} \\ \vec{F}^{(3)} \\ \vec{F}^{(4)} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ X^{(0)} & X^{(3)} & X^{(4)} \\ Y^{(0)} & Y^{(3)} & Y^{(4)} \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 \\ X^{(R)} \\ Y^{(R)} \end{pmatrix} \cdot F_g \quad (3.35)$$

$$\begin{pmatrix} \vec{F}^{(1)} \\ \vec{F}^{(2)} \\ \vec{F}^{(5)} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ X^{(1)} & X^{(2)} & X^{(5)} \\ Y^{(1)} & Y^{(2)} & Y^{(5)} \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 \\ X^{(R)} \\ Y^{(R)} \end{pmatrix} \cdot F_g \quad (3.36)$$

Mit diesen beiden Gleichungen lassen sich die Kraftverteilung über die Fußspitzen bestimmen. Die nun berechenbaren zu erwartenden Kraftverteilungen auf die Stützfüße werden verwendet, um die vertikale „Vorspannung“ für die aktive Nachgiebigkeit (siehe Kapitel 3.5.1) einzustellen. Dies wird erreicht, indem die vertikalen Sollkräfte für die z -Komponente in Bezug auf das *Symmetriekoordinatensystem* mit den berechneten Stützkräften aus Gleichung (3.35) und (3.36) substituiert werden:

$$\vec{F}_{soll}^{(i)} = \begin{pmatrix} 0 \\ 0 \\ \vec{F}^{(i)} \end{pmatrix} \quad (3.37)$$

3.6.3 Kraftverteilung für mehr als drei Stützfüße

Für den Fall, dass mehr als drei Stützfüße verwendet werden, kann die Kraftverteilung nicht direkt auf dem erläuterten Weg bestimmt werden, da das statische System unbestimmt ist. Um die Kraftverteilung dennoch zu bestimmen, sind zusätzliche Bestimmungsgleichungen notwendig, um eine eindeutige Lösung aller Kräfte zu erhalten. Diese Bedingungen können dann beispielsweise Kriterien für eine möglichst gleichmäßige Kraftverteilung enthalten. Dieses Ziel wird über eine Minimierung der Stützkräfte erreicht. Folglich hat dies den Effekt dass der Energieverbrauch minimiert wird. Folgende Gleichung stellt dies mathematisch dar:

$$\left| \vec{F} \right|^2 = \sum_i \left| \vec{F}^{(i)} \right|^2 \rightarrow \min \quad (3.38)$$

Diese Bedingung entspricht gleichzeitig der besten Näherungslösung für das Gleichungssystem (3.22), (3.23) und (3.24). Zur Lösung des Gleichungssystems wird die pseudoinverse Matrix M^+ eingeführt [32].

$$\vec{F}^{(i)} = \underbrace{\begin{pmatrix} 1 \\ \vec{X}^{(i)} \\ \vec{Y}^{(i)} \end{pmatrix}}^+ \cdot \begin{pmatrix} 1 \\ \vec{X}^{(R)} \\ \vec{Y}^{(R)} \end{pmatrix} \cdot F_g \quad (3.39)$$

$$\begin{pmatrix} 1 \\ \vec{X}^{(i)} \\ \vec{Y}^{(i)} \end{pmatrix}^T \cdot \left(\begin{pmatrix} 1 \\ \vec{X}^{(i)} \\ \vec{Y}^{(i)} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \vec{X}^{(i)} \\ \vec{Y}^{(i)} \end{pmatrix}^T \right)^{-1}$$

Mit den beschriebenen Methoden ist es möglich, die Kraftverteilung auf die Stützfüße zu bestimmen. Diese Stützkräfte können dann für die aktive Nachgiebigkeit verwendet werden.

3.7 Regelungstechnische Grundlagen

Das vorliegende Kapitel stellt eine Einführung in die Regelungstechnik dar. Dabei sollen die wichtigsten Begriffe und mathematischen Grundlagen erläutert werden.

3.7.1 Laplace–Transformation

Die Laplace-Transformation ist eine Integraltransformation, die sich besonders zur Lösung von linearen Differentialgleichungen mit konstanten Koeffizienten eignet [19]. Die Eingangsfunktion wird vom Zeit- in den Frequenzbereich überführt. Die Transformation erfüllt mehrere Bedingungen:

- Sie wandelt lineare zeitinvariante Differential- und Integralgleichungen in eine algebraische Gleichung mit der neuen Variablen s um.
- Man erhält sofort die Gesamtlösung der Differential- bzw. Integralgleichung im Bildbereich ohne Anwendung spezieller Lösungsansätze.
- Die Lösung der Differential bzw. Integralgleichung ist eine gebrochen rationale Funktion s, deren Rücktransformation in den Zeitbereich mittels Partialbruchzerlegung und Korrespondenztabellen ohne Schwierigkeiten durchgeführt werden kann.
- Die Analyse und die Synthese rückgekoppelter Systeme werden im Bildbereich gegenüber dem Zeitbereich wesentlich erleichtert.

Der Zeit (t)-Bereich wird auch als Originalbereich oder Oberbereich und der Bildbereich als Unterbereich oder Frequenz (s) bezeichnet. Für beide Transformationsrichtungen gibt es umfangreiche Tabellen, in denen die wichtigsten Paare ($f(t), F(s)$) gelistet sind [19].

Das folgende Integral definiert die Laplace–Transformation:

$$L\{f(t)\} = F(s) = \int_0^{\infty} f(t) \cdot e^{-st} dt \quad (3.40)$$

mit $s := \sigma + j\omega$. s wird als komplexe Bildvariable oder komplexe Kreisfrequenz bezeichnet.

Die Laplace–Transformation besitzt zahlreiche Eigenschaften, von denen in der Regelungstechnik insbesondere der Differentiations–, der Integrations– und der Faltungssatz benutzt werden. Nach dem Differentiationssatz entspricht eine Differenzierung im Zeitbereich nach der Transformation einer Multiplikation mit s :

$$L\left\{\frac{d^n f(t)}{dt^n}\right\} = s^n \cdot F(s) \quad (3.41)$$

Eine Integration im Zeitbereich entspricht nach der Transformation einer Division durch s :

$$L\left\{\int f(t) dt\right\} = \frac{1}{s} \cdot F(s) \quad (3.42)$$

Der Faltungssatz besagt, dass die Faltungsoperation im Zeitbereich in eine Multiplikation der Laplace–Transformierten im Frequenzbereich übergeht.

$$y(t) = g(t) \cdot u(t) \quad (3.43)$$

$$Y(s) = G(s) \cdot U(s) \quad (3.44)$$

3.7.2 Grundbegriffe

Im Allgemeinen werden Regelungen für technische Systeme eingesetzt, wenn diese zeitveränderliche Systemgrößen besitzen, die sich auf eine bestimmte Art und Weise verhalten sollen. In der Praxis tritt häufig der Fall auf, dass eine Größe einen konstanten Wert erreichen soll; die einzustellende Größe wird als *Regelgröße*, der Sollwert als *Führungsgröße* bezeichnet. Das Einwirken von äußeren Störeinflüssen (*Störgrößen*) soll dabei so kompensiert werden, dass die Regelgröße möglichst schnell und genau wieder ihren Sollwert einnimmt. Regelungen messen im Gegensatz zu Steuerungen fortlaufend die einzustellende Regelgröße und vergleichen sie mit der Führungsgröße. Die auftretende Differenz wird herangezogen, um daraus mit einem geeigneten Algorithmus (z.B. digitaler PID–Stellalgorithmus) eine Ausgangsgröße zu berechnen, mit deren Hilfe die Regelgröße wieder dem Sollwert angeglichen wird. Insgesamt ergibt sich ein geschlossener Wirkungsweg, den man als *Regelkreis* bezeichnet. Die Reaktion auf Störeinflüsse kann zwar erst bei einer daraufhin auftretenden Veränderung der Regelgröße erfolgen, jedoch werden alle einwirkenden Größen kompensiert und es ist nur ein relativ geringer Aufwand an Sensorik nötig. Außerdem sind nur Regelungen in der Lage, von sich aus instabile Systeme zu handhaben. Ein

System wird als *stabil* bezeichnet, wenn es in seiner Ruhelage verweilt, solange es nicht von außen angeregt wird bzw. nach Zurücknahme aller Einwirkungen in diese zurückkehrt. In Abbildung 3.13 sind die Elemente eines Regelkreises im sogenannten *Blockschaltbild* dargestellt. Dabei bezeichnet die *Regelstrecke* den Teil des Systems, in dem die physikalischen Größen beeinflusst werden sollen. Sie liegt grundsätzlich zwischen *Stellort* und *Messort*. Störgrößen, die die Regelgröße x beeinflussen, greifen an *Störorten* an. Die *Messeinrichtung* erfasst die Regelgröße am Messort und gibt sie als eine der beiden Eingangsgrößen in den *Vergleicher*, der die Differenz zur Führungsgröße bildet. Diese wird vom Regler als Reglerausgangsgröße y_R an das *Stellglied* geschickt. Hierbei handelt es sich in der Regel um einen Aktor, dessen Ausgangsgröße (*Stellgröße*) y am Stellort auf die Regelstrecke wirkt. Bei der Differenzbildung im Vergleicher sind zwei Vorgehensweisen möglich. Bei dem implementierten Regler der Laufmaschine Lauron IVb wird die *Gegenkopplung* benutzt, bei der der gemessene Istwert vom Sollwert abgezogen wird ($e(t) = w(t) - \tilde{y}(t)$). Dadurch wird den auftretenden Störungen entgegengewirkt, indem eine beispielsweise zu große Regelgröße in einer negativen Reglerausgangsgröße bzw. Stellgröße resultiert.

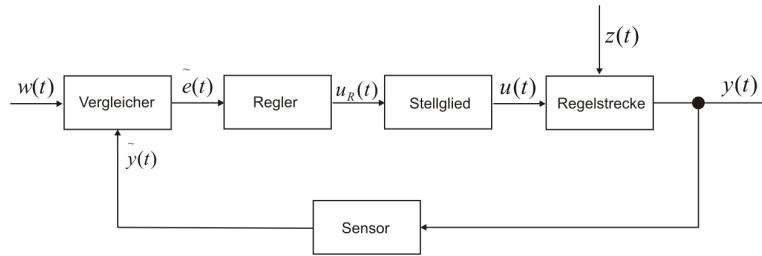


Abbildung 3.13: Grundstruktur eines Regelungskreises

Zusammenfassung der verwendeten Bezeichnungen:

- $w(t)$ = Führungsgröße = Sollwert
- $y(t)$ = Regelgröße = Istwert
- $\tilde{y}(t)$ = durch den Sensor gemessene Regelgröße y
- $u(t)$ = Stellgröße (Stellglied); $u_R(t)$ = Stellsignal (Regler)
- $\tilde{e}(t)$ = Regeldifferenz: $\tilde{e}(t) = w(t) - \tilde{y}(t)$
- $e(t)$ = bleibende Regelabweichung: $e(t) = w(t) - y(t)$
- $z(t)$ = Störgröße

3.7.3 Prinzip des kontinuierlichen PID-Regler

In der Industrie werden Standardregler eingesetzt, deren Übertragungsfunktionen proportionales (P-Glied) oder integrales (I-Glied) Übertragungsverhalten

zeigen, oder die eine Kombination aus beiden mit möglichem zusätzlichem D–Anteil (D–Glied) enthalten. Die allgemeinste Form dieses Standardreglers ist der PID–Regler [19][29][45].

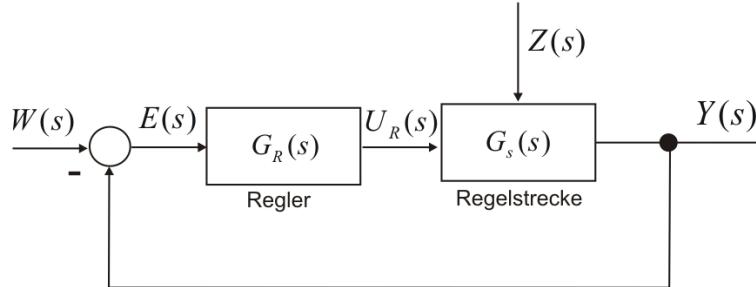


Abbildung 3.14: Blockschaltbild des geschlossenen Regelungssystems

Der geschlossene Regelungskreis muss bestimmte Bedingungen erfüllen:

1. Er soll stabil sein, d. h. im gesamten Arbeitsbereich stabil bleiben und eine ausreichende Stabilitätsreserve für das Führungs– und das Störverhalten zeigen,
2. unempfindlich gegenüber Parameterschwankungen der Regelstrecke sein,
3. ein gutes dynamisches Führungs– und Störverhalten besitzen, d. h. die Sprungantwort der Regelgröße muss ausreichend gedämpft und der Einfluss der Störgrößen auf die Regelgröße gering sein,
4. eine kurze Anregelzeit und Ausregelzeit und geringes Überschwingen besitzen und
5. im stationären Zustand soll die bleibende Regelabweichung für die Führungs– und die Störübertragungsfunktion gegen null gehen.

Der Regler bildet die Regeldifferenz $E(s) = W(s) - Y(s)$ und berechnet entsprechend seiner Übertragungsfunktion $G_R(s)$ die Stellgröße $U_R(s)$. Abbildung 3.14 zeigt das Blockschaltbild des geschlossenen Regelungssystems. Hierbei beschreibt $G_R(s)$ die Übertragungsfunktion des Reglers im Bildbereich und $G_S(s)$ die Übertragungsfunktion der Regelstrecke im Bildbereich.

Der kontinuierliche PID–Regler wird durch Parallelschaltung eines P–, eines I– und eines D–Gliedes aufgebaut. Abbildung 3.15 zeigt das Blockschaltbild des PID–Regler im Bildbereich.

Die Übertragungsfunktion des PID–Reglers wird aus dem Blockschaltbild ermittelt.

$$G_R(s) = \frac{U_R(s)}{E(s)} = K_P + K_D s + \frac{K_I}{s} ; \quad m > n \quad (3.45)$$

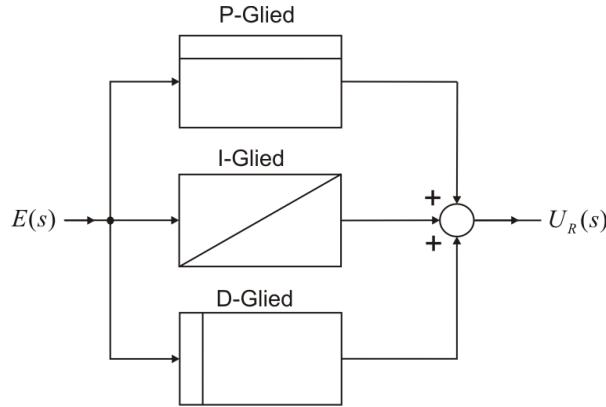


Abbildung 3.15: Blockschaltbild PID-Regler im Bildbereich

Mit den Ausdrücken für die Nachstellzeit T_N und Vorhaltzeit T_V geht Gleichung (3.45) über in:

$$G_R(s) = \frac{U_R(s)}{E(s)} = K_P \left[1 + T_V s + \frac{1}{T_N s} \right] \quad (3.46)$$

Die Reglergleichung lautet somit:

$$u_R(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{de(t)}{dt} = K_P \left[e(t) + \frac{1}{T_N} \int_0^t e(\tau) d\tau + T_V \dot{e}(t) \right] \quad (3.47)$$

mit

$$\begin{aligned} T_N &= \frac{K_P}{K_I} \\ T_V &= \frac{K_D}{K_P} \end{aligned}$$

Der Graph der Sprungantwort

$$h(t) = K_R + K_D \delta(t) + K_I t \quad (3.48)$$

und das Symbol des PID-Reglers sind in Bild 3.16 dargestellt.

Die Zeitkonstanten T_N, T_V und der Verstärkungsfaktor K_P werden an den Reglertypen direkt eingestellt.

3.8 Modular Controller Architecture (MCA)

Modular Controller Architecture (*MCA*)^[9] [39] ist eine am Forschungszentrum für Informatik^[4] (*FZI*) entwickelte Steuerungsarchitektur für autonome Robo-

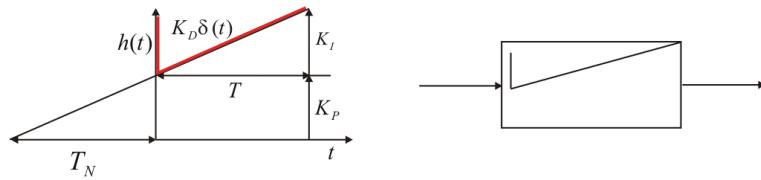


Abbildung 3.16: Sprungantwort und Symbol des PID-Reglers

ter. Für eine effiziente Entwicklung und Realisierung von Roboterprototypen ist es von Vorteil transferfähige Softwarekomponenten zu verwenden; wieder-verwendbare Komponenten mit standardisierten Schnittstellen im Hinblick auf erweiterbare Architekturen. *MCA* ist ein modulares Software Framework welches diese Kernpunkte realisiert. *MCA* ist kein automatisches Codegenerierungstool, weiterhin enthält es keine grafische Programmieroberfläche. Diese Steuerungsarchitektur ist grundsätzlich aus drei Schichten aufgebaut:

Linux Komponente:

Die oberste der drei Schichten läuft auf einem PC unter dem Betriebssystem *Linux*. Diese Schicht enthält die Benutzerschnittstelle und alle Steuerungsaufgaben, die keine Echtzeitbedingungen erfordern.

RT-Linux-Teil:

RT-Linux (Real-Time-Linux) [14][15] ist eine Linux Erweiterung, die unter Echtzeitbedingungen arbeitet. Der RT-Linux-Teil bildet die mittlere Schicht der Steuerung und übernimmt alle Aufgaben, die in Echtzeit ablaufen müssen. Darunter fällt unter anderem die Trajektorienplanung und die Kinematikberechnungen.

DSP56F803 μController:

Die DSP56F803 Mikrocontroller stellen die unterste hardwarenahe Schicht dar. Sie übernehmen die Ansteuerung der Motoren und lesen Sensorsignale aus. Weiterhin realisieren sie die ersten Regler und Filter. Die Mikrocontroller sind über einen CAN-Bus mit dem PC und dem dort laufenden RT-Teil verbunden.

3.8.1 Architektur

Der MCA Aufbau gleicht einer klassischen hierarchischen Kontrollstruktur. Auf der untersten Ebene befinden sich die Hardwareschnittstellen. Je weiter man sich im System nach oben bewegt, desto abstraktere Einheiten findet man vor. Auf der obersten Ebene des Systems ist die Benutzerschnittstelle zu finden. Generell existieren innerhalb des MCA-Systems zwei Datenstromrichtungen:

- **Sensor-Daten:** Der Fluss der Sensordaten beginnt in den Sensoren und geht über alle Module, welche diese Sensordaten erforderlich bis hin zur Benutzerschnittstelle. Die Module zwischen dieser Datenstromrichtung können Sensordaten modifizieren und/oder neue Sensorwerte ableiten.

- **Control-Daten:** Der Controldatenfluss entspringt der obersten Ebene und bewegt sich abwärts zu den Aktoren.

Die MCA Basiseinheit wird durch *Module* bereitgestellt. Diese Module können in *Gruppen* zusammengefasst werden. Die Module innerhalb einer Gruppe kommunizieren über frei konfigurierbare *Kanten* miteinander. Eine Gruppe kann in eine andere Gruppe gepackt werden oder innerhalb eines *Parts* als main loop eingesetzt werden. Ein Part stellt die Hauptausführungsumgebung innerhalb der Modular Controller Architecture dar. Im folgenden werden diese Komponenten diskutiert.

3.8.1.1 Module

Ein Modul bildet die Basiseinheit des MCA Kontrollsysteams. Ein Modul nimmt Sensorinformationen von unteren Ebenen oder der Hardwareebene über die *sensor input* Schnittstelle an, verarbeitet diese in einer *Sense()* Funktion und über gibt sie an die *sensor output* Schnittstelle. Controldaten werden über die *Control input* Schnittstelle angenommen, in der *Control()* Funktion verarbeitet und an *Control out* weitergeleitet. Die *Sense()* und *Control()* Funktionen können Daten via lokale Variablen austauschen. Diese Modulstruktur wird in Abbildung 3.17 dargestellt. *Control input*, *Control output*, *Sensor input* und *Sensor output* werden als double Vektoren modelliert. Jedes Vektorelement beinhaltet eine Beschreibung. Dem gesamten Vektor ist ein Flag zugeordnet, der dem Modul signalisiert ob sich Inhalte der Vekorelemente geändert haben. Somit wird der *Sensor()* bzw. *Control()* Programmcode nur ausgeführt, wenn neue Werte zur Verarbeitung anstehen. Eine zweite Schnittstelle zum setzen von Kontrollsystemparamtern steht innerhalb des MCA-Systems zur Verfügung. Diese wird als *Parameters* bezeichnet. Jedes Modul kann über beliebig viele Paramterwerte verfügen. Innerhalb der Modulklasse werden diese Modulparameter als Floata rray repräsentiert. Auf diese Modulparameter kann von außerhalb zugegriffen werden. Das *MCAbrowser* Tool stellt die Funktionalität zur Verfügung, Parameter zu lesen und gegebenenfalls während der Systemausführung zu ändern. Daher ist es nicht notwendig das System zwecks Feintuning zu stoppen. Ist der beste Wert eines Parameters gefunden, so kann dieser als Standardwert fest in das System eincompiliert werden.

3.8.1.2 Gruppen

MCA beruht konzeptionell auf der Grundlage, funktionale Basisblöcke in Module zusammen zu fassen. In den meisten Fällen besteht ein Steuerungssystem aus mehr als einem Funktionsblock. Aufbauend auf dieser Grundlage bietet das MCA System die Möglichkeit Funktionsblöcke in Modulgruppen zu kombinieren. Module werden innerhalb einer Gruppe (engl. *group*) in eine bestimmte Ebene eingefügt. Eine Modulgruppe bietet die gleichen Schnittstellen wie Module, bestehend aus *Control input*, *Control output*, *Sensor input* und *Sensor output* Vektoren und einer Modulgruppenbeschreibung. Eine Gruppe organisiert die enthaltenen Module inklusive ihrer Kanten in einer hierarchischen Struktur. Innerhalb einer Modulgruppe werden die enthaltenen Module und Kanten Level

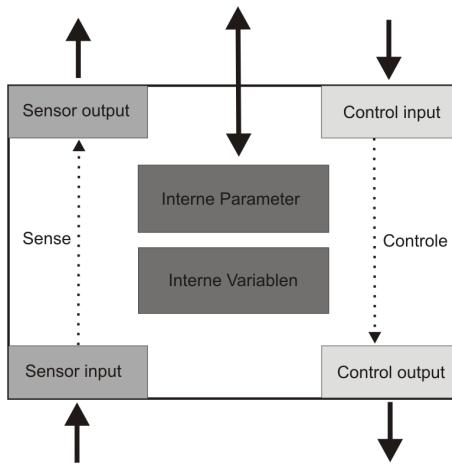


Abbildung 3.17: Modulaufbau Inputs/Outputs

für Level iterativ ausgeführt. Der Aufruf der *Control()* Funktion einer Gruppe resultiert in der Ausführung aller Abwärtsverkantungen und *Control()* Funktionen aller Module von oben nach unten. Die *Sense()* Funktion führt alle Aufwärts kanten und *Sense()* Funktionen der Module von unten nach oben aus.

