Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jan Ruhnke

Entwicklung und Realisierung eines vierbeinigen USAR-Roboter-Laufsystems

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jan Ruhnke

Entwicklung und Realisierung eines vierbeinigen USAR-Roboter-Laufsystems

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. rer. nat. Gunter Klemke
Zweitgutachter: Prof. Dr. Ing. Andreas Meisel

Abgegeben am 02.08.2011

Thema der Bachelorarbeit

Entwicklung und Realisierung eines vierbeinigen USAR-Roboter-Laufsystems

Stichworte

Vierbeiniger Roboter inspiriert durch Säugetiere, Search and Rescue Robotics

Kurzzusammenfassung

In dieser Arbeit wird ein Laufsystem mit vier Beinen entwickelt und realisiert. Dieses Laufsystem ist für unebenes Gelände in einem USAR Szenario konzipiert. Das Laufsystem besteht aus dezentralisierten Beinen für einen Roboter in der Größe eines Schäferhunds. Diese Beine verhalten sich autonom und lösen rudimentäre Probleme. Kollisionen und mechanische Fehler werden erkannt und bearbeitet. Das Laufsystem ist robust und durchsetzungsfähig.

Title of the paper

Development and implementation of a four-legged USAR robot walk system

Keywords

Quadruped robot inspired by mammals, Search and Rescue Robotics

Abstract

In this bachelor thesis, a running system is developed and implemented with four legs. This walk system is designed for rough terrain in a USAR scenario. The walking system consists of decentralized legs for a German shepherd sized robot. These legs are autonomously and independently. Each leg solves rudimentary problems. Collisions and mechanical failures are detected and processed. The running system is robust and assertive.

Danksagungen

Diese Arbeit wurde durch meine Familie erst ermöglicht. Herzlichen Dank für die Geduld!
Auch möchte ich mich für die Geduld bei meinem Betreuer und Professor Gunter Klemke bedanken.
Mein Professor Herr Andreas Meisel hat meine Sicht für den „einfachen Weg“ durch seine „Praktiker Formeln“ in den Seminaren geebnet.
Besonders bedanke ich mich auch bei meinem Vater, der mir mit Rat und Tat immer zur Hilfe geeilt ist wenn ich mit der Mechanik nicht mehr weiter kam. Auch bei Herrn Bernd Bethke aus der Zentral Werkstatt der HAW, weil Baumarktwerkzeug nicht alles kann.
Diese Arbeit wurde durch folgende Firmen mit Material gesponsert:
Firma Hoffmann + Krippner GmbH. Herr Hans-Jürgen Horst hat dies unkompliziert ermöglicht.
Firma Distrelect GmbH.
Firma Ott GmbH & Co KG. Herr Manfred Ott war prompt zur Stelle.
Firma Microsoft Deutschland GmbH und besonders die Microsoft Student Partners.
Ohne Ihre / Eure Unterstützung wäre diese Arbeit finanziell und zeitlich gescheitert.

Inhaltsverzeichnis

1 Einführung

1.1 Vorwort

Diese Arbeit beschäftigt sich mit einem sehr kleinen Teil der Entwicklung von autonomen Robotern. Diese kleine Teilentwicklung stellt ein Laufsystem vor. Dieses Laufsystem ist die Grundlage für einen vierbeinigen Roboter, der die Größe eines Schäferhundes hat. Seit der Katastrophe von Fukushima stehen vierbeinige Roboter im Fokus der Fachpresse, da diese Roboter als Helfer agieren können. In den folgenden Kapiteln, werden die möglichen Vorteile und Probleme eines Laufsystems im USAR-Szenario diskutiert.

Umfang

Bedingt durch den Umfang einer Bachelorarbeit wird nur ein Teil der Designentscheidungen diskutiert. In dieser Arbeit wird ein komplettes Laufsystem beschrieben, aber aus finanziellen Gründen wurde nur ein Bein realisiert. Aus den Erkenntnissen dieser Arbeit wurden die anderen Beine simuliert. Die Betrachtungen enden mit der Koordination von einzelnen Schritten des Laufsystems.

Konzepte

Die Designentscheidungen wurden stark durch andere Konzepte beeinflusst. Die Mechanik orientiert sich an der Natur und verfolgt angepasste Konzepte [Nac10], [Sic08], [Jan10] der Bionik. Das Grundkonzept des Softwaredesigns wurde durch mehrere Quellen [Bro86], [Bro87], [Rod87], [Sic08] beeinflusst. Diese Grundkonzepte wurden an die Gegebenheiten und mit eigenen Ideen angepasst. Teilweise werden die Quellen so stark vermischt, dass eine einzelne Quelle nicht mehr erkennbar ist. Zudem wurden einige Verfahren aus der professionellen Spieleentwicklung [Juc10] verwendet. Diese Verfahren sind nicht wissenschaftlich belegt.

Übersicht

Diese Arbeit ist in sechs inhaltliche Schwerpunkte gegliedert. Das Kapitel 1 beschäftigt sich kurz mit der Zielsetzung dieser Arbeit. In Kapitel 2 werden die Vor- und Nachteile eines Laufsystems erläutert. Es werden erste Verhaltensweisen entwickelt. Kapitel 3 stellt den mechanischen Aufbau und die Realisierung der Beine vor. Zudem wird der Grundaufbau des kompletten Roboters angeschnitten, um ein schlüssiges Gesamtkonzept darzustellen. Das Hardwaredesign wird in Kapitel 4 vorgestellt. Der größte Teil dieser Arbeit wird vom Kapitel 5 eingenommen. Hier werden die Softwaregrundlagen für ein dezentrales Laufsystem aufgezeigt. Das Unterkapitel 5.3.1 prüft die Machbarkeit einer dezentralen inversen Kinematik auf einer 8Bit MCU. Im Unterkapitel 5.3.5 wird das komplette System getestet und erste Punkte für die Weiterentwicklung aufgezeigt. Eine kritische Bewertung dieses Laufsystems findet im Kapitel 6 statt. Dieses Kapitel bündelt auch die Ansatzpunkte für eine Weiterentwicklung des kompletten Roboters.

1.2 Motivation

Autonom agierende Roboter werden in naher Zukunft einen bedeutenden Anteil im internationalen Massenmarkt bekommen. Über den Einsatzzweck und Nutzen wird schon lange gestritten. Unstrittig ist der Einsatzzweck in für Menschen gefährlichen Umgebungen. Roboter werden heute nicht nur in der Industrie eingesetzt, sondern auch in Anti-Terroreinsätzen zum Bombenentschärfen oder in Katastrophengebieten, beispielsweise nach Erdbeben oder Atomreaktorunfällen. Diese Roboter sind heute meistens keine autonomen Systeme. Als Helfer im Katastrophenfall hat sich eine eigene Disziplin der Robotik gebildet. Diese autonomen Robotersysteme werden als „Urban Search And Rescue (USAR) Robots“ bezeichnet. Die Hauptaufgabe dieser Roboter soll es sein, verletzte oder eingeklemmte Personen autonom aufzufinden und die Position einer zentralen Leitstelle zu melden. Eine Entwicklung in diesem Bereich wurde von meinen Kommilitonen Björn Bettzüche [Bet10], David Schnell und mir als „Project AMEE“(1.4) im Jahr 2009 begonnen.

1.3 Problemstellung

Das Hauptproblem heutiger USAR Roboter ist die autonome Mobilität in unebenem Gelände. Viele Roboter können sich nur auf befestigtem Untergrund fortbewegen. Der erste Entwicklungs-schwerpunkt ist ein Fortbewegungssystem (Laufsystem), bei dem Agilität, Robustheit und Kosteneffizienz abgewogen werden müssen. Die Mechanik, Elektronik und Software wird für den USAR Einsatzzweck entwickelt.

1.3.1 Umweltbedingungen

Ein USAR Roboter ist Outdoor-Bedingungen ausgesetzt. Diese Bedingungen entsprechen größtenteils denen im Automobilbau. Es treten Temperaturen für geschützte elektronische Hardware von -45°C bis 80°C bei einer Luftfeuchtigkeit bis zu 80% auf. Es muss mit Wasser und anderen Verunreinigungen gerechnet werden. Der Roboter muss starke Erschütterungen überstehen.

1.4 Projekt AMEE

Das Projekt AMEE[1] (**A**utonomous **M**apping **E**xploration and **E**vasion) beschäftigt sich mit der Entwicklung eines autonomen USAR-Roboters (AMEE, *Abbildung 1-1*), der in einem Katastrophenfall den Helfern vor Ort die gefährliche Suche nach Überlebenden erleichtert. Dabei liegt der Schwerpunkt auf der Mobilität in rauem und unebenem Gelände, wie beispielsweise zerstörte Gebäude nach einem Erdbeben. Der Roboter (AMEE) ist als Sensoren Plattform konzipiert und ermöglicht vielfältige Einsatzmöglichkeiten.

Abbildung 1-1 Simulation von AMEE im Einsatzgebiet

AMEE soll dabei von einem Startpunkt einen entfernten Zielpunkt erreichen, den Weg kartographieren und die Umwelt erfassen. Dabei sollen, je nach Einsatzparametern, relevante Informationen gesammelt und Objekte untersucht werden. AMEE soll diese Daten an einen Leitstand übermitteln. Ist keine Funkverbindung möglich, bewegt sich der Roboter zu einem Punkt der eine Funkverbindung ermöglicht.
Als Fortbewegungssystem wurde ein Laufsystem gewählt und wird in der vorliegenden Arbeit vorgestellt. Die Designentscheidung für ein Laufsystem wird im Kapitel 2 genauer untersucht.

Die Zielsetzung ist es, Erkenntnisse zu sammeln, um einen Roboter zu realisieren der für einen realen USAR-Einsatz genutzt werden kann.

[1] Der Name *A.M.E.E.* wurde erstmals in dem Kinofilm „Red Planet“ [Ant00] benutzt. In einer entfernten Vision soll eine Weiterentwicklung von *AMEE*, die Agilität der Kinovorlage erreichen.

1.5 Andere Projekte

Bevor das Laufsystem von AMEE diskutiert wird, werden andere (weiter fortgeschrittene) Roboter mit Laufsystemen vorgestellt und mit den Zielen von AMEE verglichen. Sie zeigen die Grundzüge und teilweise auch die Schwierigkeiten bei der Entwicklung von auf.

1.5.1 Honda ASIMO

Abbildung 1-2 Honda ASIMO [Hon07]

Der Roboter ASIMO ist ein autonomer Roboter, der mit Menschen agieren soll. Laut Honda ist das Entwicklungsziel [Hon071] einen Partner / Freund für Menschen zu schaffen, der sich in das soziale Umfeld integrieren kann. Für ASIMO wurde die Größe aus psychologischen Gründen auf 120cm Höhe festgelegt. Er ist ein bipedaler dynamischer Läufer (2.4.1). Das Projekt begann 1987 mit der Entwicklung eines Laufsystems, dass das menschliche Laufen nachahmt. Im Jahr 2000 wurde der Roboter ASIMO der Öffentlichkeit vorgestellt. Bis heute wird der Roboter weiterentwickelt [Jan10] und soll in einigen Jahren als kommerzielles Produkt vermarktet werden.

Das zweibeinige Laufsystem und die Zielsetzung bei ASIMO decken sich nicht mit dem Projekt AMEE. Aber ASIMO hat einige grundlegende Probleme und Lösungen aufgezeigt, zum Beispiel bei den Antrieben. Für sicheres zweibeiniges Laufen darf das Laufsystem kein Spiel in den Antrieben besitzen. Hierfür wurden spezielle sogenannte „verspannte Getriebe“ entwickelt [Jan10]. Diese sind, wie bei AMEE, Gleichstromgetriebemotoren.

ASIMO besitzt keine Sensoren in den Füßen [Jan10]. Die Kollisionskontrolle erfolgt über ein Verfahren, das auf der „Anti-Pitch-Detection“ [Hon07] (5.3.4.3.3) basiert.

Honda entwickelte für ASIMO ein dezentrales Steuerungssystem, das im Aufbau einem Verteilten System [Jan10] gleicht. Einige zeitkritische Aufgaben wurden in spezialisierte Prozessoren ausgelagert [Jan10]. Der Hauptkontroller ist ein Multiprozessorsystem mit eigener Lastverteilung [Jan10].

1.5.2 Boston Dynamics Big Dog

Abbildung 1-3 Boston Dynamics Big Dog [Bos10]

Der Roboter „Big Dog“ der Firma Boston Dynamics ist ein sog. „rough-terrain robot”. Dieser Roboter wurde für das „Tactical Technology Office” der DARPA (USA) entwickelt. Seine Aufgabe soll es sein, Einsatzkräfte mit Equipment zu versorgen. Der „Big Dog“ ist einen Meter lang und hat eine Schulterhöhe von ca. 75cm. Er ist ein dynamischer Läufer und für die schnelle Fortbewegung im Gelände konzipiert. Auch wenn die Zielsetzung des Laufsystems zum Projekt AMEE sehr ähnlich ist,

unterscheidet sie sich in vielen Punkten. Die Antriebe in den Beinen werden hydraulisch betrieben. „Big Dog“ soll nicht wie AMEE erkunden, sondern schnell zu einem Zielpunkt gelangen. Leider sind durch den militärischen Hintergrund der Entwicklung keine Details über die Software bekannt. In der Weiterentwicklung des Projekts AMEE könnten viele Lösungen von „Big Dog“ als Ideengeber genutzt werden.
Das Ziel der „Boston Dynamics“ Entwickler deckt sich, trotz anderer Ansätze mit dem Projekt AMEE:

„The ultimate goal for BigDog is to develop a robot
that can go anywhere people and animals can go.” [Bos10]

1.5.3 Projekt ARAMIES

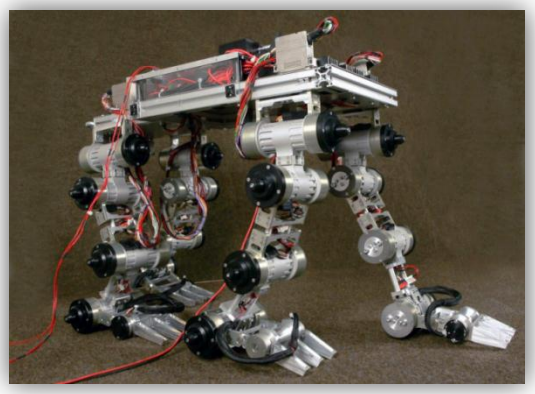

Abbildung 1-4 Projekt ARAMIES

Das Projekt ARAMIES ist eine Entwicklung des „Deutschen Forschungszentrums für Künstliche Intelligenz (DFKI GmbH)“ in Zusammenarbeit mit der Universität Bremen. Das Projekt ARAMIES und das Projekt AMEE ähneln sich sehr stark. Die Dezentralisierung der Steuereinheiten verfolgt einen ähnlichen Ansatz wie der Honda ASIMO (1.5.1). Die Konstruktion der Beine ist, vom Grundaufbau dem Projekt AMEE ähnlich. Auch die Planung der Software ähnelt in Ansätzen.
“*Das ARAMIES-Projekt beschäftigt sich mit der Entwicklung eines Laufroboters, der in schwierigstem Gelände, vor allem in unebenen oder steilen Bereichen, autonom operieren kann… Der vierbeinige ARAMIES-Roboter besitzt 26 Gelenke: sechs pro Bein und zwei für die Bewegung des Kopfes. … Ein großer Vorteil des ARAMIES Roboters im Vergleich zu anderen Laufrobotern ist sein aktuierter Fuß, der es ihm ermöglicht, in steilem Gelände sicheren Halt zu finden. In Labortests war der Roboter in der Lage, eine Sprossenwand mit einer Steigung von 70° zu bewältigen. Jeder Fuß ist ausgestattet mit fünf Drucksensoren sowie einem nach unten gerichteten Infrarot-Abstandssensor zur zuverlässigen Feststellung des Bodenkontaktes. Die modulare Steuerungs- und Leistungselektronik besteht aus einem PC104-System für komplexe Navigations- und Planungsaufgaben, einer MPC565/FPGA-Platine für die reaktive, verhaltensbasierte Kontrolle und fünf Motorsteuerungs-platinen, die von je einem FPGA gesteuert werden. Jede dieser Platinen kann bis zu sechs Motoren gleichzeitig ansteuern und alle analogen Sensorsignale eines Beines einlesen. Unter Benutzung von LVDS-Kommunikation können bis zu acht der Motorsteuerungsplatinen an die MPC565/FPGA-Platine angeschlossen werden. Die bioinspirierte Software steuert die 26 Gelenke simultan mit zentralen Mustergeneratoren und Reflexmodellen, was eine wesentliche Verbesserung im Vergleich zur üblichen modell-basierten Robotersteuerung darstellt und den nötigen Rechenaufwand zugunsten hoher Energieeffizienz stark minimiert*.“ [Spe07]
Starke Unterschiede werden erst im Detail deutlich. ARAMIES ist für Einsätze in der Raumfahrt konzipiert worden. Dadurch sind die Größe und das Gewicht stark begrenzt (2.4.2). Es werden keine industriellen Serienbauteile verwendet. Alle mechanischen und elektronischen Komponenten sind Spezialanfertigungen. ARAMIES verwendet keinen kommerziellen Softwareunterbau (OS, Framework). Das Projekt wurde im April 2007 beendet.

2 Laufsystem

In diesem Kapitel werden die Designentscheidungen *für* ein Laufsystem des Roboters AMEE (1.4) erörtert. Danach erfolgen Abwägungen über das Grundkonzept eines Laufsystems. Im Abschnitt 2.4.2 wird versucht, erste technische Spezifikationen abzuschätzen. Mit diesen Daten werden Laufrhythmen für unebenes Gelände entwickelt (2.5).

2.1 Anforderungen an das Fortbewegungssystem

Um autonomen Robotern Mobilität zu ermöglichen, werden als erstes der Einsatzzweck und die Alternativen betrachtet. In der Robotik hat sich noch kein Fortbewegungssystem als allgemeintauglich erwiesen. Verglichen mit dem Automobilbau wird es immer spezielle Lösungen für unterschiedliche Anforderungen geben.
Das gesuchte Fortbewegungssystem für AMEE soll folgende Anforderungen erfüllen:

a. Der Roboter soll sich im extremen Gelände bewegen können, wie Landformationen im Hochgebirge.
b. Der Roboter soll beispielsweise über Geröllhalden steigen können.
c. Der Roboter soll langsam ein Gebiet erkunden.
d. Der Roboter soll immer in einem sicheren Zustand sein.
e. Der Roboter soll jederzeit gestoppt werden können.
f. Der Roboter soll sich von jedem Punkt fortbewegen können. (Ausgenommen Wasser.)

Einige dieser Forderungen sind absichtlich zu hoch gewählt worden, um das ideale Laufsystem zu finden.

2.2 Fortbewegungssysteme

Für einen USAR-Roboter-Fortbewegungssystem wären Räder, Omni-Wheels, Kettenantriebe oder Beine möglich. Fliegende USAR-Roboter werden nicht betrachtet[2]. Die Fortbewegungssysteme werden im Stand und bei langsamer Fahrt betrachtet.

Räder:
Radsysteme sind sehr energieeffizient, da sie das Gewicht des Roboters auf den Achsen tragen. Es müssen nur die Beharrungskräfte des Roboters und die Walkarbeit der Räder überwunden werden [Gro11]. Radsysteme mit drei oder mehr Rädern befinden sich immer in einem sicheren Zustand (2.4.1)([Sic08] ab Seite 1115, ab 404). Aus diesem Grund werden Radsysteme bei Robotern auf Langzeitmissionen, wie beispielsweise in der Raumfahrt eingesetzt. Die aktuelle Marserkundungs-Mission der NASA verwendet ein Fahrsystem (*Abbildung 2-1*) und verdeutlicht das ein Radsystem immer noch Forschungsgegenstand ist [Nat11].

[2] Flugsysteme wurden aus Kostengründen nicht betrachtet. Kostengünstige Flugsysteme haben eine zu geringe Nutzlast für die Zielsetzung von AMEE([Sic08]z.B Seite 1158).

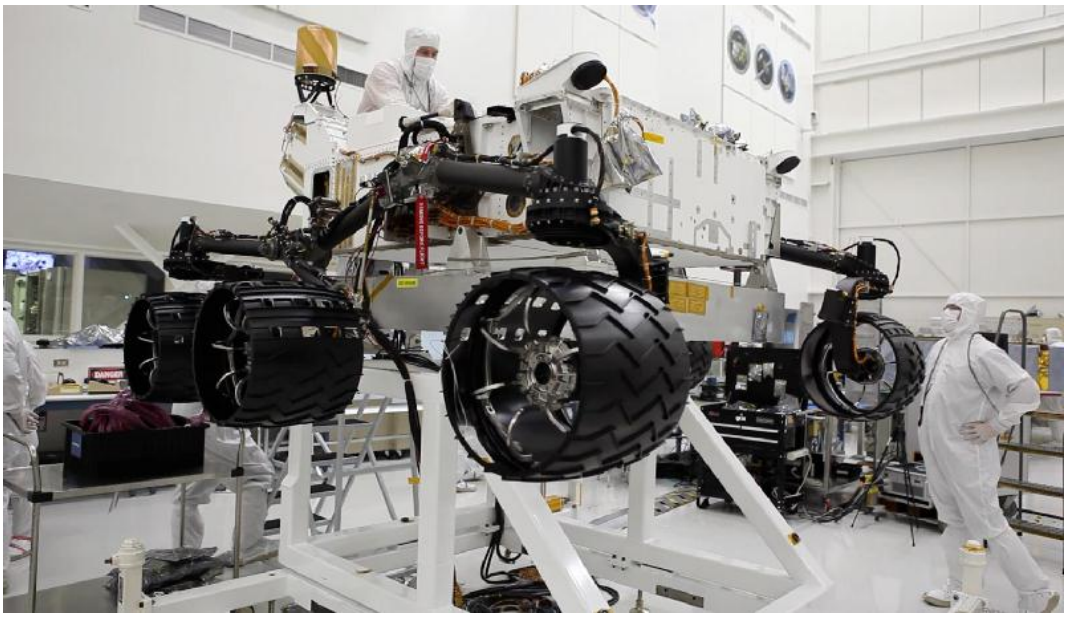

Abbildung 2-1 Mars Rover "Curiosity" [Nat11]

Das Einsatzgebiet von Radsystemen beschränkt sich aber auf ein Gelände, was mit heutigen geländegängigen Automobilen erkundet werden kann [Nat11]. Räder neigen zum Schlupf auf losem Untergrund, durch ihre relativ kleine Auflagefläche([Sic08]ab Seite 1054,) [Wet96]. Wie auch immer Anordnung und Durchmesser der Räder gewählt werden, gibt es Probleme, die im Gelände(2.1a) von einer autonomen Maschine schwer gelöst werden können([Sic08] ab Seite 403). Räder sind aus diesen Gründen für den Roboter AMEE als Fortbewegungssystem ungeeignet.

Omni-Wheels:

Omnidirektionale Räder (Omni-Wheels) sind ein zweiter Ansatz. Besonders die Abwandlung, das sogenannte Mecanumrad (*Abbildung 2-2*), bietet sich für einen Roboter an. *„…Diese Räder bestehen aus einer Felge, auf der unter einem Winkel von 45° lose, ballige Rollen so angebracht sind, dass sie über den Abrollumfang wieder einen exakten Kreis bilden. Durch die Schräganordnung der Rollen entstehen beim Antreiben des Rades 2 Kraftkomponenten. Gegeneinander gerichtete Kräfte der einzelnen Räder werden über die Achsen und den Rahmen kompensiert. Die übrigen Kräfte addieren sich zur resultierenden Fahrtrichtung. Auf diese Weise kann durch entsprechendes Ansteuern der einzelnen Räder bezüglich Drehrichtung und -geschwindigkeit jedes beliebige Fahrmanöver erzeugt werden.“* [MIA10]

Abbildung 2-2 Mecanumrad [MIA10]

Der Energiebedarf von Systemen mit Omni-Wheels ist vergleichbar mit Radsystemen [Gro11]. Aber Omni-Wheels bietet keinen Halt, wenn der Roboter sich auf einer Schrägen bewegen soll. Es kann kleine Steigungen nur in Fahrtrichtung überwinden [MIA10]. Für unebenes Gelände sind Omni-Wheels nicht verwendbar([Sic08]ab Seite 304, Seite 407).

Kettenantreibe:

Kettenantriebe bieten eine hohe Mobilität im Gelände, da das Fahrzeuggewicht auf eine große Fläche verteilt wird. Das Problem des Schlupfs von Rädern auf losem Untergrund wird damit umgangen([Sic08]ab Seite 1055) [Wet96]. Die Antriebsräder sind bei vielen Kettenfahrzeugen erhöht angeordnet, damit sich das Fahrzeug über kleinen Hindernissen schieben kann [Gro11]. Spitze Gegenstände werden durch beweglich gelagerte Sekundärräder innerhalb der Kette ausgeglichen [Sic08]. Der Energiebedarf von kettengetriebenen Systemen ist höher als bei Radsystemen, da die Kette mehr Reibung im Antriebssystem erzeugt [Sup10]. Roboter mit einem Kettensystem, wie beispielsweise der „HD2 SWAT / EOD Tactical Treaded Robot“ (*Abbildung 2-3*) der Firma SuperDroid Inc., erreichen eine Geschwindigkeit von 1,5m/s.

Abbildung 2-3 HD2 SWAT / EOD Tactical Treaded Robot, Bild © 2010 SuperDroid

Die Leistungen von Kettenantreiben erfüllen fast die Anforderungen des Projekts AMEE(1.4), aber bei extremen Steigungen auf Geröll oder Felsen ergeben sich oft nur zwei oder drei Auflagepunkte pro Kette [Wet96]. Kettengetriebene Roboter überwinden Schrägen mit losem Untergrund, durch hohe Geschwindigkeiten [Sup10]. Ein Anfahren dieser kettengetriebenen Roboter auf einer rutschigen Schräge führt immer wieder zu Problemen und ist nicht zuverlässig genug [Wet96]. Dieser Fall tritt bei einem USAR Roboter oft ein, da beispielsweise eine Treppe mit Geröll verschüttet ist, und der Roboter sich neu orientieren muss. Aus diesem Grund muss es dem Roboter möglich sein, auf einer Schrägen auch anzuhalten und dann wieder zuverlässig seinen Kurs fortzusetzen. Der Schwachpunkt von Kettenantrieben tritt in einem USAR Szenario regelmäßig auf und damit können auch Kettenantreibe nicht vorbehaltlos verwendet werden.

Beine (Laufsystem):

Beine bzw. Füße können exakt auf einen Punkt platziert werden. Es ist möglich, über ein Loch oder Hindernis zu schreiten ohne dies zu berühren. Abstrakt betrachtet, bewegen sich Fahrsysteme auf Linien, die das Gewicht des Roboters tragen müssen[3]. Laufsysteme belasten den Untergrund nur punktuell [Wet96] [Bru08]. Damit kann das Laufsystem selbst einen tragenden Untergrund gezielt auswählen und im Vorfeld auf Tragfähigkeit testen (5.3.4.4.11). Da gezielt ein Angriffspunkt für die tragenden Kräfte ausgewählt wird, kann ein Laufsystem steigen. Ein steigfähiger Roboter unterliegt nicht den Grundsätzen von Schlupf, um sich fortzubewegen. Der Vortrieb beim Steigen wird durch Hebelwirkung der Beine erzielt [Wet96]([Sic08]ab Seite 378, ab Seite 1067).

Der Nachteil von Beinen ist der Konstruktions- [Gro11] und Implementierungsaufwand. Um einen Fuß zu positionieren wird ein relativ komplexes Verfahren benötigt [Bru08]. Gegenüber anderen Fortbewegungssystemen ist der Energieverbrauch eines Laufsystems höher, da mehr Antriebe benötigt werden([Sic08] Seite 386). Beine tragen die Masse des Roboters durch eigenen Kraftaufwand. Ein Laufsystem muss in der Lage sein, sein eigenes Gewicht zu heben.

Auswertung:

Die vier betrachteten Fortbewegungssysteme werden anhand einer gewichteten Tabelle (*Abbildung 2-4*) bewertet.

-	Räder	Omni Wheels	Ketten	Beine	Gewichtung
Energiebedarf	10	9	8	5	5
Verschleiß	10	2	7	4	1
Geschwindigkeit	10	6	8	4	1
Sicherer Zustand	10	2	10	10	10
Agilität im Gelände	4	0	8	10	10
Extreme Agilität	0	0	4	10	10
Aufwand Mechanik	10	8	7	5	5
Aufwand Software	6	10	8	4	2
Kosten	10	5	8	4	7
Summe	342	168	382	394	-

Abbildung 2-4 Bewertung der Fortbewegungssysteme

Die Punkte sind von 0 bis 10 vergeben worden, wobei *0 = schlecht* und *10 sehr gut* entspricht. Die Gewichtung orientiert sich am USAR-Einsatzszenario für den Roboter AMEE (*0 = unwichtig, 10 = wichtig*). Die Betrachtung geht von vier Rädern mit Allrad, drei Omni-Wheels, zwei Ketten und vier Beinen als statischer Läufer(2.4.1) aus. Der Energiebedarf des Systems wurde über einen Zeitraum von einer Stunde geschätzt. Der Verschleiß des Laufsystems ist niedrig gewichtet, da in einem USAR-Szenario das Equipment nur kurzeitig belastet wird [Bru08]. Die Hauptkriterien für die Gewichtung sind die „Agilität im Gelände“ und die „extreme Agilität“, mit der die mögliche hohe Steigfähigkeit verdeutlicht wird.

Das Laufsystem mit Beinen ist für den Einsatzzweck am besten geeignet, dicht gefolgt vom

[3] Dieser Zusammenhang ist besonders auf rutschigen Geröllhalden und an Abbruchkanten von Bedeutung. Zudem können gefährliche Objekte, wie uninsolierte Kabel „übergangen“ werden.

Kettensystem. Diese Auswertung deckt sich mit der Betrachtungen anderer Untersuchungen [Wet96] [Sic08], wobei das Kettensystem in dieser Arbeit deutlich positiver bewertet wird.
Als Schlussfolgerung wird der Roboter AMEE mit einem Laufsystem realisiert.

2.3 Anzahl der Beine

Die Anzahl der Beine ist eine weitere Abwägung. Es wären zwei, vier oder sechs Beine denkbar. Auch hier ist wieder der Einsatzzweck entscheidend. Die Betrachtung konzentriert sich auf den Energiebedarf, Aufwand und vor allem auf die Einhaltung des stabilen / sicheren Zustands(2.4.1).