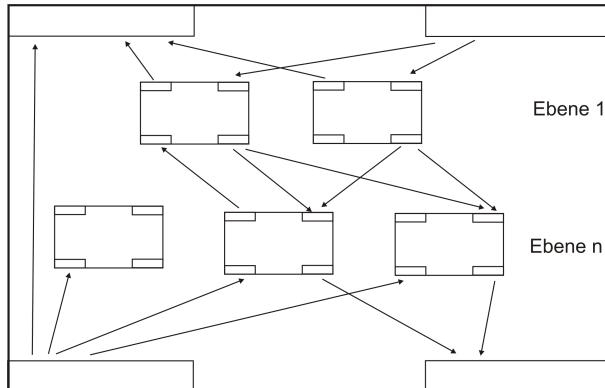


Abbildung 3.18: Beispiel einer MCA Gruppe

3.8.1.3 Kanten

Datenhandling zwischen Modulen wird über Kanten (engl. *edge*) realisiert. Eine Kante verbindet *Sensor output* mit *Sensor input* oder *Control output* mit *Control input* zweier Module. Input und Output Schnittstellen werden als Vektoren bereitgestellt. Eine Kante kopiert entweder einzelne Werte, einen Teil des Vektors oder den gesamten Vektor einer Schnittstelle zu einer Schnittstelle eines anderen Moduls. Der Kopiervorgang wird nur ausgeführt, falls sich Werte im Quellvektor geändert haben. Die Übertragung einzelner Werte mit spezifizier-

ten Indizes gewährleisten eine flexible Permutation der Werte. Diese Eigenschaft bietet eine schnelle und einfache Möglichkeit eine komplexe Kontrollstruktur innerhalb einer Gruppe aufzubauen. Die *Control()* und *Sense()* Funktionen regeln die Datenstromrichtung und die Modulausführungen innerhalb einer Gruppe. Dieses Verhalten wird am Beispiel der *Sense()* Funktion erläutert:

1. Passiere alle Datenkanten beginnend der *Sensor input* Schnittstelle der Gruppe zu allen Modulen des untersten Levels n der Gruppe.
2. Rufe alle *Sense()* Funktionen der Module auf der untersten Ebene n auf.
3. Passiere alle Sensordaten zwischen den Modulen der Ebenen n und $n-1$ entsprechend ihrer Verbindungskanten.
4. Rufe alle *Sense()* Funktionen der Module in der $n-1$ Ebene auf.
5. Wiederhole Schritt 3 und 4 für alle $n = n - 1$ Ebenen.
6. Aktualisiere den *Sensor output* der Gruppe.

Die Ausführung der *Control()* Funktion geschieht analog mit dem Unterschied, dass mit dem ersten Level begonnen und absteigend gearbeitet wird.

3.8.1.4 Part

Ein Part ist die Ausführungsumgebung des MCA Systems. Jeder Part beherbergt ein oder mehrere Module bzw. Modulgruppe(n). Der Linux Prozesskontext d. h. die notwendige Ausführungsumgebung wird vom Part bereitgestellt. Parts können mit anderen Parts, welche auf verschiedenen Computern ausgeführt werden bzw. in anderen Umgebungen laufen Verbindungen eingehen. Der Datenfluss innerhalb eines Parts wird vom Part selbst über die *Sensor* und *Control* Daten bewältigt. Die folgenden Schritte werden periodisch ausgeführt.

1. Aufruf der *Sense()* Funktionen der verwalteten Module
2. Aufruf der *Control()* Funktionen der verwalteten Module

Mit der Ausführung jeder einzelnen dieser Funktionen wird erst nach dem Ablauf einer vorgegebenen periodischen Zeit begonnen. Diese periodische periodische Ausführungszeit in μs kann für jeden Part separat eingestellt werden. Um die Kommunikation via GUI (Grafische Benutzer Schnittstelle) mit der Ausführungsumgebung dem *Part* zu gewährleisten verwendet MCA so genannte inter-part Verbindungen. Diese setzen zur Übertragung das TCP/IP Protokoll ein. Daher können Steuerungsapplikationen auf verschiedene PCs verteilt werden. Selbstverständlich kann ab diesem Punkt keine Echtzeitperformance garantiert werden.

3.8.1.5 Echtzeit– und Netzwerktransparenz

MCA bietet vollständige Netzwerk und Echtzeittransparenz. Ausgehend vom Kontrollsysteem betrachtet macht es keinen Unterschied wo ein bestimmter Part ausgeführt wird, solange jedem neu zugefügten Part mitgeteilt wird wo die anderen Parts ausgeführt werden. Es obliegt dem Systemdesigner wo und unter welchen Bedingungen der Part ausgeführt werden soll. Dieses Konzept erlaubt die Entwicklung einer Kontrollstruktur in einer sicheren Linuxumgebung. Somit können Fehler mit normalen Werkzeugen wie einem Debugger gefixt und danach das gesamte entwickelte System ohne Modifikationen des Quellcodes auf einem Echtzeitsystem zur Ausführung gebracht werden. Die Benutzerschnittstelle bzw. der *MCAbrowser* können auf einem zweiten PC ausgeführt werden und sich mit dem Robotersystem via wireless LAN (TCP/IP) verbinden.

3.8.2 Tools

Das MCA Framework stellt zwei essentielle Tools zur Verfügung. Dabei handelt es sich um *MCAGUI* und den *MCAbrowser*. *MCAGUI* stellt eine grafische Oberfläche zur Verfügung, mit der *Control input* und *Sensor output* Werte bzw. Parameter der Softwaresteuerung angezeigt bzw. gesetzt werden können. Mit Hilfe des *MCAbrowsers* können beispielsweise mehrere Module mit der gleichen Funktionalität parallel integriert und mühelos verglichen werden, indem alle anderen Module bis auf das zu testende deaktiviert werden. Dies geschieht ohne Neucompilierung oder Neustart der Steuerungssoftware. Im folgenden werden die beiden Tools vorgestellt.

3.8.2.1 MCAGUI

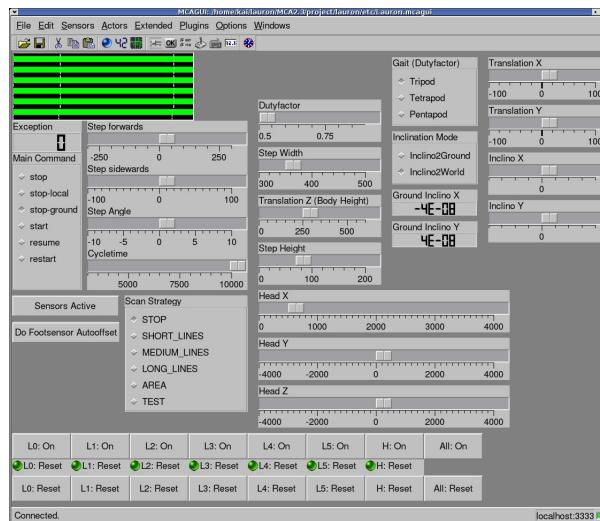


Abbildung 3.19: MCAGUI Steuerungssystem für LAURON IVb

MCA liefert ein frei konfigurierbares Benutzerinterface namens *MCA GUI*. *MCA-GUI* verbindet sich mit dem MCA Part des Kontrollsystems und nimmt Einfluss auf die *Control inputs* und *Sensor outputs*. Die GUI kann dynamisch konstruiert werden, indem Eingabemasken mit den *Control inputs* bzw. *Sensor outputs* eines Parts verbunden werden. Es stehen folgende Eingabemasken zur Verfügung: Schieberegler, Buttons, LCD Anzeigen, radio Buttons, LEDs, und eine Open Inventor [44] Schnittstelle zur 3D Repräsentation eines Roboters und seiner Umgebung. Diese GUI kann auf jedem PC mit funktionsfähiger Netzwerkkarte ausgeführt werden. Der PC mit der GUI verbindet sich mit der Steuerung und tauscht über das TCP/IP Protokoll Daten aus.

3.8.2.2 MCAbrowser

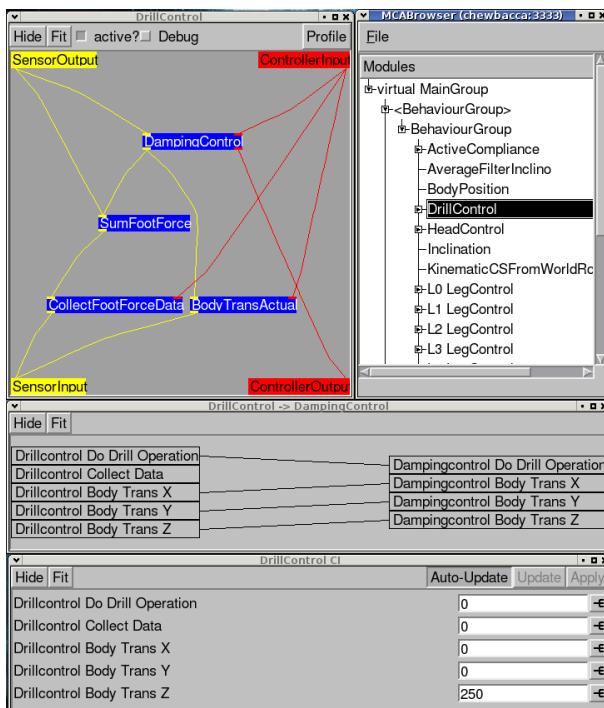


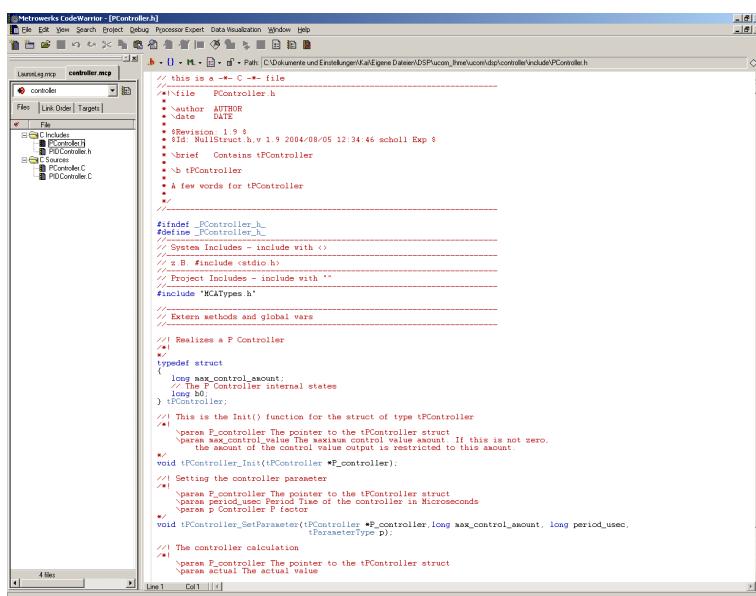
Abbildung 3.20: *MCAbrowser* untersucht Kanten, *Control input* einer Gruppe

Das QT [13] basierende Tool *MCAbrowser* visualisiert die komplette MCA Struktur. Der Browser verbindet sich via TCP/IP mit dem MCA Part und stellt das System in einer Baumstruktur dar. Ferner können verschiedene Parametereinstellungen einfach getestet und verglichen werden ohne das System neu zu starten oder erneut compilieren zu müssen. Des weiteren bietet das Tool die Möglichkeit, Modulparameter darzustellen und ggf. zu ändern. Zusätzlich können die Kanten der Module visualisiert werden, die Inhalte der *Control* bzw. *Sensor input* und *outputs* der Module angezeigt werden, sowie einzelne Module aktiviert bzw. deaktiviert werden ohne das System neu zu starten. Eine weitere

Stärke liegt darin, die einzelnen Kanten zwischen Modulen per Maus auszuwählen und die Verbindungen inklusive ausgetauschter Werte nachzuvollziehen.

3.9 Metrowerks Codewarrior

Für die Programmierung der Motorola DSP56F803 Mikrocontroller kommt die von Metrowerks entwickelte Entwicklungsumgebung namens Codewarrior [8] in der Version 7.2 zum Einsatz. Zusätzlich ist es erforderlich das Metrowerks Software Development Kit (kurz. SDK) in der Version 2.5 zu installieren. Die Codewarrior Entwicklungsumgebung bietet die Möglichkeit sowohl in der 56800 Assembler Sprache zu arbeiten, sowie einen für diese Prozessorfamilien optimierten ANSI C Compiler zu entwickeln. Die Entwicklungsumgebung bietet eine grafische Oberfläche, einen Quellcodeeditor und eine Kommandozeileingabe. Der Projektmanager stellt alle notwendigen Tools zur Verfügung die erforderlich sind um komplexe Projekte zu konfigurieren und letztlich zu managen. Hierzu zählt die komplette Kontrolle über die Quelldateien sowie aller Bibliotheken und Abhängigkeiten. Die Projektabhängigkeiten werden automatisch verwaltet. Dies macht die Verwendung von komplizierten Makefiles überflüssig. Vorgefertigte Templates erleichtern die Erstellung eines neuen Projektes. Der Quellcodeeditor bietet Syntaxhighlighting und drag-and-drop Support. Eine grafische Ausgabe stellt komplexe Datenstrukturen in einer Baumstruktur dar. Der enthaltene Debugger erfüllt die Funktionalität eines Standard-C-Debuggers, inklusive das uneingeschränkten setzen von Breakpoints. Das SDK bietet Bibliotheken welche die Konfiguration und den Einsatz von Mikrocontroller-spezifischen I/O Schnittstellen einfacher gestaltet. Dazu zählt z.B. die Konfiguration der GPIOs, der Puls-Weiten-Modulation (PWM) und der CAN-Bus Schnittstelle. Siehe DSPF803 Spezifikationen in Kapitel 5.1.1.



```

//<> this is a C file
//<> \file tPController.h
//<> \author AUTHOR
//<> \date DATE
//<> \revision 1.9 $ 
//<> $Id: tPController.h,v 1.9 2004/08/05 12:34:46 scholl Exp $
//<> \brief Contains tPController
//<> \b tController
//<> \a A few words for tPController
//<> \e

#ifndef _tPController_h_
#define _tPController_h_
//<> System Includes - include with <>
//<> #include <stdio.h>
//<> Project Includes - include with <>
#include "MCATypes.h"

//<> External methods and global vars
//<>

//<> Realizes a P Controller
//<> 
//<> \typedef struct
//<> {
//<>     long max_control_wcount;
//<>     // The P Controller internal states
//<>     long period_usec;
//<> } tPController;

//<> This is the Init() function for the struct of type tPController
//<> \param p_controller The pointer to the tPController struct
//<> \param wmax_control_value The maximum control value amount. If this is not zero,
//<>     the amount of the control value output is restricted to this amount.
//<> \param p_controller F factor
void tPController_Init(tPController *p_controller);

//<> Setting the controller parameters
//<> \param p_controller The pointer to the tPController struct
//<> \param period_usec The time of the controller in Microseconds
//<> \param p Controller F factor
void tPController_SetParameter(tPController *p_controller, long max_control_amount, long period_usec,
                             tParameterType p);

//<> The controller calculation
//<> \param p_controller The pointer to tPController struct
//<> \param actual The actual value

```

Abbildung 3.21: Codewarrior Entwicklungsumgebung

Kapitel 4

Aufbau des sechsbeinigen Laufroboters Lauron IVb

Mehrbeinige Laufmaschinen sollten in der Lage sein sich auf ebenem und unebenem Gelände fortzubewegen ohne Kenntnis der Umgebungsmerkmale. Des Weiteren sollten sie Unebenheiten und Unwegsamkeiten selbstständig erkennen und bis zu einem gewissen Maße umgehen können. Ferner sollten sie in der Lage sein bestimmte technische Aufgaben erledigen zu können für die sie ausgelegt wurden.

Um dieser Aufgabenstellung gerecht zu werden, wurde der Roboter Lauron IVb (Abbildung 4.1) entwickelt. Dabei handelt es sich um einen Prototypen der vierten Generation – einen Laufroboter mit sechs Beinen. Entwickelt wurde diese Laufmaschine am **Forschungszentrum für Informatik[4]** (*FZI*) in Karlsruhe. In diesem Kapitel werden die technischen Grundlagen, die dieser Laufmaschine zu Grunde liegen vorgestellt.

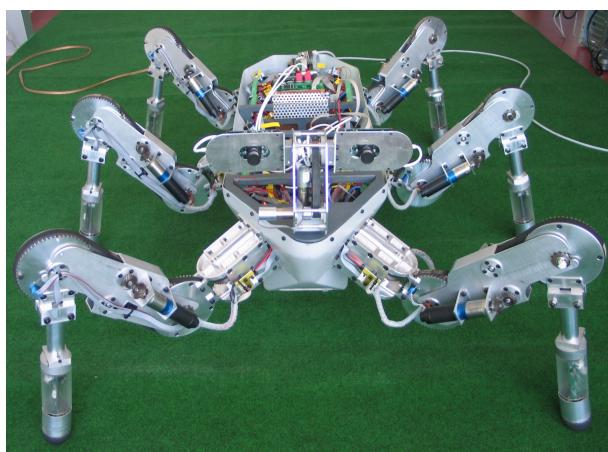


Abbildung 4.1: Die Laufmaschine Lauron IVb

4.1 Mechanischer Aufbau des Laufroboters Lauron IVb

Um den Anforderungen aus Kapitel 4 Genüge zu tun erfüllt der Laufroboter Lauron IVb gewisse Randbedingungen:

- sechs Beine die eine statische stabile Gangart garantieren, entsprechend dem Laufvorgang einer Stabheuschrecke.
- Verwendung von Sensoren (Gelenkwinkelsensoren, Kraftsensoren, Neigungssensor).

Die Grundform des Laufroboters ist dem Körper der Stabheuschrecke nachempfunden. An diesem sind sechs Beine befestigt, welche jeweils drei aktive Freiheitsgrade besitzen (siehe Abbildung 3.2). Ferner besitzt der Laufroboter noch eine Stereobildkamerakopf der über zwei aktive Freiheitsgrade verfügt. Die Tabelle 4.1 gibt einen Überblick über die mechanischen Parameter die dem Laufroboter Lauron IVb zu Grunde liegen.

4.1.1 Der Körper des Laufroboters Lauron IVb

Wie bereits eingangs erwähnt ist der Körper der Laufmaschine Lauron IVb Stabheuschreckenartig. Die Beine sind an den Seitenflächen des Körpers befestigt. Wie aus der schematischen Abbildung 3.4 zu erkennen ist, sind die Beine 0,1,4,5 im 30° Winkel zur Längsseite des Körpers angebracht (vgl. Kapitel 3.2.2). Ferner sind alle sechs Beine bezogen auf das *SKS* nicht parallel zum Untergrund, sondern mit einer Neigung nach unten in der *x-z-Achse* von 30° für die Beine 2,3 und einer Neigung von 25° für die Beine 0,1,4,5 angebracht. Die Laufmaschine Lauron IVb kann sowohl in Bezug auf das *SKS* in positive x-Richtung wie auch in negative x-Richtung gleich schnell laufen. Weiterhin beherrscht Lauron IVb noch den Seitwärtsgang (positive y- und negative y-Richtung) welcher allerdings bedingt durch die Bauart nicht der Geschwindigkeit des Vor- bzw. Rückwärtsgangs gleich kommt. Wie in Abbildung 4.1 zu sehen ist, wird durch die Befestigung des Kamerakopfes die Vorförtslaufrichtung festgelegt. Folglich bezogen auf das *SKS* in positiver x-Achsenrichtung.

Das hardware Steuerungssystem ist im Körper der Laufmaschine Lauron IVb untergebracht. Die 6 Mikrocontrollerplatten für die Beine wie auch die Mikrocontrollerplatine für den Kamerakopf sind an den inneren Seitenwänden des Körpers angebracht. Der Industrie PC/104 ist an einer Rahmenverstrebung im mittleren Bereich des Körpers aufgehängt. Abbildung 4.2 zeigt diesen Aufbau.

4.1.2 Die Beine des Laufroboters Lauron IVb

Die Beine stellen die Fortbewegung sicher. Mit diesen ist es möglich den Laufroboter auf ebenem wie auch auf unebenem Gelände bis zu einem gewissen Grad stabil zu bewegen. Im Nachfolgenden wird auf den Aufbau der Beine genauer eingegangen.

Basis Parameter	
Grundform	Stabheuschreckenartig
Anzahl der Beine	6
Aktive Bein-Freiheitsgrade (gesamt)	$6 \cdot 3 = 18$
Max. theoretische Geschwindigkeit ca.	0.5 km/h
Gewicht (Grundkonfiguration)	19 kg
Nutzlast ca.	10 kg
Körper	
Grundkörperabmessungen	
Höhe ca.	19 cm
Seitenlänge ca.	72 cm
Breite ca.	24 cm
Sensoren	
Neigungssensoren	1
Beine	
Aktive Freiheitsgrade	3
Arbeitsbereich der Gelände	
α-Gelenk	$-60^\circ \dots +60^\circ$
β-Gelenk	$-30^\circ \dots +60^\circ$
γ-Gelenk	$-30^\circ \dots +60^\circ$
Antiebe	Gleichstrommotoren (24V)
Abmessungen eines Beines	
Gesamt	64,5cm
A-Segment	7cm
B-Segment	7,5cm
C-Segment	20cm
D-Segment	30cm
Masse eines Beines ca.	2,2kg
Sensoren	
Gelenkwinkelsensoren	3 optische Inkrementalsensoren pro Beingelenk
Fußkraftsensoren	3 orthogonale Komponenten im Unterschenkel
Motorstromsensoren	3 pro Gelenkmotor
Kopf	
Aktive Freiheitsgrade	2
Sensoren	
Kameras	2 am Kopf angerbrachte Firewire-Stereokameras

Tabelle 4.1: Mechanische Parameter des Laufroboters Lauron IVb

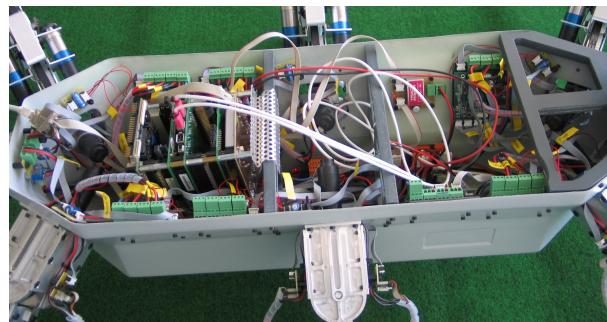


Abbildung 4.2: Der Körper der Laufmaschine Lauron IVb

4.1.2.1 Kinematischer Aufbau der Beine

Um die räumlichen Trajektorien zu erzeugen besitzt jedes Bein entsprechend dem Vorbild der Stabheuschrecke aus der Natur (vgl. Kapitel 3.1.1) drei Freiheitsgrade. Somit kann im Arbeitsbereich eines Beines jede Position mit der Fußspitze erreicht werden. Die hier verwendeten Freiheitgrade werden als Rotationsfreiheitsgrade bezeichnet. Beim Laufroboter Lauron IVb werden die drei Rotationsfreiheitsgrade hintereinander verkettet. Das erste Gelenk realisiert eine Bewegung bezogen auf das *BBKS* in der *x-y-Achse*. Das zweite und dritte Gelenk stehen parallel zueinander und bilden die Bewegung in der *y-z-Achse*.

Da es mit den 3 Freiheitsgraden im Bein nicht möglich ist, die räumliche Ausrichtung der Fußspitze zu beeinflussen wird am unteren Ende des Fußes eine Gummihalbkugel angebracht. Damit wird eine punktförmige Auflagefläche geschaffen. Nachteilig ist zu erwähnen, dass bedingt durch die geringe Reibung des Fußes auf dem Untergund die Laufmaschine auf rutschigem Material in der Stemmpphase leicht den stabilen Bodenkontakt verliert. Abbildung 4.3 zeigt den Aufbau eines Beines.

4.1.2.2 Arbeitsbereich eines Beines

Die Abbildung 4.4 zeigt den Arbeitsbereich eines Beines. Betrachtet man die Abbildung 4.4a so sieht man den Arbeitsbereich eines Beines in *y-z-Ebene* im *BBKS*. Daraus ist zu erkennen, dass der Arbeitsbereich des Beines durch Kreisbögen beschränkt wird, welche durch die minimal und maximal möglichen Gelenkwinkel entstehen. Die Geraden, die die beiden Kreisbögen aus Abbildung 4.4a schließen beschränken ebenfalls den Arbeitsbereich des Beines. Abbildung 4.4b zeigt den Arbeitsbereich in der *x-y-Ebene* des *BBKS* und ist analog zu der Abbildung 4.4a zu verstehen.

4.1.2.3 Aktoren und Sensoren

Aktoren

Die Aktoren realisieren die Winkeländerung der einzelnen Beinsegmente. Somit

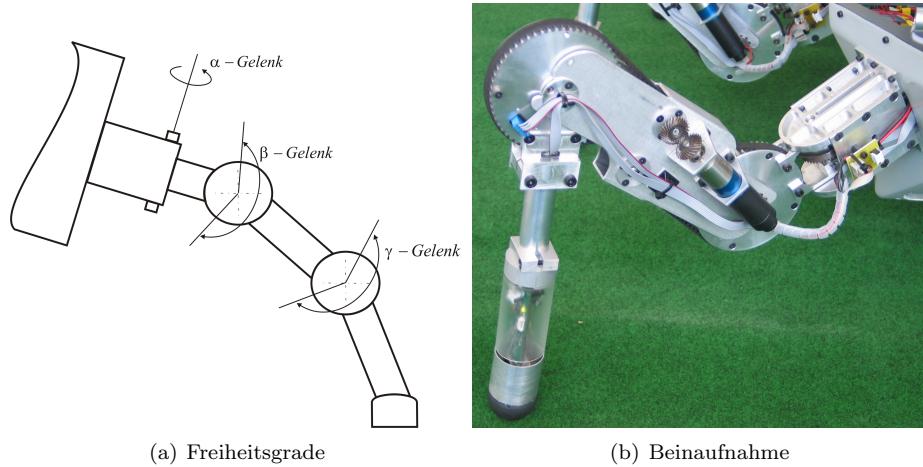


Abbildung 4.3: Beinaufnahme

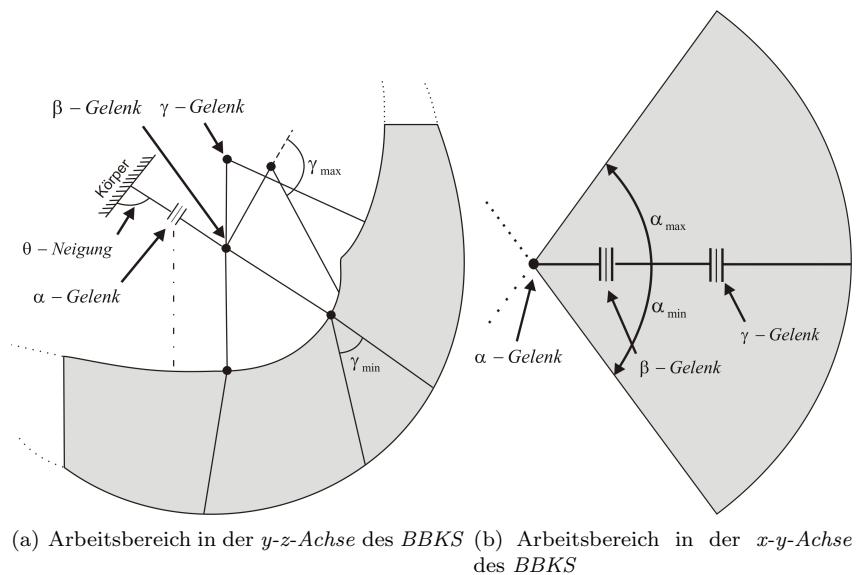


Abbildung 4.4: Arbeitsbereich eines Beines

wird für jedes Gelenk (α, β, γ) jeweils ein Motor benötigt. Bei dem Laufroboter Lauron IVb werden hierfür 12 V, 17 W DC Kleinstmotoren von *FAULHABER* mit Planetengetriebe eingesetzt [26][27]. Die nominale Untersetzung beträgt 134:1.

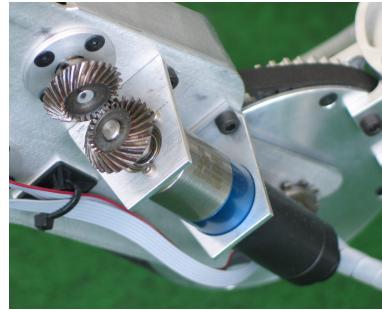


Abbildung 4.5: *Faulhaber*-Kleinstmotor

Positionssensoren

Um die Gelenkwinkel, die mit Hilfe der Motoren angefahren werden zu messen werden Inkrementalsensoren verwendet. Diese Gelenkwinkelpositionssensoren befinden sich direkt auf der Gelenkkachse. Die beim Überstreichen eines Winkels auftretenden Impulse werden per Hardware gezählt, so dass der gemessene Winkel schließlich als Zählerwert zur Verfügung steht. Die Umrechnung von Winkelwert W zu Impulswert I und umgekehrt wird mittels einer Linearfunktion durchgeführt [33].

$$I = a + W + i_0 \quad (4.1)$$

Dabei ist die Steigung $a = \frac{900}{360}$ die Anzahl der Impulse pro Winkelgrad. Der Achsenabschnitt i_0 entspricht dem Impulswert bei einer Gelenkstellung von 0° . Um die initiale Gelenkwinkelposition zu bestimmen wird ein Referenzpunkt angefahren. Dieser befindet sich am mechanischen Anschlag eines Gelenkes. Von dort werden für jede Bewegung die Impulse gezählt.

Kraftsensoren

Die Kraftsensoren an den Füßen von Lauron IVb bestehen aus Stahlbauteilen mit definierten Biegesetzen und darauf angebrachten Dehnungsmessstreifen (kurz DMS), die in Vollbrücke zusammengeschaltet sind. Die schwachen Signale werden durch einen Differenzverstärker geleitet, bevor die A/D-Wandlung auf den Mikrocontrollern durchgeführt wird (vgl. Kapitel 5.3). In Abbildung 4.6 zeigt die schematische Darstellung des Verformungskörpers, der wie ein „G“ geformt ist. Wirkt nun eine Kraft in negative z -Richtung, so wird der linke DMS gedehnt und der rechte gestaucht. Zur Filterung dieser Signale wird ein Medianfilter verwendet um die Ausreißer und das Rauschen in den Griff zu bekommen. Diese Signalwerte werden in Kraftwerte mit der Einheit Newton umgerechnet. Somit können die auf das Bein wirkenden Kräfte (x, y, z -Richtung) gemessen und in Newton angegeben werden. Die Umrechnung der Signalwerte S^i der Komponenten $i \in \{x, y, z\}$ in Kraftwerte F^i (Newton) wird wie auch bei den Positionssensoren mit Hilfe einer Linearfunktion durchgeführt [33]:

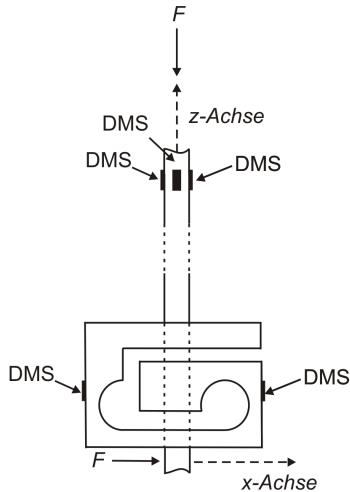


Abbildung 4.6:
Verformungskörper des Kraft-
sensors



Abbildung 4.7: Fußsensor

$$F^i = m^i \cdot (S^i - f_0^i) \quad (4.2)$$

Die Steigung m^i wird mit Hilfe von Referenzkraftgrößen bestimmt und die Achsenabschnitte f_0^i ergeben sich aus den Sensorwerten im entlasteten Zustand. Diese Kalibrierung ist notwendig, da die Parameter der Sensoren von Fertigungstoleranzen der verwendeten Bauteile, der Positionierung der DMS und der Verbindung des Dehnmessstreifens mit dem Verformungskörper abhängen. Da es sich hier um Mehrkomponentenkraftsensoren handelt muss die Referenzkraft in einer definierten Raumrichtung wirken. Um eine gerichtete Kraft zu erzeugen, kann mit Hilfe von Umlenkrollen die Kraftwirkungsrichtung geändert werden. Abbildung 4.8 zeigt diesen Vorgang für die $x-y-z$ -Achsenkalibrierung.

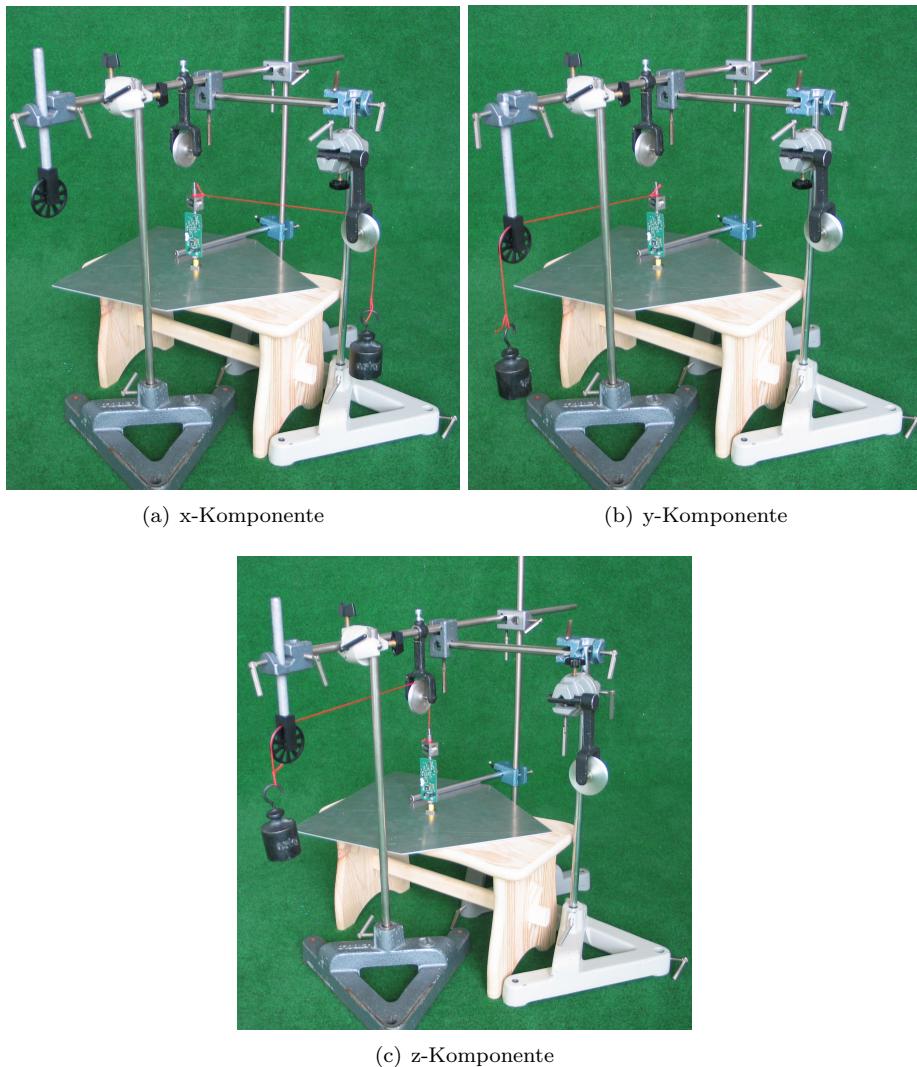


Abbildung 4.8: Kalibrierung des Kraftsensors

Kapitel 5

Aufbau des Steuerungssystems für den Laufroboter Lauron IVb

Im Folgenden wird die Hardware des Steuerungssystems für den Laufroboter Lauron IVb genauer eingegangen. Bei embedded Systems (eingebettete Systeme¹) spielt die Leistungsfähigkeit und die vorhandenen Ressourcen sowie die Input/Output Komponenten eine wichtige Rolle. Bei solchen Systemen werden nicht nur Daten manipuliert sondern auch angeschlossene Aktoren und Sensoren und deren Daten verarbeitet. Bei Embedded Systems handelt es sich um spezialisierte Systeme die spezielle Dienste zur Bearbeitung der gestellten Anforderungen enthalten. Daher ist eine genaue Kenntnis der Rechnerarchitektur und deren gezielte Nutzung von essentieller Grundlage um erfolgreich embedded Echtzeitsysteme² zu entwickeln.

5.1 Mikrocontrollerplatinen

Die sieben auf der Laufmaschine Lauron IVb untergebrachten Rechnereinheiten, enthalten DSP-Mikrocontroller des Typs Freescale DSP56F803 [30]. Diese Platten besitzen weiterhin Speichereinheiten (RAM, Flash-ROM), ein Altera FLEX 10K FPGA [18] und Signalverarbeitungseinheiten für Input/Output sowie für die Kommunikation.

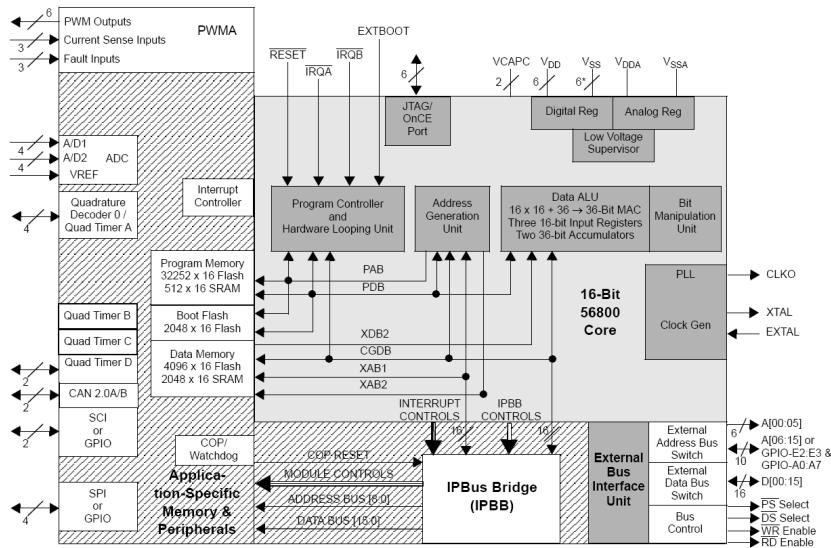


Abbildung 5.1: DSPF803 Blockdiagramm [30]

5.1.1 Mikrocontroller

Der Mikrocontroller 56F803 entstammt der 56800 Prozessorfamilie. Er kombiniert auf einem einzigen 16 Bit Chip, die Prozessorleistung eines Digitalen Signal Prozessors (kurz DSP) und die Funktionalität eines Mikrocontrollers. Der 56F803 enthält die notwendige Peripherie um speziellen Applikationen gerecht zu werden wie z.B. Bewegungskontrolle, Encoding, Stromversorgung, Fahrzeugsteuerung, Motormanagement und die Industrielle Steuerung und Überwachung von Maschinen. Der 56800 Kern basiert auf der Harvardarchitektur und erlaubt bis zu sechs Operationen pro instruction cyle. Ferner unterstützt der 56F803 die Programmausführung entweder im internen oder externen Speicher. Auf zwei Operanden kann pro Anweisungszyklus vom on-chip RAM zugegriffen werden. Der Controller enthält zwei externe, getrennte Interruptzugänge und bis zu 16 General Purpose Input/Output (kurz. GPIO). Er enthält 31.5K word (16-bit) Programm Flashspeicher und 4K Word Datenflashspeicher (jeweils über die JTAG Schnittstelle programmierbar). Des Weiteren enthält er 512 Word Programm RAM und 2K Word Daten RAM. Ein weiteres Feature des 56F803 ist die Integration eines Puls-Weiten-Modulator Moduls (kurz. PWM). Dieses Modul stellt drei unabhängige, individual programmierbare PWM Ausgangssignale zu Verfügung um eine verbesserte Motorsteuerungsfunktionalität zu gewährleisten. Zusätzlich enthält der Controller eine serielle Kommunikationsschnittstelle (kurz. SCI), ein Serial Peripheral Interface (kurz. SPI) und vier Timer. Jedes dieser Schnittstellen kann als GPIO genutzt werden falls ihre regulären Funktionen nicht benötigt werden. Der 56F803 verfügt über eine Controller Area Network

¹ Embedded Systems vereinigen durch ihre sehr hardwarenahe Konstruktion die große Flexibilität von Software mit der Leistungsfähigkeit der Hardware

² Echtzeitanforderungen sind zeitliche Rahmen, die ein Echtzeit-System beim Bearbeiten der Daten nicht überschreiten soll oder darf.

DSP	FPGA
Regelung der Gelenkwinkel (PID-Regler). Erzeugung der PWM-Signale.	Fortlaufende Auswertung des Absolutcodes. Generierung der Statuswerte der Motoren.

Tabelle 5.1: Funktionsblöcke von DSP und FPGA

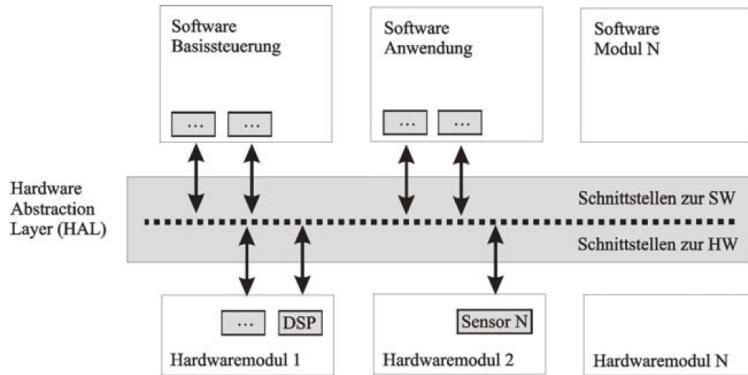
Schnittstelle (kurz. CAN Version 2.0) und einen internen Interrupt Controller [30].

5.1.2 Beincontroller



Abbildung 5.2: Physikalischer Aufbau des Beincontrollers

Der Beincontroller ist mit dem DSP56F803 Mikrocontroller bestückt und besitzt zu dem noch weitere periphere Komponenten. Abbildung 5.2 zeigt den physikalischen Aufbau des Beincontrollers. Das Steuerungskonzept der Basisregelung sieht die Durchführung von Regelungsaufgaben, Kontrollaufgaben und Rechenaufgaben hardwaretechnisch unter Verwendung von DSP- und FPGA-Bausteinen vor. Aufgaben werden im Sinne einer schnellen Abarbeitung bei geringem Stromverbrauch auf beide Bausteintypen verteilt. Die Kommunikation zwischen den beiden Komponenten erfolgt über den Daten- und Adressbus. Die Aufteilung der Aufgaben ist in Tabelle 5.1 dargestellt. Um die Adaption und Erweiterbarkeit der Struktur auch für zukünftige Systeme zu gewährleisten wird als Ebene zwischen Hard- und Softwarekomponenten eine *Hardware Abstraction Layer* (HAL) eingeführt. Aufbauend auf der Software MCA können so Schnittstellen zwischen der HAL und den Hard- bzw. Softwarekomponenten definiert werden (siehe Abbildung 5.3). Durch die Implementierung solcher MCA-Module ist die Entwicklung der Software von der Entwicklung der Hardware völlig entkoppelt. Erweiterungen und Modifikationen einzelner Module haben keinen Einfluss auf das Gesamtsystem.

Abbildung 5.3: *Hardware Abstraction Layer*

5.2 Industrie-PC/104



Abbildung 5.4: PC/104 [7]

PC/104 ist ein Industriestandard [12] in dem die Spezifikationen einer kompakten PC-Architektur definiert sind. Bei diesem PC-Standard handelt es sich um einen Industrie-PC mit sehr geringen Abmessungen der jedoch mit allen wesentlichen Komponenten eines standard Desktop PC ausgerüstet. Somit unterstützt der Cool RoadRunner III PC/104 der Lippert GmbH [42] embedded PC Applikationen mit der Rechenleistung eines Pentium PCs. Der PC/104 besitzt ein AGP4x Slot, 10/100 BaseT Ethernet und einen AC-97 I/O on-Board Soundchip. Ferner bietet er einen CompactFlash Sockel und ein Ultra DMA-100 EIDE Interface und 512 MB SDRAM Systemspeicher. Ausgestattet ist der PC/104 mit einem Pentium III 933 MHz CPU und bietet somit genug Rechenleistung für eine Vielzahl an Anwendungen. Die Infrastruktur der Hardware wird mit dem VIA TwisterT Chipsatz realisiert, welcher eine integrierten Savage4 Grafikbeschleuniger von S3 enthält. Dieser besitzt 32MB Grafikspeicher und kann Auflösungen bis zu 1600 x 1200 mit 16.7 Millionen Farben darstellen. Die Platine bietet Anschlussmöglichkeiten für TFT und CRC Monitore und bietet

noch einen S-Video Out Anschluss. Zusätzlich zu den üblichen seriellen und parallelen Ports sind zwei USB Ports und ein IrDa auf dem Board integriert. Lokale Speichermedien können über das CompactFlash Modul angeschlossen werden. Systemerweiterungen sind über den PC/104 Bus möglich.

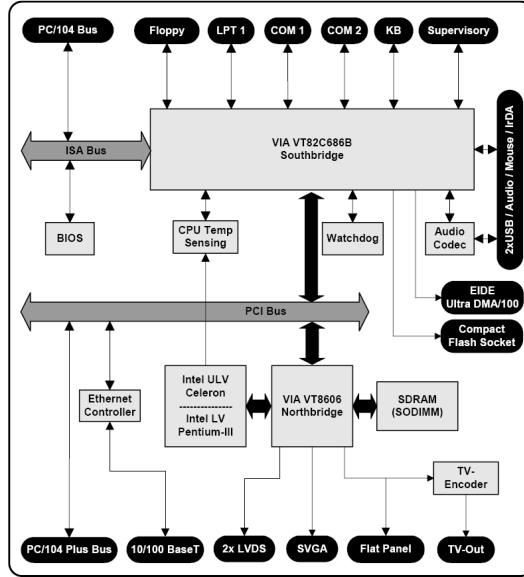


Abbildung 5.5: PC/104 Blockdiagramm [42]

5.2.1 Hauptcontroller PC/104

Zu den weiteren Stärken des PC/104 zählt die modulare Erweiterbarkeit über den PC/104 Bus. Auf dem PC/104 sind Steckverbindungen angebracht auf die andere Platinen mit bestimmten Schnittstellen angeschlossen werden können. Diese Platinen bieten wieder die Möglichkeit eine weitere Platine auf sich zu fassen. Somit entsteht eine Art Turm aus Platinen die eine gewisse gewünschte Funktionalität bieten. Der bei der Laufmaschine Lauron IVb eingesetzte PC/104 wurde mit verschiedenen Platinen erweitert. Hierzu gehört eine PC/104-PCA-1 PCMCIA Karte [43] mit zwei Slots zur Aufnahme einer wireless LAN Karte um eine drahtlose Kommunikation zu ermöglichen. Zusätzlich dazu ist noch eine eNet-CAN-Bus Platine [46] aus dem Hause Phytec, welche zur Kommunikation mit den Mikrocontrollerplatinen eingesetzt wird aufgesetzt. Diese besitzt zwei CAN Bus Interfaces. Für die Stereo-Firewirecamera am Kopf wurde eine Firewireplatine (IEEE 1394-Interface) benötigt, welche drei mögliche Anschlüsse beherbergt. Der PC/104 wurde noch um eine Platine mit PS/2 Tastatur und Monitor Anschluss, sowie einer Platine die als Netzteil agiert erweitert. Da das Platzangebot im Laufroboter Lauron IVb sehr begrenzt ist, wird auf eine IDE Festplatte verzichtet. Statt dessen kommt ein 1GB Microdrive von IBM zum Einsatz. Somit steht ein System zur Verfügung, das an Rechenleistung ausreichend ist um die planerischen Aufgaben wie z.B. Transformationen

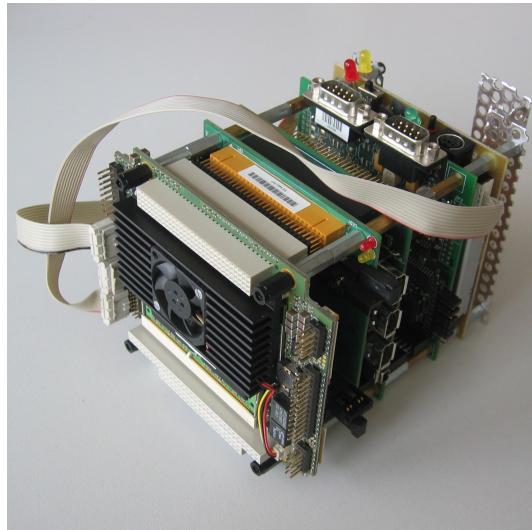


Abbildung 5.6: PC/104 mit Erweiterungsplatinen

und Generierung von Laufzyklen zu berechnen und die Input/Output Daten der Mikrocontroller zu verarbeiten.