Zweibeiner

Ein bipedales Laufsystem (Zweibeiner) ist in allen Konstruktionsvarianten fast nie in einem stabilen Zustand, da der Schwerpunkt beim Laufen fast immer die Auflagefläche verlässt (dynamischer Läufer, 2.4.1) [Bru08]. Vorteil des Zweibeiners ist der geringere Energiebedarf [Hon07]. Dies ist aber nicht bedingt durch die Anzahl der Antriebe, sondern durch den möglichen Laufstil (2.4.1, ZMP).
Wie der Roboter ASIMO [Hon07] (1.5.1) zeigt, ist der Software-Entwicklungsaufwand für dynamisches Laufen sehr hoch. Honda ASIMO kann nicht klettern. Leichte Unebenheiten im Boden und Treppen werden zwar vom „I-Walking“ [Hon07] überwunden, aber die Zielsetzung der Honda Entwickler ist keine Outdoor- bzw. USAR Umgebung. Die Erfahrungen des HRI [Jan10] zeigen, das ein zweibeiniges Laufsystem für ein USAR Szenario ungeeignet ist. Einfache Beobachtungen in der Natur und am Menschen verdeutlichen dies. Bewegen Menschen sich auf schrägem und rutschigem Untergrund, gehen sie auf die Knie und benutzen zusätzlich die Hände.

Vierbeiner

Vierbeinige Roboter können sich sicher im Gelände bewegen, da immer eine Dreipunktauflage gewählt werden kann [Gro11](2.5.1). Das heißt, der Roboter kann laufen, indem er nur ein Bein zurzeit vom Boden hebt([Sic08] ab Seite 378). Ob dieser Zustand immer stabil und sicher ist, hängt von der Beinlänge und der Lage des Schwerpunkts ab([Sic08]ab Seite 378)(2.4.1). Ein Vierbeiner ist ein guter Kompromiss zwischen Energiebedarf und Stabilität im Gelände([Sic08] ab Seite 385) [Wet96].

Sechsbeiner

Sechsbeinige Laufsysteme für Roboter bilden meist das Laufverhalten von Insekten nach([Sic08]ab Seite 380). Die Konstruktion der Beine kann auf unter vier Freiheitsgrade vereinfacht werden [Wet96] [Sic08]. Ab einem bestimmten Gewicht wird aber der Energieverbrauch der Antriebe im Verhältnis zum Nutzen ineffizient [Sic08]. Kontrovers zu diesem Thema stehen die Arbeiten von David Wetterberg, der ein Laufsystem realisiert [Wet96](Projekt Amber) hat, das einen geringeren Energieverbrauch aufweist als ein Fahrsystem. Da dieses Laufsystem einen anderen Einsatzzweck hat, wird dieses nicht weiter betrachtet.
Paradoxerweise sind sechsbeinige Laufsysteme für kleine, leichte Roboter (unter 1kg) und für sehr große Roboter (über 1.000kg) gut geeignet [Sic08].
Die Stabilitätsbetrachtung, des Vierbeiners treffen auch auf den Sechsbeiner zu. In einigen extremen Situationen im Gelände bietet ein Sechsbeiner, durch das zusätzliche Beinpaar mehr halt. Dies könnte sich durch das höhere Gewicht wieder als nachteilig erweisen [Sic08].
Die Vor- und Nachteile von Vier- und Sechsbeinern werden in der Wissenschaft kontrovers diskutiert. Es kann in dieser Arbeit kein stichhaltiger Beweis für oder gegen einen Sechsbeiner geführt werden.

Designentscheidung:
Die obige Betrachtung setzt wieder das USAR-Szenario voraus. Ein Zweibeiner ist für den Einsatzzweck ungeeignet.
Das Verhältnis zwischen Energiebedarf und Nutzen bei einem *vierbeinigen* Laufsystem ist für einen USAR Roboter gut([Sic08] ab Seite 385). Zudem ist das Verhältnis zwischen Kosten und Nutzen vertretbar[4].
Die Entscheidung gegen ein sechsbeiniges Laufsystem kann nicht direkt wissenschaftlich begründet werden. Die Entscheidung kann nur über die Ästhetik und den mechanischem Aufwand getroffen werden. Das AMEE Laufsystems soll ein Säugetier nachbilden, da in der Natur ein vierbeiniges Laufsystem ab circa einem Kilogramm Körpergewicht bevorzugt wird [McG97]. Aus diesen Gründen wird für das AMEE Laufsystem ein vierbeiniges Laufsystem verwendet.

2.4 Vierbeiniges Laufsystem

Ein vierbeiniges Laufsystem kann konstruktiv auf einen stabilen Zustand (2.4.1) optimiert werden. Dies wird durch die Lage des Schwerpunkts beeinflusst([Sic08] ab Seite 378). In diesem Unterkapitel werden konstruktive Grundentscheidungen getroffen, die Auswirkungen bis in die Software haben.

2.4.1 Dynamischer Läufer oder statischer Läufer

In der Informatik und Bionik werden zwei Arten von Laufsystemen unterschieden([Sic08] ab Seite 361). Diese hängen vom Einsatzzweck des Roboters ab.
Dynamischer Läufer:
Diese Art von Laufsystem hat einen hohen Schwerpunkt. Die Beine sind lang und stehen rechts und links eng zusammen. (Siehe Big Dog, 1.5.2) Der Roboter ist damit in der Lage, zu springen und sehr schnell zu laufen. Der Energiebedarf ist auf glattem Untergrund gering. Beim Laufen fällt der Roboter-Torso ständig in Laufrichtung und wird nur von einem hervorschnellenden Bein abgefangen. Die Hauptarbeit wird von der Schwerkraft erbracht. Dieses Prinzip wird als „Zero-Moment-Point“ (ZMP [Hon07]) bezeichnet. Der ZMP ist der Punkt, an dem der Roboter über den Schwerpunkt in Laufrichtung anfängt zu kippt. Ein ZMP basierendes System muss ständig regeln. Ein Systemausfall bringt den Roboter zum Stürzen. Die Mechanik ist sehr Aufwendig. Die Beine müssen schnell bewegt werden. Bedingt durch die Massenträgheit sollten die Beine bei einem dynamischen Läufer sehr leicht sein. Die Beine müssen aber auch sehr robust sein, da immer harte Schläge auf die Beine einwirken. Diese Anforderungen wiedersprechen sich. Die Leistung der Antriebe steigt fast exponentiell mit dem Gewicht der Beine, da der dynamische Läufer die Masse der Beine schnell beschleunigen muss [Jan10]. Aus diesem Grund verwendet beispielsweise der „Big Dog“ [Bos10] ein hydraulisches Antriebsystem. Im *extremen* Gelände ist ein dynamischer Läufer schneller als ein statischer Läufer, aber der Energieverbrauch steigt auch an, da beim Eintritt in den Zustand des ZMP sofort ein sicherer Halt für den nächsten Schritt „ertastet“ werden muss. Es ist ein sehr genaues Timing erforderlich. Die nötigen Reaktionszeiten für den Kontroller werden kürzer. Das ganze Laufsystem wird aufwendiger. Für einen Software-Kontroller ist es schwer zu erkennen, wann eine Situation kritisch wird [Jan10]. Die Realisierung stellt sehr hohe Anforderungen an die Entwickler und die finanziellen Mittel.

[4] Dies gilt für das hier realisierte Laufsystem.

Statischer Läufer:
Der Roboter befindet sich immer in einem sicheren Zustand. Die Bezeichnung „sicherer Zustand“ stammt aus dem Bereich der verteilten Systeme und der Betrachtung von Multi-Rechnersystemen im Flugzeug- und Fahrzeugbau. Er ist auch hier wie im Flugzeugbau zu verstehen und bezeichnet einen Zustand, in dem eine Abschaltung des gesamten Systems keine physischen Folgen hat. Das vierbeinige AMEE Laufsystem soll immer in einem sicheren / stabilen Zustand bleiben, was mit vier Beinen realisiert werden kann(2.3, Vierbeiner). Dies wird allgemein als statischer Läufer bezeichnet [Sic08]. Der Laufrhythmus kann den sicheren Zustand gefährden, dies wird im Kapitel 2.5 erläutert.
Designentscheidung:
Der Roboter AMEE soll nicht rennen oder springen können. Der Roboter AMEE soll ein Gebiet erkunden und Daten sammeln. Dafür wird es nötig sein, an einigen Stellen zu verharren. Der Roboter soll unter extremen Umweltbedingungen agieren, bei dem es zu unerwarteten Ereignissen (Systemausfälle usw.) kommen kann. Aus diesen Gründen wird der Roboter AMEE ein statischer Läufer. Zudem ist der Entwicklungsaufwand für einen dynamischen Läufer erheblich höher([Sic08] ab Seite 361) [Jan10] [Hon07].

2.4.2 Gewicht

Die Masse des Roboters ist ein ausschlaggebender Faktor bei der Konstruktion der Beine (3.3) und der Entwicklung von Laufmustern / Laufrhythmen (2.5). Es müssen Eckdaten ermittelt werden, um die Spezifikationen des Laufsystems zu ermitteln. Ein autonomer USAR Roboter muss über wesentlich mehr eigene Rechenleistung verfügen als andere autonome Roboter. Ein erstes Konzept wäre, die Rechenleistung extern zur Verfügung zu stellen. Im USAR Szenario sind aber meist keine störungsfreien Funkverbindungen herstellbar. Ein USAR Roboter muss demnach autonom agieren können. Die nötige Rechenleistung für diesen Roboter kann in diesem Entwicklungsstadium nur geschätzt werden. Der Roboter AMEE wird für drei Intel ATOM D510MO ITX Mainboards konzipiert. Das Gewicht von ca. 200g pro Mainboard ist nicht ausschlaggebend.
Für Spezifikationen des Laufsystems gilt folgende logische Kette (*Abbildung 2-5* [Nac10], vereinfacht): Steigt die Rechenleistung, steigt die elektrische Leistungsaufnahme des Roboters. Steigt die Leistungsaufnahme, werden Akkus mit höherer Kapazität benötigt. Akkus mit höherer Kapazität, haben ein höheres Gewicht zur Folge. Ein höheres Gewicht benötigt stärkere Antriebe in den Beinen. Werden die Antriebe stärker, habe auch diese eine höhere Leistungsaufnahme und mehr Gewicht. Damit schließt sich der Kreis.

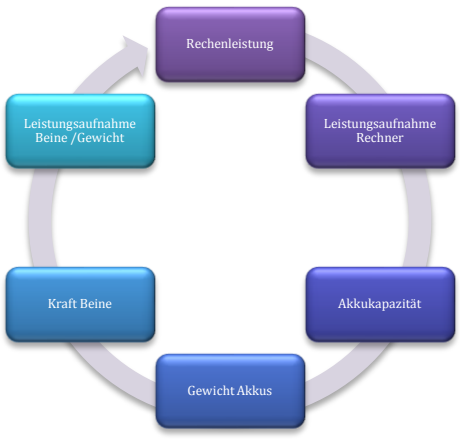

Abbildung 2-5 Leistungs- / Gewichtskreislauf

Die Abschätzung des Gewichts wird über die geschätzte Rechenleistung ermittelt. Die Bild- bzw. 3D Objekterkennung wird einen erheblichen Teil der vorhandenen Rechenleistung nutzen. Ein Intel® D510MO ITX Mainboard hat eine durchschnittliche Leistungsaufnahme, bei

Verwendung von 4GB RAM und Windows CE 6 auf einer SSD von ca. 11W – 25W [Int09]. Die maximale Leistungsaufnahme für die Hauptrechner beträgt ca. 3 x 25W = 75W plus Wandlerverluste. Damit ergeben sich circa 80 Watt maximale Leistungsaufnahme für drei Mainboard mit insgesamt 12 logischen „x64 ATOM“ CPUs. Der Roboter soll mindestens eine Betriebsdauer von 60 Minuten haben. Es werden für die Hauptrechner unter Dauervolllast 7Ah bei 12V benötigt. Die Tests im Kapitel 5.3.5 haben ergeben, dass die Beine eine durchschnittliche Stromaufnahme von 100W auf ebenem Boden haben[5]. Damit kann die Leistung auf ca. 15 - 20Ah bei 12V geschätzt werden.
Die Auswahl der Akku-Technologie wird hier durch den Einsatzzweck bestimmt. Obwohl Li-ion (Lithium Ionen) oder Li-ion Polymer Akkus eine erheblich höhere Energiedichte liefern als andere Technologien, fallen die möglichen Optionen aufgrund des angestrebten Temperaturbereiches klein aus. Unter dem Gefrierpunkt (0°C) liefern nur noch Blei-Gel oder Nickel-Metall-Hybrid Akkus ausreichend Energie. Beim Kosten-/Leistungsfaktor sticht der Blei-Gel Akku hervor, da er eine beachtliche Leistung mit geringen Kosten kombiniert. Die Technologie ist durch den Automobilbau ausgereift und hat sich als robust erwiesen [Gro11]. Leider ist der Blei-Gel Akku auch der schwerste Akku.
Eine Worst Case Überschlagsrechnung für 60 Minuten Betriebszeit wird in Tabelle 2-1 dargestellt:

Beine (realisiertes Bein 4,8kg)	19,2
Akku Panasonic Blei LC1224P 12V 24Ah	8,6
Torso inkl. Abdeckungen geschätzt	3,0
Kopf inkl. Antriebe geschätzt	3,0
Mainboard, Wandler, Switch, Sensoren	1,5
Kabel	0,3
Summe	35,6

Tabelle 2-1 Gewichtsschätzung in kg

In der jetzigen Form hätte der Roboter AMEE eine Masse von ca. 36kg. Als Material wurde AlMg zugrunde gelegt. In der Weiterentwicklung sollen mehr Kohlefaserverbundwerkstoffe genutzt werden, womit das Gewicht auf unter 30kg reduziert werden kann [Gro11]. Bei USAR-Einsatz-szenarien über 0°C Außentemperatur, könnte eine leichtere Akkutechnologie genutzt werden.

2.5 Gleichgewicht und sicherer Zustand

Ein Laufsystem mit vier Beinen kann in mehreren Laufmodi betrieben werden, die an Anforderungen des Untergrunds angepasst sind. Konzeptionell wurden für AMEE zwei Lauf-Modi entwickelt. Der „Fast Static Walk“ (2.5.2) und der „Save Walk“(2.5.1). Die zugrundeliegenden Laufmuster wurden durch Filmsequenzen eines Wolfes inspiriert.

[5] Es wird eine Impulslast von ca. 10% angenommen. Diese Annahme ist eine Abschätzung, um einen ersten Anhaltspunkt für die Entwicklung zu erhalten.

2.5.1 Save Walk

Dieser Laufmodus wird genutzt, wenn der Roboter sich auf unsicherem Untergrund bewegt. Im USAR Szenario wäre dies eine Schutthalde, eine teilweise verschüttete Treppe oder ein eingestürzter Boden. Es wird nur ein Fuß zurzeit angehoben. Die anderen drei Füße bleiben auf dem Boden, damit der Schwerpunkt immer in einem stabilen Bereich bleibt. Der „Save Walk“ ist dem „creeping gaint [Sic08] Seite 379“ ähnlich, wie er im „Handbook of Robotics“ beschrieben wird. Dies wird in *Abbildung 2-6* dargestellt.

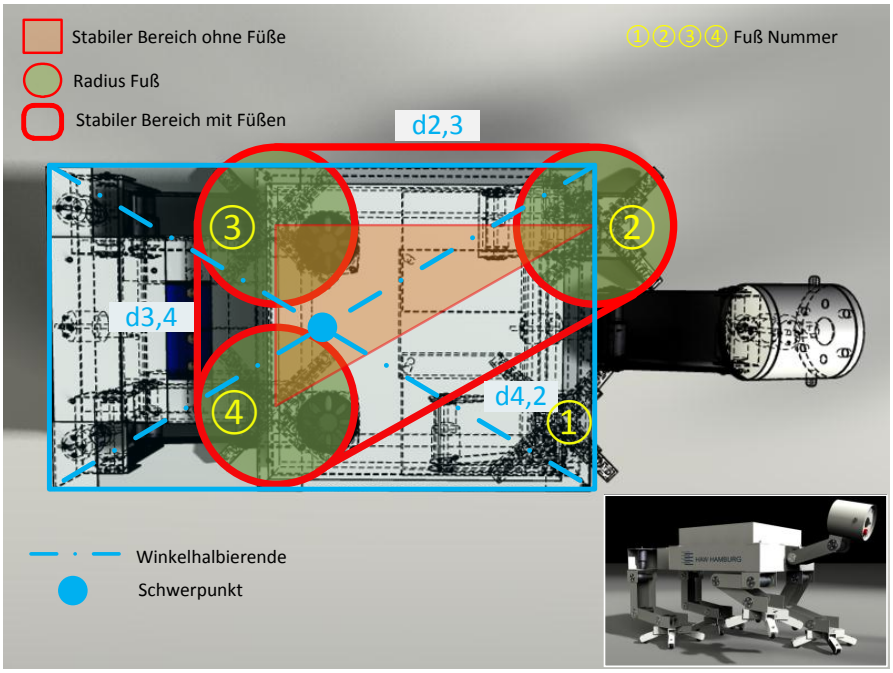

Abbildung 2-6 Save Walk

Die folgenden Betrachtungen des Schwerpunkts sind eine Abschätzung und gehen davon aus, dass das Gewicht gleich verteilt ist. Da AMEE relativ kurze Beine hat, ist der Schwerpunkt in der Seitenansicht nicht kritisch. Der Schwerpunkt liegt unterhalb des Torsos, weil die Beine durch die Antriebe sehr viel Masse haben .

In der Praxis werden die Schwerpunkte der einzelnen Bauteile bestimmt und summiert bzw. subtrahiert. Die realen Neigungswinkel des Roboters sollen mit einem Gyroskop ermittelt werden. Bei Robotern wie AMEE wird der Schwerpunkt in einer Simulation ermittelt und als Funktion in den Kontroller implementiert. Dies ist aber nicht mehr Bestandteil dieser Arbeit und wird in der Weiterentwicklung erarbeitet.

In der *Abbildung 2-6* ist der Fuß ① angehoben. Die anderen Füße stehen fest auf dem Boden und bilden damit ein Dreieck über den Mittelpunkten der Füße. Durch die relativ großen Füße von AMEE wird dieses Dreieck um die Abmessungen der Füße vergrößert. Um vereinfacht den Schwerpunkt des Roboters zu ermitteln, wird der Roboter als einfache geometrische Form betrachtet. Wird der Kopf vernachlässig, ergibt sich ein Rechteck. Liegt der Schwerpunkt in dem Dreieck, das durch die Füße gebildet wird, steht der Roboter stabil und befindet sich in einem sicheren Zustand. Bewegt sich der Roboter über die am Boden stehenden Füße, verschiebt sich dieses Dreieck mit diesen Füßen. Der Schwerpunkt bleibt fest an seiner ermittelten Position im Roboter. Liegt der Schwerpunkt außerhalb

des Dreiecks, kippt der Roboter. Durch die Massenträgheit des Roboters, würde der Roboter langsam anfangen zu kippen. Dies könnte durch eine schnelle Reaktion der Beine abgefangen werden, wie beim Honda ASIMO „i-Walking“ [Hon07]. Das ist aber nicht die Zielsetzung, der Roboter soll jederzeit in einem stabilen Zustand gehalten werden.

Ablauf:
Das Laufschema in *Abbildung 2-6* kann als Sequenz dargestellt werden, bei der die beiden Hinterbeine die Bewegung schon ausgeführt haben. Der vordere rechte Fuß ① befindet sich in der Bewegung zum Zielpunkt, und wird vor den Körper platziert. Fuß ② wartet bis Fuß ① die Bewegung abgeschlossen hat. Ist dies abgeschlossen, folgt Fuß ② auf die gleiche Weise wie Fuß ①. Danach wird der ganze Körper über eine synchronisierte Bewegung (5.3.3.5) aller Beine nach vorn gezogen. Die Sequenz beginnt von vorn mit dem Setzen der hinteren Füße. Die Sequenzfolge der Füße ist: (④; ③; ①; ②; pull; ↺). „Pull“ bedeutet in dieser Sequenz die synchrone Bewegung der Beine.
Wird davon ausgegangen, dass der Roboter sich auf einem losen Untergrund bewegt, ist diese Laufsequenz sicher aber auch langsam. Denkbar wäre auch die Sequenz (④; ②; ③; ①; pull; ↺) [Sic08] bei der das Dreieck zwischen den Beinen möglichst groß gehalten wird, da die Sequenz die Beine über Kreuz anspricht.
Durch diesen Laufrhythmus wird es möglich, den Boden zu prüfen. Werden die Füße bzw. Zehen mit einfachen Kontakten ausgestattet (3.3.1), kann bei jedem Auftreten geprüft werden, ob der Boden tragfähig ist. Im Normalzustand werden die Kontakte am Fuß geschlossen, sobald der Fuß belastet wird. Tritt nun der Roboter auf einen nicht tragfähigen Untergrund, öffnen sich die Kontakte wieder, worauf reagiert werden kann (5.3.4.4.11). Der Fuß kann dann zurückgezogen werden und an einer näheren, Position erneut die Tragfähigkeit prüfen.

2.5.2 Fast Static Walk
Genutzt wird dieser Laufmodus auf relativ ebenem Boden im Gelände. Im „Fast Static Walk“ Mode wird das Laufen gegenüber dem „Save Walk“ durch parallel ablaufende Bewegungen beschleunigt. Zwei Beine bewegen sich gleichzeitig zu ihren jeweiligen Zielpunkten. Synchron dazu wird der Körper von den zwei übrigen Beinen nach vorn gezogen. Hier wird der natürliche Laufrhythmus eines vierbeinigen Tieres nachgebildet. In der Wildbiologie wird dies als „Schreiten“ bzw. „Passgang“ bei einem Wolf bezeichnet [Sch96]. Der „Fast Static Walk“ ähnelt dem „Trot gait“([Sic08] Seite 381).

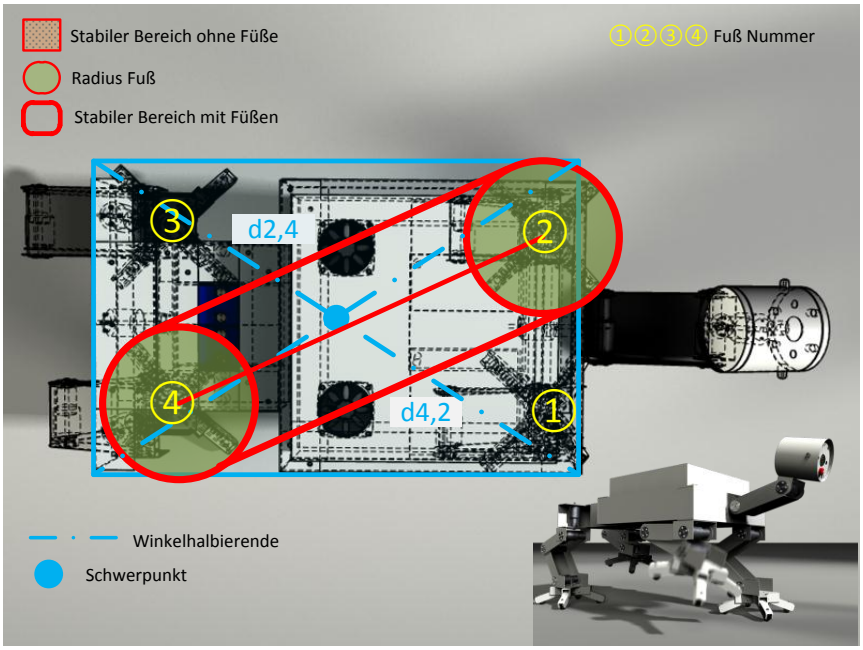

Abbildung 2-7 Fast Static Walk

Werden zwei Füße ① und ③ (wie in *Abbildung 2-7*) gleichzeitig angehoben, ist der stabile Bereich nur eine Linie zwischen den Füßen am Boden. Durch die Größe der Füße wird der sichere Bereich soweit vergrößert, dass der Roboter im stabilen Bereich bleibt. Da der Roboter eine synchrone Bewegung ausführt und die tragende Massen bzw. die in der Luft stehenden Massen symmetrisch verteilt sind, könnte der Roboter das Gleichgewichtsprinzip wie eine Waage nutzen. Streng genommen handelt es sich damit um dynamisches Laufen. In einem USAR Szenario wird aus Stabilitätsgründen auf diese Möglichkeit verzichtet. Wird der stabile Bereich betrachtet, fällt auf dass der Roboter seinen vollen Bewegungsumfang nicht nutzen kann. Würde die volle mechanisch mögliche Schrittweite genutzt, kippt der Roboter zwar nicht, aber wäre wie oben beschrieben in einem nichtsicheren Zustand. Kleinste Unebenheiten im Boden könnten dann schon zum Kippen führen. Aus diesem Grund wird auf ca. 30% der Schrittweiter verzichtet.
In *Abbildung 2-8* wird die Sequenz in einem Gant ähnlichen Diagramm dargestellt, um die Parallelität der Abläufe abzubilden. Das Diagramm stellt die Abläufe der Bild-Sequenz *Abbildung 2-9* dar. Eine unterbrochene Line bedeutet, dass der Fuß in der Luft ist. Eine durchgezogene Linie bedeutet, der Fuß ist auf dem Boden und in Bewegung.

Ablauf:

Fuß ① und ③ sind schon in der vorgreifenden Position platziert. Fuß ② und ④ sind in der hinteren Position, d.h. nach hinten ausgestreckt. Nun werden Fuß ② und ④ angehoben und in die vordere Position bewegt. Gleichzeitig werden Fuß ① und ③ in die hintere Position gezogen, bleiben aber immer auf dem Boden. Der Torso wird von diesen Beinen nach vorn „gezogen“. Nun erfolgt die gleiche Bewegung mit vertauschten Beinpaaren. Es arbeiten immer die Beinpaare (①, ③) und (②, ④) zusammen.

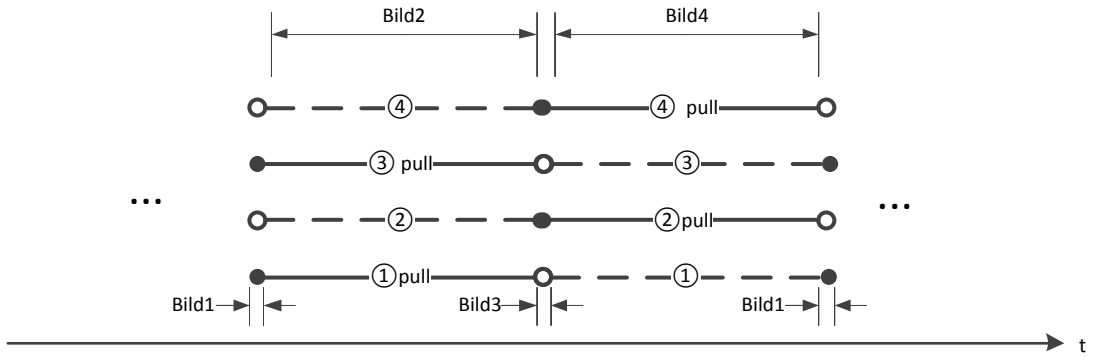

Abbildung 2-8 Fast Static Walk Diagramm Sequenz

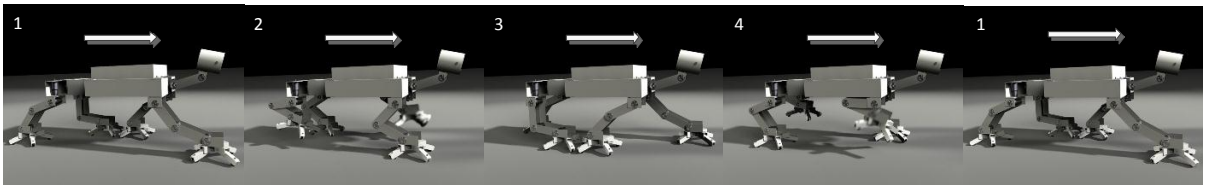

Abbildung 2-9 Fast Static Walk Sequenz

Läuft der Roboter, kommt der inversen Kinematik (5.3.1.1) eine besondere Bedeutung zu.
Der Roboter-Torso soll während des Laufens weitestgehend auf einer Höhe gehalten werden. Eine geplante Bild- bzw. Objekterkennung liefert qualitative hochwertigere Daten, wenn die Kameras im Kopf ruhig und gleichmäßig bewegt werden[6].
Bewegt sich das auf dem Boden stehende Beinpaar direkt von der Anfangsposition in die Endposition, würden die Beine nicht gebeugt werden (5.3.4.3.2). Der Roboter-Torso würde sich in Wellenbewegungen über den Boden bewegen. Als Folge müsste beispielsweise eine Mustererkennung der Bildverarbeitung das fokussierte Objekt in den Bilddaten neu suchen.
Um diese Probleme zu vermeiden, wird die Ansteuerung der Beine verändert, da diese das Problem verursachen (Verursacherprinzip). Um den Roboter-Torso immer auf derselben Höhe zu halten, werden die Füße betrachtet (*Abbildung 2-9*). Die Füße, die auf dem Boden stehen und den Roboter-Torso über den Boden bewegen, müssen eine Bewegung auf einer Geraden beschreiben. Dies wird erreicht, indem dieser Bewegungsablauf in möglichst kleine Einzelschritte (5.3.4.4.4) zerlegt wird. Als Konsequenz daraus, müssen die Beine in schneller Abfolge angesteuert werden, um eine flüssige Bewegung zu erzielen. Ein Hauptproblem ist dabei, dass jede dieser Einzelbewegungen neue Winkel zwischen den einzelnen Gliedern in einem Bein benötigen. Die Winkel müssen durch die inverse Kinematik möglichst schnell ermittelt werden. Ein Verfahren hierfür wird im Kapitel 5.3.1 beschreiben.
Durch dieses Vorgehen wird der Roboter-Torso auf einer Höhe über dem Boden gehalten und bewegt. Es wird der Bewegungsablauf in *Abbildung 2-9* erzielt, bei dem die Beine in Bild 2 und 4

[6] Auch jagende oder suchende Tiere wechseln in dieses Verhalten um ein Ziel zu fokussieren [McG97]. Tiere die nicht jagen, wählen ein energieeffizienteres Verhalten [McG97]. Eine mögliche Analogie wäre, das AMEE Daten „jagt“.

eingeknickt sind.
Die Betrachtung aus energetischer Sicht wird hier nicht berücksichtigt. Bei diesem Verfahren werden in schneller Folge die Antriebe beschleunigt und abgebremst, um den Bewegungsablauf auf dem Boden einzuhalten. In der Realisierung konnte kein geringerer Energiebedarf ermittelt werden. Es besteht die Möglichkeit, dass dieser Sachverhalt durch den verwendeten Antriebstyp (3.3.2) verursacht wird. Mehr dazu im Fazit (6.4, Untersetzung).