5.3 Messwerterfassung

Bevor die Daten der Sensoren als Informationen im Mikrcontroller verarbeitet werden können, müssen die Messgrößen der Sensoren in elektrische Größen umgewandelt werden. Welche dann im nächsten Schritt in binäre Informationen gewandelt werden müssen. Hierfür kommt der A/D Wandler (Analog/Digital Wandler) zum Einsatz. Die Wandlung der Sensorsignale auf der Laufmaschine Lauron IVb wird von den DSP56F803 Mikrocontrollern übernommen. Dabei werden die analogen Messspannungen im Bereich von 0..5V auf binäre Werte zwischen 0 und 4095 (12 Bit) abgebildet. Daraus ist zu erkennen, dass alle zu messenden Werte in Form einer Spannung vorliegen müssen.

5.4 Kommunikation

Um die Kommunikation der in diesem Kapitel vorgestellten Hardwarekomponenten zu realisieren, werden zwei zentrale Kommunikationsmedien eingesetzt. Für die Kommunikation innerhalb der Laufmaschine Lauron IVb zwischen der PC/104 Komponente und den Mikrocontrollern wird das Controller Area Network (kurz. CAN) aus dem Hause Bosch eingesetzt. Der Datenaustausch der PC/104 Einheit mit externen PCs wird über das TCP/IP Protokoll realisiert. Abbildung 5.7 zeigt die Anwendung der Kommunikationsmethoden. Im folgenden wird auf die zwei Übertragungsmethoden genauer eingegangen.

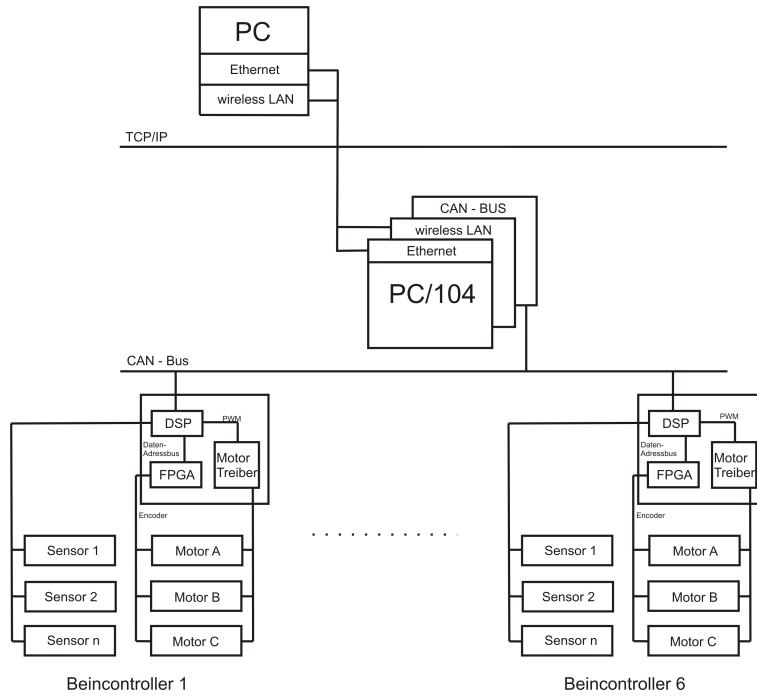


Abbildung 5.7: Kommunikation der Hardwarekomponenten

5.4.1 Das TCP/IP Referenzmodell

In diesem Abschnitt wird das Referenzmodell für die TCP/IP-Architektur vorgestellt [20]. Das *TCP/IP-Referenzmodell* – benannt nach den beiden primären Protokollen TCP und IP der Netzarchitektur beruht auf den Vorschlägen, die bei der Fortentwicklung des ARPANETs gemacht wurden. Das TCP/IP Modell ist zeitlich vor dem OSI-Referenzmodell entstanden. Das TCP/IP Modell besteht im Gegensatz zum OSI-Modell aus nur vier Schichten: Application Layer, Transport Layer, Internet Layer und Network Layer. Tabelle 5.2 zeigt das TCP/IP Referenzmodell. Folgende Ziele wurden bei der Entwicklung der Architektur definiert:

- Unabhängigkeit von der verwendeten Netzwerk-Technologie.
- Unabhängigkeit von der Architektur der Hostrechner.
- Universelle Verbindungsmöglichkeiten im gesamten Netzwerk.
- Ende-zu-Ende Quittungen.
- Standardisierte Anwendungsprotokolle.

Applikationsschicht (application layer):

Die Applikationsschicht (auch Verarbeitungsschicht genannt) umfasst alle höherschichtigen Protokolle des TCP/IP-Modells, die mit Anwendungsprogrammen

TCP/IP-Schicht	Protokolle
Anwendungsschicht	Telnet, FTP, SMTP, NFS
Transportschicht	TCP, UDP
Internet	IP
Netzwerkschicht	MAC, Ethernet, DSSS

Tabelle 5.2: TCP/IP–Referenzmodell

zusammenarbeiten und die Netzwerkinfrastruktur für den Austausch anwendungsspezifischer Daten nutzen.

Transportschicht (transport layer):

Die Transportschicht ermöglicht die Kommunikation zwischen den Quell- und Zielhosts. Im TCP/IP-Referenzmodell wurden auf dieser Schicht zwei Ende-zu-Ende-Protokolle definiert: das Transmission Control Protocol (TCP) und das User Datagram Protocol (UDP). TCP ist ein zuverlässiges verbindungsorientiertes Protokoll, durch das ein Bytestrom fehlerfrei zu einem anderen Rechner im Internet übermittelt werden kann. UDP ist ein unzuverlässiges verbindungsloses Protokoll, das vorwiegend für Abfragen und Anwendungen in Client/Server-Umgebungen verwendet wird, in denen es in erster Linie nicht um eine exakte, sondern schnelle Datenübermittlung geht.

Internetschicht (internet layer):

Die Internetschicht ist für die Weitervermittlung von Paketen und die Wegewahl (Routing) zuständig. Auf dieser Schicht und den darunterliegenden Schichten werden Punkt-zu-Punkt-Verbindungen betrachtet. Die Aufgabe dieser Schicht ist es, zu einem empfangenen Paket das nächste Zwischenziel zu ermitteln und das Paket dorthin weiterzuleiten. Kern dieser Schicht ist das Internet Protocol (IP), das einen unzuverlässigen, verbindungslosen Paketauslieferungsdienst bereitstellt. Das Internet Control Message Protocol (ICMP) ist fester Bestandteil jeder IP-Implementierung und dient zur Übertragung von Diagnose- und Fehlerinformativen für das Internet Protocol.

Netzwerkschicht (network layer):

Unterhalb der Internetschicht befindet sich im TCP/IP–Modell eine große Definitionsfläche. Das Referenzmodell sagt auf dieser Ebene nicht viel darüber aus, was hier passieren soll. Festgelegt ist lediglich, dass zur Übermittlung von IP–Paketen ein Host über ein bestimmtes Protokoll an ein Netz angeschlossen werden muss. Dieses Protokoll ist im TCP/IP–Modell nicht weiter definiert und weicht von Netz zu Netz und Host zu Host ab. Das TCP/IP–Modell macht an dieser Stelle Gebrauch von bereits vorhandenen Protokollen, wie z.B. Ethernet (IEEE 802.3) etc.

Da im TCP/IP Protokoll die Netzwerkschicht Platzhalter für verschiedene Techniken zur Datenübertragung liefert, wird die Host-an-Netz Schicht bei der Laufmaschine Lauron IVb durch das Ethernet Protokoll für eine kabelgebundene Übertragung mittels 10/100 BaseT Ethernet Controller bzw. mit dem DSSS (Direct Sequence Spread Spectrum) Frequenzspritzverfahren [34] für die PCMCIA wireless LAN Übertragung ausgefüllt.

5.4.2 Controller Area Network

Der CAN-Bus (Controller Area Network) [47] gehört zu der Familie der Feldbusse. Es handelt sich dabei um ein asynchrones, serielles Bussystem, das 1983 von Bosch für die Vernetzung von Steuergeräten im Automobilbau entwickelt wurde. Der CAN-Bus arbeitet nach dem CSMA-CA (Carrier Sense Multiple Access-Collision Avoidance) Verfahren. Die Daten sind NRZ-L codiert. Des Weiteren kommt zur Datensicherung das CRC-Verfahren zum Einsatz. Zur fortlaufenden Synchronisierung der Busteilnehmer wird Bit-Stuffing verwendet. Der Bus ist entweder mit Kupferleitungen oder über Glasfaser ausgeführt. Im Falle von Kupferleitungen arbeitet der CAN-Bus mit Differenzsignalen. Er wird normalerweise mit 3 Leitungen ausgeführt: CAN_HIGH, CAN_LOW und CAN_GND (Masse). CAN_LOW enthält den komplementären Pegel von CAN_HIGH gegen Masse. Dadurch können Gleichtaktstörungen unterdrückt werden. Die Übertragung der Daten erfolgt so, dass ein Bit, je nach Zustand, entweder dominant oder rezessiv auf den Busleitungen wirkt. Ein dominantes Bit überschreibt dabei ein rezessives. Die maximale Datenübertragungsrate beträgt 1Mbit/s. Dabei ist eine maximale Leitungslänge von 40m möglich. Bei 500kBit/s sind zum Beispiel 100m möglich und bei 125kBit/s 500m. Alle Teilnehmer kommunizieren mit der gleichen Baudrate. Der Buszugriff wird verlustfrei mittels der bitweisen Arbitrierung auf Basis der Identifier der zu sendenden Nachrichten aufgelöst. Dazu sensiert jeder Sender den Bus während er gerade den Identifier sendet. Senden zwei Teilnehmer gleichzeitig, so überschreibt das erste dominante Bit eines der beiden, das entsprechend rezessive des anderen, welcher dieses erkennt und seinen Übertragungsversuch beendet, damit der andere seine Daten übertragen kann.

Durch dieses Verfahren ist auch eine Hierarchie der Nachrichten untereinander gegeben. Die Nachricht mit dem niedrigsten Identifier darf immer übertragen werden. Für die Übertragung von zeitkritischen Nachrichten kann also ein Identifier hoher Priorität (niedrige ID, z.B. 0) vergeben werden um ihnen so Vorrang bei der Übertragung zu gewähren. Dennoch kann selbst bei hochpriorem Botschaften der Sendezeitpunkt zeitlich nicht genau vorher bestimmt werden. Die einzelnen Aufgaben des CANBus sind in sogenannten Schichten (Layer) definiert:

Bitübertragungsschicht (Physical Layer):

Diese Schicht beschreibt die physikalischen Eigenschaften, wie z.B. Stecker, Kabel, Signalpegel und wie ein Bit auf der Leitung dargestellt wird.

Übertragungsschicht (Transfer Layer):

Die Übertragungsschicht hat die Aufgabe, das spezifizierte Busprotokoll abzuarbeiten. Dazu gehören die Erzeugung eines Übertragungsrahmens (Pakets), die Anforderung des Busses mit der nötigen Erkennung des Buszustands (frei, belegt) und evtl. die Durch- bzw. Weiterführung des Zugriffs (dezentrale Buszuteilung). Dazu kommen die Aufgaben der Fehlererkennung und Fehleranzeige.

Objektschicht (Object Layer):

Diese Schicht hat als Hauptaufgaben die Botschaftenverwaltung und Zustandsermittlung. Sie entscheidet, welche Botschaften momentan zu übertragen sind. Auf der Empfängerseite nimmt sie eine Botschaftenfilterung anhand der Ken-

nung im Identifikationsfeld vor, d.h. eine Entscheidung, welche Botschaften vom Knoten akzeptiert werden müssen und welche nicht.

Anwendungsschicht (CAN Application Layer CAL):

In dieser Schicht werden die zu übertragenden Daten als Botschaften bereit gestellt und mit einer Kennung versehen, die eine inhaltsbezogene Adressierung ermöglichen. Durch die Wahl der Kennung wird jede Nachricht mit einer festgelegten Priorität versehen.

Kapitel 6

Betriebssystem

In diesem Kapitel wird auf das eingesetzte Betriebssystem eingegangen. Da es sich bei dem in der Laufmaschine Lauron IVb eingesetzten Hauptprozessor respektive PC/104 um ein x86 kompatibles System handelt, liegt es nahe ein ausgereiftes und stabiles Betriebssystem zu verwenden, welches bedingt durch seine Komplexität einen großen Funktionsspektrum abdeckt. Ferner muss das Betriebssystem Systemtreiber für die eingesetzte Hardware (vgl. Kapitel 5.2.1) zur Verfügung stellen. Ein weiterer zu erfüllender Aspekt ist die Echtzeitfähigkeit (vgl. MCA Kapitel 3.8.1.5). Alle diese Anforderungen werden von dem „open source“ Betriebssystem Linux bzw. RT-Linux erfüllt. Bei der schematischen Abbildung handelt es sich um eine Erweiterung der Abbildung 6.1, um den Einsatz des Betriebssystems und des darauf ausgeführten MCA basierenden Steuerungssystems. Im folgenden wird auf das Betriebssystem Linux, den Begriff Echtzeitfähigkeit und RT-Linux eingegangen.

6.1 Echtzeitsystem

„Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Erarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.“ [2] Der Begriff Echtzeit ist jedoch ein nicht mehr klar definierter, überbenutzter Begriff, welcher je nach Anwendungsgebiet „sofort“, „schnell“ oder „in Echtzeitschritten“ bedeuten kann. Man spricht besser von „harten“ und „weichen“ Echtzeitbedingungen. Der Begriff „weiche“ Echtzeit bedeutet, dass das Erreichen der Zeitschranke (Deadline) zwar erwünscht ist, jedoch durch das Verpassen dieses Zeitlimits kein ernsthafter, direkter Schaden für das betroffene System entsteht und das korrekte Systemverhalten ebenfalls nicht gestört wird. Bei Systemen, die „harten“ Echtzeitbedingungen unterliegen, muss das Einhalten von Zeitpunkten garantiert werden. Einzelne Zeitpunkte dürfen nicht verpasst werden, da dieses Verpassen der Zeitschranke direkte, katastrophale Auswirkungen haben kann. Ein Echtzeitsystem ist ein System, das seine Funktionen und Antworten auf

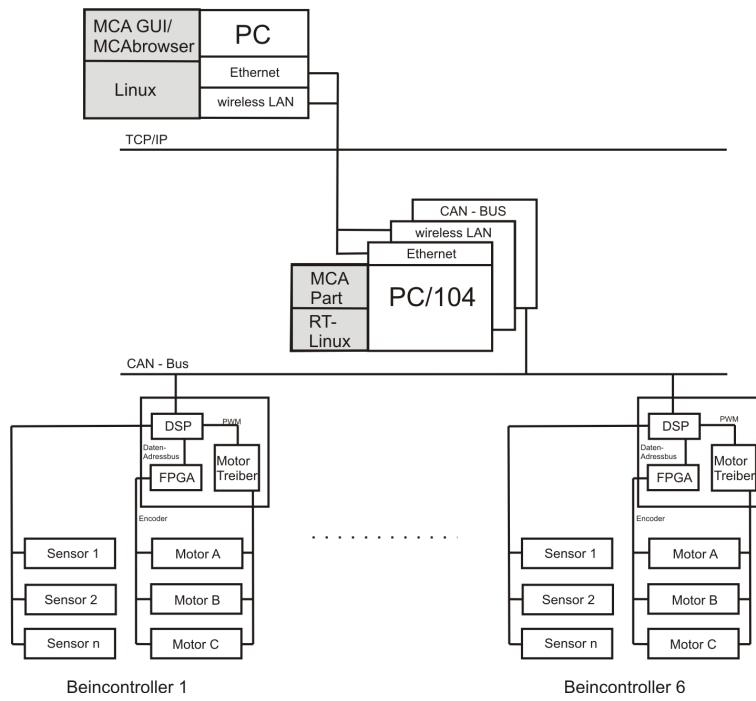


Abbildung 6.1: Kommunikation

externe, asynchrone Ereignisse innerhalb einer festgelegten Zeitspanne erledigt. Es wird von diesem System also ein determiniertes Verhalten (Pünktlichkeit, Rechtzeitigkeit) gefordert. Ein Echtzeitsystem ist kein System, das in Echtzeit arbeitet, was einen Eindruck von „Wirklichkeit“ und eines besonders schnellen Systems hervorruft. Demnach muss es nicht besonders schnell arbeiten, sondern muss in erster Linie zuverlässig zu einem definierten Zeitpunkt seine Arbeit beendet haben. Weitere Anforderungen, die an ein Echtzeitsystem gestellt werden, unterscheiden sich entsprechend ihres Anwendungszwecks sehr stark. Echtzeit-Betriebssysteme erfüllen grundsätzlich die gleichen Aufgaben wie „normale“ Betriebssysteme. Sie verwalten die Ressourcen eines Rechners, gestatten den Aufruf von Standardprogrammen, die zu diesem Betriebssystem gehören oder zusätzlich bereit gestellt werden und stellen Programmierschnittstellen zur Verfügung, mit deren Hilfe eigene Programme die Funktionen des Betriebssystems nutzen können. Besondere Anforderungen, die an ein Echtzeitsystem gestellt werden sind das deterministische Task-Management und Scheduling, die schnelle Bereitstellung von Speicher, die gesonderte Behandlung von Interrupts, die Gewährleistung der Korrektheit und der Reaktionszeiten, die unabdingbare Stabilität und Fehlertoleranz, die Vorhersagbarkeit von Entscheidungen und die Wartbarkeit.

Standard–System	Echtzeit–System
Hohe Antwortzeiten	Latenzzeiten gering (typ. 20 μ s)
Hohe Schedulingzeiten (ab 10ms)	Scheduling (Zeitplanung) im μ s Bereich (30 - 200 μ s)
Interrupts lange sperren	spezielle, schnelle Interruptbehandlung (tw. mit Speicherung)
nichtunterbrechbare Kernelaufrufe	tw. preemptive Kernel
„gerechte“ Rechenzeitaufteilung	Prioritätsklassen mit „ungerechten“ Schedulern
Optimiert für Effizienz und Durchsatz	optimiert für Determinismus
Sammeln von Systemaufträgen	Aufträge werden direkt abgearbeitet
Abschnitte werden längere Zeit gelockt	Kurzes Locking
Kein deterministisches Schedulingverfahren	Deterministisches Schedulingverfahren
Beim Scheduling wird jeder Task berücksichtigt	Solange hochpriore Tasks lauffähig sind, bleiben andere Tasks unberücksichtigt

Tabelle 6.1: Standard–System und Echtzeit–System im Vergleich

6.1.1 Linux

Heute sind mehrere Echtzeit-Betriebssysteme erhältlich. Was jedoch vermisst wird, ist ein offenes, standardisiertes, unterstützendes, effizientes und billiges Multitasking-System mit harten Echtzeitfähigkeiten. Viele UNIX Systeme erfüllen die ersten 3 Anforderungen. Linux, ein freies UNIX ähnliches Betriebssystem, beinhaltet exzellente Stabilität, Effizienz, Quellcodeverfügbarkeit, keine beschränkenden quelloffenen Lizenzen und eine umfangreiche Benutzerbasis. Des Weiteren besitzt es alle Eigenschaften eines modernen UNIX Systems: mehrere X-Windowsysteme sind implementiert, Werkzeuge für grafische Benutzerschnittstellen, Netzwerkfähigkeit, Datenbanken, Programmiersprachen, Debugger, und eine Fülle von Anwendungen. Es kann eingebettet werden und ist in der Lage relativ kleine Mengen RAM und andere Computerressourcen effizient zu nutzen. Kurz gesagt: Linux hat das Potential für eine exzellente Entwicklungsumgebung und für eine breite Fülle von Anwendungen, die beschriebenen Echtzeitssysteme mit eingeschlossen.

Linux wurde als monolithisches Betriebssystem entwickelt. Verschiedene Programme werden also kompiliert und anschließend zusammengelinkt, sodass ein Betriebssystemkern entsteht (englisch: *kernel*). Auch wenn Linux für den Benutzer als preemptives (unterbrechbares) Betriebssystem erscheint, ist es von der Kernel-Architektur her nicht preemptive, da Systemfunktionen vom Scheduler (Zeitplaner) nicht zu unterbrechen sind. Linux besteht zur Laufzeit nur aus Prozessen, d.h. Programmen, die sich in der Ausführung befinden. Zur Verwaltung

müssen weitere Dienste zur Erzeugung und Zerstörung der Prozesse zur Verfügung gestellt werden. Der Scheduler spielt somit eine zentrale Rolle. Das System ist nur schwer wartbar, da bei der kleinsten Änderung der gesamte Kernel neu kompiliert und das System neu gestartet werden müssen. Daher wurde das Konzept von Linux um eine Modulararchitektur erweitert. Sie ermöglicht, Treiber, die bisher im Kernel eingebunden waren, auszulagern, bei Bedarf im laufenden Betrieb einzubinden und jederzeit wieder zu entfernen. Der Kernel wird vollständig in einem einzigen Thread (ein separater Prozess) ausgeführt, sodass keine unerwarteten Kontextwechsel auftreten können, während der Kernel-Code läuft. Jeder Prozess arbeitet innerhalb eines ihm eigens vom Kernel zugewiesenen virtuellen Adressraums und ist geschützt vor Zugriffen anderer Prozesse.

Jedoch hat Linux mehrere Probleme, die es davon abhalten ohne Änderungen als hartes Echtzeit-Betriebssystem genutzt zu werden. Das offensichtlichste Problem ist, dass die meist nützliche Designregel für den „allgemeinen Zweck“ des Betriebssystems (den am häufigsten vorkommenden Fall) entwickelt wurde. So ist z.B. der Standard Linux-Interrupt-Handler für den „allgemeinen Zweck“ von Betriebssystemen wirklich schnell, jedoch kommt es durch häufiges Abschalten der Interrupts im schlimmsten Fall zu erheblichen Verzögerungen. Solange also Echtzeit- und Betriebssysteme für „alltägliche Zwecke“ widersprechende Designziele besitzen, ist es unvermeidlich, dass das, was im einen System clever, im anderen untauglich ist. Die Kombination beider Designziele in einem System führt zu keinem akzeptablen Ergebnis.

Es existieren jedoch noch weitere Widersprüche zu wichtigen Echtzeitvoraussetzungen. Linux benutzt eine „grobkörnige“ Zeitplanung für einige seiner Kerndatenstrukturen, dies hat zur Folge, dass Aufgaben mit Exklusivrechten sehr lange Intervalle zum Abarbeiten bekommen. Des Weiteren gibt Linux manchmal sogar den unwichtigsten Aufgaben ein Zeitstück, auch wenn eine wichtigere Aufgabe läuft. Dies ist ein großer Unterschied zu Echtzeitsystemen, bei denen man nicht annehmen kann, dass eine Aufgabe mit niedrigerer Priorität jemals Fortschritte macht.

Mit der Nicht-Unterbrechbarkeit des Kernel hängt das Problem der Synchronisation eng zusammen. Um Daten, auf die asynchron zugegriffen werden müssen, zu schützen, werden die Interrupts während kritischen Sektionen oft ausgeschaltet, da dies sehr einfach ist. Dieses Ausschalten beeinträchtigt jedoch die Systemfähigkeit zum direkten Antworten auf Ereignisse. Die meisten UNIX Systeme benutzen einen virtuellen Speicher, der es möglich macht, Programme, die mit ihrer Größe den zur Verfügung stehenden RAM überschreiten, laufen zu lassen, indem nur der arbeitende Teil des Programms im Speicher behalten wird. Ist der real vorhandene Arbeitsspeicher zu klein um alle angeforderten Speicherseiten zu speichern, so kann der Kernel Speicherseiten, die lange nicht benutzt wurden auf ein anderes Medium - meist die Festplatte - auslagern. Für Echtzeitsysteme führt die Auslagerung auf den virtuellen Speicher (Festplatte) jedoch zu einem nicht tolerierbaren Grad von Unvorhersagbarkeit. Linux ordnet Anfragen von Aufgaben neu und „stapelt“ Operationen, um das Benutzen der Hardware effizienter zu gestalten. Es lässt Aufgaben mit hoher Priorität auf Aufgaben mit niedrigen Prioritäten warten, um Ressourcen zu befreien. Dies ist für Echtzeitsysteme jedoch unakzeptabel. Aufgrund dieser Faktoren ist das traditionelle UNIX-System Linux ohne tief greifende Änderungen, nur schwer für Echtzeitverarbeitung geeignet.

Linux ist wie die meisten anderen Betriebssysteme entworfen, um die durch-

schnittliche Leistung des Systems zu erhöhen und somit nicht für die Erfüllung von harten Echtzeitbedingungen geeignet.

6.1.2 Real Time Linux

RT-Linux [14][15] ist eine Variation der Basisidee. Man hat versucht, Linux soweit umzubauen, dass es auch harten Echtzeitbedingungen genügt, also Zeit-Deadlines einzubauen, welche nicht überschritten werden können. Während Betriebssysteme wie UNIX danach streben, gute durchschnittliche Leistung bereitzustellen, ist für Echtzeit-Betriebssysteme das korrekte Timing das Hauptziel. Durchsatz ist dabei zweitrangig. Der Linux-Kernel wurde jedoch nicht total verworfen, da man sonst völlig von den Entwicklungen bei Standard Linux-Kernel abgeschnitten wäre. Aus diesen Gründen agiert Linux nur als Gastsystem. Dadurch wird Linux zu einem Task mit niedrigster Priorität. Er läuft nur dann, falls kein Echtzeit-Task läuft, und wird zudem immer unterbrochen (preemptive), falls ein Real-Time Task den Prozessor benötigt. Der nicht echtzeitfähige Linux-Kernel ist somit der Idle-Task des darunter liegenden Echtzeit-Betriebssystems, bzw. RTLinux ist ein kleines, in Echtzeit laufendes, ausführbares Programm, das unter dem Betriebssystem Linux als vollständig vorgelagerter Prozess läuft.

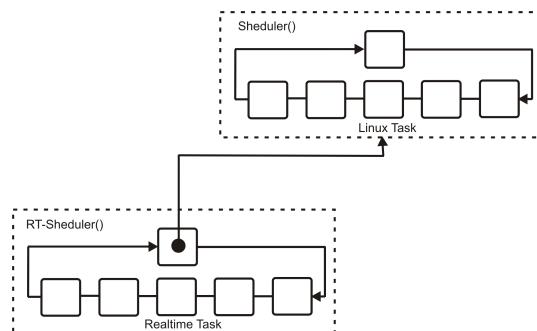


Abbildung 6.2: Echtzeit-Kernel mit Linux-Kernel als Idle-Task [41]

Parallel können weitere RT-Tasks laufen, die von einem eigenen Scheduler verwaltet werden. Das Betriebssystem Linux läuft nur noch im Hintergrund und bearbeitet die Applikationen, denen weiche Echtzeitbedingungen genügen, wohingegen RT-Linux die Anteile übernimmt, die auf harte Echtzeit setzen. Somit müssen aber sämtliche Applikationen explizit für dieses System geschrieben werden.

Das System wurde so gestaltet, dass Linux selbst nur wenig modifiziert werden musste. Realisiert wird RT-Linux über das Linux Modulkonzept. Um RT-Linux zu benutzen lädt man Module, die, was auch immer RT-Linux für Fähigkeiten benötigt, diese implementieren. Wenn die Dienste dieser Module nicht die Bedingungen der Anwendung erfüllen, können sie durch andere Module ersetzt werden. Diese Module sind:

- Scheduling Modul.
- Kommunikations-Modul (FIFO, Shared Memory).
- Modul mit Synchronisations- und Interprozesskommunikationsprimitiven (optional).
- Ende-zu-Ende Quittungen.
- Modul, durch das ausgewählte Parameter des Echtzeitsystems aus dem Gast-Betriebssystem (Linux-proc-Schnittstelle) heraus verändert werden können (optional).

Eine Ausnahme ist das RT-Linux Kernel-Modul, da es nicht zur Laufzeit wie ein Modul, sondern als Patch in den Quellcode des Kernels eingebracht wird. Das verlangt ein anschließendes Kompilieren des Kernels. Ansonsten kann RT-Linux zur Laufzeit durch das Modulkonzept in beliebiger Weise skaliert werden. Solange keine Echtzeitmodule geladen sind, verhält sich Linux wie gewöhnlich und es ist mit keinerlei Einbußen bzgl. Stabilität oder Leistungsfähigkeit zu rechnen. Module sind Teile des Betriebssystems, die während der Laufzeit eingebunden werden. Nach diesem Konzept funktioniert der Linux-Kernel. Linux wird also durch RT-Linux kontrolliert. Beide Kerne werden in der jeweils nächsten Schale durch Module erweitert. Es existieren zwei Arten von dynamisch ladbaren Modulen. Die Kernel-Module des Gastbetriebssystems Linux und die RT-Linux-Module. Die Linux-Module interagieren im Gegensatz zu den RT-Linux-Modulen nicht mit dem RT-Linux-Kern. In der äußersten Schale befinden sich die Userprozesse, durch sie werden die nicht zeitkritischen Komponenten des Gesamtsystems realisiert.

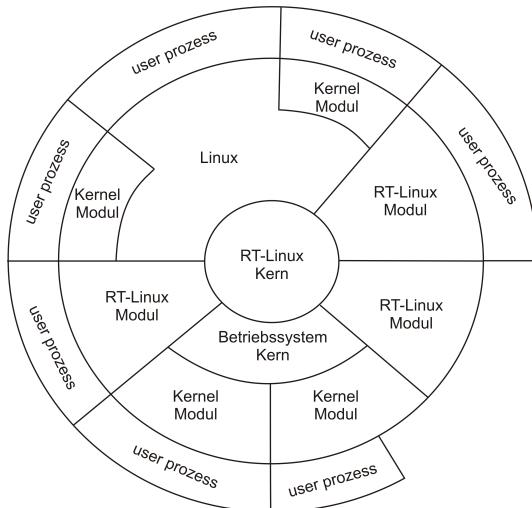


Abbildung 6.3: RT-Linux Schalenmodell [41]

6.1.3 RTLinux und RTAI

Für besonderes hohe Zeitanforderungen stehen die beiden Real-Time Erweiterungen RT-Linux und RTAI zur Verfügung, welche die gleiche Architektur aufweisen. Die am DIAPM [1] entwickelte RTAI (Real Time Application Interface) [16] ist ein Split-off des am NMT [10] initiierten RT-Linux (Real-Time-Linux) [5]. Beide Varianten sind beinahe vollständig eigenständige Echtzeitbetriebssysteme, mit dem Linux-Kernel als Idle-Task. Im folgenden werden die beiden Real-Time Erweiterungen RTLinux und RTAI einander gegenübergestellt [50].

Lizenzen

RTLinux:

RTLinux unterliegt zwei verschiedenen Lizenzen (Open RTLinux Patent License und Non-free License).

1. Open RTLinux Patent License: Jede Modifikation des Softwarecodes muss GPL (GNU General Public License) sein, d.h. der Code muss offen im Web verfügbar sein. Die GPL fordert, dass der Anwender Änderungen an RTLinux selbst wieder unter der GPL freigeben muss.
2. Non-free License: Diese Lizenz ist für Anwender gedacht, die RTLinux kommerziell betreiben möchten. Wenn dies der Fall ist muss von FSM Labs diese kostenpflichtige Lizenz erworben werden.

RTAI:

RTAI unterliegt lediglich der LGPL (GNU Lesser General Public License). Diese Lizenz besagt, dass Benutzer den Code verändern können und auch in kommerzieller Hinsicht keine Lizenz erwerben müssen.

API-Kompatibilität

RTLinux:

RTLinux ist als ein POSIX 1003.13 konformes minimales Echtzeitbetriebssystem implementiert.

RTAI:

RTAI besitzt eine eigene API, die von der RTLinux Version 1 API abgeleitet ist. RTAI versucht die Kompatibilität zur RTLinux Version 1 API zu erhalten. Zahlreiche neue Funktionen (message queues, mailboxes, etc.) sind zueinander inkompatibel, so dass man sich für eine Variante jeweils immer entscheiden muss. Es existiert ein POSIX-Modul, das POSIX 1003.1c und 1003.1b-Konformität bietet.

Speichermanagement

RTLinux:

Es wird keine dynamische Speicherzuweisung unterstützt. Es muss also explizit vor Ausführung des Realtime-Threads der Speicher reserviert werden. Durch das nicht-posix-konforme MBUFF-Modul wird shared memory unterstützt, so dass sich verschiedene RT-Threads den Speicher teilen können.

RTAI:

Bei RTAI wird sowohl die dynamische Speicherzuweisung, als auch shared memory unterstützt. Diese Funktionalität ist nicht posix-konform. Zusätzlich zum MBUFF-Modul, wie bei RTLinux, wird auch ein eigenes SHMEM-Modul von RTAI zur Unterstützung des shared memory angeboten. Beide Module funktionieren jedoch nach einem ähnlichen Prinzip.

Interprozesskommunikation

Als Interprozesskommunikation wird die Kommunikation zwischen User-Space und Realtime-Threads bezeichnet. Auch die Kommunikation der Realtime-Threads untereinander gehört dazu. Um diese Kommunikation zu ermöglichen, werden verschiedene Verfahren von RTLinux und RTAI bereitgestellt.

RTLinux:

Bei RTLinux erfolgt die Kommunikation der Tasks untereinander hauptsächlich mit Hilfe von FIFOs. Dabei werden Unix- Pipes als Kommunikationsmechanismus verwendet. Zusätzlich bietet RTLinux noch Message-Queues, die jedoch durch eine veraltete Jerry Epplin IPCImplementierung realisiert sind.

RTAI:

Bei RTAI gibt es eine Vielzahl von Mechanismen, die benutzt werden können, um mit anderen Prozessen zu kommunizieren. Sie sind jedoch untereinander inkompatibel, so dass man sich für eine Variante entscheiden muss. Zum einen gibt es FIFOs deren Funktionalität denen der RTLinux entsprechen. Zusätzlich existiert ein flexibles Mailboxsystem mit verschiedenen Sende- und Empfangsfunktionen. Sender und Empfänger können verschieden große Nachrichten in dieselbe Mailbox schreiben und lesen. Message-Queues sind in RTAI in vier verschiedenen Ausführungen vorhanden, die unterschiedlich flexibel sind.

Synchronisierung

RTLinux:

Es werden POSIX-thread-mutex-Variablen verwendet. Außerdem wird ein PRIORITY_PROTECT-Protokoll angeboten, um das Priority-Inversion-Problem zu behandeln. Die Zustandsvariablen und die Semaphoren sind jeweils auch POSIX-konform.

RTAI:

RTAI benutzt genau wie RTLinux POSIX thread mutex Variablen und ebenso ein PRIORITY_INHERIT-Protokoll, um das Priority-Inversion-Problem zu behandeln. Zusätzlich zu den POSIX-konformen Zustandsvariablen und Semaphoren gibt es spezielle RTAI-Semaphoren.

User-Space Realtime

User-Space Realtime bedeutet die Kompilierung und Ausführung hart-echtzeitfähiger Programmteile im User-Space. Normalerweise ist es erforderlich hart-echtzeitfähige Programmteile explizit auszulagern, um sie im Kernel als Modul einzubinden. Dieses würde bei einem Echtzeitbetriebssystem, das User-Space Realtime-Unterstützung bietet, entfallen. Die Vorteile einer solchen Unterstützung sind wie folgt: Ein abgestürzter Task würde aufgrund eines Programmierfehlers nichts an der Systemstabilität ändern. Die Echtzeitapplikation wird konform eines normalen Unix-Prozesses programmiert und gestartet. Ein Nachteil ist jedoch die erhöhte Interruptlatenz, die durch den Umweg über den User-Space entsteht und folglich der hart-echtzeitfähige Teil nicht direkt im Kernel ausgeführt wird.

RTLinux:

RTLinux stellt die Möglichkeit der Verwendung von User-Space Realtime Signalen zur Verfügung. Hardware-Interrupts werden dabei von user-space signal handlers behandelt. Jedoch ist es nicht möglich, einen Linuxsystemaufruf oder RTLinux-Services von diesen Handlern aufzurufen.

RTAI:

RTAI unterstützt volles User-Space Realtime und stellt für diese Unterstützung ein Modul namens LXRT bereit. Es gibt drei verschiedene Implementierungen (LXRT-Services, Extended LXRT und Mini-LXRT).

Bei der Laufmaschine Lauron IVb kommt die Real-Time Erweiterung RTAI zum Einsatz. Nicht zuletzt wegen der vereinfachten Lizenzbestimmung gegenüber RTLinux.

Kapitel 7

Analysye der vorhandenen Steuerung

Für die Integration kraftgeführter Bewegungen in das Steuerungssystem ist es erforderlich, die vorhandene Steuerung zu analysieren. Somit wird im folgenden auf den Aufbau des vorhandenen Steuerungssystems der Laufmaschine Laaron IVb eingegangen. Dem Steuerungskonzept liegt die Theorie des natürlichen Laufvorganges einer Stabheuschrecke zugrunde (vgl. Kapitel 3.1.1). Prinzipiell ist die Steuerung in zwei Teile gegliedert:

1. Verhaltenssteuerung auf Basis des MCA-Systems, ausgeführt auf dem PC/104.
2. Beincontrollersteuerung, zuständig für die Sensordatenerfassung und Aktorensteuerung, ausgeführt auf jedem der sechs Beincontroller.

7.1 MCA-Steuerung

Im Folgenden wird der Aufbau der auf MCA basierenden Steuerung beschrieben. Die Aufgaben der MCA-Steuerung sind hauptsächlich bestimmt durch Transformationen, Filterungen von Sensorsignalen, kinematische Berechnungen sowie planerische Aufgaben, etwa der Generierung von Laufzyklen.

7.1.1 MCA-Part

Der MCA-Part bildet den Container der einzelnen Gruppen und ist somit die höchste Abstraktionsebene der Steuerung. Die *MCAGUI* verbindet sich über TCP/IP mit dem MCA-Part unter dem Namen „virtual MainGroup“ tritt. Die „virtual MainGroup“ beinhaltet nur zwei Schnittstellen, *Sensor output* und *Control input*. Alle Steuerungsinformationen der *MCAGUI* sind über das TCP/IP Protokoll direkt mit dem *Control input* der „virtual MainGroup“ verbunden.

Dem entsprechend werden ebenfalls alle relevanten Sensorinformationen der „virtual MainGroup“ über den *Sensor output* der *MCAGUI* zur Verfügung gestellt. Die „virtual MainGroup“ ist in drei Gruppen unterteilt:

- Ebene 1: „BehaviourGroup“.
- Ebene 2: „SignalProcessingGroup“.
- Ebene 3: „DSP–Remote-Part“.

Abbildung 7.1 verdeutlicht diese Konstellation. Aus der Abbildung ist weiterhin ersichtlich, dass die *Control input* Daten der *MCAGUI* über eine Verkantung mit dem *Control input* der „BehaviourGroup“ in Verbindung steht; der *Control output* der „BehaviourGroup“ mit dem *Control input* der „SignalProcessingGroup“ verbunden ist; letztlich der *Control output* der „SignalProcessingGroup“ mit dem *Control input* des „DSP–Remote-Part“ verkantet ist. Auf der Sensor Seite findet man das Gleiche Verkantungsmodell vor. Nur dass dieses von unten beginnend jeweils *Sensor output* mit *Sensor input* verkantet. Letzlich geht *Sensor output* der Ebene 1 der „BehaviourGroup“ eine Verbindung mit dem *Sensor output* der „virtual MainGroup“ ein.

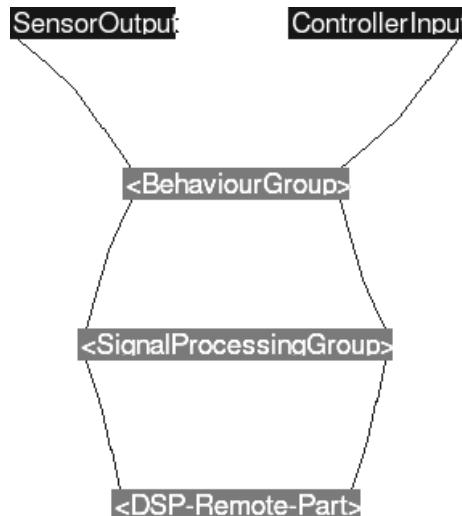


Abbildung 7.1: „Virtual MainGroup“ der MCA–Steuerung

7.1.2 Group–“BehaviorGroup“

Die „BehaviourGroup“ stellt das Herz der Steuerung dar. Wie der Name schon sagt, wird hier das Verhalten der Laufmaschine geplant bzw. berechnet (engl.

Behaviour = deutsch *Verhalten*). Die im Rahmen dieser Arbeit entstandenen Erweiterungen der Steuerung sind ebenfalls in dieser Gruppe eingegliedert worden. Um die Struktur und die Interaktion der Modulgruppe „BehaviourGroup“ besser zu verstehen, werden im folgenden die Aufgaben bzw. Funktionsweisen der wichtigsten Module bzw. Modulgruppen vorgestellt. Abbildung 7.2 zeigt den Aufbau der „BehaviourGroup“.

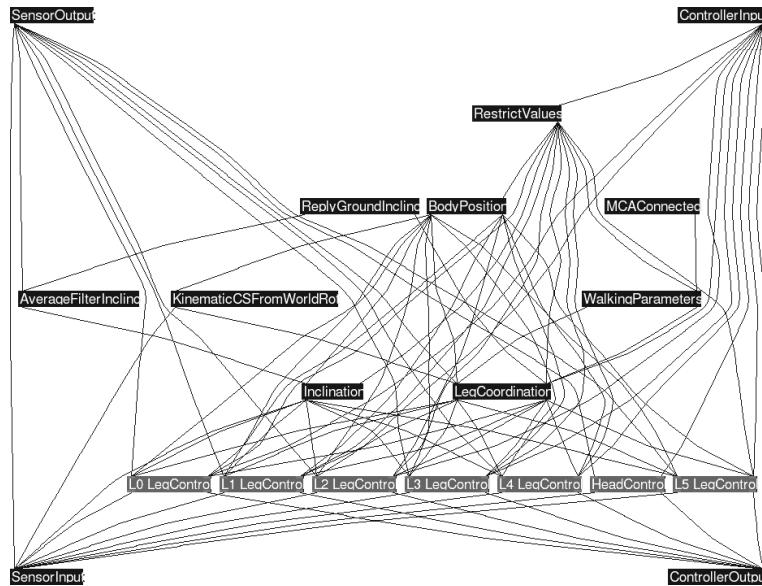


Abbildung 7.2: „BehaviorGroup“ der MCA–Steuerung

RestrictValues:

Dieses Modul führt eine Begrenzung von Eingabewerten auf bestimmte MIN MAX Werte durch. Wird über die *MCAGUI* ein Wert eingestellt, der ausserhalb eines vorher festgelegten Bereichs liegt, wird der eingestellte Wert auf das entsprechende MIN bzw. MAX gesetzt. Somit wird eine Fehleingabe von Seiten des Benutzers unterbunden, um mögliche Schäden an der Laufmaschine zu verhindern.

BodyPosition:

In diesem Modul wird die Haltungskontrolle durchgeführt.

WalkingParameters:

Dieses Modul berechnet aus der Gangart und dem Belastungsfaktor die Soll-Phasenverschiebung.

LegCoordination:

Dieses Modul führt die Beinkoordination durch.

Inclination:

Dieses Modul schätzt die aktuelle Untergrundneigung auf Basis der Ist-Neigung und des Ist-Fußpunkts ab. Je nach Neigungsmodus und Ist- Bzw. Soll-Neigungswinkel wird der Neigungswinkelkorrekturvektor für das „BodyPosition“ Modul berechnet.

KinematicCSFromWorldRot:

Dieses Modul führt eine Koordinatentransformation von einem Ausgangs- in ein Zielkoordinatensystem durch. Als Ausgangs- und Zielkoordinatensystem kommt das *Weltkoordinatensystem*, *Symmetriekoordinatensystem*, *Beinkoordinatensystem* oder *Beinbezugskoordinatensystem* in Frage.

LegControl:

Diese Modulgruppe repräsentiert jeweils die Steuerung eines einzelnen Beins und liegt somit in sechsfacher Ausführung vor. In dieser Modulgruppe wird vor allem die Einzelbeintrajektorie geplant und die Beintrajektorie umgesetzt. Abbildung 7.3 zeigt die Modulgruppe mit Untermodulen. Auf die einzelnen Untermodule wird nicht genauer eingegangen, da sie für den Inhalt dieser Arbeit nicht von essentieller Bedeutung sind. Die in dieser Arbeit entstandenen Steuerungserweiterungen werden teilweise mit dem *Control input* bzw. *Sensor output* der einzelnen „LegControl“ Modulgruppen verbunden. Wird diesbezüglich eine genauere Erläuterung notwendig, wird dies im entsprechenden Kapitel vorgenommen.

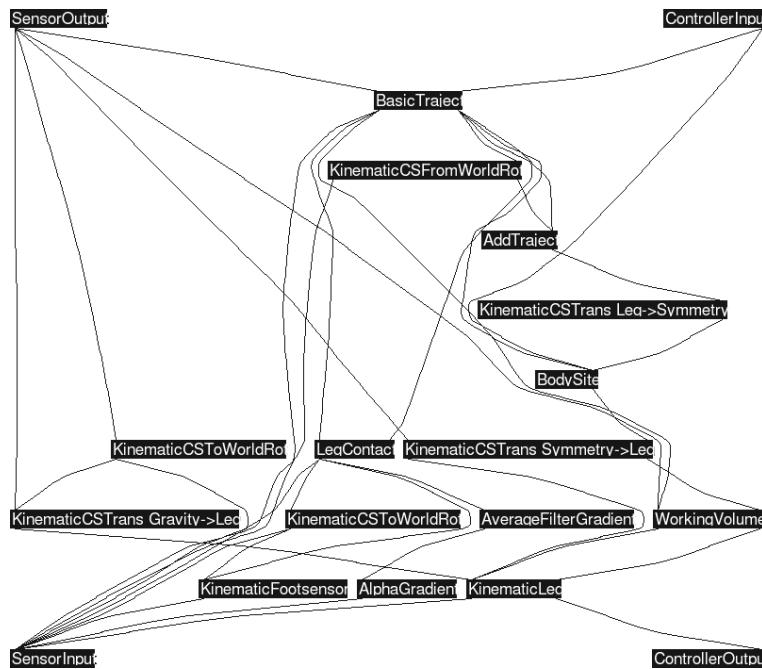


Abbildung 7.3: „LegControl“ der MCA-Steuerung

7.1.3 Group—“SignalProcessingGroup“

Diese Modulgruppe führt die Signalverarbeitung aus. Die gemessenen Fußkräfte, Motorströme und Neigungswinkel werden zuerst mit Hilfe von Median- und Averagefiltern geglättet und anschließend auf Basis der Kalibrierungsdaten in die Einheiten Newton, Milliampere und Winkelradian umgerechnet. Weiterhin findet die Umsetzung von Winkelgeberimpulsgrößen zu Gelenkwinkeln und um-

gekehrt statt. Somit wird eine gewisse Unabhängigkeit der höheren Ebenen von der speziellen Ausprägung der eingesetzten Sensoren erreicht. Auf die einzelnen Module der Modulgruppe „SignalProcessingGroup“ wird auch hier nicht eingegangen, da sie für diese Arbeit nicht von Bedeutung sind. Abbildung 7.4 zeigt den Aufbau der „SignalProcessingGroup“.