2.5.3 Besondere Bedingungen für ein USAR Roboter Laufsystem

Bewegungsablauf

Ein Laufsystem für unebenes Gelände muss über genügend Bodenfreiheit verfügen, damit in einem „Fast Static Walk“ Laufmodus (2.5.2) nicht zu früh in einen Klettermodus (Save Walk, 2.5.1) gewechselt wird. Dafür wird bei jedem neuen Schritt eines Beins der Oberschenkel nach hinten und an den Torso gezogen (5.3.4.4.10). Der Unterschenkel hat damit genügend Raum, um nach vorn gedreht zu werden. Es bleibt ein relativ großer Freiraum zwischen Fuß und Boden. Die anderen Beine können den Torso über kleine Hindernisse ziehen. Der Fuß wird mit seiner Auflagefläche immer parallel zum Torso gehalten, damit die Zehen nicht mit kleinen Hindernissen kollidieren (5.3.1.7.3).

Kollisionen

Die Kollisionskontrolle der Beine darf nicht auf kleine Hindernisse reagieren. Die Beine müssen nachgebende Hindernisse wegschieben. Beispielsweise könnte sich der Roboter nicht durch einen Wald bewegen, wenn jede Kollision zum Abbruch der Bewegung führe. Dies wird mit einer Kollisionskontrolle erreicht, die der „Anti-Pitch-Detection“ (5.3.4.3.3) aus dem Fahrzeugbau ähnelt. Bei Kollisionen mit harten Objekten hat sich in einer Testreihe (5.3.5.5) gezeigt, dass die Wiederholung der Kollision meistens zum Erfolg führt. Bei einer Kollision mit einem nicht nachgiebigen Objekt wird die Teilbewegung, die zur Kollision geführt hat, bis zu dreimal wiederholt (5.3.4.5). Es wird versucht, das Objekt zu verschieben. Der Roboter muss durchsetzungsfähig sein.

2.6 Zusammenfassung

In diesem Kapitel wurden grundlegende Designentscheidungen für den Roboter des Projekts AMEE ermittelt und erläutert. Zusammengefasst wurden folgende Entscheidungen getroffen:

a. Der Roboter wird ein Laufsystem haben (2.2).
b. Der Roboter wird vier Beine haben (2.3).
c. Der Roboter wird ein statischer Läufer (2.4.1).
d. Das Gesamtgewicht des Roboters wurde auf ca. 35,6kg geschätzt (2.4.2).
e. Die grundlegenden Laufmuster und erste Problemstellungen wurden identifiziert (2.5).

Der Roboter AMEE würde in die „Man-portable UGV“ [Bet10] USAR-Klassifizierung eingeordnet werden.

3 Mechanische Hardware

Durch die Ermittlung der Spezifikationen (2.6) kann mit der Konstruktion des Laufsystems begonnen werden. In diesem Kapitel wird mit der schematischen Konstruktion der Beine begonnen und schrittweise, bis zur Realisierung, vervollständigt. Auf konkrete Berechnungen und Konstruktionsdetails wird in diesem Kapitel verzichtet. Um das Gesamtkonzept darzustellen, wird kurz der komplette Roboter erläutert.

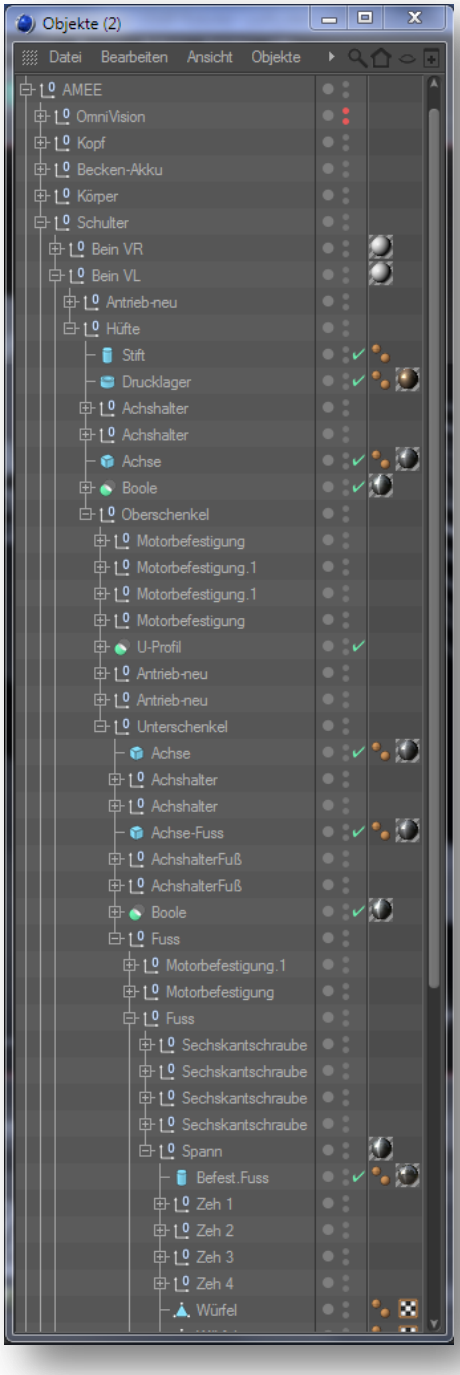

Abbildung 3-1 Hierarchisches Modell eines Beins

3.1 Simulation

Für die Konstruktion wurde ein hierarchisches 3D Simulationsmodell aller Bauteile erstellt. Die Einzelteile sind gliederweise hierarchisch in Baugruppen angeordnet (*Abbildung 3-1*). Die Simulation wurde mit einer Kollisionskontrolle und realen physikalischen Größen erweitert. Ein Ausschnitt des kompletten 3D-Modells wird in *Abbildung 3-2* dargestellt.

Mit Hilfe der Simulationssoftware konnten alle Bewegungsabläufe simuliert werden. Die Beine können an allen Achsen oder über die Position der Füße bewegt werden. Durch die 3D Echtzeitdarstellung konnte die Konstruktion stückweise optimiert werden. Mit Hilfe des Simulationsmodells wurden auch die Bewegungsabläufe für die Softwaremodellierung ermittelt. Aus dem Simulationsmodell konnten auch die resultierenden technischen Zeichnungen exportiert werden. Die Darstellungen in dieser Arbeit wurden mit diesem Modell erstellt. Das Modell ist schrittweise erweitert worden.

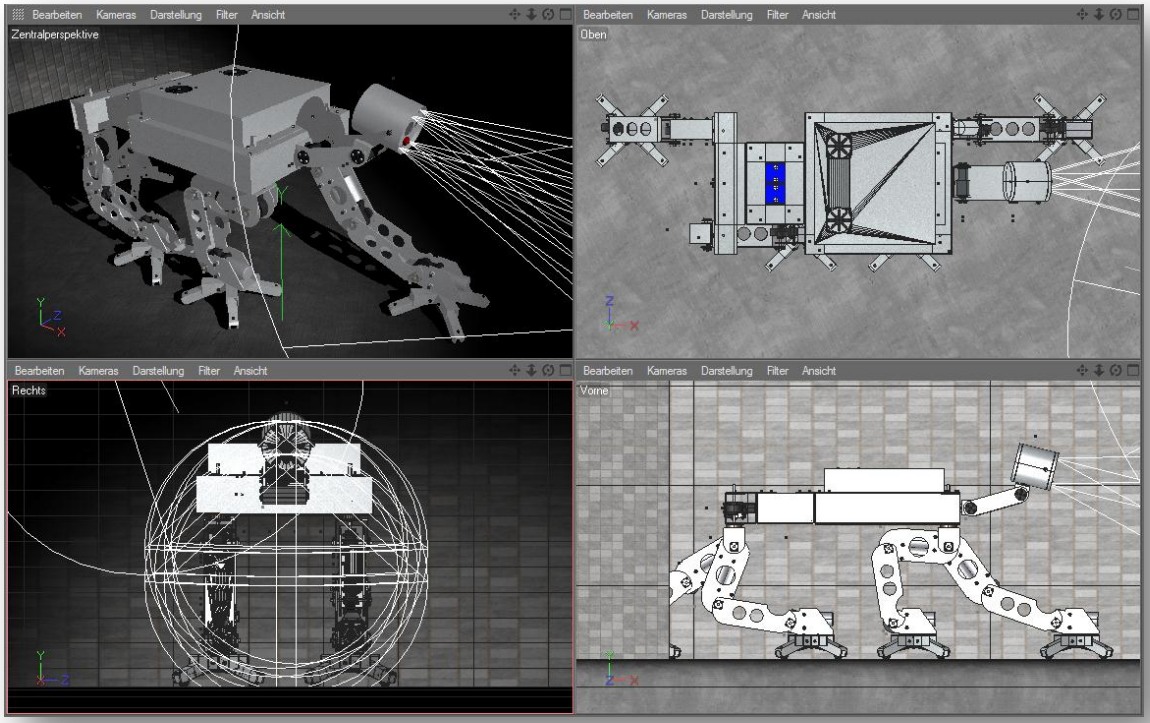

Abbildung 3-2 ,3D-Modell AMEE

Anmerkung:
Das Grundmodell wurde in der Vorbereitungsphase zu dieser Arbeit erstellt. Die realen physikalischen Größen und das korrekte Funktionsprinzip konnten erst mit der Realisierung erstellt werden.

3.2 Grundkonzept

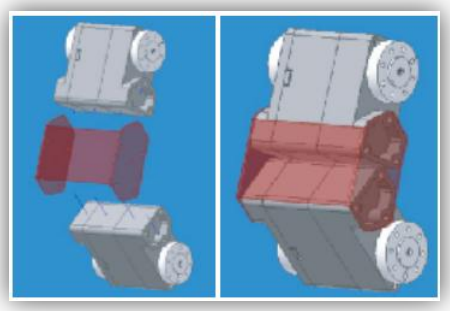

Abbildung 3-3 Robonova-I, Bein [Hit08]

Die Grundidee stammt aus der Konstruktion des kleinen Roboters „Robonova-I“ der Firma HiTec Robotics [Hit08]. In dieser Konstruktionsart (*Abbildung 3-3*), werden die sonst üblichen Gelenke und Lager direkt durch die Achsen und Lager der Antriebe ersetzt. Das Motorgehäuse wird als tragendes Teil verwendet. Die Antriebe werden mit Aluminiumblechen verbunden. Mit diesen Blechen wird die Form des Roboters modelliert. Um den Blechen eine höhere Stabilität zu geben, werden die Bleche zu U-Profilen gebogen. Diese Art des Aufbaus ist günstig und robust. Der AMEE Roboters wird in Anlehnung an diese Idee aufgebaut.

3.3 Aufbau des Beins

Ähnlich einem Papierbastelbogen können fast alle Teile aus Aluminium-Blech flach ausgeschnitten werden. Diese Teile können mit Hilfe eine Abkantbank zu U-Profilen gebogen werden. *Abbildung 3-4* stellte die benötigen Bauteile für ein Gelenk dar.

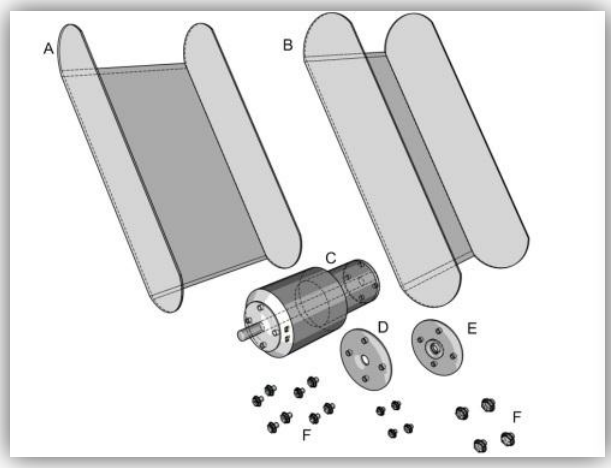

Abbildung 3-4 Grundgelenk

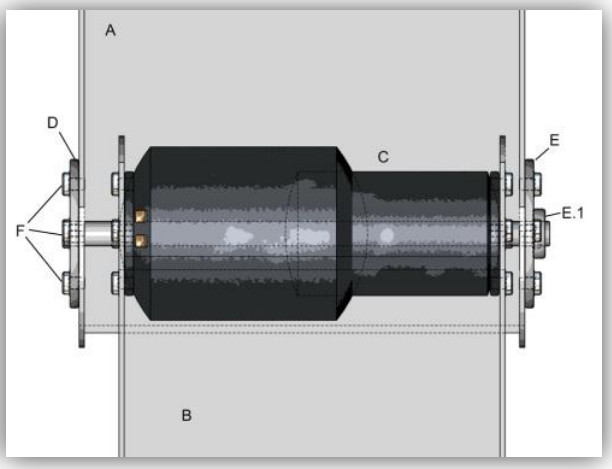

Abbildung 3-5 montiertes Grundgelenk

A: U-Profil 1
B: U-Profil 2
C: Motor mit Getriebe und durchlaufender Achse
D: Achshalter 1
E: Achshalter 2 mit Klemmring durch eine Madenschraube
F: Sechskantschrauben

Wird dieser einfache Aufbau zusammengesetzt wie in *Abbildung 3-5* dargestellt, ergibt dies ein Grundgelenk, dass den Ober- und Unterschenkel darstellt. Die Funktion ergibt sich dadurch, dass das Motorgehäuse C am U-Profil B fixiert ist. Die Achse des Motors C ist am U-Profil A fixiert. Wird der Motor nun aktiviert, verdreht sich das U-Profil A gegenüber dem U-Profil B. Die Stabilität der Konstruktion kann mit der Materialstärke und der Profilgröße fast beliebig erhöht werden. Je nach Antriebstyp muss die Achse durch ein zusätzliches Lager, im Profil B abgestützt werden.

Es werden keine kostenintensiven Spezialanfertigungen benötigt. Durch diese simple Konstruktion können einfach und kostengünstig komplexe mechanische Bauteile erstellt werden. Weiterhin ist es einfach, Veränderungen an der Konstruktion vorzunehmen.

Werden an dieses Grundgelenk zwei weitere Antriebe und Halter montiert (*Abbildung 3-6*, links), sind die Grundkomponenten für ein Bein gegeben. Der obere Halter ist der Fixpunkt zum Roboter-Torso. Um das Bein in drei Dimensionen bewegen zu können, wird am Fixpunkt noch ein Antrieb montiert, der das ganze Bein dreht.

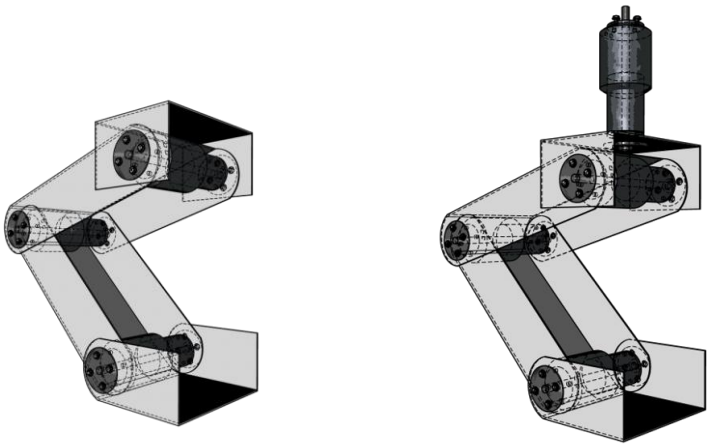

Abbildung 3-6 Grundkörper Bein

3.3.1 Der Fuß

Im Kapitel 2.5.1 wurde die besondere Bedeutung der Füße ermittelt. Auch der Fuß kann aus einem Blech ausgeschnitten und gebogen werden (*Abbildung 3-7*a). Der Grundkörper für den Fuß ist etwas aufwendiger. Auf den Menschen übertragen, entspricht dieser dem menschlichen Spann.

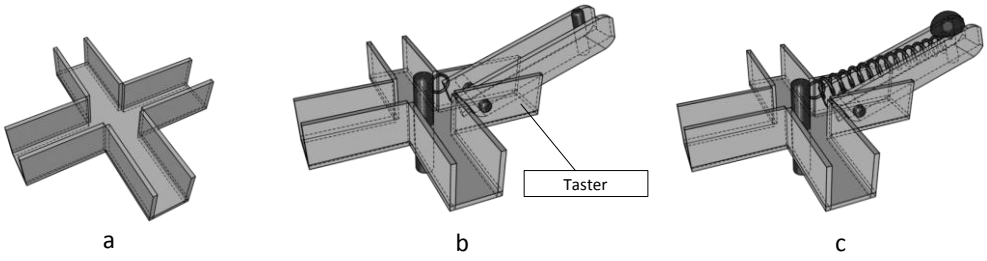

Abbildung 3-7 Grundkörper Fuß, gesehen von unten

Der Fußgrundkörper wird mit einer zentralen Achse erweitert. Es wird ein schmaleres U-Profil, beweglich auf einer Achse, als Zeh montiert. (*Abbildung 3-7*b) Der Zeh wird durch eine „Nase“ an der Rückseite des Profils gegen Umklappen gesichert. Zwischen dem Fußgrundprofil und dem Zeh wird ein Taster montiert. Der Taster wird so positioniert, dass er ausgelöst wird, wenn der Zeh gegen das Fußgrundprofil geklappt wird. Dieser Taster wird in der Bodenerkennung genutzt (5.3.4.4.11). Wird die zentrale Achse mit der Außenkante des Zehs durch eine Zugfeder verbunden (*Abbildung 3-7*c), wird der Taster erst ab einem bestimmten Druck ausgelöst. Diese Schritte werden für weitere drei Zehen

wiederholt. Das fertige Bein, mit montiertem Fuß[7], ist in *Abbildung 3-8* als schematische Zeichnung dargestellt.

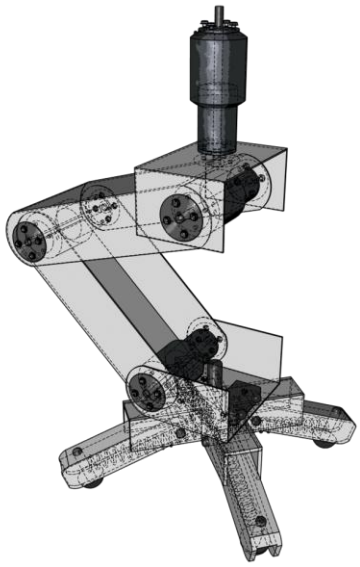

Abbildung 3-8 Zeichnung komplettes Bein

3.3.2 Elektroantriebe

Bei Robotern der mittlerer Größe (beispielsweise ASIMO(1.5.1) ca. 1,2m) werden oft Gleichstrom-Getriebemotoren verwendet([Sic08] ab Seite 79). Sie bieten hohe Effizienz, sind robust und können einfach angesteuert werden. Diese Antriebstypen haben keine eigene Steuerelektronik, wodurch in der späteren Programmierung die Antriebscharakteristika begrenzt verändert werden können. Die Drehzahl wird durch die Spannung geregelt. Ein Drehrichtungswechsel wird durch Umpolen der Versorgungsspannung erreicht. Für den Automobilbau werden diese Antriebe in großer Stückzahl hergestellt und sind kostengünstig zu beschaffen. Durch die Spezifikationen im Fahrzeugbau sind die Antriebe in wasserdichten und staubgeschützen Gehäusen verfügbar. Der verwendete Antrieb muss für ein Laufsystem eine große *Unter*setzung des Getriebes haben. Bei zu kleiner Untersetzung würden die Beine im Ruhezustand durch das Eigengewicht des Roboters einknicken. Durch Verwendung von weiteren Automobilbauteilen sind 12V Gleichspannung in diesem Projekt die maximal zur Verfügung stehende Spannung. Bedingt durch den Aufbau des Beins, wird ein Antrieb mit durchgehender Welle / Achse benötigt. Nach einer Überschlagsrechnung und Beratung eines Motor-Herstellers wird ein Antrieb mit einem Drehmoment von ca. 20Nm benötigt.

[7] Die Idee, den Fuß mit ausladenden und beweglichen Zehen zu entwerfen, stammt aus dem Kinofilm „Red Planet“ [Ant00].

Für das Bein wurden Antriebe der Baureihe 0225 GMPG der Firma Ott GmbH gewählt, der folgende Eckdaten [Ott10] bietet:

Nennspannung	**12V**
Leerlaufdrehzahl	21 U/min
Anlaufdrehmoment	22 Nm
Stromaufnahme	ca. 12A
Einschaltdauer	10%
Getriebeuntersetzung	210:1
Gewicht	0,71kg
Länge	162mm
Breite	40mm

Tabelle 3-1 Technische Daten Antrieb [Ott10]

Dieser Antrieb hat ein geringeres Eigengewicht als vergleichbare Antriebstypen. Nicht dargestellt ist das Drehmoment-Diagramm. Die Motorcharakteristik hat einen steilen Drehmomentanstieg gegenüber der anliegenden Spannung. Die Getriebe sind verspannt (1.5.1) und haben kein Spiel.

3.3.3 Konstruktion

In der Konstruktion des Beins wird das Modell (3.1) an die realen Bauteile angepasst. Für die Realisierung wurde eine leichte und zähe Aluminium-Magnesium-Legierung (AlMg [Gro11]) gewählt. Aufgrund der Gehäusebauart der Antriebe, kann der Aufbau des Beins (3.3) optimiert werden.

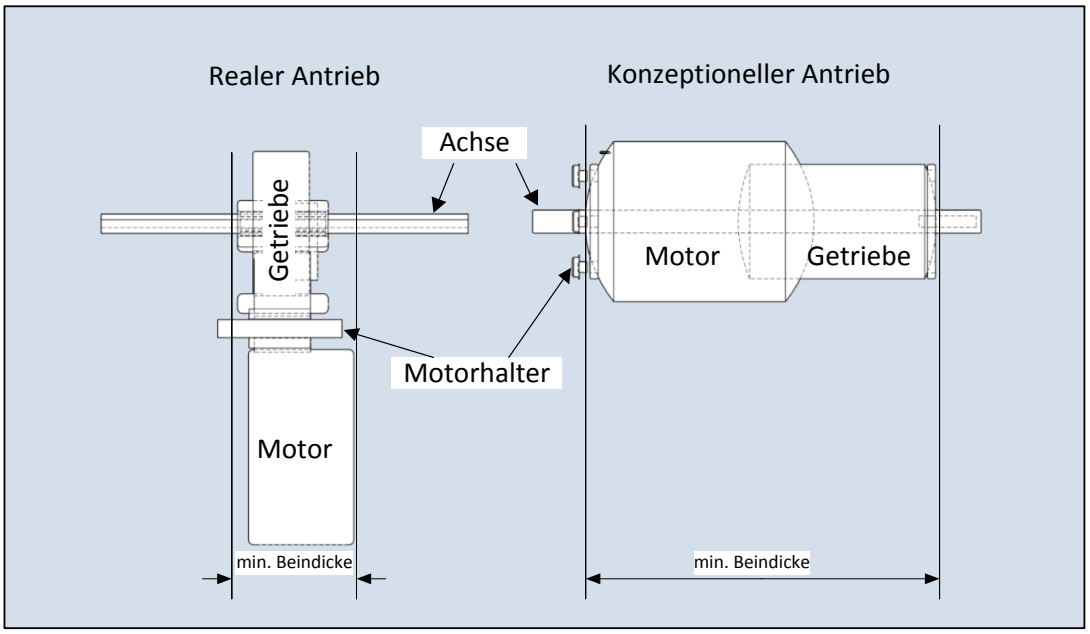

Abbildung 3-9 Umsetzung des Antriebskonzepts

In *Abbildung 3-9* sind der konzeptionelle (3.3) und der reale Antrieb (3.3.2) gegenübergestellt. Um die Beindicke möglichst schmal zu halten, wurde ein Antriebstyp gewählt, bei dem das Getriebe um 90° verdreht zum Motor ist. Dadurch kann bei kleineren Abmessungen ein leistungsstärkerer Motor

verbaut werden. Das diskutierte Konstruktionskonzept wird damit nicht verändert, aber das Bein bietet in der Frontansicht weniger Angriffsfläche bei einer Bewegung.
Die Antriebe werden liegend in einem Vierkantrohr montiert (*Abbildung 3-10*). Durch die Antriebslänge wird die minimale Länge der Glieder bestimmt. Die Konstruktion wurde an die Antriebe angepasst.

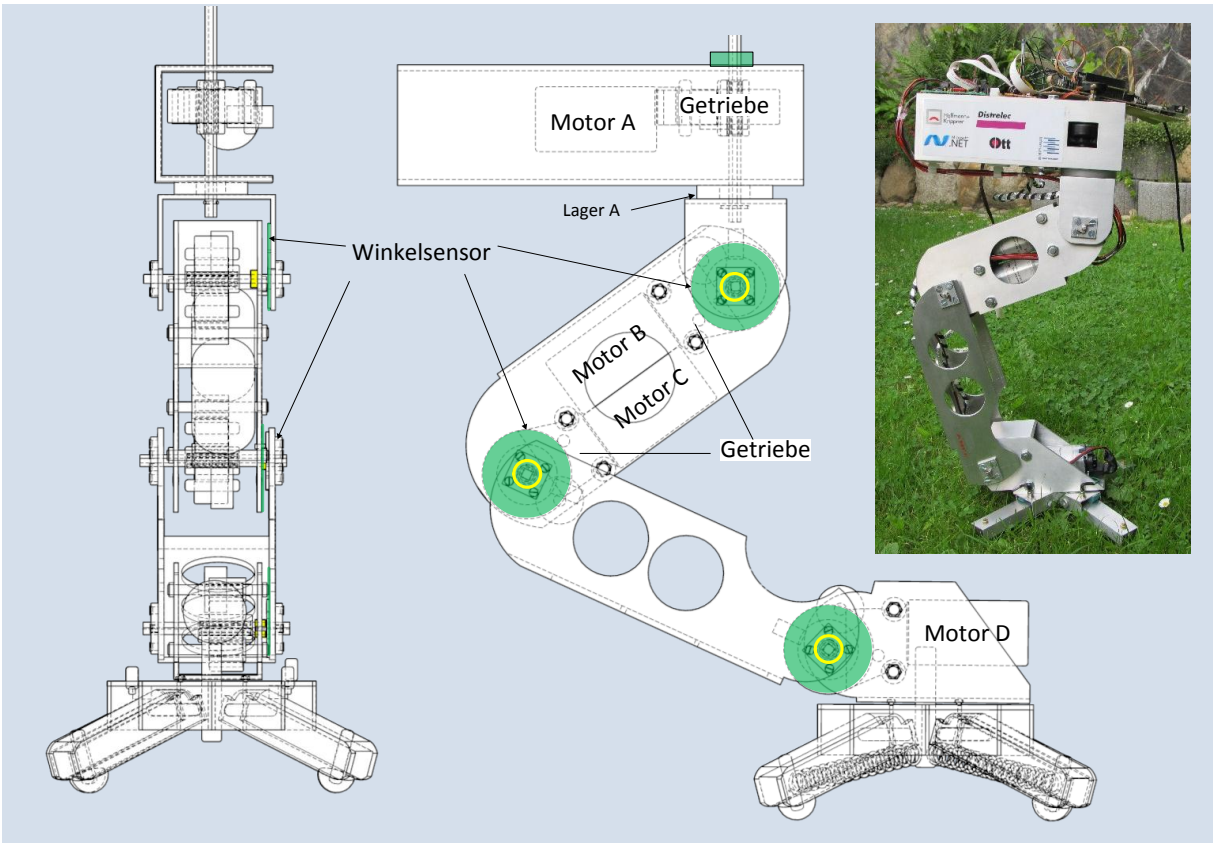

Abbildung 3-10 reale Konstruktion des Beins

Der Antrieb besitzt ein robustes Kunststoffgetriebe. Deshalb wird das Getriebe mit Gleitlagern der Firma igus GmbH (Typ iguldur), an den Achsen entlastet. In *Abbildung 3-10* sind diese als gelbe Hülsen an der Achse dargestellt. Für die Rückmeldung des jeweiligen Drehwinkels werden Sensoren (grün, *Abbildung 3-10*) auf den Achsen montiert. Als Sensor wird ein Magnetfolienpotentiometer (MFPotis) der Firma Hoffmann + Krippner GmbH vom Typ „SENSOFOIL® Magnet FR4“ verwendet. Diese Sensoren werden im Kapitel 4.2.4.1 genauer beschrieben. Durch die Bauart der MFPotis, mit 2,5mm Dicke, ist die kompakte Bauart des Beins möglich geworden. Sie wurden zwischen die Glieder geklebt. Es wird ein Neodym Magnet als „Schleifkontakt“ verwendet, der in eine Bohrung im gegenüberliegenden Glied versenkt wird. Nicht dargestellt ist der Winkelsensor für den Antrieb A. Hier wurde ein Folienpotentiometer mit integriertem „Schleifkontakt“ der Firma Contelec® Typ WAL 305 über der Achse vom Antrieb A montiert. Die Achse A dreht die Hohlachse des Folienpotis.
Zwischen Motor A und dem Halter der Schulter wurde ein breites Kunststofflager montiert (*Abbildung 3-10*, Lager A). Das Lager hat einen großen Durchmesser, um die wirkenden Kräfte an der Schulter besser zu verteilen, da der Hebelarm mit ausgestrecktem Bein relativ lang ist.