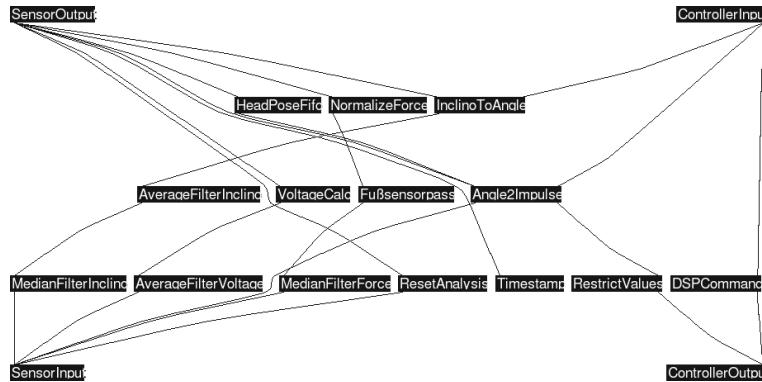


Abbildung 7.4: „SignalProcessingGroup“ der MCA–Steuerung

7.1.4 Group–“DSP–Remote–Part“

Diese Module verbinden sich direkt mit den Mikrocontrollern der Laufmaschine Lauron IVb und stellen somit die unterste Ebene des MCA–Steuerung dar. Ferner bilden diese Module die Schnittstelle des *Hardware Abstraction Layer* (HAL) (vgl. Kapitel 5.1.2). Die Module beinhalten die Soll-Winkel und Reset-Befehle für die zu steuernden Motoren sowie die Ist-Winkel, Motorenströme, Reset-Antworten der Motoren, Sensorsignale der Fußsensoren bzw. des Neigungssensors. Abbildung 7.5 zeigt diese Modulgruppe mit ihren Modulen.

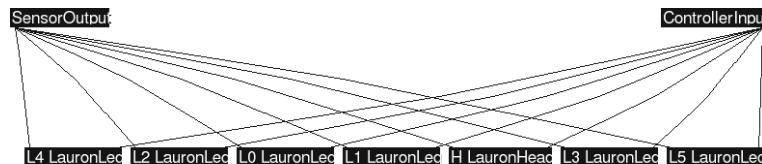


Abbildung 7.5: „DSP-Remote-Part“ der MCA–Steuerung

7.2 Beincontroller

Die Beincontrollersoftware wurde mit dem Codewarrior ProgrammierTool (siehe Kapitel 3.9) entwickelt und auf jeden Beincontroller geladen. Die Software bildet den *Hardware Abstraction Layer* (HAL) (vgl. Kapitel 5.1.2) auf der Controller Seite. Die Controllersoftware geht eine Verbindung mit dem „DSP-Remote-Part“ und dem entsprechenden Controller ein. Die auf den Controllern ausgeführte Software nimmt die Soll-Winkel und Reset-Befehle für die zu steuernden Motoren zur Verarbeitung an. Um die Soll-Winkel mit Hilfe der Motoren anzufahren, wird ein PID-Regler eingesetzt welcher ein PWM Signal erzeugt. Die Controllersoftware gibt die Ist-Winkel der Beine, sowie die Reset-Antworten an den „DSP-Remote-Part“ (siehe Kapitel 7.1.4) weiter. Ferner werden die Sensordaten des an den Controller angeschlossenen Fußsensors an den „DSP-Remote-Part“ zur weiteren Verarbeitung weitergegeben. Die Controller kommunizieren mit dem „DSP-Remote-Part“ über den CAN-Bus. Im Folgenden wird auf den implementierten digitalen PID-Regler und auf die Funktionsweise der Motorenansteuerung unter Einsatz eines PWM-Signals eingegangen.

7.2.1 Puls–Weiten–Modulation (PWM) Theorie

Zur Ansteuerung der Motoren wird eine Puls–Weiten–Modulation verwendet. Bei der Puls–Weiten–Modulation bleibt die Periodendauer der aufeinander folgenden Pulse n_{PWM} konstant. Um eine Änderung des Signals zu erhalten wird die Pulsweite n_{ON} verändert. Für die Ansteuerung des Motors, werden die Schaltsignale der PWM direkt vom Mikrocontroller erzeugt. Die Pulsweite n_{ON} wird durch den digitalen PID–Regler (vgl. Kapitel 7.2.2) berechnet. Das Verhalten vom Motor ist abhängig von der Periodendauer sowie vom Tastverhältnis. Die PWM wird erzeugt, indem ein Aussgangssignal periodisch eingeschaltet und nach Verstreichen einer bestimmten Dauer wieder ausgeschaltet wird. Maßgeblich ist hierbei die Periodendauer n_{PWM} und die Einschaltzeit n_{ON} . Der Anteil der Einschaltzeit an der Gesamtzeit wird als Tastverhältnis (X_{PWM}) bezeichnet. Abbildung 7.6a zeigt ein Zeitdiagramm zur PWM–Erzeugung.

$$X_{PWM} = \frac{n_{ON}}{n_{PWM}} \quad (7.1)$$

Die Periodendauer ergibt sich aus:

$$n_{PWM} = n_{ON} + n_{OFF} \quad (7.2)$$

Der Mittelwert der erzeugten Spannung wird mit

$$U_m = U_{OFF} + (U_{ON} - U_{OFF}) \cdot \frac{n_{ON}}{n_{PWM}} \quad (7.3)$$

berechnet. Diesen Zusammenhang verdeutlicht Abbildung 7.6b.

Bei der Laufmaschine Lauron IVb werden folgende Werte für n_{ON} , n_{OFF} , n_{PWM} , U_{ON} und U_{OFF} verwendet:

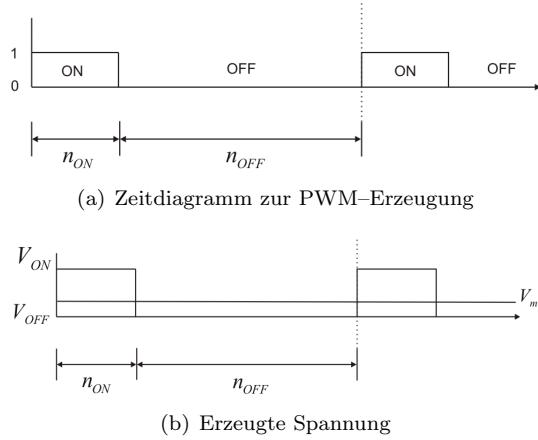


Abbildung 7.6: PWM-Erzeugung

- n_{ON} wird durch den digitalen PID-Regler erzeugt und liegt zwischen 0 und 32000.
- $n_{PWM} = \text{Anzahl der maximalen Impulse pro Zyklus.}$
- $n_{OFF} = n_{PWM} - n_{ON}$
- $U_{ON} = 24 \text{ Volt.}$
- $U_{OFF} = 0 \text{ Volt.}$

Abbildung 7.7 zeigt eine fiktive PWM-Drehzahlkennlinie. Diese zeigt das Verhältnis zwischen Tastverhältnis und Winkelgeschwindigkeit bzw. Drehzahl eines Motors.

Wie in der Abbildung zu sehen, handelt es sich nicht um ein lineares Verhalten der Funktion. Durch die PWM Ansteuerung des Gelenkantriebes treten Induktionskräfte der Ankerwicklung des Motors auf. Nach Anlegen eines Pulses muss in der Ankerwicklung erst ein Magnetfeld aufgebaut werden, bevor es zu einem Antriebsmoment kommen kann. Es ist zu erkennen, dass bei kurzen Impulsen keine Drehzahl aufgebaut wird. Dieses Verhalten ist Frequenzabhängig, d.h. bei niedrigen Frequenzen ist die „Totzone“ geringer als bei größer werdenden Frequenzen. Ferner ist zu erkennen, dass nach der „Totzone“ zwischen Winkelgeschwindigkeit und Tastverhältnis eine direkt-lineare Beziehung besteht.

7.2.2 Digitaler PID-Stellungsalgorithmus

In die Mikrocontroller der Laufmaschine Lauron IVb ist ein digitaler PID-Regler implementiert. Dieser PID-Regler erzeugt eine PWM Einschaltzeit n_{ON} für jeden der drei Motoren pro Bein. Diese Einschaltzeit wird zum Ansteuern des PWM Controllermoduls eingesetzt. Die Motorenbewegung erfolgt über das PWM Signal, welches vom Controller an die entsprechenden Motoren geleitet wird. Der

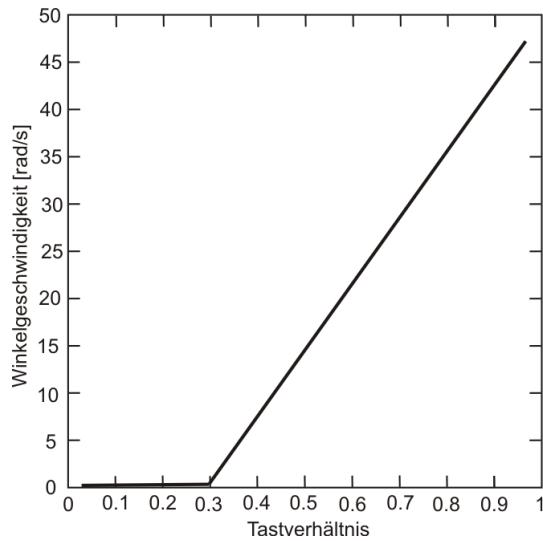


Abbildung 7.7: PWM–Drehzahl–Kennlinie

Proportionalfaktor (P–Faktor), Integrationsfaktor (I–Faktor) und Differentiationsfaktor (D–Faktor) lassen sich für jeden der drei Motoren eines Beines mit Hilfe des *MCAbrowsers* über die Modulparameterschnittstelle setzen. Die entsprechenden sechs Beinmodule „Li LauronLeg“ (für $i=0\dots 5$) sind in der MCA–Gruppe „DSP–Remote–Part“ (siehe Kapitel 7.1.4) zu finden. Im Folgenden wird auf die Theorie der Regelungstechnik und auf den implementierten digitalen PID–Regler eingegangen. Abbildung 7.8 zeigt den digitalen Regelungskreis.

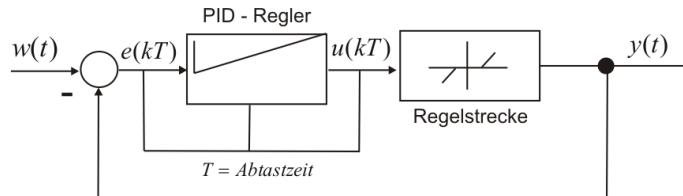


Abbildung 7.8: Digitaler PID–Gelenkregelkreis

Im Folgenden werden die in Abbildung 7.8 verwendeten Bezeichnungen vorgestellt:

- $w(t)$ Der Gelenkwinkelsollwert.
- $y(t)$ Der Gelenkwinkelwert gemessen mit Hilfe der Gelenkwinkelpositionssensoren (vgl. Kapitel 4.1.2.3).
- $e(kT)$ Die Gelenkwinkelabweichung zwischen Soll– und Istwert ($e(t) = w(t) - y(t)$).
- $u(kT)$ Das PWM n_{ON} Motorenstellsignal.

- Die Regelstrecke stellt die PWM–Drehzahl–Kennlinie dar.

Die Übertragungsfunktion eines kontinuierlichen PID–Reglers (Gleichung (3.47)) kann mit folgenden Näherungen, die für kleine Abtastzeiten T gelten, diskretisiert werden [19].

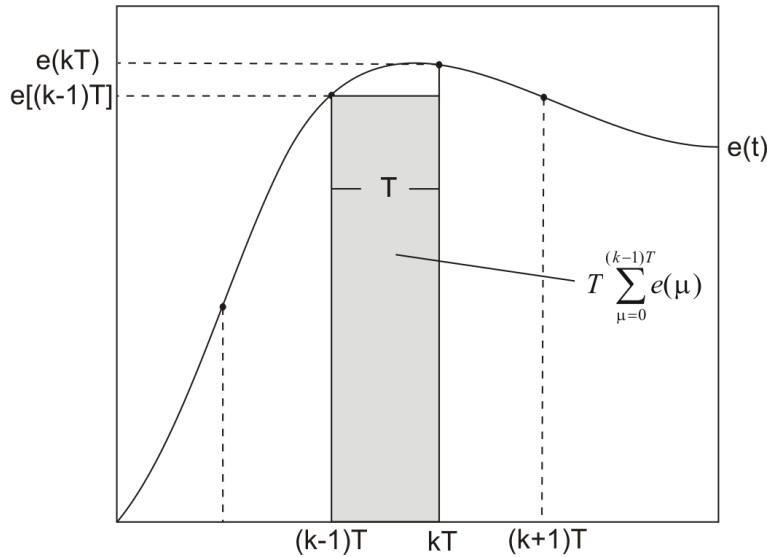


Abbildung 7.9: Integration durch Bildung der Rechtecksumme

Das Integral wird durch die Rechtecksumme (Abbildung 7.9)

$$\int_0^t e(\tau) d\tau \approx \sum_{\mu=0}^{(k-1)T} e(\mu) \quad (7.4)$$

und der Differentialquotient wird durch den *Rückwärtsdifferenzenquotienten*

$$\frac{d}{dt} [e(t)] \approx \frac{e(kT) - e[(k-1)T]}{T} \quad (7.5)$$

angenähert. Die Differenzengleichung des PID–Reglers lautet mit den Näherungen:

$$u(kT) = K_P \left[e(kT) + \frac{T}{T_N} \sum_{\mu=0}^{(k-1)T} + \frac{T_V}{T} \{e(kT) - e[(k-1)T]\} \right] \quad (7.6)$$

Diese Gleichung ist für eine Realisierung im Rechner ungeeignet, da es sich hier nicht um einen rekursiven Algorithmus handelt. Deshalb wird erneut die Differenzengleichung für

$$u[(k-1)T] = K_P \left[e[(k-1)T] + \frac{T}{T_N} \sum_{\mu=0}^{(k-2)T} + \frac{T_V}{T} \{e[(k-1)T] - e[(k-2)T]\} \right] \quad (7.7)$$

gebildet. Durch die Differenzbildung $u(kT) - u[(k-1)T]$ erhält man einen rekursiven Algorithmus den sogenannten PID-Stellungsalgorithmus (7.8).

$$u(kT) = u[(k-1)T] + d_0 e(kT) + d_1 e[(k-1)T] + d_2 e[(k-2)T] \quad (7.8)$$

mit

$$d_0 = K_P + \frac{K_D}{T}; \quad d_1 = - \left[K_P + 2 \frac{K_D}{T} - T K_I \right]; \quad d_2 = \frac{K_D}{T} \quad (7.9)$$

Der Regelungsalgorithmus (7.8) wird in den Mikrocontroller zur rekursiven Berechnung der Stellgröße $u(kT)$ für die gewählte Abtastzeit T implementiert. Im Mikrocontroller müssen dann lediglich die zurückliegenden Regeldifferenzen $e[(k-1)T]$ und $e[(k-2)T]$ sowie die Stellgröße $u[(k-1)T]$ abgespeichert werden. Die Parameter K_P , K_D und K_I können über den *MCA browser* verändert werden.

Kapitel 8

Lösungskonzept

Ausgehend von der Vorstellung der „active Compliance“, „damping control“ und der „Kraftverteilungen“ werden im Nachfolgenden die Steuerungsalgorithmen für den Laufroboter Lauron IVb, auf Basis des MCA–Systems und der Anpassungen der Mikrocontrollersoftware vorgestellt.

8.1 Zusammenstellung der Anforderungen

Für die Erweiterung der vorhandenen MCA–Steuerung müssen folgende planerischen Vorüberlegungen angestellt werden:

- Integration der Erweiterungen in die vorhandene Steuerung, so dass bestehende Funktionalität nicht beeinträchtigt wird.
- Entwicklung einer geeigneten Modul– und Gruppen–Struktur im Sinne des MCA–Systems, zur Realisierung der Steuerungserweiterungen.
- Finden von geeigneten Positionierungen in die vorhandenen primären Modulgruppen („BehaviourGroup“, „SignalProcessingGroup“, „DSP–Remote–Part“) zur Integration der Erweiterungen.
- Umlegung und Neuerstellung von Kanten, so dass Messgrößen und Stellgrößen an den entsprechenden neuen Modulgruppen zur Verfügung stehen.

Unter dem Gesichtspunkt der aufgeführten Vorüberlegungen wird im folgenden gemäß des Top–Down–Entwurfs die Steuerungserweiterungen entsprechend des MCA Konzepts entwickelt.

8.2 Top–Down–Entwurf

Entsprechend der Aufgabenstellung werden zwei Erweiterungen in das Steuerungssystem aufgenommen. Zum einen wird die „active Compliance“ zur Mini-

mierung von Querkräften umgesetzt. Zum Anderen findet eine technische Bohr-operation unter Verwendung des „damping control“–Reglers Einzug in die Steue-rung. Beide Erweiterungen unterliegen dem Konzept kraftgeführter Bewegungen (vgl. Kapitel 3.5.1). Aus der Sicht des Top–Down–Entwurfs führen bei beiden Erweiterungen Kraftinformationen der Fußsensoren zu einer durch den entspre-chenden Kraftregler bestimmten Roboterbewegung.

Der Top–Down–Ansatz ist für die Entwicklung des hierarischen Steuerungssys-tems entsprechend der Modular Controller Architecture (MCA) (vgl. Kapitel 3.8) vorteilhaft. Bis grundlegende Basisfunktionen vorliegen, werden ausgehend von der abstrakten zu lösenden Aufgabe schrittweise Funktionsmodule erzeugt.

8.2.1 Implementierung kraftgeführter Bewegungen

Durch die Beine werden kinematische Ketten geschlossen [38]. Dies bedeutet, falls ein Gelenk eine Bewegung ausführt, müssen auch andere Gelenke synchron mit bewegt werden. Dadurch treten in der Praxis immer Fehler auf. Bei me-chanischen, steifen Strukturen treten durch kleine Abweichungen bereits große Kräfte auf, die in die mechanische Struktur einwirken und Energie binden. In Kapitel 3.5.1 wurde eine Methode vorgestellt, mit der sich solche auftretenden Querkräfte zwischen den Beinen minimieren lassen. Für die Implementierung auf dem Laufroboter Lauron IVb wurde die „active Compliance“ ausgewählt. Diese Methode arbeitet mit einer kinematischen Steuerung. Die Informationen über die gemessene Kraftwirkung werden zur Modifikation des Sollwertes der Bein-position herangezogen. Die Richtung der Bewegung entsteht bei einem Bein erst durch das Zusammenwirken aller drei aktiven Freiheitsgrade. Die resultierende Reaktionskraft kann eine beliebige räumliche Richtung besitzen. Die räumliche Orientierung wird mittels der im Bein angebrachten Fußkraftsensoren gemessen und ist abhängig von der Position des Beins. Um mit diesen Größen arbei-ten zu können, müssen sie auf einer gemeinsamen Koordinatenbasis abgebildet werden. Hierzu wird das *Untergrundkoordinatensystem* bzw. das *Symmetrieko-ordinatensystem* verwendet. Die Sollposition der Beine liegt im *Symmetrieko-ordinatensystem* vor. Die wirkenden Kräfte liegen bezüglich der Orientierung im *Untergrundkoordinatensystem* vor. Da jedoch die „active Compliance“ nur für den Fall angenommen wird, dass sich der Roboterkörper parallel zur Ebene befindet, welche senkrecht zum Gravitationsvektor steht, weisen *Untergrundko-ordinatensystem* und *Symmetriekoordinatensystem* die gleiche Orientierung auf. Die von der Steuerung vorgegebene Sollposition des Beines wird dann von der kraftabhängigen Positionsänderung überlagert. Der über die „active Complian-ce“ modifizierte Sollwert wird nun in die entsprechenden Koordinatensysteme überführt und über die Positionsgelektregler realisiert. Zusammenfassend be-deutet dies, dass die „active Compliance“ nach Aktivierung überlagernd zu allen via *MCA GUI* eingestellten Bewegungen fungiert und die durch die Roboterbe-wegung entstehenden Querkräfte minimiert.

8.2.1.1 MCA–Group: ActiveCompliance

Entsprechend der Top–Down–Analyse und der hierarchischen Strukturierung von Modulen in Gruppen werden die zur Realisierung der „active Compliance“ notwendigen Module zu einer höheren Abstraktionsebene der „ActiveCompliance–Group“ zusammengefasst. Zur Lokalisierung einer geeigneten Position der „ActiveCompliance–Group“ muss festgestellt werden, welche *Control* und *Sense* Informationen zur Realisierung der „active Compliance“ benötigt werden, bzw. wo die Informationen durch Umlegung und Neuverbindung von Kanten abgegriffen werden können. Somit wird die „ActiveCompliance–Group“ in der bestehenden „BehaviourGroup“ untergebracht. Um Redundanzen im Steuerungssystem zu vermeiden, werden die bereits vorhandenen translatotrischen Bewegungen des Roboterkörpers in x–, y– und z–Achsenrichtung des *Symmetriekoordinatensystems* verwendet und entsprechend erweitert. Erweitert heißt in diesem Zusammenhang, dass eine über die *MCAGUI* eingestellte Translation des Roboterkörpers im gleichen Verhältnis auf alle sechs Beine übertragen wird. Dies ist jedoch für die „active Compliance“ nicht ausreichend, da Kräfte auf jedes einzelne Bein mit verschiedener Orientierung wirken können und folglich die Ausweichbewegung entlang der wirkenden Kräfte jedes Beins einzeln berechnet werden muss. Die hierfür notwendigen x–, y– und z–Kraftkomponenten der sechs Fußsensoren liegen bezüglich der Orientierung des *Symmetriekoordinatensystems* vor und werden von den Beinmodulgruppen „Li LegControl“ (für $i=0\dots5$) (vgl. Kapitel 7.1.2) der „BehaviourGroup“ zur Verfügung gestellt. Abbildung 8.1 zeigt mögliche Krafteinwirkungen auf den Roboterkörper und die entsprechenden Ausweichbewegungen. Ferner wird die in Kapitel 3.6 vorgestellte Kraftverteilung implementiert. Die zu erwartenden Kraftverteilungen auf die in der Stützphase befindlichen Beine werden als „Vorspannung“ der „active Compliance“ in Bezug auf die z–Achse der *Beinkoordinatensysteme* eingestellt. Hierbei wird Gleichung (3.39) zur Bestimmung der Kraftverteilungen für die in der Stützphase befindlichen Beine 3, 4, 5 und 6 implementiert. Ausgehend von den benötigten *Control* und *Sense* Daten wird die „ActiveCompliance–Group“ direkt vor die „LegControl“–Gruppen der einzelnen Beine geschaltet. Abbildung 8.2 zeigt die Integration der Gruppe in die „BehaviourGroup“. Somit kann die „ActiveCompliance–Group“ via *MCAGUI* eingestellte Bewegungen abfangen und unter Verwendung der Kraftsensorinformationen entsprechend dem berechneten Ergebnis des „active Compliance“–Reglers modifizieren. Die modifizierten Sollpositionen werden dann an die entsprechenden Beinmodulgruppen „Li LegControl“ (für $i=0\dots5$) weitergeleitet. Die benötigten Kraftvektoren und Ist–Positionen der Beine werden direkt von den sechs Beinmodulgruppen an die „ActiveCompliance–Group“ geleitet.

Entsprechend dieser Anforderungen werden zur Realisierung gemäß der hierarchischen Aufteilung des Gesamtsystems, der „ActiveCompliance–Group“ die vorgestellten Aufgaben in verschiedene Aufgabenbereiche unterteilt. Diese Abstraktionsebene entspricht der Modulebene des MCA–System.

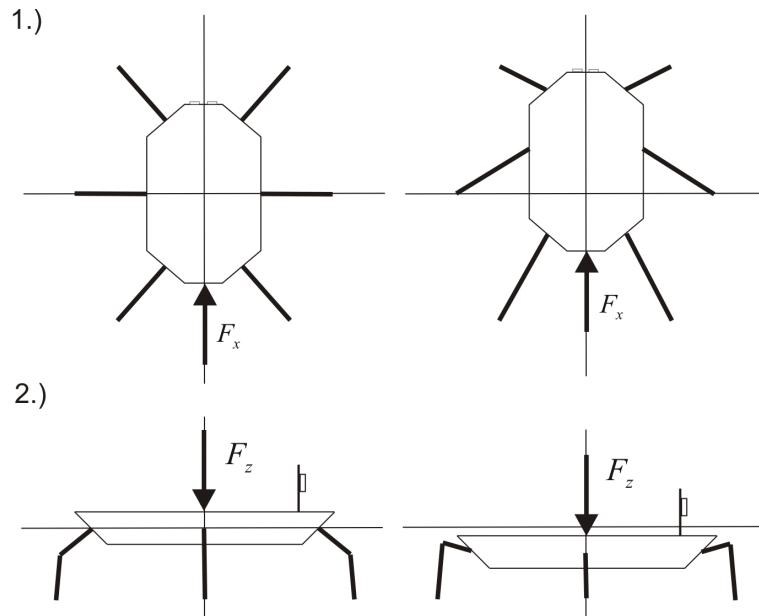


Abbildung 8.1: Mögliche Krafteinwirkungen auf den Roboterkörper und die entsprechenden Ausweichbewegungen

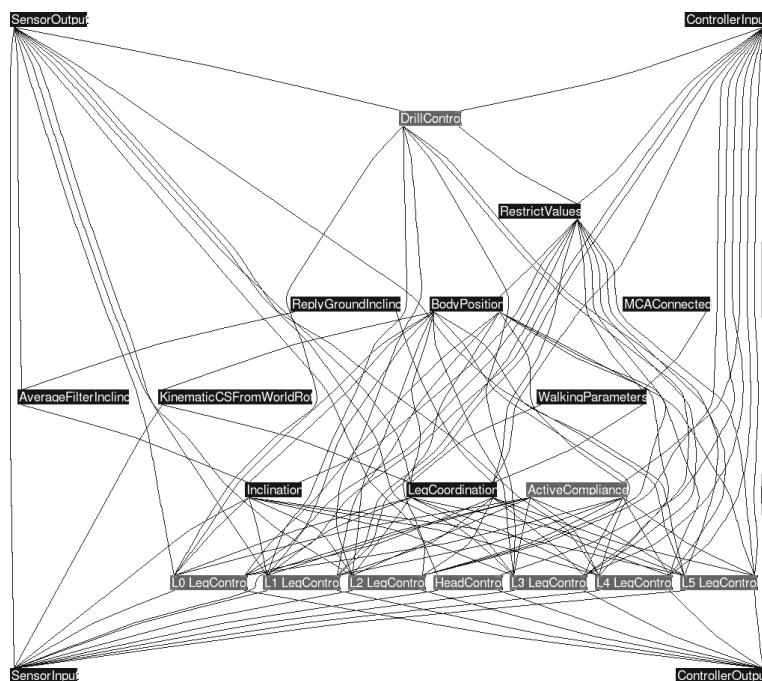


Abbildung 8.2: „BehaviourGroup“ mit Steuerungserweiterung

8.2.2 Bohroperation

Da die Laufmaschine Lauron IVb die Fähigkeit besitzt eine von außen wirkende Kraft unter Einsatz der Kraftsensoren in den Füßen zu messen, wird der Bohrvorgang selbst mit Hilfe der gemessenen Kräfte gesteuert. Für diesen Zweck wird eine Bohrmaschine parallel der x-Achse des *Symmetriekoordinatensystems* in Laufrichtung am Roboter Lauron IVb befestigt. Die Bohroperation erfolgt an einer senkrecht stehenden Holzwand parallel zum Untergrund. Der Bohrvorgang wird hierfür in drei Phasen, eine Ausrichtungsphase, eine Bohrphase und eine Rückstellphase gegliedert. In der Ausrichtungsphase wird der Bohrer der Bohrmaschine an die gewünschte Bohrstelle mit Hilfe von Körperbewegung positioniert. Für die Bewegung des Bohrers entlang der Bohrachse wird ein Dämpfungsregler (vgl. Kapitel 3.5.1) nach Gleichung (8.1) eingesetzt.

$$\dot{x} = k_x \cdot (F_{soll}^{(R)} - \sum F_{ist}^{(i)}_x) \quad (8.1)$$

Dabei ist \dot{x} die Geschwindigkeitskomponente des Roboterkörpers entlang der x-Achse des *Symmetriekoordinatensystems*, der die Bewegung zum Objekt realisiert. Diese Bewegung erfolgt so lange, bis sich ein Gleichgewicht zwischen den Sollkräften $\sum F_{soll}^{(i)}$ und den aktuell wirkenden Kräften $\sum F_{ist}^{(i)}$ erreicht ist. Die Bewegungsrichtung ist abhängig von dem Anpassungsparameter k_x und der Kräftedifferenz. Ist der Kontakt zwischen Bohrer und Wand erfolgt, wird in die Bohrphase gewechselt. Die Bohroperation wird durch einen Dämpfungsregler für alle drei Koordinatenachsen des *Symmetriekoordinatensystems* gesteuert. Die gewünschte Bohrkraft wird als Sollkraft in Bohrrichtung eingestellt. Zur Minimierung der beim Bohren entstehenden Querkräfte wird für die entsprechenden Kraftkomponenten die Sollkraft ($F_{soll}^{(R)} y$, $F_{soll}^{(R)} z$) auf null gestellt. Die Steuergesetze für die Bohrphase lauten für die y- und z-Geschwindigkeitskomponenten des Roboterkörpers im *Symmetriekoordinatensystems*:

$$\dot{y} = k_y \cdot (\overbrace{F_{soll}^{(R)} y}^0 - \sum F_{ist}^{(i)} y) \quad (8.2)$$

$$\dot{z} = k_z \cdot (\overbrace{F_{soll}^{(R)} z}^0 - (\sum F_{ist}^{(i)} z - F_g)) \quad (8.3)$$

Dabei sind \dot{y} und \dot{z} die Bewegungen des Roboterkörpers entlang der Körperachsen und liefert die Parameter für die Soll-Körpertranslationen entlang der y- und z-Achse des *Symmetriekoordinatensystems*. Da es sich bei den Positionsänderungen um Geschwindigkeiten (\dot{x} , \dot{y} und \dot{z}) handelt müssen diese erst in Strecken überführt werden. Hierzu wird die Gleichung (8.4) eingesetzt.

$$v = \frac{s}{t} \Rightarrow s = v \cdot t \quad (8.4)$$

s beschreibt hierbei die Positionsänderung als Strecke. v ist die Geschwindigkeit und wird entsprechend mit den berechneten Geschwindigkeitswerten \dot{x} , \dot{y} und \dot{z} ersetzt. Die Zeit t beschreibt die vergangene Zeit zwischen der periodischen

Ausführung des Dämpfungsreglers. Da die Bohrmaschine am Roboterkörper befestigt ist und die Bohrbewegung über Körpertranslationen realisiert wird, werden die wirkenden Kräfte bezüglich Orientierung des *Symmetriekoordinatensystems* dargestellt. Da für die Bohroperation nur für den Fall angenommen wird, dass sich der Roboterkörper parallel zur Ebene befindet, welche senkrecht zum Gravitationsvektor steht, lässt sich die Gewichtskraft F_g des Laufroboters aus Gleichung (8.3) von der Summe der gemessenen z-Kraftkomponenten der Füße subtrahieren. In der Rückstellphase erfolgt eine Bewegung zum Ausgangspunkt der Bohroperation. Dabei ist die Bewegung entlang der Bohrachse nicht kraftgeführt, während für die Querachsen weiterhin die Dämpfungsregelung zur Minimierung der Querkräfte aktiv ist.

8.2.2.1 MCA–Group: DrillControl

Analog zu der „ActiveCompliance“ werden alle zur Realisierung der Bohroperation notwendigen Module in einer Gruppe der „DrillControl“ zusammengefasst. Da es sich bei der Bohroperation um einen verhaltensbasierten Vorgang handelt, wird die „DrillControl“–Group in die „BehaviourGroup“ (vgl. Kapitel 7.1.2) integriert. Nach Aktivierung der Bohroperation via *MCAGUI* arbeitet die „DrillControl“ für die Dauer der Durchführung der drei Phasen autonom und übernimmt stellvertretend für die *MCAGUI* die Robotersteuerung. Daher kann die „DrillControl“ als ein höheres Verhaltenskonzept angesehen werden, welches auf Grundlage des „damping Control“–Kraftreglers agiert. Zur Erfüllung dieser Gegebenheiten wird die „DrillControl“–Group direkt zwischen die *MCA-GUI* und den Modulen bzw. Modulgruppen der „BehaviourGroup“ geschaltet. Entsprechend dieser Überlegung erfolgt die Positionierung in der Ebene 1 (vgl. Abbildung 8.2) der *BehaviorGroup*. Für die Bewegung des Roboterkörpers während der Bohroperation werden die im Steuerungssystem bereits vorhandenen translatorischen Bewegungen im x-, y- und z-Achsenrichtung bezüglich des *Symmetriekoordinatensystems* eingesetzt. Dabei werden die translatorischen Positionsänderungen des Roboterkörpers im *Symmetriekoordinatensystem* mit den Gleichungen (8.1), (8.2) und (8.3) auf alle sechs Beine übertragen. Der über die „DrillControl“ eingestellte translatorische Sollwert wird an die entsprechenden Beinmodulgruppen „Li LegControl“ (für $i=0\dots 5$) (vgl. Kapitel 7.1.2) der „BehaviourGroup“ geleitet und letztendlich über die Positionsregler der Beine realisiert. Zusätzlich ist die translatorische Ist–Position des Roboterkörpers im *Symmetriekoordinatensystem* erforderlich, diese wird zu der über den Dämpfungsregler berechneten Soll–Positionsänderung addiert. Die während der Bohroperation wirkenden Kräfte werden mit den Fußkraftsensoren gemessen und bezüglich Orientierung des *Symmetriekoordinatensystems* aufsummiert. Die hierfür notwendigen x-, y- und z-Kraftkomponenten der sechs Fußsensoren liegen bezüglich der Orientierung des *Symmetriekoordinatensystems* vor und werden von den Beinmodulgruppen „Li LegControl“ (für $i=0\dots 5$) der „BehaviourGroup“ zur Verfügung gestellt. Die Bohroperation arbeitet primär also nicht überlagernd zu anderen Bewegungen des Roboters. Entsprechend dieser Anforderungen werden die zur Realisierung vorgestellten Aufgaben in verschiedene Aufgabenbereiche unterteilt. Diese Aufgabenbereiche werden durch MCA–Module realisiert.

Eine Alternative zur Organisierung des Dämpfungsreglers in der

„DrillControl“–Group liegt in der Positionierung des Dämpfungsreglers in der „ActiveCompliance“–Group. Somit würden sich beide Kraftregler in der gleichen Ebene befinden. Da es sich aber bei der Bohroperation um einen autonomen Vorgang handelt dem der Dämpfungsregler zu Grunde liegt, es sich hingegen bei der „active Compliance“ um eine überlagernde kraftgeführte Bewegung zur Minimierung von Querkräften auf Grundlage des „active Compliance“–Reglers handelt, werden beide kraftgeführten Bewegungen gemäß der grundlegend unterschiedlichen Aufgaben getrennt in zwei Modulgruppen organisiert.

8.3 Beincontroller Anforderungen

Durch die Kraftregler der „active Compliance“ und der „damping Control“ aus Kapitel 3.5.1 werden Positionsänderungen der Beine bestimmt. Da diese Änderungen der Soll–Positionen teilweise sehr fein sind, muss der auf den Beincontrollern implementierte digitale PID–Regler entsprechend gut entworfen und eingestellt sein. In der Praxis wurde allerdings am Laufroboter Lauron IVb beobachtet, dass geringe Abweichungen zwischen Soll– und Ist–Gelenkwinkel zu keiner Drehzahl der Motoren führte. Zur Lösung dieses Problems wird im folgenden ein Konzept vorgestellt, das den vorhandenen digitalen PID–Regler entsprechend verbessert.

8.3.1 PWM–Kennlinienlinearisierung

Wie in Kapitel 7.2.1 beschrieben erzeugt der digitale Positions PID–Regler eine PWM Einschaltzeit n_{ON} um die Gelenkwinkelabweichung zwischen Soll– und Ist–Wert auszuregeln. In der Praxis führen allerdings geringe Abweichungen zwischen Soll– und Ist–Gelenkwinkel zu keiner Drehzahl der Motoren. Dieses Verhalten entspricht der PWM–Drehzahl–Kennlinie (Abbildung 7.7). Da aber auch geringe Abweichungen zwischen Soll– und Ist–Gelenkwinkel zu einer Drehzahl des Motors führen soll, muss die „Totzone“ ausgeglichen werden. Aus der PWM–Drehzahl–Kennlinie ist zu erkennen, dass der Kurvenanstieg außerhalb des Totbereichs zunächst linear ist. Das Modell der PWM–Drehzahl–Kennlinie kann somit wie folgt dargestellt werden.

$$f = \begin{cases} -X_{PWM,MAX} \leq X_{PWM} \leq -X_{PWM,0} & : \omega = m \cdot (X_{PWM} + X_{PWM,0}) \\ -X_{PWM,0} < X_{PWM} < +X_{PWM,0} & : \omega = 0 \\ +X_{PWM,0} \leq X_{PWM} \leq +X_{PWM,MAX} & : \omega = m \cdot (X_{PWM} - X_{PWM,0}) \end{cases} \quad (8.5)$$

mit

- | | |
|-------------|---|
| X_{PWM} | – Betrag der Puls–Weiten Modulation. |
| $X_{PWM,0}$ | – Betrag der Stelle des Kennlinienknicks im positiven und negativen Bereich der Streckenübertragungsfunktion. |

- $X_{PWM,MAX}$ – Maximaler Betrag der Puls–Weiten Modulation.
 m – Steigungsfaktor der Kennlinie außerhalb des „Totbereiches“.
 ω – Drehzahl.

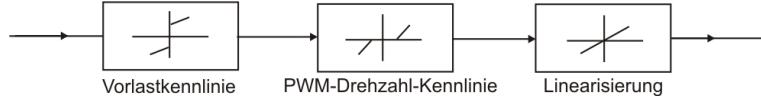


Abbildung 8.3: Linearisierte PWM–Drehzahl–Kennlinie

Im Hinblick auf die Verbesserung der Qualität der Regelung wird ein nicht lineares System durch das Vorschalten einer ausgleichenden Kennlinie linearisiert [45][38]. Das Grundprinzip verdeutlicht Abbildung 8.3. Die nicht lineare statische Kennlinie der PWM–Drehzahl–Kennlinie erfährt im Anfangsbereich eine Linearisierung indem eine Vorlastkennlinie dem System hinzugefügt wird. Die Vorlastkennlinie hebt bzw. senkt die PWM–Kennlinie. Somit verhält sich das Gesamtsystem weitgehend linear, insbesondere um den Nullpunkt. Dieses Verhalten ist z.B. für das langsame Anfahren und das Halten einer Position günstig. Voraussetzung für den Einsatz dieser Vorschaltkennlinie ist die Möglichkeit der Aufteilung eines Systems in einen linearen und einen nichtlinearen Teil. Der nicht lineare Systemteil wird mit Hilfe der gespiegelten (inversen) Vorschaltkennlinie kompensiert bzw. linearisiert. Nach dem Inneren–Modell–Prinzip werden die nichtlineare Regelstrecke und die Kompensations–Kennlinie zu einem linearen Modell der Regelstrecke abstrahiert.

$$G_S = G_{S,NL} \cdot G_{S,LIN} \quad (8.6)$$

Hierbei beschreibt $G_{S,NL}$ den nichtlinearen Anteil der Streckenübertragungsfunktion und $G_{S,LIN}$ die Kompensationsfunktion zur Linearisierung der Streckenübertragungsfunktion. Das Ergebnis ist eine lineare Übertragungsfunktion der Regelstrecke. Abbildung 8.4 zeigt die Erweiterung der Abbildung 7.8 um die Kennlinienlinearisierung.

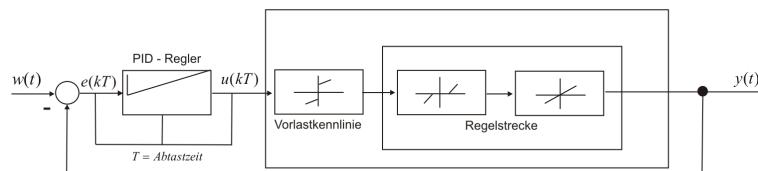


Abbildung 8.4: Digitaler PID–Gelenkregelkreis mit Linearisierung der Regelstrecke

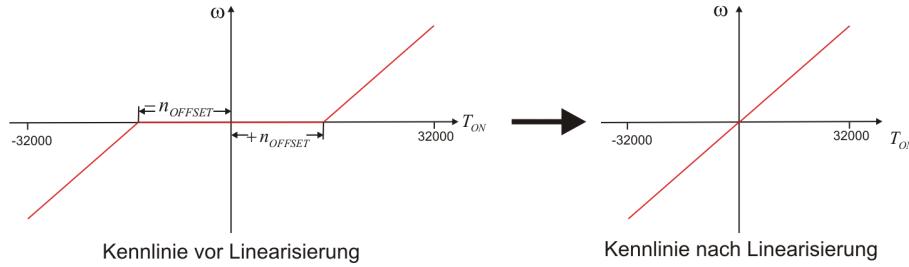


Abbildung 8.5: Kennlinienlinearisierung

8.3.1.1 Beincontroller PWM–Offset

Um den „Totbereich“ der PWM–Drehzahl–Kennlinie (Abbildung 7.7) auszugleichen wird die PWM–Drehzahl–Kennlinie linearisiert. Um diese Linearisierung zu erreichen wird eine konstante Größe der n_{OFFSET} eingesetzt. Dieser $OFFSET$ wird zu der vom digitalen PID–Regler erzeugten Einschaltzeit n_{ON} addiert bzw. subtrahiert. Abbildung 8.5 zeigt die PWM–Drehzahl–Kennlinie und den Einsatz des Parameters n_{OFFSET} .

Wie aus Abbildung 8.5 ersichtlich müssen positive n_{ON} Einschaltzeiten sowie negative n_{ON} Einschaltzeiten gesondert betrachtet werden. Die Gleichung (8.7) stellt diesen Zusammenhang dar:

$$f = \begin{cases} -n_{ON,MAX} \leq n_{ON} < 0 & : n_{ON,NEW} = n_{ON} - n_{OFFSET} \\ n_{ON} = 0 & : n_{ON,NEW} = 0 \\ 0 < n_{ON} \leq +n_{ON,MAX} & : n_{ON,NEW} = n_{ON} + n_{OFFSET} \end{cases} \quad (8.7)$$

mit

n_{ON}	– Betrag der Einschaltzeit.
$n_{ON,MAX}$	– Maximaler Betrag der Einschaltzeit.
n_{OFFSET}	– Konstanter Offset.
$n_{ON,NEW}$	– Neue Einschaltzeit.

Gleichung (8.7) wird in die Beincontrollersoftware integriert um den „Totbereich“ zu kompensieren. Da die PWM–Drehzahl–Kennlinie für die bei der Laufmaschine Lauron IVb eingesetzten Motoren nicht bekannt ist müssen die PWM–Offsets für die Motoren der Beine bestimmt werden. Dadurch dass die Roboterbeine in Segmente verschiedener Masse unterteilt sind, müssen die Motoren unterschiedliche Arbeit verrichten um ein Segment und die Folgesegmente zu bewegen. Folglich sind die „Totzeiten“ der PWM–Drehzahl–Kennlinien der drei Motoren eines Beins unterschiedlich. Um unterschiedliche Offsets der Motoren zu testen, lassen sich für jeden der drei Motoren eines Beins mit Hilfe des *MCA browser* via Modulparameterschnittstelle die PWM–Offsets setzen. Hierfür wurden

die Beinmodule „Li LaurenLeg“ (für $i=0..5$) der MCA–Gruppe „DSP–Remote–Part“ (vgl. Kapitel 7.1.4) um folgende Parameter erweitert: $\alpha_{OFFSET}^{(i)}$, $\beta_{OFFSET}^{(i)}$ und $\gamma_{OFFSET}^{(i)}$. Ferner muss darauf geachtet werden, dass durch die Addition bzw. Subtraktion des PWM–Offsets der maximale Betrag der vom digitalen PID–Regler generierten Einschaltzeit nicht überschritten wird. Hierfür wird eine Schutzfunktion implementiert, welche bei Überschreitung der maximalen Einschaltzeit diese auf den Maximalwert (± 32000) setzt. Gegebenenfalls müssen die Parameter K_P , K_I und K_D des digitalen PID–Reglers für die drei Motoren eines Beins nach der Linearisierung neu eingestellt werden, da die Linearisierung das Verhalten des PID–Reglers verändert.

Kapitel 9

Entstandene Module

In diesem Kapitel werden die in dieser Arbeit entstandenen Module und Modulgruppen vorgestellt. Nach den Modulnamen werden zuerst die wichtigsten Datenwerte der Ein- und Ausgänge aufgelistet, gefolgt von einer kurzen Beschreibung der Funktionalität.

9.1 ActiveCompliance

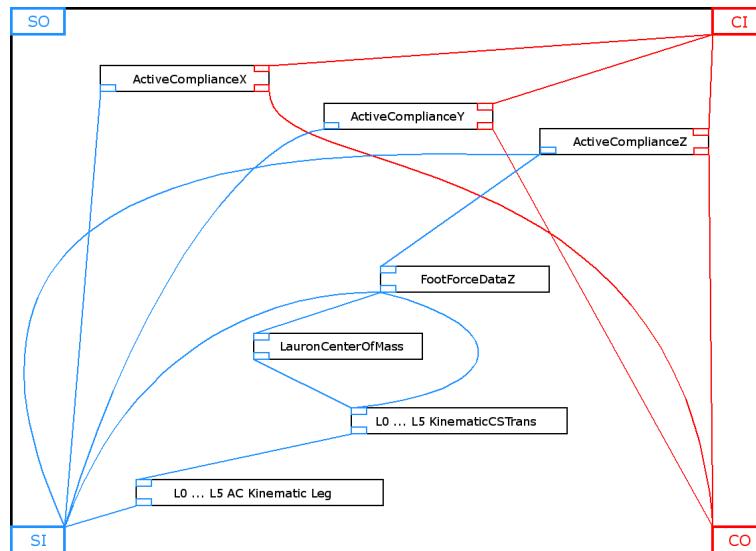


Abbildung 9.1: Aufbau der „ActiveCompliance–Group“

Im Folgenden werden die *Control* und *Sense* Daten der „ActiveCompliance–Group“ vorgestellt. Des weiteren werden die Module dieser Gruppe inkl. der *Sensor* und *Control* Daten erläutert. Abbildung 9.1 zeigt die Module der

„ActiveCompliance–Group“.

ActiveCompliance

CI: Über die *MCAGUI* eingestellte Soll–Körpertranslation X, Y und Z im *Symmetriekoordinatensystem*; Flag zum Aktivieren bzw. Deaktivieren der „active Compliance“.

CO: Berechnete translatorische–Sollpositionen X, Y und Z der sechs Beine im *Symmetriekoordinatensystem*.

SI: Ist–Beingelenkwinkel (α, β, γ) der sechs Beine; Beinkontaktzustand (0: kein Kontakt; 1: Bein befindet sich in der Stützphase); Kraftvektoren bezüglich Orientierung des *Symmetriekoordinatensystem* der sechs Beine.