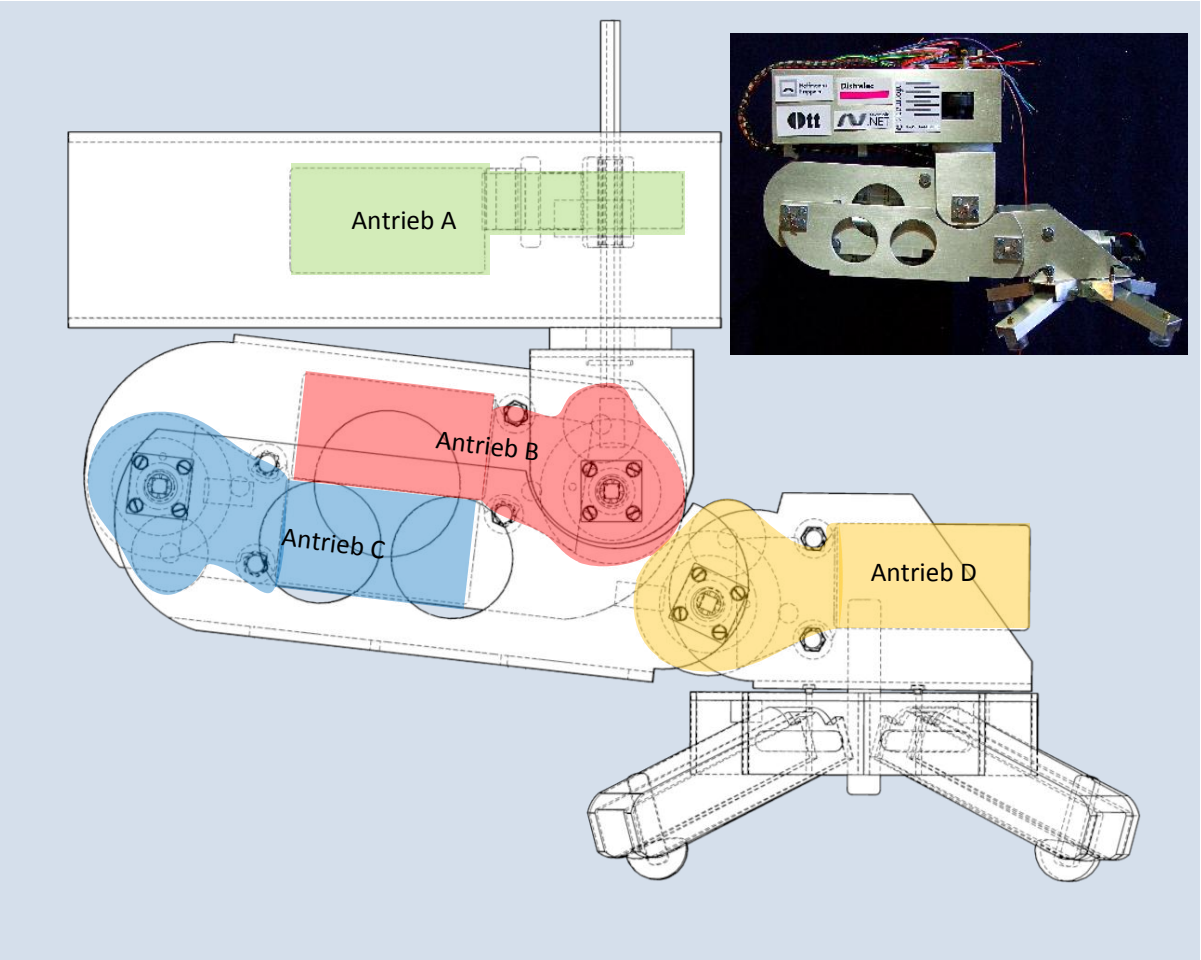

Abbildung 3-11 Bein in Parkposition

Um das Bein möglichst nah an den Körper ziehen zu können, sind die Antriebe B, C (Motor in *Abbildung 3-11*) im Oberschenkel platziert. Der Unterschenkel besteht aus einem U-Profil (nach oben geöffnet) und enthält keinen Antrieb. Der Antrieb D für den Fuß wurde im Fuß montiert. Die Länge des Unterschenkels ist so gewählt worden, dass Oberschenkel und Unterschenkel ineinander passen. Dies wird in *Abbildung 3-11* dargestellt. Die dargestellte Position der Glieder ist auch die sogenannte Parkposition des Beins. Diese bietet einen guten Schutz der Getriebe bei harten Schlägen auf das Laufsystem. Die Kräfte, die auf den Fuß einwirken, werden auf die Profile abgeleitet, da diese aufeinander liegen und sich gegenseitig abstützen. Damit ist dies die Position der Beine, die bei einem Sturz eingenommen wird. Der Hebelarm ist so kurz wie möglich und bietet Anschläge.

Fuß

Abbildung 3-12 stellt die Konstruktion des Fußes dar. Damit die Software erkennen kann, wann und wie der Fuß auf den Boden aufsetzt, sind wasserdichte Taster montiert worden.

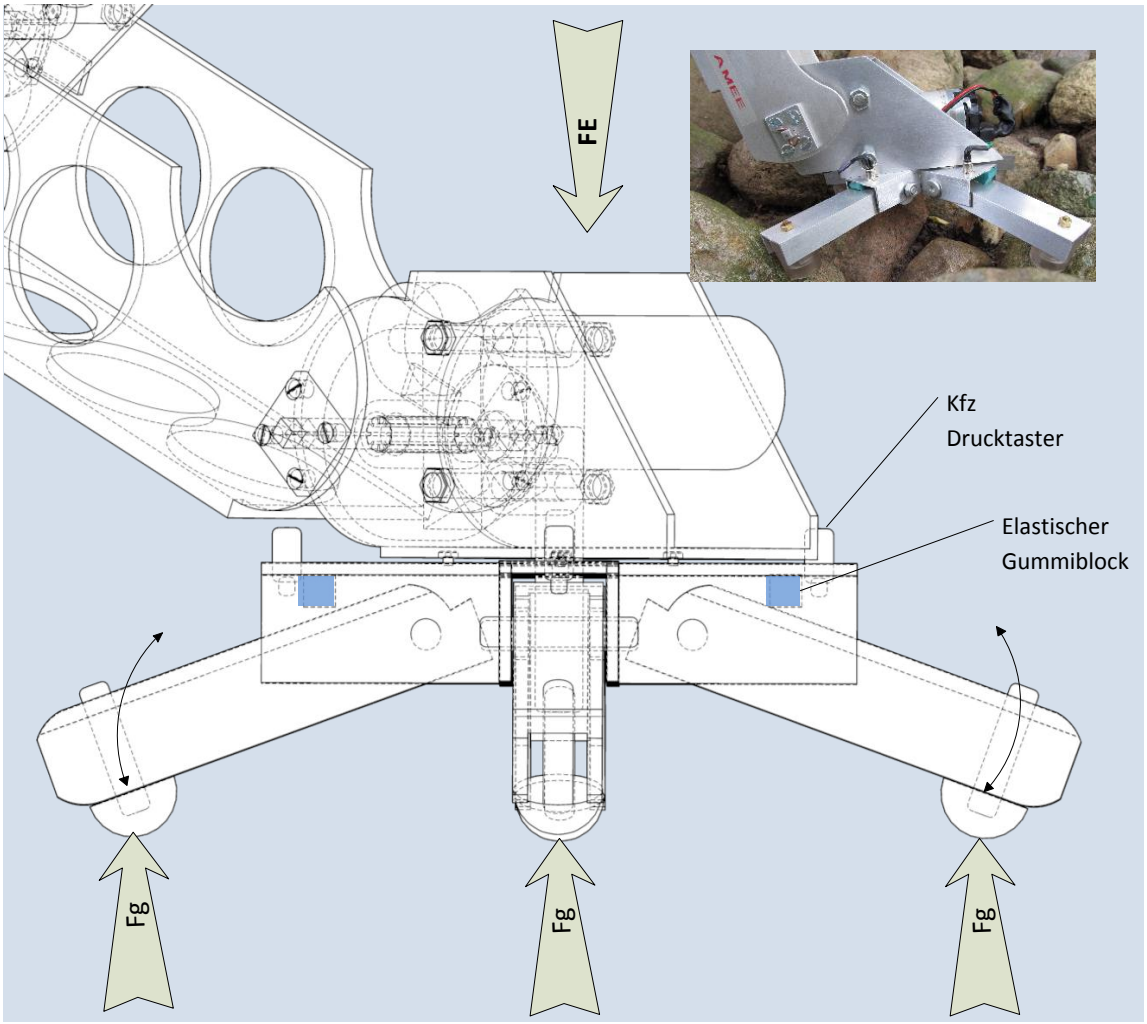

Abbildung 3-12 Fußkontakte

Zusätzlich wurden elastische Gummiblöcke zwischen Spann und Zeh geklebt. Eine Belastung des Fußes drückt die Gummiblöcke zusammen, bis der Taster im Spann geschlossen ist. Der Elastizitäts-koeffizient (gerechnet über den kompletten Gummiblock) des aufgeschäumten Gummis steht in Beziehung zur Dicke des Materials. Durch Verändern der Dicke der Gummiblöcke, kann das Auslösegewicht für die Taster eingestellt werden.

Workspace

Das Bein hat eine Gesamtlänge von 69,5cm und einen halbkugelförmigen Workspace mit einem Radius von 66,5cm, unterhalb des Torsos. Ergänzt wird dieser Workspace über dem Torso durch eine deformierte Viertelkugel mit einem oberen Scheitelpunkt von 44cm. Der in *Abbildung 3-13* skizzierte Workspace (blau) zeigt nicht, dass die obere Viertelkugel zur Achse A hin abflacht.

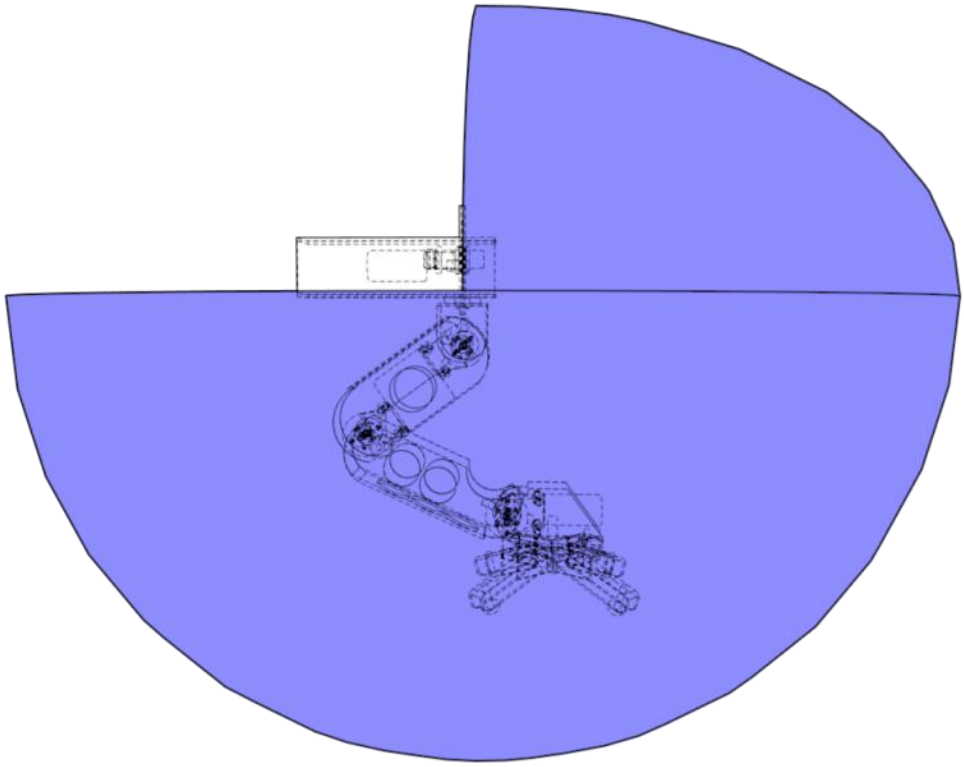

Abbildung 3-13 Workspace des Beins

Das Bein hat ein Gewicht von 4,6kg ohne den angedeuteten Torso. Die Impulsleistungsaufnahme erreicht ca. 90 – 120W.

Schutz

Noch nicht realisiert wurde ein Schutzstrumpf. Dieser Schutzstrumpf besteht aus wasserdichten und elastischem Neopren®. Dieser Schutzstrumpf ist geschnitten wie ein unten geschlossener Schlauch. Der Schlauch wird ohne Fuß über das Bein gezogen. Danach wird der Fuß über dem Schlauch montiert. Dieses Detail ist eine Abwandlung der Schutzstrümpfe des Big Dog [Bos10]. Damit kann das Bein bis zur Schulter in Wasser eintauchen. Zudem ergibt sich ein guter Schutz gegen Sand und wird auch gegen die Umwelt elektrisch isoliert.

Für ein bessere Übersicht werden die Antriebe weiterhin in der konzeptionellen Bauform (z.B. Abbildung 3-8) als Zylinder dargestellt.

3.4 Grundgehäuse des Roboters

In dieser Arbeit wird der Roboter-Torso nur als Träger für das Laufsystem betrachtet. Darüber hinaus werden einige Punkte für die Weiterentwicklung angeschnitten. Um den Aufbau von AMEE kostengünstig zu halten, wird für den Torso ein einfacher Quader als Grundkörper gewählt. In diesen Quader müssen alle elektronischen Komponenten passen. Eine einfache Konstruktion ist hier nötig, da die Elektronik vor Spritzwasser geschützt werden soll. Gerade Linien an den Öffnungen erleichtern das Abdichten des Torsos. Der Rahmen des Torsos wird aus U-Profilen gefertigt. An den Außenkanten greifen die Kräfte der Beine an, da sie hier befestigt werden. Weiterhin werden diese U-Profile als Kühlkörper für die Leistungselektronik (4.2.3) verwendet. Es soll auch die Abwärme der Hauptrechnerprozessoren per Heatpipes dorthin abgeleitet werden, damit der Torso keine Öffnungen zur Kühlung benötigt.

Die Seitenlängen des Torsos sind nicht ganz frei wählbar, da diese direkt den Schwerpunkt und den stabilen Zustand beeinflussen (2.5). Wird der Torso zu breit oder zu schmal gewählt, wird der Bereich des stabilen Zustands so verkleinert, dass die nutzbare Schrittweite im „Fast Static Walk“ (2.5.2) auf unter 50% schrumpft. Die genaue Ermittlung dieser Kantenlängen ist auch von noch nicht berücksichtigten Faktoren abhängig. Dies ist eine Aufgabe für die Weiterentwicklung des Roboters. Hier werden nur Abschätzungen dargestellt, wie sie im Prototypenbau üblich sind.

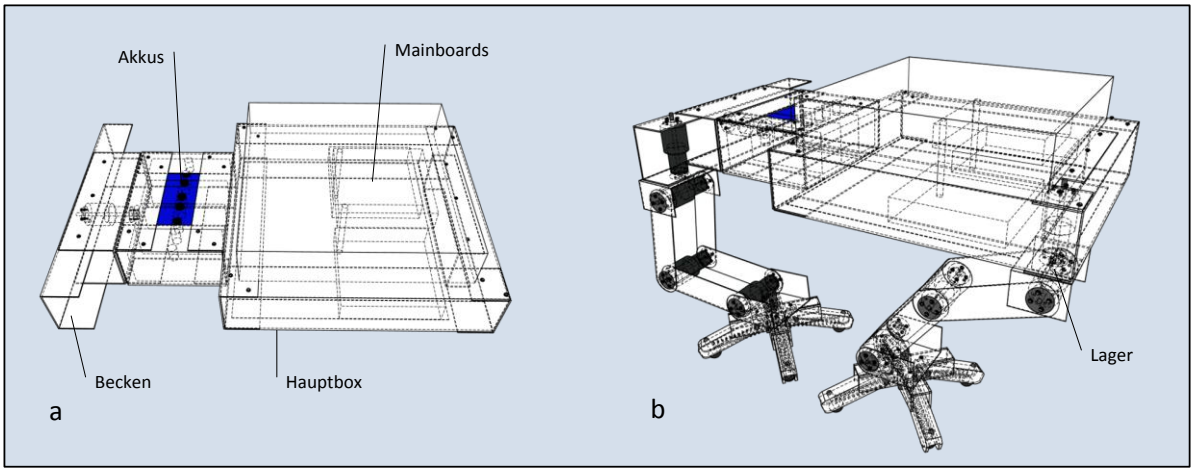

Abbildung 3-14 Grundkörper

In der Hauptbox, *Abbildung 3-14*a, ist die gesamte Elektronik montiert. In der Abbildung wird dargestellt, wie die U-Profile ineinander gesteckt werden, um einen stabilen Rahmen zu erhalten. Als Bodenplatte wird ein Blech unter diesen Rahmen geschraubt und mit diesem wasserdicht verklebt. Der Deckel der Hauptbox ist leicht erhöht, um mehrere Mainboards übereinander anzuordnen. An die Hauptbox wird ein weiterer Rahmen montiert, der die Akkus enthält. Durch diese Bauart kann der Akku-Typ variieren. Weiterhin wird durch die Akkus der Schwerpunkt nach hinten verschoben, was bei Steigungen vorteilhaft sein kann. Das größte Gewicht liegt damit auf den Hinterbeinen. An den Akku-Rahmen wird das Becken befestigt, das aus zwei U-Profilen besteht. Es entspricht dem Beckenknochen und ist breiter als der Akku-Rahmen, damit die Hinterbeine neben den Körper gezogen werden können. In *Abbildung 3-14*a ist noch ein weiterer Antrieb dargestellt, der das Becken gegenüber dem restlichen Körper verdreht. Damit kann der Torso nach einem Sturz wieder auf die Beine gedreht werden. Dieser Fall wird von der höheren Logik des Hauptkontrollers bearbeitet und

ist, wie auch der letztgenannte Motor, nicht mehr Bestandteil dieser Arbeit.
In *Abbildung 3-14*b ist zu erkennen, wie die Beine am Torso befestigt werden. Der Antrieb für eine Beindrehung, wird im Torso montiert.

3.5 Kopf und Hals des Roboters

Der Kopf des Roboters ist als Träger für richtungsabhängige Sensoren geplant. Je nach Ausrichtung des Halses werden andere Daten erfasst. Weiterhin ist ein Kohlenmonoxid Sensor am Kopf geplant, der zur Personensuche genutzt werden soll. Da dies aber nicht direkt im Zusammenhang mit dem Laufsystem steht, wird der Kopf nur oberflächlich betrachtet. Auch die Form ist nur eine Annäherung an die endgültige Konstruktion. Der Hals soll dem Kopf eine gute Bewegungsfreiheit ermöglichen. Er kann auch dazu genutzt werden, um AMEE nach einen Sturz wieder aufzurichten.

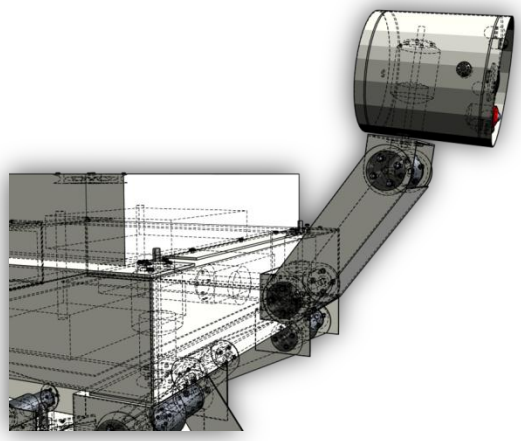

Abbildung 3-15 Kopf und Hals

Der ganze Hals kann am Torso um 360° gedreht werden, wie auch der eigentliche Kopf. Für die Positionierung wird eine modifizierte inverse Kinematik aus den Beinen verwendet. (5.3.1.1) Der Hals muss im Verhältnis zum Körper relativ lang sein, damit der Roboter über Hindernisse „sehen“ kann.

3.6 Zusammenfassung

In diesem Kapitel wurde der mechanische Aufbau des Roboters konkretisiert. Es wurden folgende Punkte ermittelt.

a. Durch ein Simulationsmodell wird die Realisierung beschleunigt (3.1).
b. Ein konzeptionelles Konstruktionsmodell vereinfacht die Realisierung der Mechanik (3.2).
c. Die Mechanik des Beins wurde realisiert (3.3.3).
d. Es wurden der mögliche Workspace und das Gewicht eines Beins ermittelt (3.3.3).
e. Durch den realen Aufbau des Beins wurde der Aufbau des Laufsystems ermittelt (3.4).
f. Es wurden erste Konstruktionsdetails für den kompletten Roboter festgelegt.

In der Realität ist der Entscheidungsprozess von verknüpften Designentscheidungen abhängig, die in anderen Kapiteln getroffen wurden. Durch die strukturierte Betrachtung werden die Abhängigkeiten leicht verwischt.

4 Hardwaredesign

Diese Kapitel beschäftigen sich mit dem Aufbau der elektronischen Hardware. Im Kapitel 4.1 wird das grundlegende Gesamtkonzept vorgestellt. Das Kapitel 4.2 stellt die dezentrale Beinelektronik vor, die dann bis zur Realisierung konkretisiert wird.

4.1 Aufbau des Gesamtsystems

Das Hardwaredesign für einen autonomen USAR Roboter muss an die hohen Anforderungen angepasst werden. Das Gesamtsystem ist als verteiltes System (VS) konzipiert, um eine Lastverteilung zu ermöglichen. Diese Designentscheidung wird im Kapitel 5.2 näher diskutiert.

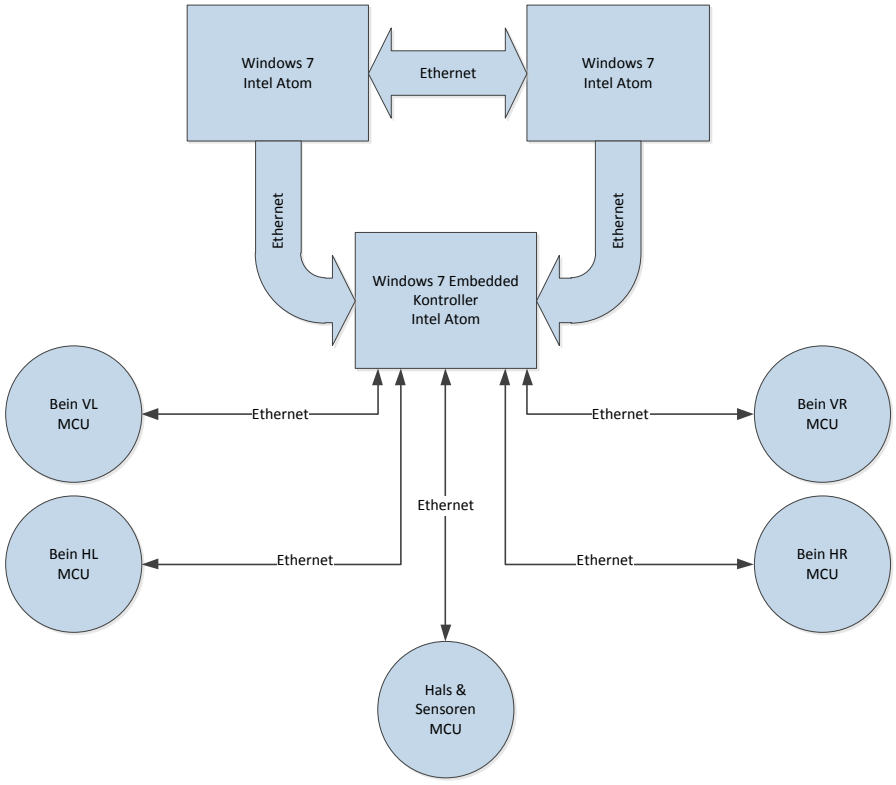

Abbildung 4-1 Logischer Aufbau der Komponenten

In *Abbildung 4-1* ist schematisch das Zusammenspiel der Komponenten gezeigt. Alle Komponenten kommunizieren über einen Ethernet-Switch (nicht dargestellt) miteinander. Die drei Intel-Atom® Rechner werden als eine Einheit betrachtet und sollen als Cluster arbeiten. In der Weiterentwicklung des Systems kommt das „Microsoft Robotics Development Studio®“ (RDS) zum Einsatz [Bet10]. Die daraus verwendeten Komponenten „Concurrency and Coordination Runtime“ (CCS) und „Decentralized Software Services“ (DSS) vereinfachen die Entwicklung eines VS. In den Hauptkontrollern sollen zukünftig die 3D Objekterkennung, Spracherkennung, höhere Logik-funktionen und der Koordinator des Laufsystems implementiert werden.

Die Extremitäten des Roboters werden als eigenständige Geräte betrachtet. Jedes Gerät soll seine eigenen Detailproblemstellungen weitestgehend selbständig lösen können. Beispielsweise ist es nicht Aufgabe des Hauptkontrollers, die Antriebe anzusteuern. Er bestimmt nur, wohin sich eine Extremität bewegen soll. Wie dieses Problem gelöst wird, ist die Aufgabe des jeweiligen Geräts

(5.3.2). Dazu erhält jedes Gerät einen eigenen Sub-Prozessor, der über Ethernet mit dem System vernetzt ist. Das Laufsystem besteht aus dem Hauptkontroller und vier getrennten Beinen, die jeweils einem Sub-Prozessor besitzen.

4.2 Elektronischer Aufbau eines Beins

Jedes Bein ist gleich und unterscheidet sich nur durch die Software-Konfiguration. So wird das Grundkonzept der Modularität verfolgt.

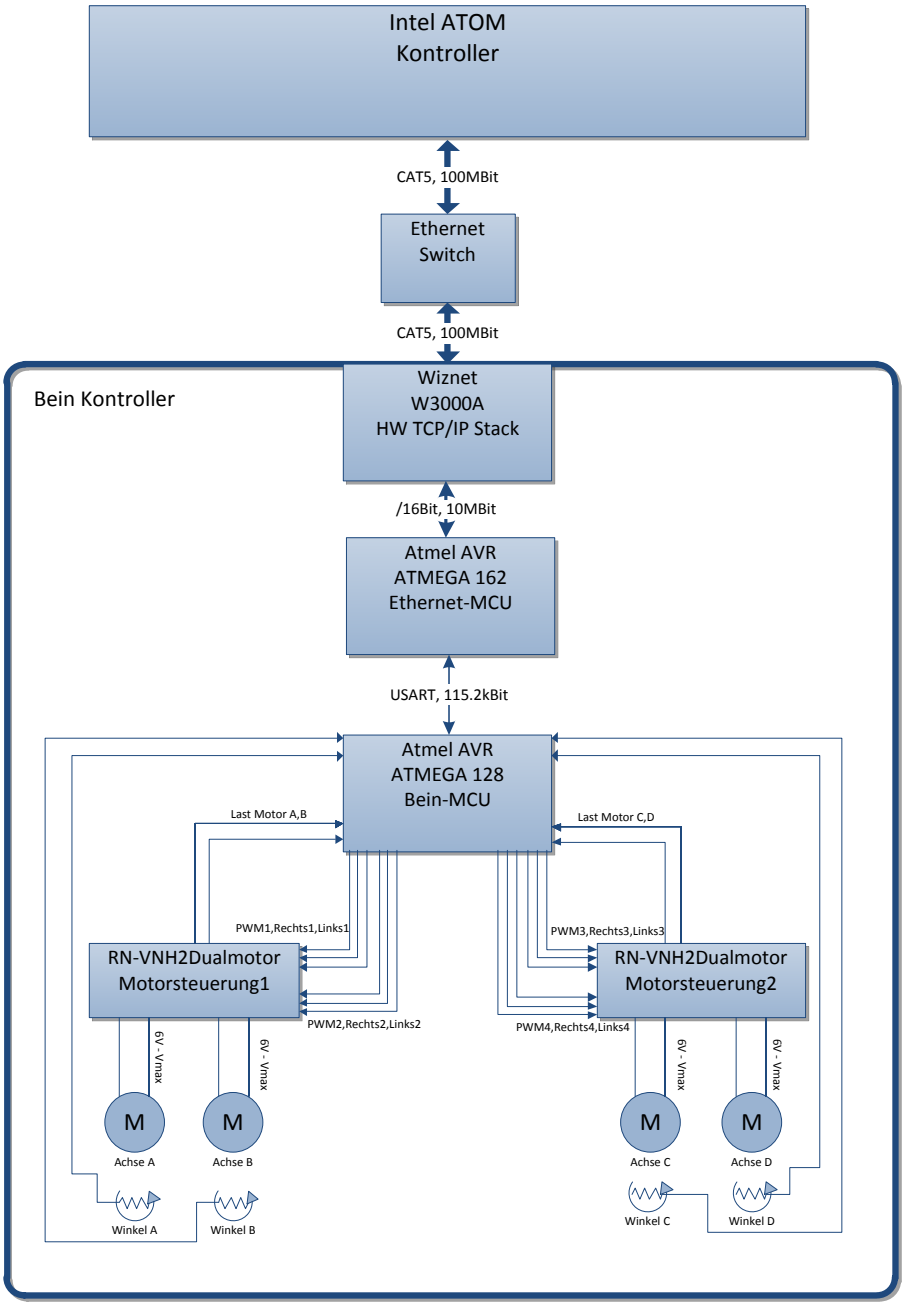

Abbildung 4-2 Anschlussplan eines Beins

In *Abbildung 4-2* ist der Anschlussplan für die Verkabelung eines Beins schematisch dargestellt. Der Hauptkontroller ist über Ethernet mit dem Bein-Kontroller verbunden. Im Bein-Kontroller ist die MCU

über einen Hardware TCP/IP Stack (4.2.2) angebunden. Die zwischengeschaltete „Ethernet-MCU“ wird in Kapitel 5.3.3.2 näher erläutert und wird als Teil des Bein-Kontrollers betrachtet. Der Datendurchsatz nimmt vom Hauptkontroller bis zur Bein-MCU stetig ab, da die übertragenen Daten systemnäher und damit kürzer werden (5.3.4.4.12).
Die Motorsteuerung und die Logik für rudimentäre Bewegungsabläufe, werden in der Bein-MCU zentralisiert. Angebunden sind hier auch die Motortreiber und Sensoren des Beins.
In den folgenden Abschnitten werden die Komponenten vorgestellt und der Bein-Kontroller schrittweise aufgebaut.

4.2.1 Die Bein-MCUs

Zentraler Punkt für das Bein ist die Bein-MCU, an der alle Sensoren und Aktoren angeschlossen sind und die Bein-Steuersoftware betrieben wird. Die MCU ist vom Typ ATMEGA 128A® 16AI der Firma ATMEL. Der ATMEGA 128A ist an der HAW Hamburg bekannt durch die abgewandelte Bauart des AT90CAN128®, der den gleichen Prozessor Kern verwendet. Um die Ausfallsicherheit unter extremen Temperaturbedingungen zu erhöhen, wird die AI Version verwendet. Die MCU kann laut Datenblatt [ATM101] bis 85°C Umgebungstemperatur sicher betrieben werden.
Die MCU wurde aufgrund des Preis-Leistungs-Verhältnisses gewählt. Obwohl der ATMEGA 128 ein 8 Bit System ist, erreicht er bei einer 16MHz Taktung 16 MIPS [ATM101] Rechenleistung. Er bietet alle nötigen Komponenten in einem Gehäuse.
Aus Gründen der Störfallbehandlung (5.3.3.2) wurde eine zweite MCU vom Typ ATMEGA162® 16AI zwischen ATMEGA128 und den Ethernet TCP/IP Hardware Stack geschaltet.
Auch der 32Bit Typ AT32UA0® der Firma ATMEL wurde näher betrachtet. Mehr dazu im Fazit (6.4).

4.2.2 Wiznet W3100A TCP/IP Hardware Stack

Der W3100A® [WIZ08] ist ein „Ethernet TCP/IP Hardware Stack Single Chip“. Er ist das Hardware-interface jedes Beins. In diesem Baustein wird das komplette TCP/IP, UDP, ICMP und ARP Protokoll bis zur Application Schicht in der Hardware abgearbeitet. Es sind auch die Ethernet Protokolle DLC und MAC integriert. Parallel können vier Verbindungen mit zusammen ca. 20 MBit/s full duplex übertragen werden. Der Baustein wird in diesem Fall über das Intel® MCU Bus Interface mit der MCU verbunden (*Abbildung 4-3*, MCU Interface), was erhebliche Geschwindigkeitsvorteile gegenüber dem vorhandenen I²C Bus bietet. Der Nachteil dieser Beschaltung ist, dass viele Pins der MCU benötigt werden. Der W3100A wird mit 3,3V betrieben, bietet aber 5V tolerante I/Os, was eine direkte Anbindung an den ATMEGA 162 ermöglicht.