SO: –

AC Kinematic Leg

CI: –

CO: –

SI: Ist–Beingelenkwinkel (α, β, γ) für ein Bein.

SO: Ist–Fußpunkt im *Beinkoordinatensystem*; Ist–Massepunkte der B–C– und D–Segmente im *Beinkoordinatensystem*.

Dieses Modul liegt in sechsfacher Ausführung für jedes der Beine vor und berechnet die direkte Kinematik des Fußpunktes (vgl. Kapitel 3.3.1) sowie die Massepunkte der B–C– und D–Segmente (vgl. Kapitel 3.6). Hierfür werden die Gleichungen (3.7), (3.27), (3.28) und (3.29) umgesetzt.

KinematicCSTrans

CI: –

CO: –

SI: Ist–Fußpunkt im *Beinkoordinatensystem*; Ist–Massepunkte der B–C– und D–Segmente im *Beinkoordinatensystem*.

SO: Ist–Fußpunkt im *Symmetriekoordinatensystem*; Ist–Massepunkte der B–C– und D–Segmente im *Symmetreikoordinatensystem*.

Dieses Modul liegt in sechsfacher Ausführung für jedes der Beine vor und führt Koordinatentransformation vom *Beinkoordinatensystem* in das *Symmetreikoordinatensystem* durch. Da die Orientierung dieser Koordinatensysteme identisch ist, sind in diesem Modul nur Translationen durchzuführen.

LauronCenterOfMass

CI: –

CO: –

SI: Ist–Massepunkte der B–,C– und D–Segmente im *Symmetriekoordinatensystem* der sechs Beine.

SO: Masseschwerpunkt des gesamten Roboters.

Dieses Modul berechnet den Masseschwerpunkt des gesamten Roboters ausgehend der Berechnungsvorschrift der Gleichungen (3.33) und (3.34).

FootForceDataZ

CI: –

CO: –

SI: Masseschwerpunkt des gesamten Roboters; Ist–Fußpunkte im *Symmetriekoordinatensystem* der sechs Beine; Beinkontaktzustände der sechs Beine.

SO: Berechnete, zu erwartende Kraftverteilungen auf die in der Stützphase befindlichen Füße.

Dieses Modul berechnet die zu erwartenden Kraftverteilungen auf die Stützfüße auf Grundlage der Gleichung (3.39). Zur Lösung dieser Gleichung wird die *newmat* Bibliothek [11] eingesetzt. Diese liefert die zur Lösung notwendigen Matrixoperationen (Transponierte und Inverse einer Matrix). Die Beinkontaktzustände der sechs Beine dienen zur Feststellung der in der Stützphase befindlichen Beine.

ActiveComplianceZ

CI: Über die *MCAGUI* eingestellte Soll–Körpertranslation Z im *Symmetriekoordinatensystem*; Flag zum Aktivieren bzw. Deaktivieren der „ActiveComplianceZ“.

CO: Entstehende Positionsänderungen in Richtung der wirkenden Kraft entlang der z–Achse des *Symmetriekoordinatensystems* für jedes der sechs Beine.

SI: Berechnete, zu erwartende Kraftverteilungen auf die in der Stützphase befindlichen Füße; z–Achsen Kraftkomponente bezüglich Orientierung des *Symmetriekoordinatensystems* der sechs Beine.

SO: –

PA: z–Achsen Nachgiebigkeitsfaktor.

Dieses Modul realisiert die Positionsänderungen der „active Compliance“ entlang der z–Achse des *Symmetriekoordinatensystems* für jedes der sechs Beine. Die berechneten Kraftverteilungen werden als vertikale „Vorspannung“ eingestellt (vgl. Kapitel 3.6, Gleichung (3.37)). Die über die *MCAGUI* eingestellte Soll–Körpertranslation Z wird verwendet um bei Rücknahme der wirkenden Kraft entlang der z–Achse wieder zur Ausgangsposition zurück zu kehren. Die Gleichung der „active Compliance“ (3.12) ist für jedes der sechs Beine vorhanden und liefert somit die entstehenden Positionsänderungen in Richtung der auf die Beine wirkenden vertikalen Kraftkomponente.

ActiveComplianceX und ActiveComplianceY

CI: Über die *MCAGUI* eingestellte Soll–Körpertranslation X bzw. Y im *Symmetriekoordinatensystem*; Flag zum Aktivieren bzw. Deaktivieren der „ActiveComplianceZ“.

CO: Entstehende Positionsänderungen in Richtung der wirkenden Kraft entlang der z– bzw. y–Achse des *Symmetriekoordinatensystems* für jedes der sechs Beine.

SI: x– bzw. y–Achsen Kraftkomponente bezüglich Orientierung des *Symmetriekoordinatensystems* der sechs Beine.

SO: –

PA: x– bzw. y–Achsen Nachgiebigkeitsfaktor.

Die Module „ActiveComplianceX“ und „ActiveComplianceY“ besitzen die gleiche Funktionalität analog zu der „ActiveComplianceZ“ mit der Ausnahme, dass bei diesen Modulen keine „Vorspannung“ eingestellt wird. Wie aus den Modulbezeichnungen ersichtlich, realisieren die beiden Module die entstehenden Positionsänderungen in Richtung der wirkenden Kräfte entlang der x–, sowie der y–Komponente des *Symmetriekoordinatensystems* für jedes der sechs Beine.

9.2 DrillControl

Im Folgenden werden die *Control* und *Sense* Daten der „DrillControl–Group“ vorgestellt. Des Weiteren werden die Module dieser Gruppe inkl. der *Sensor* und *Control* Daten erläutert. Abbildung 9.2 zeigt die Module der „DrillControl–Group“.

DrillControl

CI: Über die *MCAGUI* eingestellte Soll–Körpertranslation x, y und z im *Symmetriekoordinatensystem*; Flag zum Aktivieren bzw. Deaktivieren der „DrillControl“; Flag zum Aktivieren bzw. Deaktivieren der Protokollierung der wirkenden Fußkräfte.

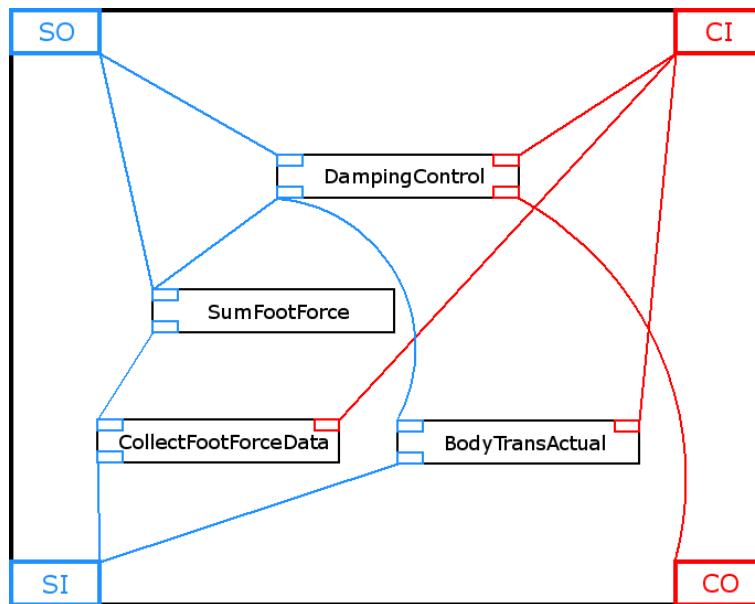


Abbildung 9.2: Aufbau der „DrillControl-Group“

CO: Berechnete translatorische Sollpositionen x, y und z des Roboterkörpers im *Symmetriekoordinatensystem*.

SI: Kraftvektoren bezüglich Orientierung des *Symmetriekoordinatensystems* der sechs Beine; Ist-Fußpunkte im *Symmetriekoordinatensystem* der sechs Beine;

SO: Berechnete translatorische-Istpositionen x, y und z des Roboterkörpers im *Symmetriekoordinatensystem*; Auf den Roboterkörper wirkende Bohrkräfte bezüglich Orientierung des *Symmetriekoordinatensystems*.

PA: –

BodyTransActual

CI: Flag zum Aktivieren bzw. Deaktivieren des „BodyTransActual“ Moduls; Über die MCAGUI eingestellte Soll-Körpertranslation x, y im *Symmetriekoordinatensystem*;

CO: –

SI: Ist-Fußpunkte im *Symmetriekoordinatensystem* der sechs Beine;

SO: Berechnete translatorische-Istpositionen x, y und z des Roboterkörpers im *Symmetriekoordinatensystem*;

Dieses Modul berechnet die translatorische-Istposition des Roboterkörpers im *Symmetriekoordinatensystem* ausgehend von den aktuellen Positionen der sechs

Fußpunkte im *Symmetriekoordinatensystem*.

CollectFootForceData

CI: Flag zum Aktivieren bzw. Deaktivieren der Protokollierung der wirkenden Fußkräfte.

CO: –

SI: Kraftvektoren bezüglich Orientierung des *Symmetriekoordinatensystems* der sechs Beine;

SO: Kraftvektoren bezüglich Orientierung des *Symmetriekoordinatensystems* der sechs Beine;

Dieses Modul schreibt nach Aktivierung via *MCAGUI* die Kräftevektoren der auf die Beine wirkenden Kräfte in eine Datei. Die Kräftevektoren werden bezüglich der Orientierung des *Symmetriekoordinatensystems* abgespeichert. Mit diesen Daten lassen sich die während des Bohrvorgangs auf den Roboterkörper wirkenden Kräfte analysieren. Bevor der Bohrvorgang autonom ablaufen kann muss der Bohrvorgang via *MCAGUI* gesteuert werden währenddessen werden die Bohrkräfte aufgezeichnet. Der Mittelwert der während der Bohrphase wirkenden Bohrkräfte entlang der x-Komponente des *Symmetriekoordinaten-systems* wird benutzt um den $F_{soll\ x}^{(R)}$ der Gleichung (8.1) einzustellen.

SumFootForce

CI: –

CO: –

SI: Kraftvektoren bezüglich Orientierung des *Symmetriekoordinatensystems* der sechs Beine.

SO: Auf den Roboterkörper wirkende Bohrkräfte bezüglich Orientierung des *Symmetriekoordinatensystems*.

PA: Gewichtskraft F_g des Roboters in Newton.

Dieses Modul summiert die einzelnen Komponenten der auf die Beine wirkenden Fußkraftvektoren bezüglich Orientierung des *Symmetriekoordinatensystems* nach den Berechnungsvorschriften:

$$\sum F_{ist\ x}^{(i)} \quad (9.1)$$

$$\sum F_{ist\ y}^{(i)} \quad (9.2)$$

$$\sum F_{ist\ z}^{(i)} - F_g \quad (9.3)$$

Somit stehen die auf den Roboter wirkenden Bohrkräfte bezüglich Orientierung des *Symmetriekoordinatensystems* zur Verfügung.

DampingControl

CI: Über die *MCAGUI* eingestellte Soll–Körpertranslation x, y und z im *Symmetriekoordinatensystem*; Flag zum Aktivieren bzw. Deaktivieren der „DrillControl“;

CO: Berechnete translatorische–Sollpositionen x, y und z des Roboterkörpers im *Symmetriekoordinatensystem*.

SI: Berechnete translatorische–Istpositionen x, y und z des Roboterkörpers im *Symmetriekoordinatensystem*; Auf den Roboterkörper wirkende Bohrkräfte bezüglich Orientierung des *Symmetriekoordinatensystems*.

SO: Berechnete translatorische–Istpositionen x, y und z des Roboterkörpers im *Symmetriekoordinatensystem*;

PA: Sollkräfte für die Kraftkomponenten $F_{soll\ x}^{(R)}$, $F_{soll\ y}^{(R)}$ und $F_{soll\ z}^{(R)}$; Dämpfungs faktoren der Komponenten k_x , k_y und k_z .

Dieses Modul stellt das Herzstück der Modulgruppe dar. In diesem Modul werden die translatorischen Positionsänderungen pro Ausführungszyklus des Moduls in Abhängigkeit der wirkenden Kräfte berechnet. Hierfür werden die Gleichungen (8.1), (8.2) und (8.3) umgesetzt. Da es sich bei den Positionsänderungen um Geschwindigkeiten (\dot{x} , \dot{y} und \dot{z}) handelt müssen diese erst in Strecken überführt werden. Hierzu wird die Gleichung (8.4) eingesetzt.

Die Zeit t beschreibt die vergangene Zeit zwischen der periodischen Ausführung des „DampingControl“ Moduls. Dazu wird die Startzeit in eine Variable geschrieben, wird das Modul erneut zur Ausführung gebracht so wird die neue Startzeit in einer Variablen abgelegt. Die Zeit t wird durch die Gleichung

$$t = t_{old} - t_{new}, \quad (9.4)$$

bestimmt.

Kapitel 10

Zusammenfassung

Mit Lauron IVb steht eine robuste Hardwarearchitektur zur Verfügung. Ausgehend von dieser sechsbeinigen Laufmaschine sowie der modularen Steuerungsarchitektur für Roboter (MCA) wurde im Verlauf dieser Diplomarbeit kraftgeführte Bewegungen implementiert. Hierbei werden die Methoden zu kraftgeführten Bewegungen aufgezeigt, die als Basis für technische Operationen dienen können. Diese Steuerungsalgorithmen müssen dabei an die vorhandene Steuerung angepasst sein und somit entsprechend der MCA Struktur integriert werden. Zur Findung geeigneter Positionierungen im vorhandenen Steuerungssystem war es erforderlich die Steuerung zu analysieren um den Aufbau und die Funktionsweise zu verstehen. Die vorgestellten kraftgeführten Bewegungsmethoden werden im Hinblick auf verschiedene Aufgabenbereiche integriert. Sie werden einerseits mit Lauf- und Körperbewegungen kombiniert um während des Laufvorgangs auftretenden Querkräfte zu minimieren. Ein weiterer Einsatzbereich sind technische Operationen wie das autonome Bohren an einer Wand. Es hat sich gezeigt, dass die in den Mikrocontrollern vorhandenen Positionsregler nicht den Anforderungen entsprechen, welche während der Ausführung der kraftgeführten Bewegungen erforderlich sind. Daher wurden die Positionsregler entsprechend angepasst und verbessert.

Da die Laufmaschine Lauron IVb der Physiologie der Stabheuschrecke nachempfunden ist und die vorhandene Steuerung auf den Mechanismen des natürlichen Laufvorgangs der Stabheuschrecke beruht wurde zum besseren Verständnis des Gesamtsystems die Stabheuschrecke vorgestellt. Zum Erreichen der Zielstellung wurden zunächst die mathematischen Grundlagen kraftgeführter Bewegungen hergeleitet. Es wurde gezeigt, dass die Kraftverteilung des Robotergewichts auf die einzelnen Stützfüße mathematisch bestimmt werden kann.

Zur Steuerung des Roboters wird ein hierarchisches modulares Steuerungssystem (MCA) eingesetzt. Der Aufbau und die mögliche Strukturierung von Modulen, Gruppen und Parts und deren Datenhandling via Kanten wurden ebenfalls vorgestellt. Ausgehend von dieser Grundlage wurde die vorhandene Robotesteuerung analysiert und Positionierungen herausgearbeitet um die vorgestellten kraftgeführten Bewegungen zu integrieren.

Für die kraftgeführten Bewegungen wurde eine modulare hierarchische Steue-

rung auf algorithmischer Basis entworfen und implementiert. Dafür wurden die mathematischen Überlegungen der kraftgeführten Bewegungen und der Kraftverteilung unter Einsatz der direkten Kinematik umgesetzt.

Für die Integration der Steuerungserweiterung in den Roboter muss die Hardware kompakt realisiert sein. Hierfür werden Mikrocontroller eingesetzt, da in ihnen Peripheriekomponenten integriert sind, die die Auswertung von Sensorsignalen und die Erzeugung von Steuersignalen erlauben. Hierfür werden Puls-Weiten-modulierte Signale zur Ansteuerung der Motoren erzeugt. Hierbei hat sich gezeigt, dass das nichtlineare Verhalten mit Hilfe von Kennliniengliedern kompensiert werden muss. Die Kennlinienrealisierung wurde durch Einsatz eines PWM Offsets durchgeführt.

Literaturverzeichnis

- [1] *DIAPM RTAI - Realtime Application Interface.* – <http://www.aero.polimi.it/~rtai/>
(Stand: 2005)
- [2] [DIN 44 300 85]
- [3] *Direkte Kinematik.* – http://lexikon.freenet.de/Direkte_Kinematik
- [4] *Forschungszentrum für Informatik in Karlsruhe.* – <http://www.fzi.de/>
- [5] *FSMLabs.* – <http://www.fsmlabs.com/index.php>
(Stand: 2005)
- [6] *Interaktive Diagnose- und Servicesysteme.* – <http://www.fzi.de/ids/>
- [7] *Lippert.* – http://www.lippert-at.com/uploads/pics/CRR3-S-01_120x88.gif
- [8] *Metrowerks.* – <http://www.metrowerks.com>
(Stand: 2005)
- [9] *Modular Controller Architecture.* – <http://mca2.sourceforge.net>
(Stand: 2001)
- [10] *New Mexico Institute of Mining and Technology.* – <http://www.nmt.edu/>
(Stand: 2005)
- [11] *Newmat.* – http://www.robertnz.net/nm_intro.htm
(Stand: 2005)
- [12] *PC/104 Embedded Consortium.* – <http://www.pc104.org/>
(Stand: 2005)
- [13] *QT.* – <http://www.trolltech.com>
(Stand: 2005)
- [14] *Real-Time-Linux Projects.* – <http://www.opentech.at/links.html.en>
(Stand: 2005)
- [15] *Real-Time-Linux Projects.* – <http://www.realtimelinuxfoundation.org/projects/projects.html>
(Stand: 2005)

- [16] RTAI - *Realtime Application Interface*. – <http://www.rtai.org/>
(Stand: 2005)
- [17] Die Stabheuschrecke. – http://www.uni-salzburg.at/did/bio_fuer_kids_und_teens/lebende_organis% men/stabheuschrecke.htm
(Stand: 02.10.2003)
- [18] Altera: *FLEX 10K Embedded Programmable Logic Family Data Sheet*. 2005.
– <http://www.altera.com/literature/ds/dsf10k.pdf>
- [19] BERGER, Manfred: *Grundkurs der Regelungstechnik*. Books on Demand GmbH, 2001
- [20] BRADEN, R.: Requirements for Internet Hosts - Communication Layers.
(1989). – <http://www.ietf.org/rfc/rfc1122.txt>
- [21] CHEN, Chun-Hung ; KUMAR, Vijar ; LUO, Yuh-Chyun: Motion Planning of Walking Robots in Environments with Uncertainty. In: *Journal for Robotic Systems* Bd. 16. 1999, S. 527–545
- [22] CRUSE, Holk: The Control of Body Position in the Stick Insect (*Carausius morosus*) when Walking over Uneven Surfaces. In: *Biological Cybernetics Vol. 24* (1976), S. 25–33
- [23] CRUSE, Holk: The Function of the Legs in the Free Walking Stick Insect, *Carausius morosus*. In: *Journal of Comparative Physiology* (1976), S. 235–262
- [24] DEAN, Jeffrey: A model of leg coordination in the stick insect *Carausius morosus*. In: *Biological Cybernetics* 64 (1991), S. 403–411
- [25] F. PFEIFFER, J. E. ; WEIDEMANN, H.: The TUM - Walking Machine.
- [26] Faulhaber: *DC-Kleinmotoren*. 2005. – http://www.faulhaber-group.com/uploadapk/pdf_metric174_de.pdf
- [27] Faulhaber: *Planetengetriebe*. 2005. – http://www.faulhaber-group.com/uploadapk/pdf_metric205_de.pdf
- [28] FERRELL, Cynthia: A comparison of three insect-inspired locomotion controllers. In: *Robotics and Autonomous Systems* (1995), S. 135–159
- [29] FÖLLINGER, Otto: *Regelungstechnik*. Hüthig, 1990
- [30] Freescale: *DSP56800 16-Bit Digital Signal Processor Family Manual*. 2005. – http://www.freescale.com/files/dsp/doc/ref_manual/DSP56800FM.pdf
- [31] FRIEDRICH PFEIFFER, Jürgen E. ; WEIDEMANN, Hans-Jürgen: Six-legged technical walking considering biological principles. In: *Robotics and Autonomous Systems* (1995), S. 223–232
- [32] GANTMACHER, Felix R.: *Matrizentheorie*. Springer Verlag, 1986

- [33] GASSMANN, Bernd: *Erweiterung einer modularen Laufmaschinensteuerung für unebenes Gelände*, Forschungszentrum Informatik an der Universität Karlsruhe, Diplomarbeit, September 2000
- [34] GOTTSCHALK, Alexander: *Wireless LAN*, Friedrich-Schiller-Universität Jena, Seminararbeit, 2003
- [35] GUDDAT, Martin: *Fortschritt-Berichte VDI*. Bd. 995: *Autonome, adaptive Bewegungskoordination von Gehmaschinen in komplexer Umgebung*. VDI Verlag GmbH, 2003
- [36] H. CRUSE, M. Dreifert-J. Schmitz D.E. Brunn J. D. ; KINDERMANN, T.: Walking: A Complex Behaviour Controlled by Simple Networks. In: *Adaptive Behaviour*, 1995, S. 385–418
- [37] HERMS, André: *Entwicklung eines verteilten Laufplaners basierend auf heuristischen Optimierungsverfahren*, University of Magdeburg, Diploma Thesis, 2004
- [38] IHME, Thomas: *Steuerung sechsbeiniger Laufroboter unter dem Aspekt technischer Anwendungen*, Universität Magdeburg, Dissertation, März 2002
- [39] K.-U. SCHOLL, J. A. ; GASSMANN, B.: MCA - An Expandable Modular Controller Architecture.
- [40] KALISIAK, Maciej ; PANNE, Michiel von d.: A Graps-based Motion Planning Algorithm for Character Animation.
- [41] LANG, Marco: *Real Time Linux*, Universität Koblenz-Landau, Seminararbeit, 2003
- [42] Lippert: *Cool RoadRunner III PC/104-Plus CPU board - Technical Manual*. 2005. – http://www.lippert-at.de/fileadmin/lippert-at/products/PC_104-Plus/Cool_Roadrunner_III/TME-104P-CRR3-R1V11.pdf
- [43] Lippert: *PC/104-PCA-1 PCMCIA*. 2005. – <http://www.lippert-at.com/>
- [44] OpenInventor: *Homepage*. Dezember 2003. – <http://oss.sgi.com/projects/inventor/>
- [45] PHILIPPSEN, Hans-Werner: *Einstieg in die Regelungstechnik*. Fachbuchverlag Leipzig, 2004
- [46] Phytec: *eNet-CAN*. 2005. – http://www.phytec.de/phytec/com_docman/task_doc_download/gid_302/Itemid_208/
- [47] Robert Bosch GmbH, Stuttgart: *CAN Specification Version 2.0*. – <http://www.semiconductors.bosch.de/de/20/can/index.asp>
- [48] STEPHAN CORDES, Rüdiger D.: Steuerungsarchitektur der sechsbeinigen Laufmaschine Lauron. (1993)
- [49] W. ILG, Th. Mühlriedel-R. D.: Hybrid learning concepts based on self-organizing neuronal networks for adaptive control of walking machines. In: *Robotics and Autonomous Systems* (1997)

- [50] WEERTS, Sören ; KRUSE, Judita: *Linux für eingebettete Systeme - RTLinux und Performance*, Universität Koblenz-Landau, Seminararbeit, 2004
- [51] WHITNEY, Daniel E.: Historical Perspective and State of the Art in Robot Force Control. In: *The International Journal of Robotics Research* Bd. 6, 1987, S. 3–14
- [52] WLOKA, Dr.-Ing. Dieter W.: *Roboter Systeme 1*. Springer-Verlag, 1992