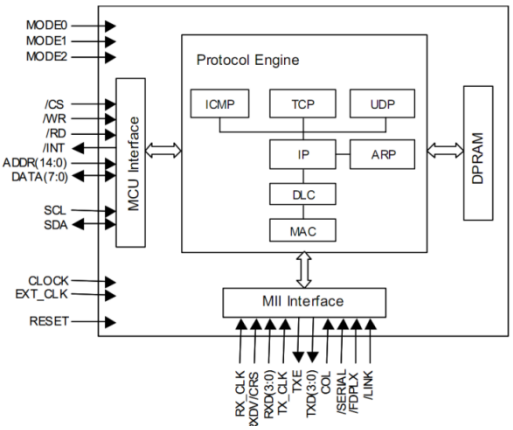

Abbildung 4-3 Block Diagramm W3000A [WIZ08]

Da der W3100A nur in einem LQFP Gehäuse hergestellt wird, wurde das vorgefertigte Modul iim7000A® der Firma Wiznet verwendet. Das Modul iim7000A bietet alle benötigten Komponenten, um den W3100A zu betreiben. Es muss nur die Ethernet-Buchse (RJ45) integriert werden. Der W3100A wird auf diesem Modul mit 25MHz und dem Realtek RTL8201BL Treiberchip betrieben, was

den maximalen Datendurchsatz auf ca. 10 – 11 MBit/s begrenzt. Da aber nur Steuerkommandos zur MCU und kurze Antwortdaten von der MCU übertragen werden, ist der Datendurchsatz höher als benötigt.

4.2.3 Motor Kontroller RN-VNH2

Für die Ansteuerung der Antriebe (Gleichstrommotor) wird eine Schaltung mit Drehrichtungswechsel benötigt. Zudem sollte auch die Drehzahl regelbar sein. Im Kapitel 3.3.2 wurde ein Motor mit einer maximalen Stromaufnahme von 12A ausgewählt. Um die Antriebe von der MCU aus zu steuern, müssen die Antriebe durch einen Transistor getrieben werden. Im einfachen Fall wird eine einfache H-Brücke mit P und N-MOSFET Transistoren als Treiber verwendet.

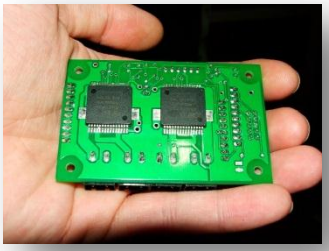

Abbildung 4-4 Motorsteuerung RN-VNH2

Die Geschwindigkeit soll über eine Pulsweitenmodulation (PWM) regelbar sein. Es ist auch nötig, einen Motor mit hoher Drehzahl schnell zu stoppen. Dies wird erreicht, indem der Motor kurzgeschlossen wird. Dabei können sehr hohe elektrische Ströme auftreten. Für die Steuerung wird die aktuelle Stromaufnahme als Rückgabewert in Echtzeit benötigt. Diese Werte werden als Überlastungsschutz und Kollisionskontrolle (5.3.4.3.3) verwendet. Weiterhin sollen die Treiber thermal überwacht werden, um ein Überhitzen und daraus folgenden Defekt vorzubeugen. Diese Anforderungen ergeben eine relativ komplexe Leistungselektronik. Aus diesen Gründen wird auf eine eigene Entwicklung verzichtet und der fertig aufgebaute Motorcontroller RN-VNH2 (*Abbildung 4-4*) der Firma „Robotikhardware“ [Bra07] verwendet. In diesem Modul wurden die Bausteine VNH2SP30-E der Firma STMicroelectronics® in der Automotive Version verbaut. Im Automobilbau wird dieser Baustein als Motortreiber für Fensterheber, Scheibenwischer und elektrische Flügelverstellung verwendet.

Bei dem Modul RN-VNH2 handelt es sich um einen dualen Motortreiber, der Antriebe bis jeweils 30A schalten kann. Die Wärmeverlustleistung der Treiberbausteine müssen ab 10A abgeleitet werden. Die Kühlung erfolgt durch das Aufschrauben der RN-VNH2 Platine in ein U-Profil der Hauptbox im Roboter Torso (3.4). Dies wird durch die kleine Bauform von 50mm x 80mm erleichtert. Gesteuert wird jeder Antrieb durch das Modul mit drei Pins. Die Schaltzustände sind über 2 Bit kodiert, wie in *Tabelle 4-1* dargestellt. Über den dritten Pin wird die Drehzahl durch eine Pulsweitenmodulation (PWM) mit maximal 20kHz gesteuert.

IN1	IN2	Funktion
0	0	Stillstand
1	0	Linksdrehung
0	1	Rechtsdrehung
1	1	Bremsen / Halten

Tabelle 4-1Kodierung des RN-VNH2

Besonders an dieser Motorsteuerplatine ist die geringe Eigenstromaufnahme von 0,025mA im Kodierungszustand 0,0 mit einer PWM von 0Hz.

Die integrierten Sensoren liefern die Stromaufnahme des jeweiligen Motors in Echtzeit als Spannung

von 0 bis 2,5V über einen Output-Pin. Da sich das Drehmoment eines Gleichstrommotors linear zur Stromaufnahme verhält, kann die mechanische Belastung der Antriebe errechnet werden. In *Abbildung 4-5* ist das anliegende Drehmoment des verwendeten Antriebs mit der resultierenden Sensorausgabespannung dargestellt.

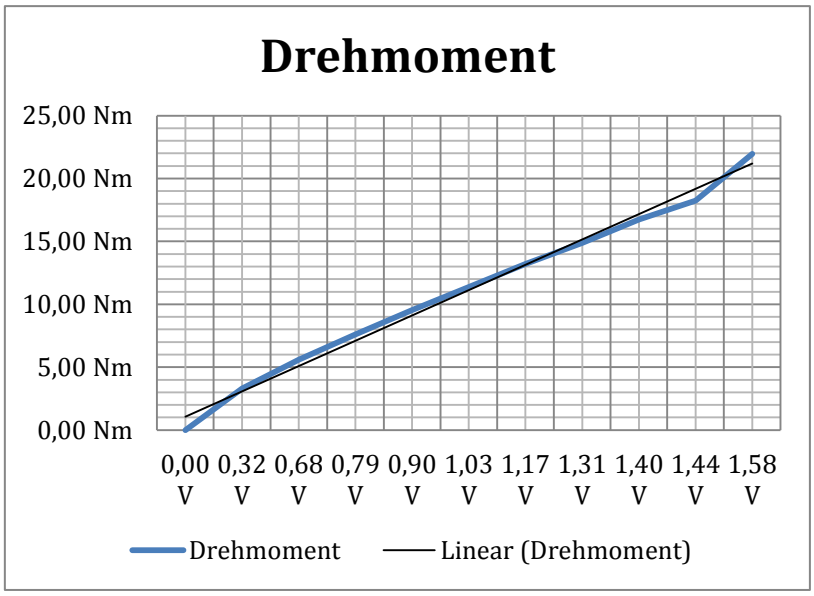

Abbildung 4-5 Messdaten und Drehmoment

Laut Hersteller wird die Ausgangsspannung mit dem Faktor 6 multipliziert. Das maximale Drehmoment des ausgewählten Motors beträgt 22Nm bei 12A. Daraus folgt die Gleichung: $Drehmoment\ in\ Nm = U_{in} \cdot 6 \cdot \frac{max.22Nm}{max.12A}$.

Der Treiberbaustein ist mit einer internen Thermosicherung ausgestattet. Überschreitet der Treiberbaustein 175°C, wird er abgeschaltet. Es erfolgt aber keine Rückmeldung, aus welchem Grund die Motorsteuerung nicht mehr reagiert. Deshalb ist das Modul RN-VNH2 mit einem zusätzlichen Temperaturwiderstand ausgestattet. Es ist vom Hersteller so entwickelt worden, dass mit steigender Temperatur die Spannung am Out Pin von 0V bis auf 2,5V steigt. Der Temperatursensor verhält sich annähernd linear und es wird eine Spannung von 0,04V bei -5°C ausgegeben. Laut den Angaben des Herstellers [Bra07] sollte das Modul nicht über 80°C belastet werden, was einer Output-Spannung von ca. 1,2V entspricht.

4.2.4 Winkelerfassung der Gelenke

Das Problem der Winkelerfassung an den Gelenken muss für einen USAR-Roboter genauer untersucht werden. Die Standardlösung in der Industrie ist ein Drehimpulsgeber (Drehgeber).Es wird nur ein relativer Winkel zur Startposition ausgegeben. Dies erfordert bei jedem Systemstart eine Anfangskalibrierung. Da der Roboter AMEE aber unter extremen Umweltbedingungen eingesetzt werden soll, kann es vorkommen, dass es zu Schalt- bzw. Rechenfehlern durch extreme Temperaturen oder starke Erschütterungen kommt. Wird ein Drehgeber verwendet, müssen die Zählerstände in einem nicht flüchtigen Speicher abgelegt werden. Alternativ könnte ein Drehgeber mit eigenem Speicher verwendet werden, da dieser die Impulse unabhängig von der Bein-MCU speichert. Ein Reset der Bein-MCU würde nicht den Zählerstand löschen. Ein Drehgeber mit eigenem

Speicher wird aber aus Kostengründen nicht verwendet.
Die erste Möglichkeit, das Speichern des Impulszählers in einem nicht flüchtigen Speicher, ist problematisch. Der Flashspeicher des ATMEGA 128 kann laut Datenblatt [ATM101] nur 10.000-mal überschrieben werden. Das interne EEPROM kann 100.000-mal überschrieben werden. Da nicht vorhergesagt werden kann, wann ein Watchdog in der MCU ausgelöst wird, muss jede Veränderung des Impulszählers gespeichert werden. Wird ein Drehgeber mit 1000 Imp/U in einem Gelenk betrachtet, ergeben sich bei einer 180° Drehung jeweils 500 Impulse für das Hin- und Zurückdrehen. Dadurch wird das interne EEPROM nach $\frac{100.000\ Schreibzykeln}{1.000\ Impulse} = 100$ Bewegungen zerstört. Diese Lösung ist praktisch nicht verwendbar und ein Drehgeber wird nicht eingesetzt.
Eine andere Alternative ist ein „360° Rotary Position Sensor“ auf dem Prinzip eines Hallsensoren-Arrays, wie beispielsweise der MLX90316 der Firma Melexis® [NVM09]. Der Sensor ist nach einmaliger Kalibrierung in der Lage, den absoluten Winkel einer Achse anhand eines Magnetfeldes zu detektieren. Der Messmagnet muss über der Stirnseite des Bausteins gedreht werden. Diese Anordnung ist in diesem Roboter-Bein schwer umsetzbar (3.3.3). Daher kann dieser Sensor nicht verwendet werden. Auch DC-Antriebe mit integriertem Winkelgeber, können nicht verwendet werden, da diese wie ein Drehgeber nur Impulse relativ zur Umdrehung ausgeben.

4.2.4.1 Magnet Folien Potentiometer

Bedingt durch die Anforderungen wird ein Magnetfolienpotentiometer (MFPoti[de], MagnetoPots [us][8]) verwendet. Hierbei wurde der Typ „SENSOFOIL® Magnet FR4“ des Herstellers Hoffmann + Krippner GmbH verbaut. Der Aufbau eines MFPotis verwendet eine Epoxidharzplatte als Gehäuse. In einer breiten Rille sind mehrere Folien eingearbeitet. Zwei elektrisch leitfähige Folien dienen als Widerstand und werden durch äußere Kunststoffstege voneinander getrennt. Hinter diesen beiden Folien befindet sich eine Magnetfolie. Wird mit einem externen Magneten die Magnetfolie angezogen, knicken die leitenden Folien ein und es kommt zu einem Kontakt an der belasteten Stelle. Je nach Position des Magneten ergibt sich der elektrische Widerstand. Bedingt durch die Bauform (*Abbildung 4-6*) kann ein Winkel von 300° erfasst werden. Der verwendete Typ hat einen Widerstand von 1kΩ bis 10kΩ. Die Linearität wird mit maximal 2% angegeben [HofkA]. Durch eigene Messungen konnte keine Abweichung festgestellt werden. Die Haltbarkeit wird mit mindestens *20 Millionen* Wiederholungen angegeben [HofkA]. Auch die angegebene Wiederholgenauigkeit von 1mm wird unterschritten. In eigenen Tests konnten keine Aussetzer beim Bewegen des Magneten festgestellt werden. Das MFPoti kann laut Hersteller von 5V bis 30V betrieben werden und ist für den Automobilbau geeignet.

Abbildung 4-6 FR4 MFPoti

Bei sehr schnellen Bewegungen (>80 U/min) werden die Folien nicht mehr zuverlässig geknickt. Dies stellt sich messtechnisch wie ein fehlender „Schleifer“ dar. Der Messwert wird aber sofort nach dem Unterschreiten der kritischen Geschwindigkeit wieder korrekt abgebildet.

[8] In den USA wird die Bezeichnung „MagnetoPots“ verwendet. Sie ist die Markenbezeichnung der Firma Spectra Symbol Inc. Weiterhin wird in der US-Fachliteratur das Akronym „Pancake Pot“ verwendet. Im deutschen Sprachraum wird die Abkürzung MFPoti von einigen Herstellern verwendet. Es hat sich noch keine einheitliche Abkürzung etabliert.

Diese Magnetfolienpotentiometer werden aufgrund ihrer Robustheit und den obigen Leistungsdaten in den Gelenken als Winkelsensoren verwendet.

4.2.5 MCU-Platine mit Ethernet

In diesem Unterkapitel, wird die Hardware für den Bein-Kontroller realisiert. Der Schaltplan wird in einen MCU-Bein-Kontroller und einen MCU-Ethernet-Kontroller aufgeteilt. Die beiden Teile sind logisch ein Layout.

4.2.5.1 MCU-Bein-Kontroller

Der MCU-Bein-Kontroller ist der zentrale Knotenpunkt für die Sensoren und Aktoren eines Beins. Der Schaltplan wird in *Abbildung 4-7* dargestellt.
Der ATMEGA 128 wurde nach der „Application Note AVR042“ [ATM10] beschaltet. Der Reset ist *nicht* auf einen Taster gelegt. Bei einem Sturz des Roboters könnte der Reset-Taster z.B. durch lose Kabel fälschlich ausgelöst werden. Ein Reset kann über Pin5 am ISP Anschluss (P3) ausgelöst werden. Die ISP Schnittstelle wird auf eine 3x2 Stiftleiste gelegt und ist kompatibel zur Definition der AVR ISP [ATM101] Schnittstelle. Damit kann das STK500 direkt zur Programmierung des Flashspeichers angeschlossen werden. Der Referenzspannung-Pin AREF für die AD Wandler (ADC) wird mit 5V aus der Spannungsquelle beschaltet. Der Takt wird über einen TTL-Quarzoszillator in der Industrial Version erzeugt. Dieser bietet die höchstmögliche Abschirmung gegen elektromagnetische Felder und wird auch auf dem MCU-Ethernet-Kontroller genutzt, gekennzeichnet durch „<<CLK“ als seitenübergreifendes Signal.
Die Anschlüsse für die Motortreiber Module RN-VNH2 (4.2.3) sind 1:1 beschaltet und als Stiftleiste P1 und P2 gekennzeichnet. Die Ausgänge OC3A, B, C und OC2 des ATMEGA 128 sind mit den PWM Eingängen der Motortreiber verbunden. Die nötigen Pulldown Widerstände für die PWM sind im Motortreiber integriert. Die einzelnen Schaltfunktionen (*Tabelle 4-1*) des Motortreibers sind auf I/O Pins des ATMEGA 128 gelegt. Hier werden die internen *Pullup* Widerstände der MCU genutzt. Die Variante als Pulldown Widerstand wäre im Extremfall sicherer. Da aber bei einem Ausfall der MCU auch die PWM ausfällt, kann der betreffende Antrieb nicht mehr aktiv sein.
Die Rückgabewerte der „Drehmomentsensoren“ der Treiberplatine RN-VNH2 sind mit den ADC Eingängen (4 bis 7) des ATMEGA 128 verbunden. Das Design der Module RN-VNH2 gibt eine maximale Sensorspannung von 2,5V zurück, womit die Drehmomentwerte nur zur Hälfte in der MCU abgebildet werden. Die erfassten Werte sind aber ausreichend. Diese Designentscheidung erklärt sich durch die minimale Spannung der Winkelsensoren.
Die Winkelsensoren (MFPoti) sind an die ADC Eingänge (0 bis 3) angeschlossen. Die MFPotis (4.2.4.1) müssen laut Hersteller mit mindestens 5V betrieben werden und sind daher an den äußeren Pins mit 5V und GND versorgt. Der „Magnet-Schleifer“ ist auf den ADC geschaltet. Um einen Defekt sicher zu erkennen (5.3.4.3.1, SensorFaultCheck), wurde die Schleifer-Spannung nicht durch einen zusätzlichen Spannungsteiler angepasst. Dadurch kann beispielsweise ein Kabelbruch genauer erkannt werden, da zwischen dem minimal zu erfassenden Winkel und 0V ein relativ großer Abstand besteht. Die Winkelsensoren werden über die Stiftleiste P3 mit der MCU verbunden. Die Messspannungen werden nicht über einen Kondensator geglättet. Die Glättung erfolgt durch die Software, da hier einfacher Änderungen in der zukünftigen Weiterentwicklung möglich sind. Die Versorgungsspannung ist die Winkelsensorspannung und die Referenzspannung der MCU, wodurch sich Spannungsschwankungen nur gering auf die Messwerte auswirken.

Die Kontakte der Fußtaster (3.3.1) werden auch über die Stiftleiste P3 mit dem System verbunden und werden auf digitale Eingänge des ATMEGA 128 gelegt.
Die Verbindung zum MCU-Ethernet-Kontroller wird durch die Signale <<RXD0, <<TXD0 für die serielle Verbindung und <<CTS,<<RTS als Flusskontrolle hergestellt.

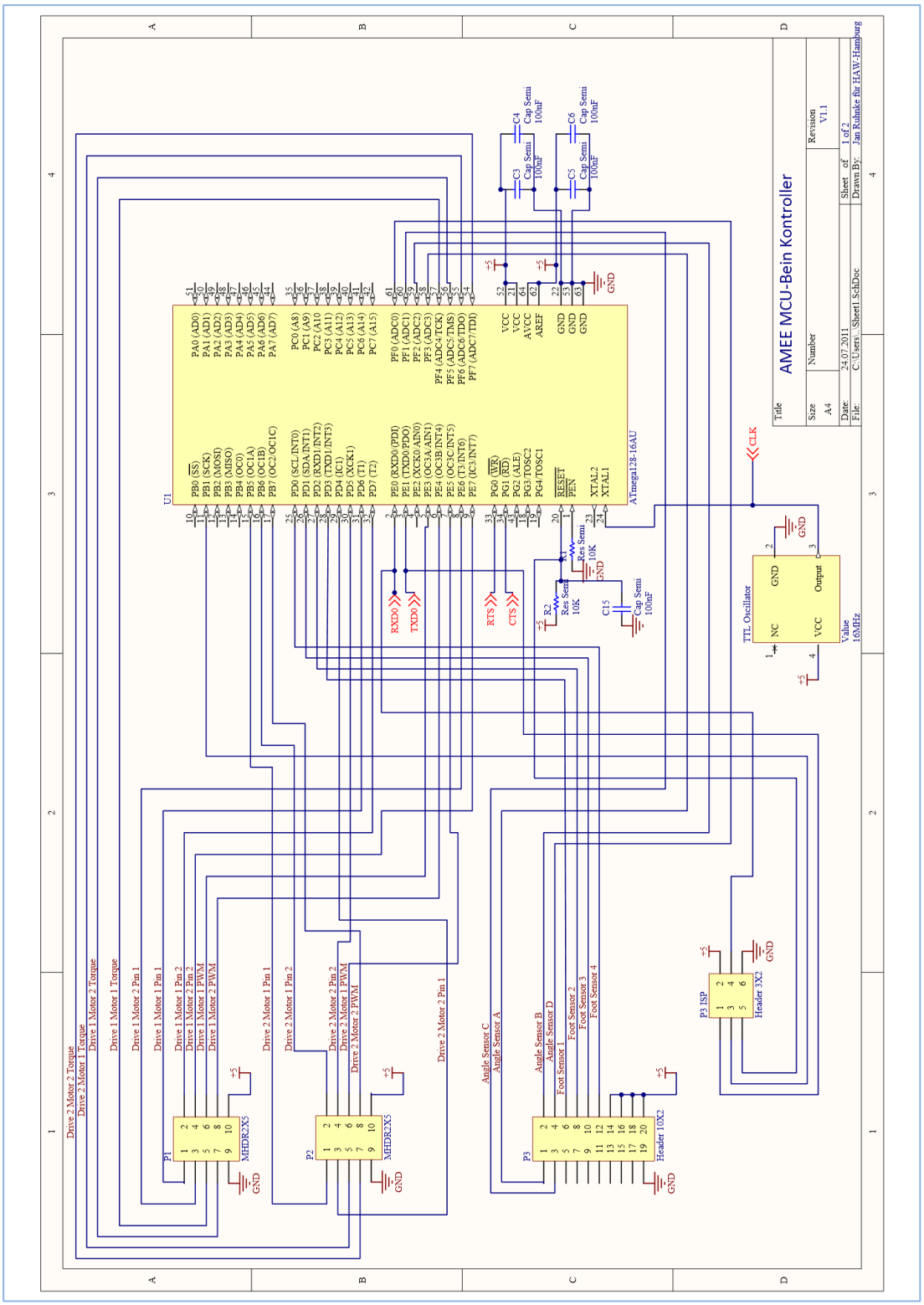

Abbildung 4-7 Schaltplan MCU-Bein-Kontroller

4.2.5.2 MCU-Ethernet-Kontroller

Der MCU-Ethernet-Kontroller ist die Schnittstelle zwischen MCU-Bein-Kontroller und Ethernet. Der Schaltplan ist in *Abbildung 4-8* dargestellt. Als MCU wird der ATMEGA 162 verwendet und nach „Application Note AVR042“ [ATM10] beschaltet. Diese MCU wurde ausgewählt, da sie eine Adressbuserweiterung hardwareseitig unterstützt. Für die implementierte Software (5.3.3.2) sind die MCU Ressourcen ausreichend. Die MCU hat eine eigene ISP Schnittstelle auf der Stiftleiste P4-ISP. Auch für diese MCU kann der Reset nur über Pin 5 der ISP Schnittstelle ausgelöst werden. Dieser Teil des Kontroller basiert auf dem Referenzdesign des „Easy TCP/IP“ Boards der Firma „MCS Electronics“ [MCSkA]. Das Modul iim7000A (4.2.2) wird im Bus-Mode betrieben. Es ist an die Adressleitungen (A0 – A15) angeschlossen. Die unteren Adressenleitungen (A0 – A7) sind über einen Latch vom Typ 74HC573 verbunden, um den W3100 (4.2.2) in den MCU Adressbereich &h8000-&hFFFF zu legen. Es werden Adressierung und Daten getrennt. Der höchste Adress-Pin A15 wird mit einem Transistor invertiert, um am W3100A den CS (Chip Select) zu generieren. Der ALE Pin (Address latch enable) wird als Steuersignal LE (Latch enable) am Latch verwendet. Durch diese Schaltung kann der W3100A direkt mit der Hardwareadresserweiterung wie ein externer RAM-Baustein angesprochen werden. Der Großteil des Adressbereichs wird für die Puffer des W3100A verwendet. Auf den höheren Adressen befinden sich die Steuerregister des W3100A. Der Reset des iim7000A kann mit dem Pin PE0 durch die MCU per Software, ausgelöst werden. Aus Sicherheitsgründen besitzt das iim7000A Modul einen Reset- und einen invertierten Reset-Eingang. Das Signal des PE0 Pins der MCU wird einmal direkt und einmal invertiert durch einen Transistor an das iim7000A Modul geleitet.

Die USART (RXD0, TXD0, CTS, RTS) des MCU-Bein-Kontrollers (4.2.5.1) wird gekreuzt an die Hardware USART des ATMEGA 162 geleitet.

In diesem Teil wurde ein Spannungswandler integriert. Aus der systemweiten Spannung von 12V werden Spannungen von 5V und 3,3V erzeugt. Die Spannungswandler (LF33 und LF50) können jeweils mit 500mA belastet werden. In einer Simulation wurde eine maximale Belastung des 5V Anteil mit ca. 350-400mA berechnet. Der 3,3V Anteil wird maximal mit 250mA belastet. Die Spannungswandler erzeugen mit einer Eingangsspannung von 6V bis 16V die geforderten Ausgangsspannungen. Durch die „Brown-out Protection“ [ATM02] der MCU, wird beim Unterschreiten von 4,7V an den MCUs das System automatisch angehalten. Dafür müsste aber die Systemversorgungsspannung von 12V auf unter 6V einbrechen.

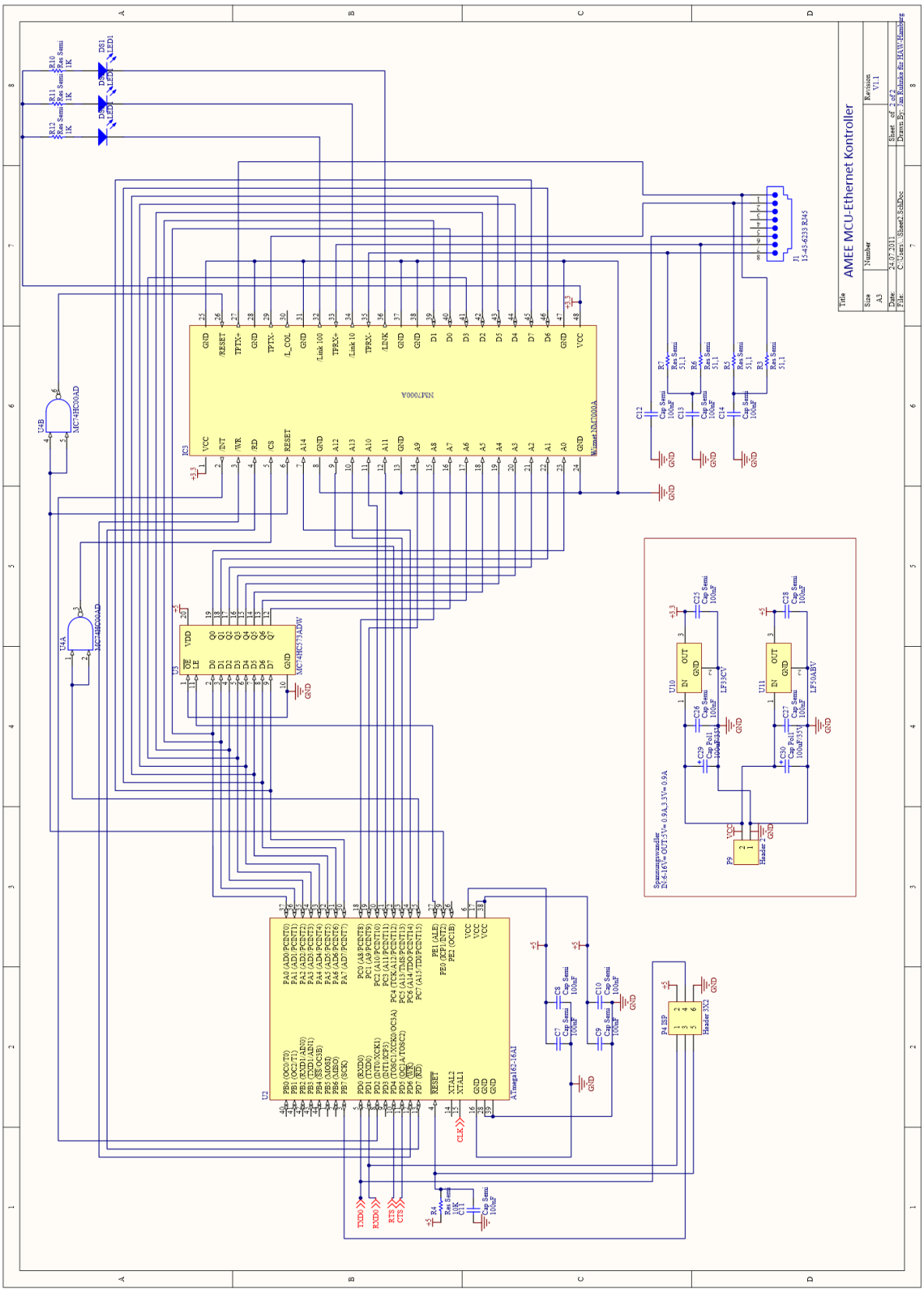

Abbildung 4-8 Schaltplan MCU-Ethernet-Kontroller

4.2.5.3 Layout des MCU-Bein und MCU-Ethernet Kontrollers

Die Platine des kompletten MCU-Kontrollers für ein Bein wird in *Abbildung 4-9* gezeigt. Die Platine ist 60mm x 190mm groß und kann in ein Vierkant-Profil des Torsos montiert werden. Die Platine wird über fünfmal 4mm Schrauben gehalten und durch die mittlere Bohrung gegen mechanische Schwingungen gesichert. Die Spannungswandler (rechts unten, *Abbildung 4-9*) werden in das Alu-Profil des Roboter-Torsos (3.4) zur Wärmeableitung geschraubt.

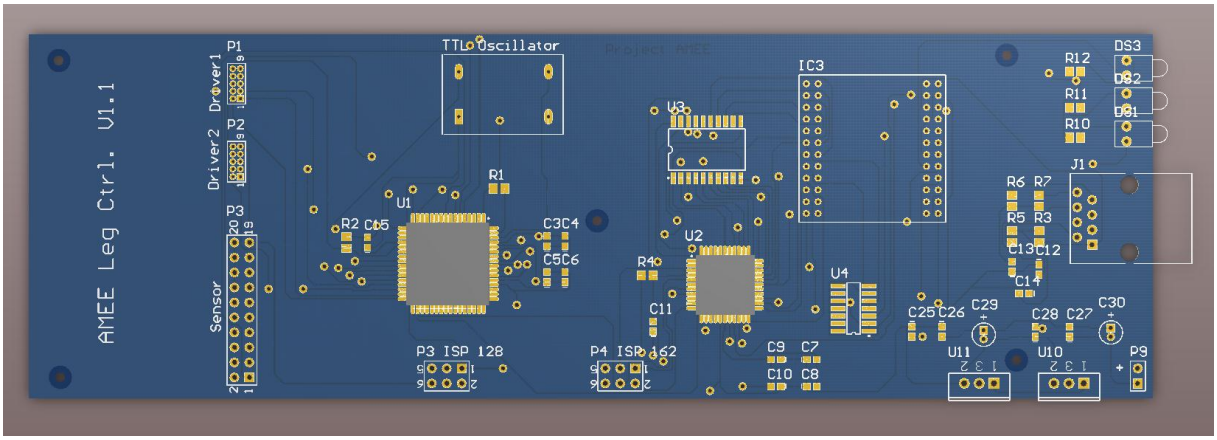

Abbildung 4-9 Layout des MCU Kontrollers

Der Ethernet-Anteil wurde rechts platziert, um Störungen durch das Ethernet in den Sensormessleitungen zu vermeiden. Netzwerkanschluss und Stromanschlüsse sind rechts. Mess- und Steuerleitungen werden links aus der Platine geführt. Die linke freie Platinen-Fläche soll für eine geklebte Kabelzugentlastung genutzt werden. Diese Platine wurde in dieser Arbeit nicht real aufgebaut (6.4). Der Prototyp wurde mit dem STK500, STK501 und dem Easy-TCP/TP-Board [MCSkA] realisiert (*Abbildung 4-10*).

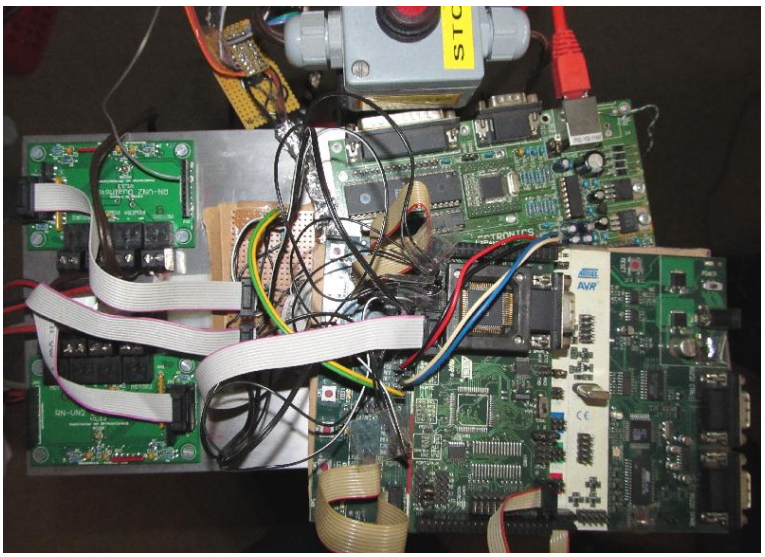

Abbildung 4-10 Prototypenaufbau auf dem Beinhalter

4.3 Zusammenfassung

Das Kapitel 4 hat die Designentscheidungen für den elektronischen Aufbau des Roboters AMEE konkretisiert. In diesem Kapitel wurde folgendes ermittelt.

a. Der Roboter wird mit einem verteilten System betrieben werden(4.1)
b. Als Bussystem wird Ethernet verwendet(4.2).
c. Die Beine werden dezentral organisiert(4.1).
d. Es wurden Antriebe, Sensoren und Bauteile spezifiziert(4.2.1 bis 4.2.4.1).
e. Jedes Bein verfügt über einen eigenen Subprozessor(4.2.1).
f. Jedes Bein verfügt über eine eigene Ethernet-Schnittstelle(4.2).
g. Es wurde eine Schaltung und eine Platine für die Beine entwickelt(4.2.5).

Die getroffenen Designentscheidungen stehen hauptsächlich mit der Modellierung im Kapitel 5 in Beziehung.

5 Software

In folgenden Abschnitten werden die Schritte der Planung und Realisierung der Software beschrieben. Die einzelnen Schritte sind nach Problemstellung strukturiert und bauen aufeinander auf. Die Hard- und Software Entscheidungen sind voneinander abhängig und haben sich beeinflusst was durch eine Gliederung teilweise verloren geht.
Das Konzept des *kompletten* Laufsystems wird in den Kapiteln 5.1 und 5.2 erläutert. Im Kapitel 5.3 wird die Software der Beine (MCU-Subprozessoren) erläutert. Da der Hauptkontroller als Koordinator für das Laufsystem in die höheren Funktionen des Roboters eingebettet werden soll, wird er im Kapitel 5.4 nur skizziert. Hier werden auch einige Kriterien für die Weiterentwicklung aufgestellt.

5.1 Kritische Grundüberlegung

Dieses Laufsystem für unebenen Untergrund bedarf einer kritischen Betrachtung, nicht nur wie im Kapitel (2.1), sondern auch bei der Herangehensweise für die Softwareentwicklung. Beobachtet man Tiere, wird schnell klar, dass sie keine Integrale und Winkelfunktionen vor jedem Schritt berechnen. Diese Fähigkeit scheint erlernt zu sein. Aus Sicht der Informatik sind ähnliche lernende Systeme (z.B. Neuronale Netze) vorhanden. Für das AMEE Laufsystem wäre ein Neuronales Netz für die inverse Kinematik (5.3.1.1) denkbar. Da aber der Roboter auch mit Menschen agieren soll, stellt sich die Frage der Zuverlässigkeit eines Neuronales Netzes, das direkt mit den Antrieben verbunden ist[9]. [Sch10](Seite 589 ff.)
Für dieses Laufsystem ist der mathematische Weg die schnellste und sicherste Lösung. Ein lernendes System ist aber für die höhere Logik über dem ausführenden Laufsystem unumgänglich.

5.2 Identifizierung der Problemstellung

Wie bereits im Kapital Hardware (4) angedeutet, wird die Software als *Verteiltes System* (VS) ausgelegt. Bei diesem Laufsystem liegt das Hauptaugenmerk nicht nur auf der Lastverteilung. Eine weitere Forderung ist, dass *jedes zu lösende Problem dort bearbeitet wird, wo es auftritt.* Es muss eine Verteilung nach logischen und physikalischen Kriterien erfolgen. Beispielsweise ist das Problem der inversen Kinematik nicht das Problem des Hauptkontrollers. Der Hauptkontroller hat nur die Aufgabe den Roboter an eine bestimmte Position zu bewegen. Er bearbeitet die dafür nötigen Schritte. Vereinfacht betrachtet, liefert der Hauptkontroller nur die Daten, wo welcher Fuß platziert werden soll. Welchen Winkel die einzelnen Glieder in einem Bein dafür haben müssen und wie welcher Motor dafür angesteuert werden muss, ist dem Hauptkontroller unbekannt. Diese Aufgabe löst ein Subprozessor (MCU) in jedem Bein. Die Machbarkeit eines MCU basierenden Bein Sub-Prozessors wird in Kapitel 5.3.1 überprüft.
Der Hauptbeweggrund für diese Teilung ergibt sich aus dem Problem der vielen kleinen zeitkritischen Teilaufgaben bei der Steuerung der Beine bzw. der einzelnen Antriebe. Zwar ist es mit heutigen höheren Sprachen und Real Time Betriebssystemen möglich, diese Aufgabe zu realisieren. Es werden aber immer Probleme mit zeitkritischen Tasks entstehen, die gleichzeitig auftreten. Als Beispiel sei hier der Honda ASIMO (1.5.1) genannt, der mit 4x4 Desktop Prozessoren und „vielen ECUs“ ausgestattet ist [Hon07]. Beim Honda Research Institut (HRI) wurde schon Ende der 1980er Jahre

[9] Dies gilt für einfache neuronale Netze. Ein sicherer lernender Automat wäre beispielsweise mit der Theorie der Markov-Ketten möglich, aber dies erscheint für diese Arbeit zu aufwendig.

bemerkt, dass die Problemstellung der zeitgleichen und zeitkritischen Tasks zwar mit einem Kontroller gelöst werden kann, aber einen hohen Entwicklungsaufwand erfordert [Jan10]. Dieses Konzept wurde für dieses Laufsystem übernommen.
Leider sind genaue Implementierungsdetails nicht veröffentlicht worden, da es sich bei ASIMO um ein kommerzielles Produkt handelt.
Zu dem obigen Problemkomplex kommt hinzu, dass wesentlich höhere Lasten auf dem Hauptkontroller verarbeitet werden sollen. Als Schlussfolgerung aus den Erfahrungen des HRI wird hier versucht, möglichst viele Aufgaben aus dem Hauptkontroller auszulagern.
Der Hauptkontroller wird nur für höhere Logik genutzt. Treten für die Beine unlösbare Probleme auf, wird der Hauptkontroller zur Unterstützung benutzt. Aus diesem Grund werden auch die einzelnen Zielsetzungen der Software in den jeweiligen Kapiteln erörtert.

5.3 MCU Bein Software

In diesem Abschnitt wird die Realisierung der Bein-MCU erläutert. Die diskutierten Design-entscheidungen sind spezialisierte Lösungen, die teilweise nur für diese Konstruktion der Beine gelten.

5.3.1 Machbarkeitsprüfung

In diesem Unterkapitel wird *vor der Modellierung* geprüft, ob eine MCU Realisierung möglich ist. Das Hauptproblem in der Entwicklung des MCU Bein Kontrollers ist die *Real Time Execution (RTX)*. Jede Verzögerung hat reale, physikalische Auswirkungen auf die Umwelt des Roboters. Bevor mit der Entwicklung eines MCU Kontrollers begonnen werden kann, werden vorab Funktionen mit hoher Laufzeit identifiziert. Dabei müssen auch die Ressourcen einer MCU beachtet werden. Weiterhin müssen sich diese Verfahren schnell synchronisieren lassen. Es dürfen keine großen Datenmengen ausgetauscht werden.
Die lastaufwendigste Funktion ist die inverse Kinematik. Es wird geprüft, ob eine inverse Kinematik unter den Vorgaben überhaupt realisierbar ist.
Die Modellierung des Bein-Kontrollers wird im Kapitel 5.3.2 fortgesetzt.

5.3.1.1 Inverse Kinematik

Die inverse Kinematik (IK) ist eines der Grundprobleme in der Robotik. Es gibt viele Lösungsverfahren, aber kein Standartverfahren. Die Verfahren müssen an den jeweiligen Anwendungsfall bzw. die Bauart des Roboters angepasst werden. Bei einer IK wird eine gewünschte Position des Endeffektors vorgegeben. Die IK ermittelt die Winkel der einzelnen Glieder einer Gliederkette. Dabei handelt es sich bei allen mathematischen Verfahren um die Lösung eines Gleichungssystems. Oft ergeben sich aus diesem Gleichungssystem mehrere korrekte Konfigurationen der Glieder.
Problematisch ist auch der Aufwand einiger Lösungsverfahren, die aus der Sicht der Laufzeit extrem teuer werden.

Für einen USAR Roboter und im Speziellen beim Projekt AMEE muss eine optimale Lösung gefunden werden, die folgende Kriterien erfüllt:

a. Das Verfahren muss *eine* eindeutige Lösung liefern.
b. Es muss einen berechenbaren und stabilen Aufwand haben.
c. Falls ein Punkt nicht erreicht werden kann, soll das Verfahren eine möglichst gute Annäherung liefern.
d. Es soll einen möglichst geringen Rechenaufwand haben.

5.3.1.2 Auswahl des IK Verfahrens

Der generische Weg wäre eine Denavit-Hartenberg-Transformation (DH) mit numerischer Lösungssuche (z.B. das Newton-Verfahren). Diese iterativen Verfahren sind für allgemeine Problemstellungen gut geeignet und basieren immer auf einer ähnlichen Implementierung. Diese Entwicklungsvereinfachung wird aufgrund massiver Matrixmultiplikationen oft mit hoher Laufzeit erkauft. Das ist für eine MCU ungeeignet und die Laufzeit ist schwer vorhersagbar. Es gibt Methoden, die Iterationen des Verfahrens zu begrenzen. Dies bewirkt aber eine ungenaue Platzierung des Endeffektors. Durch die Forderung eines stabilen Aufwands würde das DH Verfahren auf einer MCU zu stark begrenzt. Aus diesen Gründen[10] wird eine angepasste IK entwickelt. *Um dies kenntlich zu machen, wird für die Achsen die Bezeichnung A, B, C, D benutzt und nicht die üblichen Bezeichnungen* DH_1, DH_2, DH_3, DH_4.

5.3.1.3 Analytische Lösung der IK

Es muss ein Algorithmus entwickelt werden, der ohne ein numerisches Verfahren die Problemstellung lösen kann. Dieses Verfahren verfolgt einen mathematisch analytischen Ansatz, eine Vorgehensweise, die häufig in 3D Computerspielen verwendet wird [Juc10], beispielsweise für Animation von Spiele-Avataren. Zwar können heutige Spiele-Konsolen und PC Spieleentwickler beim Endprodukt auf enorme Rechenleistung zurückgreifen, aber durch die Anzahl der parallel ablaufenden Berechnungen ist ein leistungsfähiges Verfahren nötig. Der Ansatz ist, auch wenn er für virtuelle Umgebungen konzipiert wurde, auf die Robotik übertragbar. Dieses Verfahren ist eine von vielen Möglichkeiten, das Problem zu lösen. Aber das analytische Verfahren ist anschaulich und kann dadurch leicht angepasst werden.

[10] Es gibt die Möglichkeit, das DH-Verfahren so zu vereinfachen, dass eine ähnliche Lösung entsteht. Der Aufwand der Herleitung ist mindestens ebenso hoch wie das vorgestellte Verfahren. Die Implementierung des analytischen Ansatzes ist leichter zu warten.

5.3.1.4 Abstraktion des Beins

In den folgenden Abschnitten wird ein idealisiertes Modell des Beins betrachtet. Für die Abstraktion der Mechanik werden die Achsen wie in *Abbildung 5-1* mit A, B, C, D benannt und zusätzlich mit einem Farbcode versehen. Achse A (rot), Achse B (grün), Achse C (blau) und Achse D (gelb).

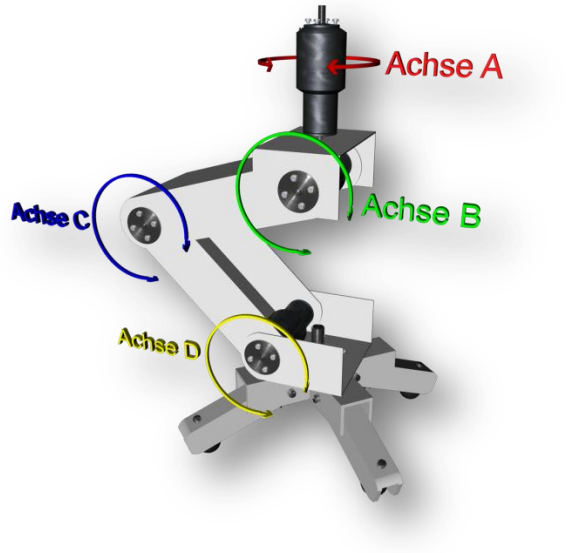

Abbildung 5-1 Achsen Bein

Das Bein in *Abbildung 5-1* hat vier Freiheitsgrade (4 DOF / Degrees of Freedom) und kann damit Zielpunkte im dreidimensionalen Raum ansteuern. Der Fuß ist als Endeffektor zu betrachten. Die Hüfte (Achse A und B) wird für die IK als Fixpunkt betrachtet. Die Winkel der Achsen sind nur durch die mechanischen Bauteile begrenzt. Die Achse A wird aus der Berechnung ausgeklammert und als Drehung wieder der Endposition des Endeffektors (Fuß) zugefügt.

5.3.1.5 Abtrennen des 3D Anteils

Das dreidimensionale Modell wird für die Berechnung vereinfacht, indem es in zwei zweidimensionale Modelle zergliedert wird: eine Rotation in der horizontalen und eine translatorische Bewegung in der vertikalen Ebene.

Es wird die Drehung um Achse A berechnet. Durch die Betrachtet des Beinmodells in der Draufsicht (*Abbildung 5-2*), bildet eine zu erreichende Position einen zweidimensionalen Punkt in einer Fläche. Vom Fixpunkt aus gesehen, wird diese Betrachtung in Gradienten zerlegt, in einen Winkel und einen Radius.

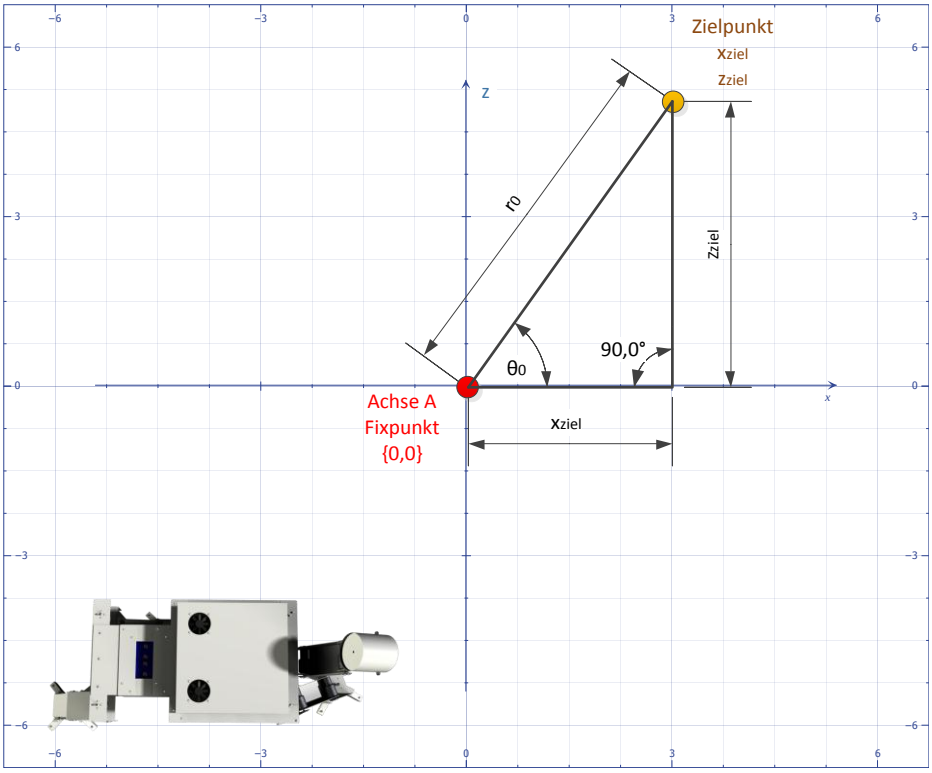

Abbildung 5-2 Draufsicht, Winkel und Radius

Der Ursprung des Koordinatensystems für die IK liegt immer am Fixpunkt des Beins. Die Zielkoordinaten sind also relativ zum Roboter und nicht zur Umwelt. Durch diese Vereinfachung gelten folgende Gleichungen:

$$Radius\ r_0 = \begin{cases} \sqrt{{x_{ziel}}^2 + {z_{ziel}}^2}\ f\ddot{u}r\ x_{ziel} \geq 0 \\ \sqrt{{x_{ziel}}^2 + {z_{ziel}}^2} \cdot (-1)\ f\ddot{u}r\ x_{ziel} < 0 \end{cases}$$

$$Winkel\ \theta_0 = \begin{cases} \tan^{-1}\left(\frac{z_{ziel}}{x_{ziel}}\right), f\ddot{u}r\ x_{ziel} \geq 0 \\ \tan^{-1}\left(\frac{z_{ziel}}{x_{ziel}}\right) + 180°\ f\ddot{u}r\ x_{ziel} < 0 \end{cases}$$ [11]

oder besser mit der atan2 Funktion

$$Winkel\ \theta_0 = \text{atan2}(z_{ziel}, x_{ziel})$$

Der Winkel θ_0 hat die x-Achse als Bezugsgerade. Dabei gilt, positive Winkel entsprechen einer Linksdrehung, negative Winkel ergeben eine Rechtsdrehung um die Achse A des Beins. Der Wert θ_0 kann direkt als Drehwinkel an den Motor (Achse A) gesendet werden und wird *nicht* mehr beachtet. Mathematisch ist diese Drehung richtig, aber durch Beschränkungen der Mechanik bzw. der Versorgungskabel, muss der Drehwinkel auf die vorderen Sektoren ① und ② begrenzt werden (siehe *Abbildung 5-3*). Muss trotzdem hinter den Nullpunkt gegriffen werden, soll sich das Bein nicht in

[11] Die Fallunterscheidung und die Prüfung auf Division durch Null, ist für die Implementierung mit der Funktion „atan2“ nicht nötig, da im Framework bzw. ANSI C und .net bereits vorhanden.

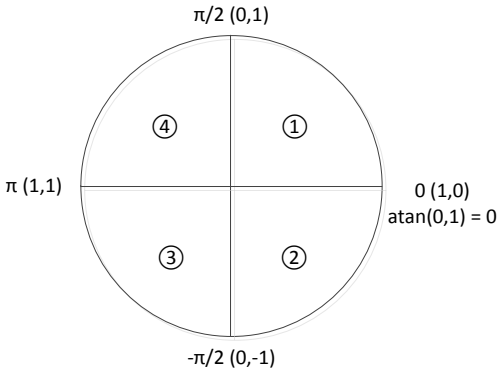

Abbildung 5-3 Sektoren der atan2 Funktion

den hinteren Bereich drehen und vorwärts greifen, sondern dreht sich stattdessen in die entgegengesetzte Richtung und greift nach Hinten (z.B. zu ① drehen, nach ③ greifen).

Dazu ist eine Korrektur des errechneten Winkels um +180° für θ < 90° beziehungsweise -180° für θ > 90° notwendig. Weitere Korrekturen sind in der Herleitung nicht nötig. Die Anpassungen an die Mechanik erfolgt im Kapitel 5.3.4.4.8.

Korrektur der X-Koordinate

Betrachtet man das entstandene 2D Modell in der Seitenansicht (*Abbildung 5-4*), wird der Radius r_0 als neue X-Koordinate in der Seitenansicht benutzt und auch in der IK so verwendet. Es gelten einige für IKs übliche Normierungen, die an die vorliegende Problemstellung angepasst wurden. Der Fixpunkt ist auch hier der Nullpunkt des Koordinatensystems. Der Winkel am Fixpunkt (Achse B, *Abbildung 5-4*) unterhalb des Torsos ist für ein Bein negativ. Der Winkel der Achse C (*Abbildung 5-4*) wird relativ zur verlängerten Oberschenkelachse angegeben. Auf der X-Achse bedeutet ein positiver Wert einen Zielpunkt vor dem Fixpunkt und ein negativer Wert hinter dem Fixpunkt.

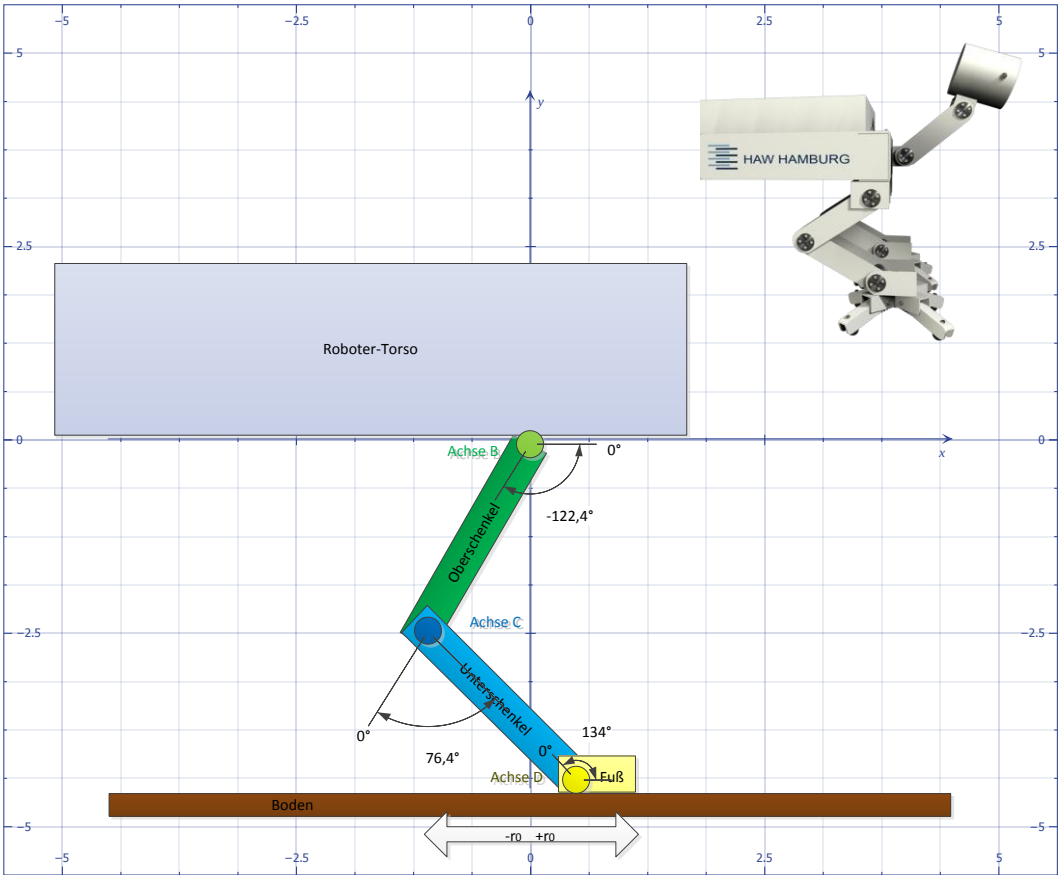

Abbildung 5-4 Seitenansicht Beinmodell

Da der Fuß mit Achse D (*Abbildung 5-1*, *Abbildung 5-4*) für die IK keine Rolle spielt, kann auch dieser außerhalb berechnet werden. Lösung von θ_3, 5.3.1.7.3.

5.3.1.6 Definition des „Two Bone" Problems

Durch die obige Abstraktion des Beins bleibt für die eigentliche IK nur noch ein sogenanntes „*Two-Bone*" Problem (zwei Glieder) bestehen. Es werden nur noch die Winkel zwischen Torso, Oberschenkel und der Winkel des Knies (Achse C) benötigt.

Das folgende Verfahren und die Herleitung beruhen auf den Ideen von Ryan Juckett [Juc10]. Der Problemkomplex besteht aus einer Gliederkette mit zwei Gliedern, wobei Glied 1 sich am Fixpunkt befindet. Glied 2 liegt hierarchisch unter Glied 1. Jedes Glied enthält zwei Variablen, den Winkel θ und die Gliedlänge d. Ziel ist es, mit dem Endpunkt des Gliedes 2 einen Zielpunkt (x_{ziel},y_{ziel}) zu erreichen, und dabei $\theta_{1,2}$ zu errechnen. Um das theoretische Modell besser zu verdeutlichen, wird in *Abbildung 5-5* eine für die Mechanik von AMEE schwer erreichbare Position der Gliedmaßen gewählt. *Die mechanischen Grenzen werden hier nicht beachtet und im Kapitel 5.3.4.3.1 erörtert*. Der Farbcode der Gliedmaßen bleibt erhalten.

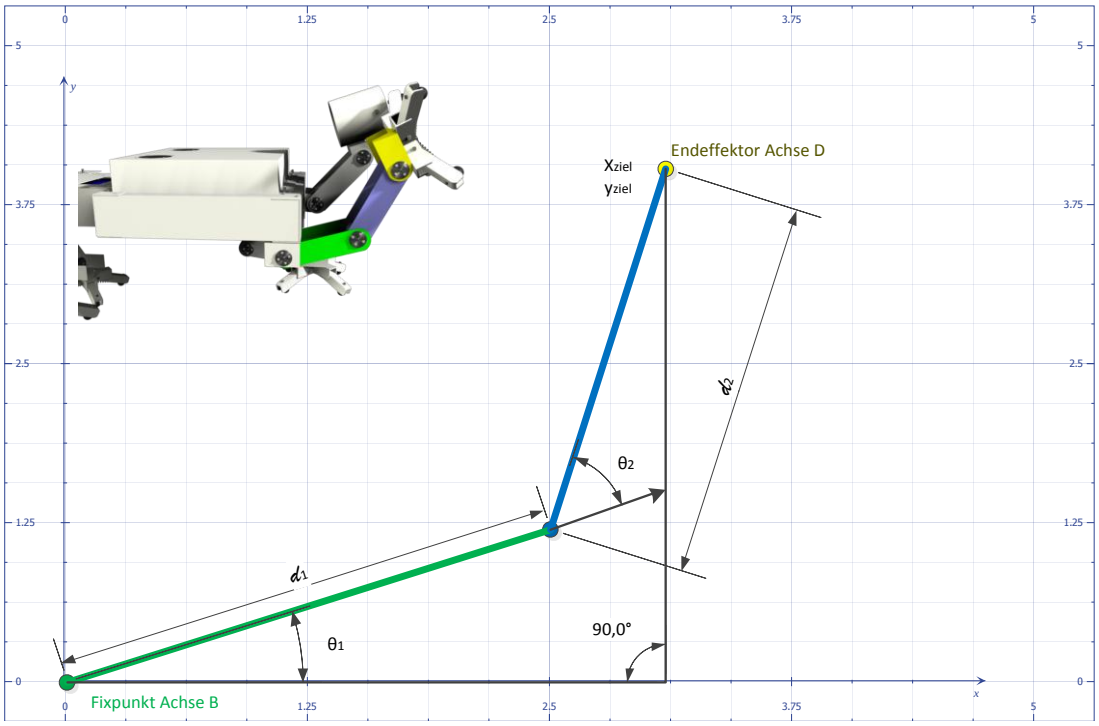

Abbildung 5-5 Modell in 2D

Wird die Gliederkette mathematisch betrachtet, ergeben sich zwei Lösung, oder keine Lösung, um vom Fixpunkt zur Zielkoordinate mit dem Endeffektor zu gelangen. Im Fall des nicht Erreichens der Zielkoordinate ergibt das Verfahren trotzdem eine eindeutige Lösung, bei der die Gliederkette auf die Zielkoordinaten zeigt. Wird die Berechnung von θ_2 (Achse C) auf den positiven Bereich eingeschränkt, ergibt dies *eine* eindeutige Lösung.

Ob eine Zielkoordinate erreicht werden kann, wird von den beiden Gliederkettenlängen d_1, d_2 festgelegt. Der minimale Abstand zum Fixpunkt wird bestimmt durch $|d_1 - d_2|$. Der maximale

Abstand zum Fixpunkt wird durch die Summe d_1, d_2 bestimmt. Der Abstand der Zielkoordinate zum Fixpunkt wird mit $r = \sqrt{{x_{ziel}}^2 + {y_{ziel}}^2}$ berechnet. Um eine Lösung bzw. den Zielpunkt zu erreichen, muss der Abstand *r* zwischen dem minimalen und dem maximalen Abstand der Gliederkette liegen. Die Bedingung $|d_1 - d_2| > \sqrt{{x_{ziel}}^2 + {y_{ziel}}^2} < d_1 + d_2$ muss wahr sein, um eine Lösung zu erhalten. Grafisch dargestellt ergibt sich ein „Lösungsring“.

5.3.1.7 Mathematische Problemdefinition

Für die IK werden die zwei unbekannten Winkel θ_1 und θ_2 gesucht. Bekannt sind die Variablen d_1, d_2 und x_{ziel}, y_{ziel}.

5.3.1.7.1 Lösung von ϑ_2

Die Lösung besteht aus zwei Gleichungen die in *Abbildung 5-6* beispielhaft veranschaulicht werden.

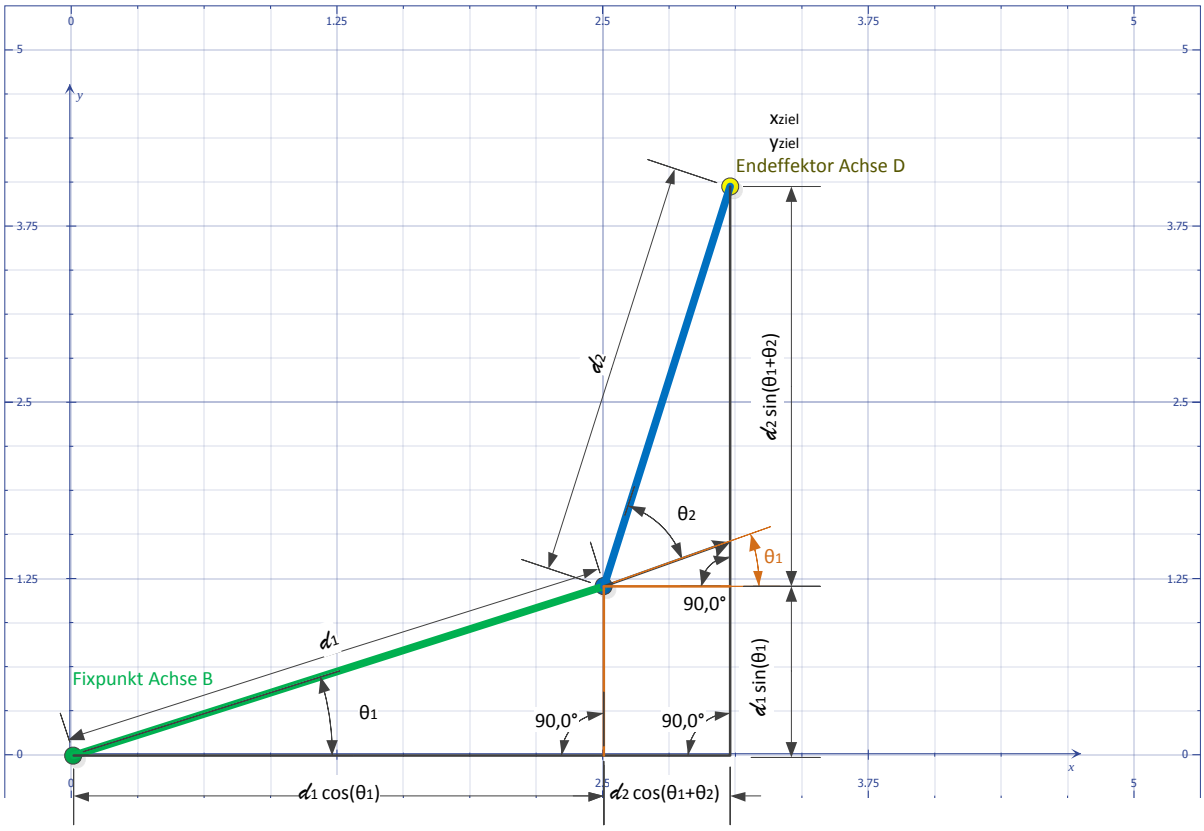

Abbildung 5-6 2D Modell erweitert mit Winkelfunktionen

Die Koordinatenwerte x_{ziel} und y_{ziel} sind die Summe ihrer anliegenden Winkelfunktionen:

$$x_{ziel} = d_1 \cdot \cos(\theta_1) + d_2 \cdot \cos(\theta_1 + \theta_2)$$
$$y_{ziel} = d_1 \cdot \sin(\theta_1) + d_2 \cdot \sin(\theta_1 + \theta_2)$$

Durch Umstellen nach θ_2 und Gleichsetzen, ergibt sich die Lösung des Kniewinkels an Achse C:

$$\theta_2 = \cos^{-1}\left(\frac{{x_{ziel}}^2 + {y_{ziel}}^2 - {d_1}^2 - {d_2}^2}{2d_1 \cdot d_2}\right)$$

Bei der Lösung für θ_2 müssen bei der Implementierung einige Ausnahmefälle beachtet werden. Wenn der $\cos(\theta_2)$ im Bogenmaß nicht in den Grenzen von [-1,1] liegt wird der Wertebereich von $\cos^{-1}$ verlassen. Dies ist der Fall wenn die Zielkoordinaten nicht erreichbar sind. Ein Wert von -1 ergibt einen Winkel von 180°, d.h. der Unterschenkel ist vollständig zum Fixpunkt gedreht. Ein Wert von 1 ergibt einen Winkel von 0°, was einen voll ausgestreckten Unterschenkel bedeutet. Ist der Wert kleiner als -1, bedeutet dies, dass die Zielkoordinaten zu nah am Fixpunkt liegen und nicht erreicht werden können. Umgekehrt verhält es sich mit einem Wert größer 1. Die Zielkoordinaten liegen zu weit vom Fixpunkt entfernt und kann nicht erreicht werden. Durch diesen Zusammenhang ist es möglich, den bestmöglichen Winkel zu errechnen, obwohl der Punkt nicht erreicht wird. D.h. der Zielpunkt wird nicht erreicht, aber der Unterschenkel zeigt in die richtige Richtung.
Die Gliederlänge ($d_1\ und\ oder\ d_2$) darf nicht Null sein, da sonst bei der Berechnung von $\cos(\theta_2)$ durch Null dividiert wird. Dies ist bei der verwendeten Mechanik nicht der Fall.
Wird $\cos^{-1}(\theta_2)$ errechnet ist das Ergebnis immer im Bereich von $[0,\pi]$. Für die betrachtete Mechanik des Beins ist dies korrekt[12].

5.3.1.7.2 Lösung von ϑ_1

Für die Lösung von θ_1 wird kein algebraischer Ansatz wie für θ_2 verwendet, da dies komplizierter und auch rechenintensiver wäre [Juc10]. Der folgende Lösungsansatz kann bei extremen Gliederlängen[13] unerwartete Ergebnisse liefern. Das Verfahren arbeitet aber bei den mechanischen Vorgaben des Beins korrekt.
In *Abbildung 5-7* werden die Winkel θ_{t3} und θ_{t4} eingeführt und sind durch die Beziehung $\theta_1 = \theta_{t4} - \theta_{t3}$definiert.

[12] Könnte das Bein in den negativen Bereich geklappt werden (das Knie hätte einen Bewegungsumfang von mehr als 180°), kann die Lösung auch mit -1 multipliziert werden. Dies hätte auch eine richtige, aber gespiegelte Lösung zur Folge.
[13] Dies gilt für die Gliederlängen 0 und das jeweilige Maximum des Datentyps.

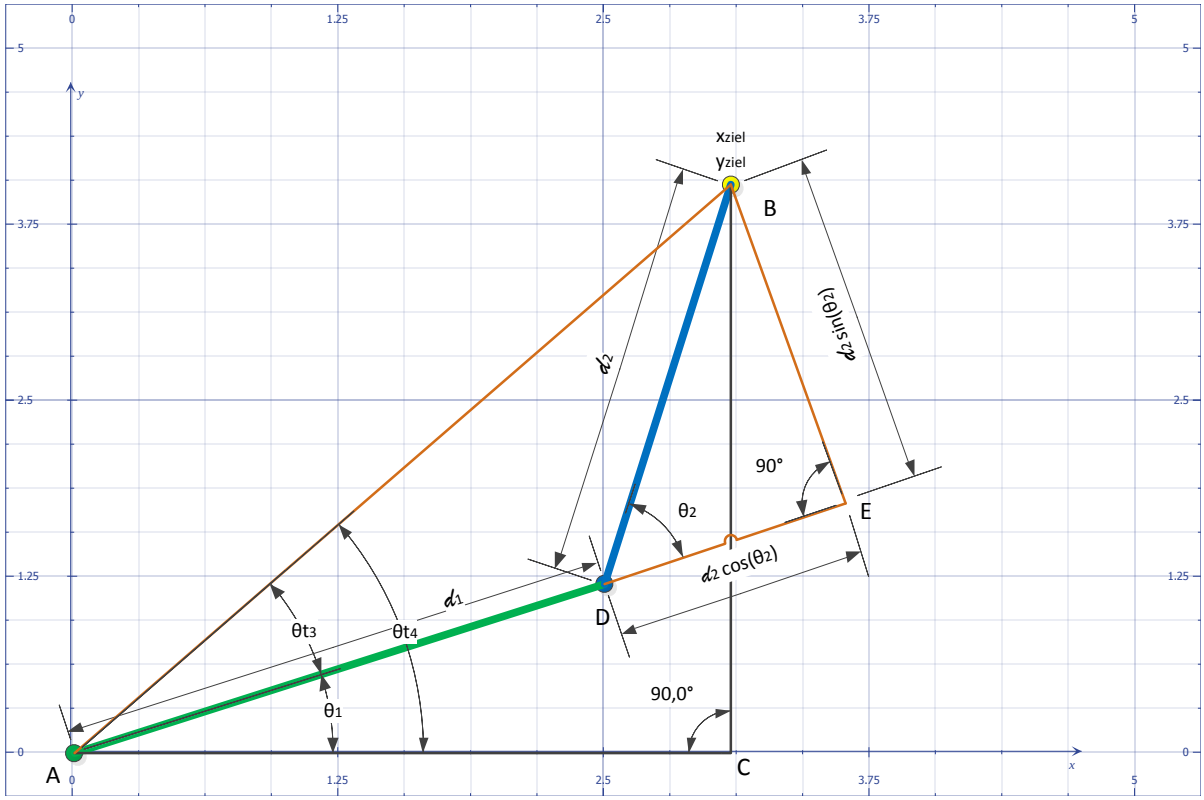

Abbildung 5-7 2D Modell erweitert um neue Winkel

Um auch negative Werte aus der Berechnung des Winkels θ_2 verarbeiten zu können, wird eine Zwischenrechnung benötigt. Für ein negatives θ_2 muss die Gleichung $\tan(\theta_4 - (-\theta_3))$ gelten. Würde $\tan(-\theta)$ gerechnet werden, ergäbe dies die falsche Lösung, da $\tan(-\theta) = \tan(\theta)$ und der Oberschenkel würde in die falsche Richtung zeigen.

Für dieses Verfahren wird die Strecke von $\overrightarrow{DE}$ und $\overrightarrow{BE}$ des Dreiecks DBE aus *Abbildung 5-7* benötigt und mit $\left|\overrightarrow{DE}\right| = d_2 \cdot \cos(\theta_2)$ und $\left|\overrightarrow{BE}\right| = d_2 \cdot \sin(\theta_2)$ berechnet. Zudem wird die Seitenlänge mit $\overrightarrow{AE} = d_1 + \left|\overrightarrow{DE}\right|$ berechnet. Damit kann $\tan(\theta_{t3})$ im rechtwinkligen Dreieck ABE ermittelt werden. Eingesetzt ergeben obige Zusammenhänge folgende Gleichung:

$$\tan(\theta_{t3}) = \frac{d_2 \cdot \sin(\theta_2)}{d_1 + d_2 \cdot \cos(\theta_2)}$$

Die Berechnung von $\tan(\theta_{t4})$ erfolgt über das rechtwinklige Dreieck ABC.

$$\tan(\theta_{t4}) = \frac{y_{ziel}}{x_{ziel}}$$

Eingesetzt in die Grundgleichung kann $\tan(\theta_1)$ gelöst werden.

$$\tan(\theta_1) = \tan(\theta_{t4} - \theta_{t3}) = \tan\left(\frac{\tan(\theta_{t4}) - \tan(\theta_{t3})}{1 + \tan(\theta_{t4}) \cdot \tan(\theta_{t3})}\right) = \frac{\frac{y_{ziel}}{x_{ziel}} - \frac{d_2 \cdot \sin(\theta_2)}{d_1 + d_2 \cdot \cos(\theta_2)}}{1 + \frac{y_{ziel}}{x_{ziel}} \cdot \frac{d_2 \cdot \sin(\theta_2)}{d_1 + d_2 \cdot \cos(\theta_2)}}$$

Vereinfacht ergibt dies:

$$\tan(\theta_1) = \frac{y_{ziel}(d_1 + d_2 \cdot \cos(\theta_2)) - x_{ziel}(d_2 \cdot \sin(\theta_2))}{x_{ziel}(d_1 + d_2 \cdot \cos(\theta_2)) + y_{ziel}(d_2 \cdot \sin(\theta_2))}$$

Die Implementierung mit der Bibliotheksfunktion atan2 berücksichtigt die notwendigen Fallunterscheidungen.

Durch das Trennen von Zähler und Nenner, kann der Winkel θ_1 isoliert werden.

$$\hat{x} = x_{ziel}(d_1 + d_2 \cdot \cos(\theta_2)) + y_{ziel}(d_2 \cdot \sin(\theta_2))$$
$$\hat{y} = y_{ziel}(d_1 + d_2 \cdot \cos(\theta_2)) - x_{ziel}(d_2 \cdot \sin(\theta_2))$$
$$\theta_1 = atan2(\hat{y}, \hat{x})$$
$$\theta_1 = atan2(y_{ziel}(d_1 + d_2 \cdot \cos(\theta_2)) - x_{ziel}(d_2 \cdot \sin(\theta_2)), x_{ziel}(d_1 + d_2 \cdot \cos(\theta_2)) + y_{ziel}(d_2 \cdot \sin(\theta_2))$$

Für $\hat{x} = \hat{y} = 0$ liefert die Bibliotheksfunktion 0, was für das gewünschte Verhalten korrekt ist.

5.3.1.7.3 Lösung von ϑ_3

Im Normalfall wird der Fuß auf geradem Untergrund immer parallel zum Torso des Roboters und damit plan auf dem Boden gesetzt. Für einen unebenen Untergrund im Gelände muss, von einer externen Bodenerkennung im Fuß (5.3.4.4.11), der Winkel θ_3 direkt an die Steuerung übermittelt werden. Der Winkel θ_3 wird in Bezug zur Unterschenkelachse gesetzt, wobei 0° einem eingeklappten Fuß entspricht und der Winkel im Uhrzeigersinn gezählt wird (*Abbildung 5-8*). Für den horizontalen Normalfall ergibt sich θ_3 direkt aus der Umkehrung der Schenkelrotationen unter Berücksichtigung der Zählrichtungen.

$$\theta_3 = 180° + (\theta_1 + \theta_2)$$
$$\theta_3 = f_{simplifyAngle}(\theta_3) f\text{ür } \theta_3 \geq 360°$$

$$f_{(simplifyAngle)} = \begin{cases} \theta = \theta + 360° \, f\text{ür } (\theta \; modulo \; 360°) < -180° \\ \theta = \theta - 360° \, f\text{ür } (\theta \; modulo \; 360°) > 180° \end{cases}$$

Durch diese Verfahren können Winkel über oder gleich 360° auftreten. Dafür wird die Funktion „SimplifyAngle“ genutzt.

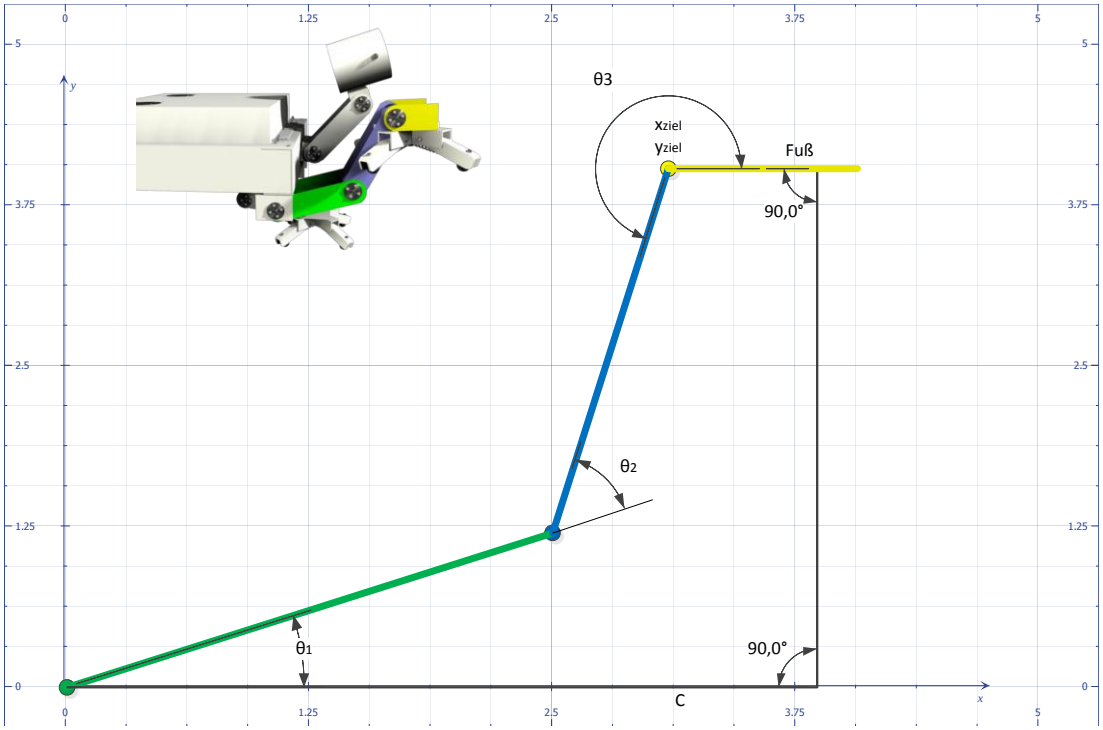

Abbildung 5-8 Berechnung der Fußstellung

In der Detailmodellierung 5.3.4.4.8 wird die Fußhöhe zur y-Zielkoordinate addiert, da die IK die Achsmittelpunkte berechnet.
Damit ist die Herleitung der benötigten IK abgeschlossen und kann für einen Geschwindigkeitstest implementiert werden.

5.3.1.8 Anmerkungen zur IK Prototypenimplementierung

In dem oben hergeleiteten Verfahren wird massiv Gebrauch von Winkelfunktionen gemacht. In der Implementierung könnten diese Winkelfunktionen auf einfache Rechenarten reduziert werden. Aber ein möglicher Geschwindigkeitsgewinn wird sich nur ergeben, wenn genaue Kenntnisse der Prozessorarchitektur angewendet werden. Dies wäre dann eine nur schwer übertragbare Lösung auf eine andere Prozessorplattform.
Auf der MCU und dem Intel Atom wird deshalb die jeweilige „math.lib“ verwendet. Für die erste Prototypenimplementierung wird C# auf einer Desktop CPU benutzt. Durch die Verwendung von C#, kann das Verhalten der IK genauer mit Visual Studio untersucht werden. Dieser Code wird nach der Untersuchung (5.3.1.11) in ANSI C portiert.
Bedingt durch die spätere Portierung wird der Datentyp *double* verwendet, da die „math.lib“ des „GCC-C Compilers für 8-Bit AVR MCUs“ [ATM11] durchgehend im Datentype *double* implementiert ist.

5.3.1.9 Modellierung und Prototypenimplementierung der inversen Kinematik

In der Herleitung (Kapitel 5.3.1.6, Seite 56 ff.) werden die reine IK und mechanische Randbedingungen vermischt. Hier werden einige Regeln des Software Engineerings (SE) verletzt, da mathematische Grundzüge mit äußeren, realen Bedingungen in einer Funktion benutzt werden. Aufgrund der eigenen Forderung nach hoher Performance und Implementierbarkeit für eine MCU werden die SE Regeln leicht „gebeugt“. Dies ist nötig, weil viele temporäre Variablen für die Einhaltung der mechanischen Randbedingungen genutzt werden. Hier ergibt sich eine hohe Abhängigkeit. In der Implementierung wurde eine gute Balance zwischen Wartbarkeit und Kopplung gefunden.

Im folgenden Flow-Chart (*Abbildung 5-9*) sind die Abhängigkeiten gut zu erkennen und begründen auch, dass es kaum eine Möglichkeit gibt den Ablauf zu parallelisieren. Wäre der Ablauf parallelisierbar, würde sich auch ein FPGA anstatt einer MCU anbieten, was aber nicht der Fall ist. Die Funktion „*SimplifyAngle*“ ist nicht in *Abbildung 5-9* dargestellt und wird in *Abbildung 5-17* erläutert.

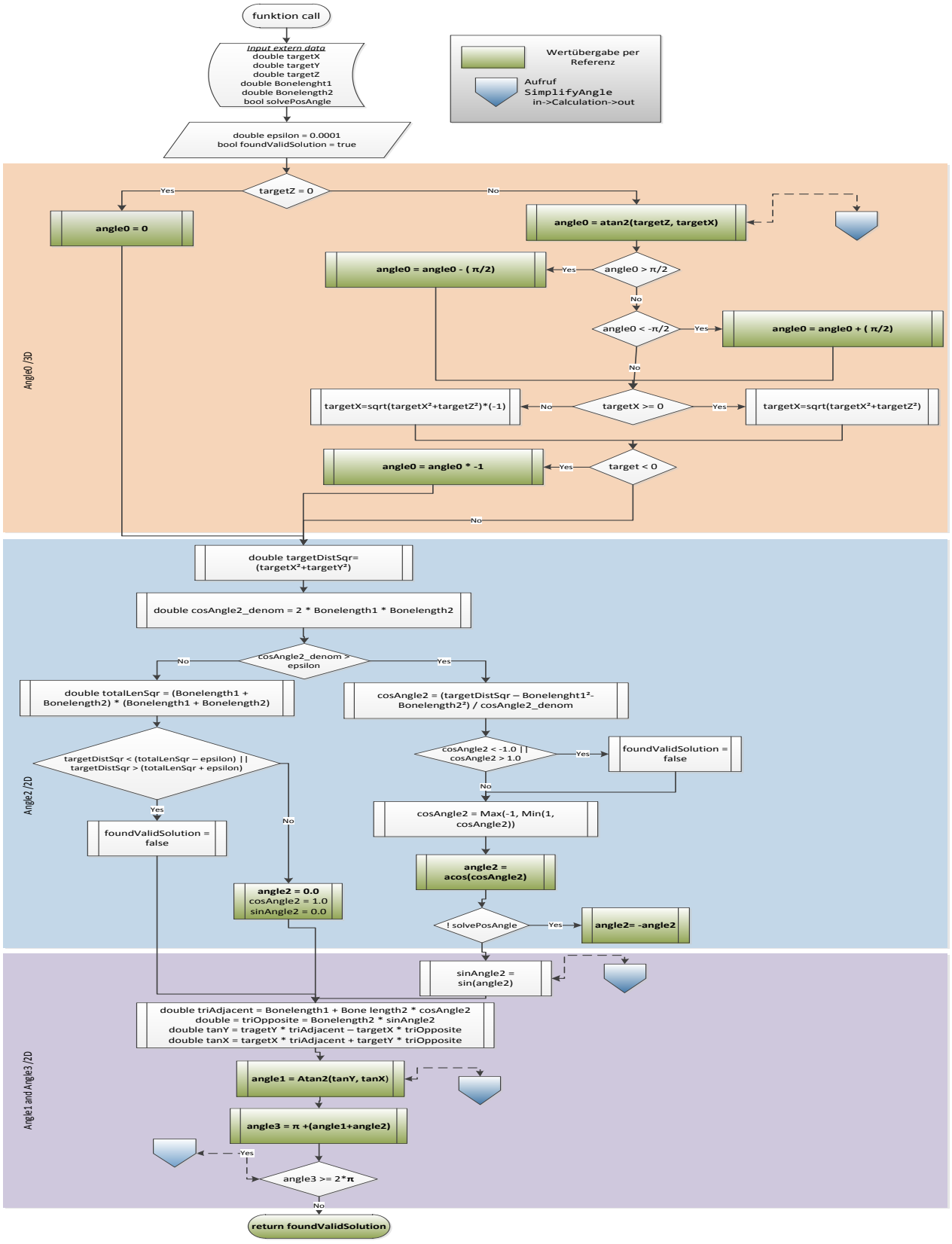

Abbildung 5-9 Flussdiagram der 3D IK

Die Implementierung der 3D-IK wurde in C# ohne erweiterte Sprachelemente umgesetzt, damit eine leichte Portierbarkeit nach ANSI C möglich bleibt.

```
public static bool CalcIK_3D
        (
            out double angle0,  //Angle A
            out double angle1,   // Angle B
            out double angle2,   // Angle C
            out double angle3,  // Angle D
            bool solvePosAngle2, // Solve for positive angle 2 instead of negative
            double length1,      // Length of bone 1. Assumed to be >= zero
            double length2,      // Length of bone 2. Assumed to be >= zero
            double targetX,      // Target x position for the bones to reach
            double targetY,       // Target y position for the bones to reach
            double targetZ        //Target z position for the bones to reach
        )
```

Abbildung 5-10

Der Funktions- / Methodenaufruf (*Abbildung 5-10*) wird mit der Übergabe der Referenzen auf die Werte gestartet. Unter C# wird durch das vorangestellten Schlüsselwort „out“ gekennzeichnet, dass diese Variable ein Rückgabewert ist. In ANSI C entspricht dies einem Pointer.
Die zu errechnenden Winkel (angle0,…,angle3) entsprechen den Achsen A bis D in *Abbildung 5-1*. Das Flag „solvePosAngle2“ wird genutzt um die Rückgabewerte in negativen Winkeln zu erhalten. Diese Möglichkeit wird bei diesem Bein nicht benötigt, kann aber verwendet werden, falls das Bein einen Bewegungswinkel über 180° hat. Dies ist bei AMEE nicht der Fall und dieses Flag wird immer mit „true“ aufgerufen. Die Variable „length1“ und „length2“ geben die Länge der Glieder in Bezug auf die Normierung an. Normierung bedeutet hier die Wahl der Maßeinheit der Koordinaten. Entspricht z.B. X=1 einem cm, wird hier die Länge der Glieder in cm angegeben. „targetX, targetY, targetZ“ sind die Zielkoordinaten relativ zum Fixpunkt, wie in der Herleitung beschrieben. Auch für die Zielkoordinaten gilt die gewählte Normierung.
Der Rückgabewert der Funktion ist das Flag „foundValidSolution“ und wird als true zurückgegeben, wenn der Zielpunkt erreicht wurde.
Um mögliche Rundungsfehler durch die Maschinengenauigkeit (EPS) auszuschließen, wird ein eigenes Epsilon definiert. Da die IK auf einer 8-Bit MCU lauffähig sein soll, wurde sicherheitshalber Epsilon auf 0.0001 begrenzt, was weit entfernt vom realen EPS ist. ($EPS \approx 2{,}2 \cdot 10^{-16}$) Unter Verwendung des „GCC-C Compilers für 8-Bit AVR MCUs“ ist der Datentyp float und double nach IEEE 754 gegeben [ATM11]. Die math.lib des GCC Compilers für AVR nutzt nur den Datentyp double. Das gewählte Epsilon ist eine Sicherheitsmaßnahme gegen Rundungsfehler in der Rechnung, da mehrfach mit Zwischenergebnissen weitergerechnet wird.
Im ersten Teil des Codes (*Abbildung 5-11*) wird der dreidimensionale Teil der IK rechnerisch abgetrennt.

```
if (targetZ != 0) //If rotation
            {
                angle0 = SimplifyAngle(Math.Atan2(targetZ, targetX));

                //Check rotation angle to prevent cutting the
                // cable, only 90 to -90 degree possible
                if (angle0 > ((Math.PI / 2)))
                {
                    angle0 -= (Math.PI / 2);
                }
                else if (angle0 < ((Math.PI / 2) * (-1)))
                {
                    angle0 += (Math.PI / 2);
                }
```

Abbildung 5-11

Der Winkel „angle0“ an Achse A wird hier errechnet. Es erfolgt eine Prüfung der Zielkoordinate Z, ob eine Drehung um Achse A nötig ist. Eine weitere Einschränkung durch die Mechanik ist eine Drehung um mehr als +/- 90°, da sonst die Kabel der unteren Glieder mechanisch abgetrennt würden. Durch die zweite Fallunterscheidung wird der Winkel beschränkt. Berücksichtigt wird dadurch auch eine negative X-Zielkoordinate. Beispielsweise errechnet atan2 für x=-100, y=-100, z= 0 korrekt 180° für Achse A, aber dies ist unerwünscht. Das Bein würde nach hinten über Achse A gedreht. Erwünscht ist aber das Verhalten, dass das Bein nur nach hinten greift und nicht geschwenkt wird.
Im folgenden Schritt wird die neue Zielkoordinate X berechnet (siehe *Abbildung 5-2*) und der Winkel A falls nötig korrigiert.

```
                 //Compute the new targetX, after rotating Angle A
                if (targetX >= 0) //Codesize VS. RAM
                {
                    targetX = Math.Sqrt(targetX * targetX + targetZ * targetZ);
                }
                else
                {
                    targetX = Math.Sqrt(targetX * targetX + targetZ * targetZ) * (-1);
                }
                if (targetX < 0.0) //check upper result again
                {
                    angle0 *= (-1); //*-1 to change the sector
                }
            }
            else
            {
                angle0 = 0.0;
            }
```

Abbildung 5-12

In *Abbildung 5-12* ist die Fallunterscheidung dargestellt, damit der Fuß immer nach vorn zeigt (5.3.1.5). Ab hier handelt es sich nur noch um ein zweidimensionales IK Problem.

Es wird Winkel C berechnet. (Lösung von θ_2 , 5.3.1.7.1)

```
            double targetDistSqr = (targetX * targetX + targetY * targetY);
            // Compute a new value for angle2 along with its cosine
            double sinAngle2;
            double cosAngle2;

            double cosAngle2_denom = 2 * length1 * length2;
            if (cosAngle2_denom > epsilon)
            {
                cosAngle2 = (targetDistSqr - length1 * length1 - length2 * length2)
                            / (cosAngle2_denom);

                // if our result is not in the legal cosine range, we cannot find a
                // legal solution for the target
                if ((cosAngle2 < -1.0) || (cosAngle2 > 1.0))
                    foundValidSolution = false;

                // clamp our value into range so we can calculate the best
                // solution when there are no valid ones
                cosAngle2 = Math.Max(-1, Math.Min(1, cosAngle2));

                // compute a new value for angle2
                angle2 = Math.Acos(cosAngle2);

                // adjust for the desired bend direction
                if (!solvePosAngle2)
                    angle2 = -angle2;

                // compute the sine of our angle
                sinAngle2 = SimplifyAngle(Math.Sin(angle2));
            }
```

Abbildung 5-13

In *Abbildung 5-13* werden einige Hilfsvariablen im Voraus berechnet, da diese Ergebnisse mehrfach verwendet werden. Hier erfolgt die erste Prüfung gegen das definierte Epsilon, um eine sehr kleine Gliederlänge abzufangen und damit einer Division durch Null vorzubeugen bzw. zu verhindern. Im Folgeschritt wird geprüft, ob der erste Wert im Bereich des Cosinus liegt. Ist dies nicht der Fall, kann daraus geschlossen werden, dass es keine Lösung gibt. (siehe *Abbildung 5-6*) Das Return-Flag wird auf „`False`“ gesetzt. Um der Forderung nach einer eindeutigen Lösung nachzukommen, werden die Maxima und Minima der Winkelfunktionen gegen die einer Periode der Winkelfunktionen um Null gebildet. Auch wenn ein Zielpunkt nicht zu erreichen ist, zeigen die Glieder in die richtige Richtung. Der Winkel C ist jetzt berechnet.

```
else
            {
                double totalLenSqr = (length1 + length2) * (length1 + length2);
                if (targetDistSqr < (totalLenSqr - epsilon)
                    || targetDistSqr > (totalLenSqr + epsilon))
                {
                    foundValidSolution = false;
                }
                angle2 = 0.0;
                cosAngle2 = 1.0;
                sinAngle2 = 0.0;
            }
```

Abbildung 5-14

Der Code für den Fall, dass die Gliederlängen kleiner als Epsilon sind[14], ist in *Abbildung 5-14* abgebildet. Es wird geprüft, ob die Glieder im Aktionsradius des Beins liegen. Falls nicht, wird das Return Flag auf „false“ gesetzt.

```
             double triAdjacent = length1 + length2 * cosAngle2;
            double triOpposite = length2 * sinAngle2;

            double tanY = targetY * triAdjacent - targetX * triOpposite;
            double tanX = targetX * triAdjacent + targetY * triOpposite;
            // Note that it is safe to call Atan2(0,0) which will happen if targetX
            // and targetY are zero
            angle1 = SimplifyAngle(Math.Atan2(tanY, tanX));
```

Abbildung 5-15

Ist der Zielpunkt erreichbar, wird aus dem Winkel C der Winkel B berechnet (*Abbildung 5-15*). Als erstes werden die Längen der Dreiecke (Abschnitt 5.3.1.7.2) berechnet. Es wird der Winkel B mit der Arkustangens-Funktion bzw. atan2 berechnet.

```
 //Compute Angle D, it is always parallel to the Body
             angle3 = (Math.PI + (angle1 + angle2));
             if (angle3 >= (2 * Math.PI))
             {
                 angle3 = SimplifyAngle(angle3);
             }

            return foundValidSolution;
```

Abbildung 5-16

Da der Winkel D immer parallel zum Torso des Roboters ist (*Abbildung 5-8*) kann auch dieser, wie in der Herleitung beschrieben, einfach berechnet werden (*Abbildung 5-16*).

[14] Dieser Fall kann im realen Betrieb nicht vorkommen, da die Gliederlänge durch die Mechanik vorgegeben ist. Diese Prüfung bildet nur die theoretische, mathematische Betrachtung in der Implementierung ab.

In der Implementierung wurde häufig die Funktion/Methode „*SimplifyAngle*“ verwendet. Die Implementierung dazu wird in *Abbildung 5-17* gezeigt.

```
private static double SimplifyAngle(double angle)
        {
            angle = angle % (2.0 * Math.PI);
            if (angle < -Math.PI)
                angle += (2.0 * Math.PI);
            else if (angle > Math.PI)
                angle -= (2.0 * Math.PI);
            return angle;
        }
```

Abbildung 5-17

Vereinfacht betrachtet, dreht diese Funktion die berechneten Winkel für das Bein in einen mechanisch erreichbaren Bereich. Es passt die Winkel in den Bereich von [-π,π] ein. Dies ist nötig, da einige Berechnungen vereinfacht wurden. Um den mathematischen Aufwand zu verringern, werden die Winkel nicht in der Rechnung korrigiert, sondern die Funktion „*SimplifyAngle*“ aufgerufen.

5.3.1.10 Benchmark

Der Zweck dieses Benchmarks ist es zu ermitteln, ob der analytische Ansatz schneller ist als ein generischer Ansatz. Es wird ermittelt, wie viele Positionsberechnungen pro Sekunde maximal errechnet werden können. Zu diesem Zweck wird als Vergleich ein *Cyclic Coordinate Descent* (CCD) [Juc10] Verfahren verwendet. Das CCD Verfahren basiert auf dem mathematischen Optimierungsproblem [Boy04]. Es wird versucht in einem Gleichungssystem ein Minimum iterativ zu finden. Dieses generische Verfahren wird bei *komplexeren* Gliederketten, aufgrund seiner hohen Leistung, in 3D Computer Spielen eingesetzt [Juc10], da nach der ersten Integration die Glieder meist schon in die richtige Richtung zeigen. Jede weitere Integration erhöht die Genauigkeit. Weiterhin lässt sich eine qualitative Aussage machen, ob das Verfahren den Zielpunkt erreicht hat bzw. ob er überhaupt zu erreichen ist. Der Aufwand dieses Verfahren ist nicht stabil. In der verwendeten Implementierung [Juc10] werden die Iterationen auf fünf Durchläufe begrenzt.

Beide Verfahren (das oben entwickelte und das CCD) wurden als erstes in C# 4.0 (.net 4.0) umgesetzt und verwenden nur den kritischen Teil einer IK. Es wird in 2D mit zwei Gliedern getestet. Obwohl die Zielplattform eine MCU sein soll, wurde die erste Implementierung in .net erstellt, da Visual Studio 2010 Ultimate® sehr nützliche Analyse Tools zur Verfügung stellt. Auch wurde bei diesem Test auf die Verwendung von Pointern unter C# (als #unsafe) verzichtet. Das CCD Verfahren wurde von Ryan Juckett [Juc10] mit OO-Methoden implementiert. Für den Test wurde der Code nur soweit wie nötig verändert.

Die Gliederlänge ist jeweils auf 100 gesetzt. Jedes Verfahren muss eine geradlinige Bewegung auf y = 100 und von x = -1000 bis x = 1000 in einer Auflösung von 0,1 Schritten durchführen und enthält damit auch Positionen, die nicht erreicht werden können. Diese Bewegung wird 10-mal wiederholt, um die Messgenauigkeit zu erhöhen. Damit muss jedes Verfahren 200.000 Zielpunkte berechnen. Die bereinigte[15] Berechnungszeit wird gemessen und in die Anzahl der Berechnungen pro Sekunde

[15] *Bereinigt bedeutet bei diesem Test, dass die Zeitmessung erst gestartet wird wenn die Methode geladen/aktiv ist und alle Variablen gesetzt sind.*

umgerechnet. Zur Abschätzung des Verhaltens werden zwei unterschiedliche Testplattformen verwendet: Ein Windows 7 Ultimate x64 System mit einem Intel Q9xxx (3,2GHz 12MB Cache mit 4GB 1066MHz RAM) und ein Windows Server 2008 R2 x64 auf einem Intel Atom 330 (1,6 GHz 512kB Cache mit 2GB 533MHz RAM). Durch die Verwendung von Mehrkernprozessoren (Q9xxx = 4 physikalische CPUs, Atom 330 = 2 physikalische + 2 logische CPUs) werden unter .net 4.0 und Windows 7 / Server 2008 R2 andere Prozesse als der Benchmark-Prozess auf die freien Kerne verdrängt, sobald das Betriebssystem einen Volllast-Prozess erkennt. Damit sind Einflüsse von Fremdprozessen minimiert.

5.3.1.10.1 IK-Benchmark Auswertung

Die Messwerte wurden fünfmal ermittelt. Zwischen den Messwerten gab es eine maximale Abweichung von 1% zum vorherigen Lauf und wird als Messfehler durch andere Prozesse und die „Garbage Collection“ von .net vernachlässigt.

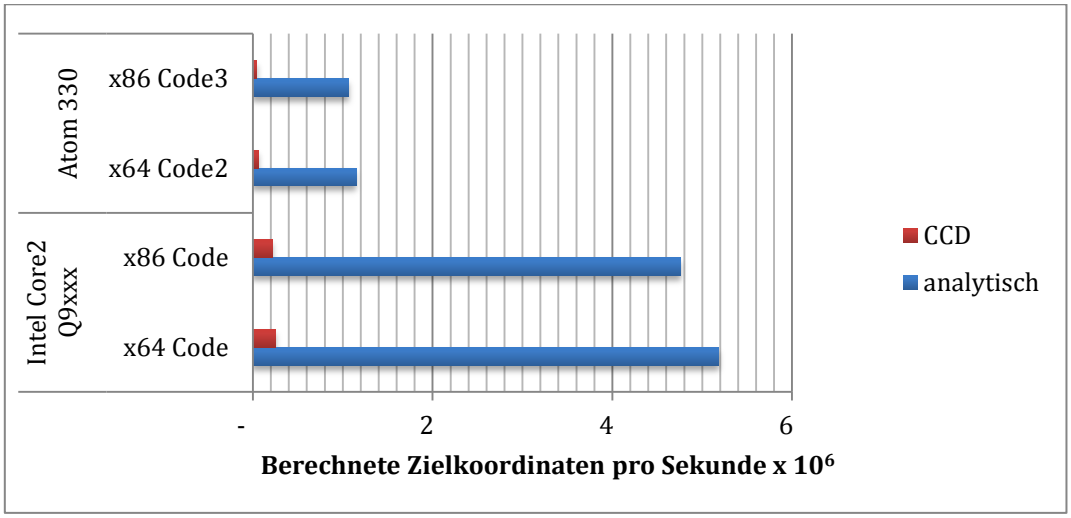

Abbildung 5-18 Kalkulationen pro Sekunde in Millionen

	Intel Core2 Q9xxx		Intel Atom 330	
	x64 Code	**x86 Code**	**x64 Code**	**x86 Code**
Analytisch calc./sec	5.181.050	4.761.632	1.148.039	1.062.638
CCD calc./sec	249.767	217.355	59.306	45.384
Analytisch/CCD Faktor	20,74353297	21,9071657	19,35788959	23,41437511
x64 / x86 Faktor		1,09		1,08

Tabelle 5-1 IK-Benchmark Messergebnisse

In *Abbildung 5-18* und in *Tabelle 5-1* ist gut zu erkennen, dass das analytische Verfahren eine Beschleunigung der Berechnung ungefähr um den Faktor 20 auf beiden Testplattformen gegenüber dem generischen Verfahren ermöglicht. Der Faktor bleibt auch auf unterschiedlichen Hardware-plattformen fast identisch. Mit diesem Benchmark kann auch das Potenzial der Intel Atom Plattform eingeschätzt werden. Der Leistungsunterschied zwischen einem Intel Desktop-Quad-Core-Prozessor und dem schwächerem Intel Atom liegt ca. bei dem Faktor 4,5. Dabei muss beachtet werden, dass

dieser Benchmark beim analytischen Verfahren die CPU interne FPU (Floating Point Unit) stark belastet. Eine allgemeine Aussage wird von diesem Benchmark nicht getroffen. Dies ist ein Spezialfall, aber für das angestrebte Laufsystem aussagekräftig.
Der Code wurde jeweils in einer x86 (32Bit) Version und in einer x64 (64Bit) Version kompiliert. Auf beiden Plattformen werden fast 10% mehr Leistung mit der x64 Version erreicht. Durch die Mehrleistung ist ein Einsatz eines x64 Betriebssystem für den Roboter gerechtfertigt. Der Irrglaube, das eine x64 Software nur bei hoher Speicherauslastung schneller arbeitet, ist durch diesen einfachen Test entkräftet.
Anmerkung: Es ergibt sich eine Beschleunigung auf der Intel Atom Plattform um 30% zwischen x86 und x64 Version beim numerischen CCD Verfahren für die x64 Version. Leider konnte auch auf Nachfrage bei Intel und Microsoft dieser Effekt nicht geklärt werden. Dies ist für die weitere Betrachtung auch nicht wichtig, da das CCD Verfahren eindeutig langsamer ist und nicht verwendet wird.
Das Ergebnis der Benchmarks ist eindeutig und der Aufwand einer analytischen Herleitung der IK ist damit gerechtfertigt. Es muss aber beachtet werden, dass das obige Verfahren nur einen lohnenden Mehrwert bringt, wenn sich der Aufbau der Mechanik extrem vereinfachen lässt. Sobald die Gliederkette nicht auf ein „Two-Bone“ Problem reduziert werden kann, wird der Aufwand der Herleitung zu groß. In so einem Fall sollte auf generische Verfahren zurückgegriffen werden.

5.3.1.10.2 Codeanalyse des „analytischen Verfahrens“

Mit der IDE Visual Studio 2010 Ultimate® (VS2010) von Microsoft wurde mit dem Performance Wizard ein CPU Sampling / Profiling erstellt, um noch vorhandene Geschwindigkeitsprobleme zu finden. Beim CPU Sampling wird eine fast atomare (zeilenweise) Betrachtung der Implementierung zur Laufzeit erstellt. Der laufende Code wird von VS2010 isoliert und die Zeit zwischen den Befehlen aufgezeichnet und ausgewertet. Durch die Isolation der zu testenden Anwendung werden verfälschende Faktoren wie andere Prozesse und Services nicht mit in die Analyse einbezogen.

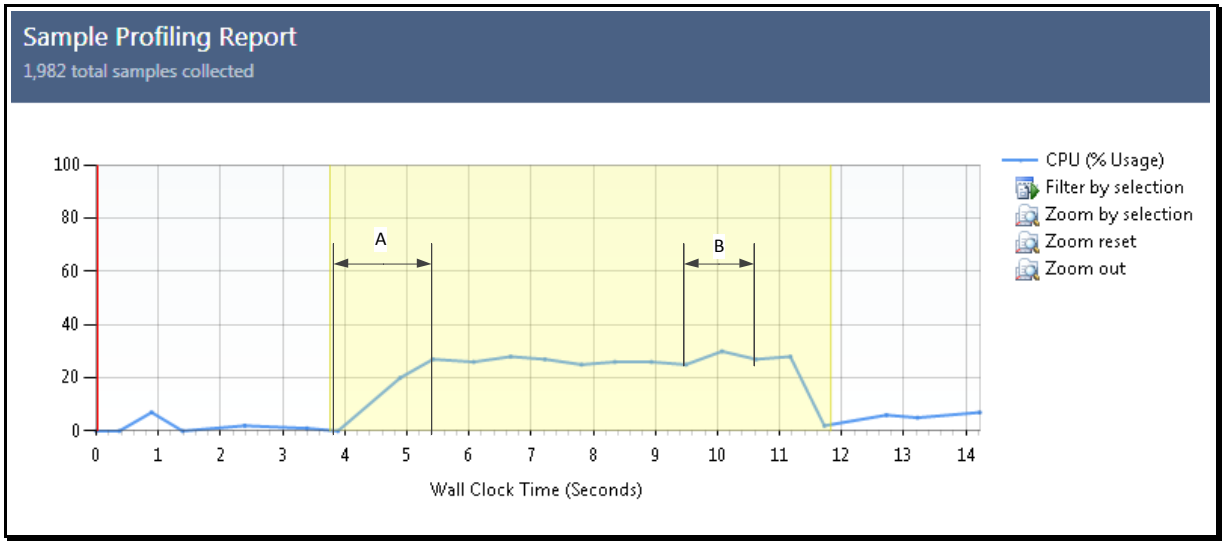

Abbildung 5-19 CPU-Last der IK (Zeitachse gestreckt)

In *Abbildung 5-19* wird die isolierte Implementierung der IK als CPU-Last in Prozent dargestellt. Die IK ist eine Single-Thread-Anwendung und hat mit 25% auf einem Quad-Core-Prozessor Volllast erreicht. Durch eine zukünftige Kollisionskontrolle der Beine im fertigem Laufsystem muss die IK viermal zentral berechnet werden (vier Beine = vier parallele IKs). Dieses Verhalten ist durchaus erwünscht. Auf der Zielplattform (Intel Atom) wird die Dual-Core-CPU durch Hyperthreading auf vier logische Kerne abgebildet. Als Hauptfunktion in der MCU spielt dieser Faktor keine Rolle, da der AVR eine Single-Core MCU ist und jedes Bein eine IK Implementierung enthält.
Im Abschnitt A (*Abbildung 5-19*) ist sehr gut das Verdrängen anderer Prozesse von diesem Kern zu erkennen. Der Scheduler des Betriebssystems (OS) erkennt durch Hilfsfunktionen, dass in diesem Fall die IK auf Volllast gehen möchte und verschiebt andere Prozesse auf freie Kerne. Dieses Verhalten ist ab Windows Version 6.0 normal. In Abschnitt B erzeugt die Arbeit der Garbage Collection einen Ausschlag. Die Last geht auf ca. 30%, da alle Services und Verwaltungsfunktionen von .net parallelisiert sind und von einem anderen freien Kern bearbeitet werden.
Die IK-Funktion(*Abbildung 5-20*) aus der Implementierung nutzt das System zu 94,8% aus.

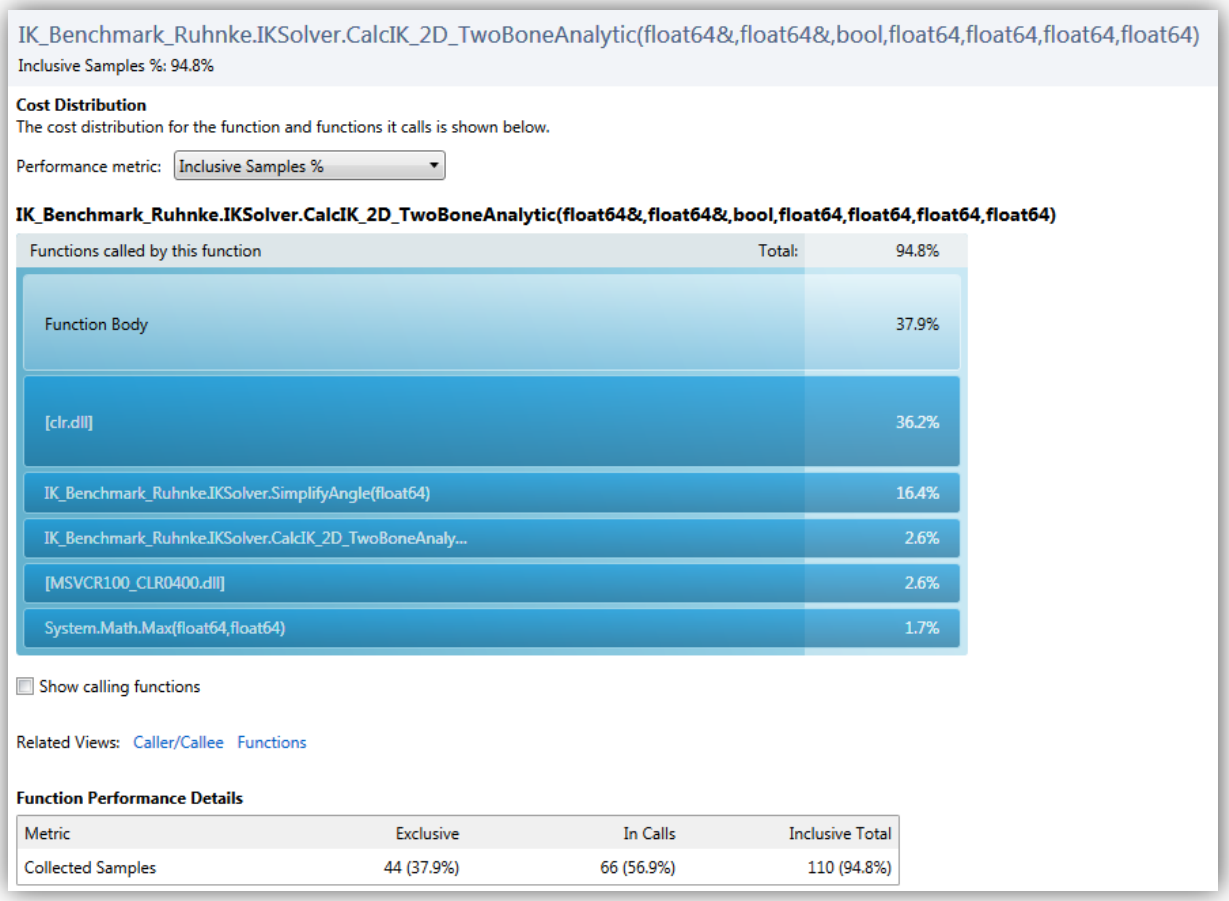

Abbildung 5-20 Analyse der IK Funktion

In *Abbildung 5-20* wird dargestellt, dass die isolierte Betrachtung der IK (...CalcIK_2D_TowBoneAnalytic) nur 2,6% Rechenleistung verbraucht. Die Methode „SimplifyAngle“ wird in der Lastrechnung (*Abbildung 5-20*) verzerrt dargestellt. Der Aufruf ist mit einer Rechnung verbunden (*Abbildung 5-15*). Das Analyse-Tool zerlegt diesen Aufruf nicht, sondern betrachtet den Aufruf als Ganzes.

In der tieferen Code-Analyse ergibt sich ein sehr ausgewogenes Bild der IK Funktion. Die Winkelfunktionen beanspruchen im Schnitt jeweils 1-2% (IK-Funktion) der Rechenleistung und sind homogen über den Code verteilet. Wie erwartet verbraucht eine verschachtelte atan2 Funktion ca. 41% der Rechenleistung in der aufgerufenen Funktion. Dort wird in einer Zeile der atan2 (Lösung von θ_1, 5.3.1.7.2) und die Normalisierung durchgeführt.

5.3.1.11 Test-Implementierung und Benchmarks auf einer MCU

Als Ziel soll die oben entwickelte IK in einer MCU lauffähig sein. Als MCU wurde ein AVR ATMEGA 128® der Firma ATMEL gewählt. Der C# Code konnte mit geringen Änderungen in ANSI C für den GCC Compiler portiert werden. Durch die Verwendung von Pointern mussten einige Funktionen leicht umgestellt werden.

Da auch auf der MCU mit double Werten (definiert als Float mit 64Bit [ATM11]) gerechnet wird, stellte sich die Frage nach der Laufzeit auf einer 8Bit MCU. Zwar handelt es sich beim ATMEGA 128® um einen RISC Prozessor, aber die Busbreite an der ALU ist auf 2 mal 8 Bit beschränkt. Für den Test wurde das AVR Studio 5 Beta2 mit dem AVR 8bit GCC Toolchain Version 3.2.1.292 verwendet. Die IK wurde auch in BASCOM Basic implementiert. Hier war die Migration von C# auf Basic aufwendiger, da der Bascom Compiler nur zwei Operanden pro Schritt / Befehlszeile erlaubt. Dadurch muss sehr viel mit temporären Variablen gearbeitet werden. *Erstaunlicherweise ist der Basic Code auf der MCU nicht so langsam, wie erwartet wurde.*

Compiler	**AVR8bit GCC**	**BASCOM AVR**
Eine Kalkulation	0,85ms	7,04ms
Größe im Flash RAM	18,4 kByte	18,5 kByte

Tabelle 5-2 3D-IK Laufzeit auf einem ATMEGA128 mit 18.432MHz

Für die Messung wurde wieder eine Bewegung auf der X-Achse von -100 bis 100 durchgeführt. Dies wurde 8-mal ausgeführt und die Zeit gemessen. Die einzelnen kompilierten Codes wurden in den ATMEGA128 geladen, der mit 18,432MHz getaktet wurde. Für diesen Test wurden am Entwicklungskit STK 500 mit STK 501 nach jedem durchlauf eine LED umgeschaltet. Der gesamte Ablauf wurde 5-mal durchgeführt und es gab keine messbaren Abweichungen zwischen den Durchläufen. Da die obige IK einen stabilen Aufwand hat, können diese Messwerte als maximal betrachtet werden. Damit kann die Entwicklung einer analytisch hergeleiteten dreidimensionalen inversen Kinematik als Erfolg bezeichnet werden. Durch die relativ geringe Rechenzeit von < 1ms kann die IK mit dem GCC Compiler in einer 8Bit MCU verwendet werden.

Die inverse Kinematik wurde in folgenden Sprachen als Funktion bzw. Methode implementiert: .net C# 4, .net Visual C / C++ 2011, ANSI C für AVR 8bit GCC Compiler und in Basic für den BASCOM-AVR Compiler[16].

[16] Bascom ist für das „fast prototyping“ gut geeignet, da fast ohne Kenntnisse der AVR Hardware z.B. eine exakte PWM Modulation mit einer Anweisung realisiert werden kann.

5.3.2 Ziele des MCU Kontrollers

Mit den Benchmarks aus dem Kapitel 5.3.1.11, wurde gezeigt dass eine inverse Kinematik verwendbare Laufzeiten in einer MCU liefern kann. Damit können die Ziele bzw. Forderungen für den Funktionsumfang des MCU Kontrollers aufgestellt werden.
Der MCU Kontroller / jedes Bein soll:

a. Bei einfachen und rudimentären Bewegungen autonom agieren.
b. Es soll Einzelbewegungen mit anderen Beinen synchronisieren können.
c. Es soll (in bestimmten Grenzen) fehlertolerant und selbstkorrigierend sein.
d. Es soll die Mechanik selbst schützen.
e. Es soll eine bestimmte Anzahl an Versuchen unternehmen, um ein Problem zu lösen.
f. Es soll über Ethernet gesteuert werden können.
g. Es soll (soweit möglich) schonend mit der Mechanik agieren.
h. Es soll auch mit einer verschlissenen Mechanik (Spiel) agieren können.
i. Es soll dem Hauptkontroller nur nötige Informationen übermitteln.
j. Es soll über semantische, einfache Befehle gesteuert werden.
k. Bestimmte Bewegungsabläufe müssen synchronisierbar sein.
l. Es soll maximale / garantierte Reaktionszeiten einhalten.
m. Es soll sicher und durchsetzungsfähig im Gelände sein.
n. Der Code soll möglichst verständlich und wartbar sein.
o. Der Code soll für andere MCUs portierbar sein.

5.3.2.1 Werkzeuge / Tools

Es wurde die Sprache C und der „GCC Compiler für 8Bit AVRs“ für die MCU Software gewählt. Als IDE wurde das AVR Studio 5® Beta 2 in der Version 5.0.1119 genutzt. Um die Software und Hardware zu testen wurde das STK 500 mit dem STK 501 der Firma ATMEL an das Bein angeschlossen. Für die Ethernet-Tests wurde das Easy-TCP/IP Kit der Firma MCS an das STK 500 angeschlossen.

5.3.2.2 Terminologie und Coding Convention

In dieser Arbeit wird mit einer Terminologie gearbeitet, die teilweise aus dem Maschinenbau stammt und sich auch in der Software ausdrücken muss, um bestimmte Zusammenhänge zu erklären. Es werden einige nicht allgemein bekannte Begriffe in diesen Kapiteln und im Glossar erklären. Auch wurde dem Abstraktionsgrad eine bestimmte Grenze gesetzt. Es soll der physikalische Hintergrund erkennbar bleiben.
Es wird in diesem Kapitel der Unterschied zwischen Motor, Antrieb und Aktor gemacht. Der *Motor* bezeichnet die elektromechanische Lösung, Energie in Bewegung umzusetzen, als einzelne Einheit. Der *Antrieb* ist die Kombination von Motor und Getriebe. Der *Aktor* ist das gesamte Glied inklusive dem Motor, Getriebe und Achse.
Die Datenstrukturen *sind in Anlehnung an die Digitaltechnik* modelliert. Es wird Register-Sharing eingesetzt und Steuerregister verwendet. Um dies zu verdeutlichen werden Variablen, die wie Steuerregister benutzt werden, als *Steuerbits* bezeichnet.
Der ANSI C Standard wurde teilweise nicht eingehalten, um den semantischen Hintergrund zu verdeutlichen.

5.3.3 Modellierung der MCU Software

Das Endergebnis dieser Entwicklung soll es sein, eine API zu erstellen, die es ermöglicht den kompletten Roboter von Punkt A nach Punkt B laufen zu lassen. Jedes Bein soll über Ethernet Befehle entgegennehmen. Diese werden von der MCU in einzelne Bewegungen der Glieder umgesetzt. Es ist die Aufgabe der MCU, die Glieder dabei zu koordinieren und zu überwachen. Beispielsweise muss der vom Hauptkontroller kommende Befehl „`DoStep (Startpunkt, Endpunkt)`“ schlussendlich ein Abfolge von wechselnden Spannungen und Richtungen der Antriebe in dynamischer Abhängigkeit von Sensordaten werden.

Da ein Schritt des Beins eine Abfolge von sich wiederholenden Sequenzen ist, wird dies mit Automaten aus der technischen Prozesslenkung realisiert. Weil einige Automaten auf andere warten müssen, bieten sich hierfür hierarchische Automaten (z.B. Harel Automaten) an. In Anlehnung an die technische Prozesslenkung, werden diese Automaten auch als *Kontroller* bezeichnet.

In einer Bottom-Up Betrachtung müssen die MCU-Register-Befehle abstrahiert werden, um effizient damit arbeiten zu können. Für diese Vereinfachung bzw. Kapselung werden höhere Befehle wie „`Motor_A(_right, speed)`“ implementiert. Diese Befehle müssen dann sukzessive zu noch abstrakteren Funktionen überführt werden, damit daraus z.B. der Befehl „`Move2Point(X,Y,Z)`“ werden kann. Diese Abstraktionsebene ermöglicht es, das Bein an eine bestimmte Position im Raum zu bewegen. Aber um die eingangs erwähnte Zielsetzung zu realisieren, werden wiederum diese Befehle in noch höherer Logik abstrahiert, um komplette Bewegungsabläufe zu modellieren. Damit stellt der Befehl „`Dostep`“ eine Abfolge von „`Move2Point`“ Befehlen mit unterschiedlichen Zielkoordinaten dar.

Durch die Forderung „o. Der Code soll für andere MCUs portierbar sein. 5.3.2“, wird ein Schichten-Modell, ähnlich dem OSI Modell (ISO 7498-2), verwendet. Die einzelnen Kontroller (Automaten) werden einzelnen Schichten, je nach Abstraktionsgrad, zugeordnet. In einer Bottom-Up Betrachtung (*Abbildung 5-21*) werden die Befehle, nach oben, von Layer zu Layer, immer abstrakter und beschreiben eine Tätigkeit.

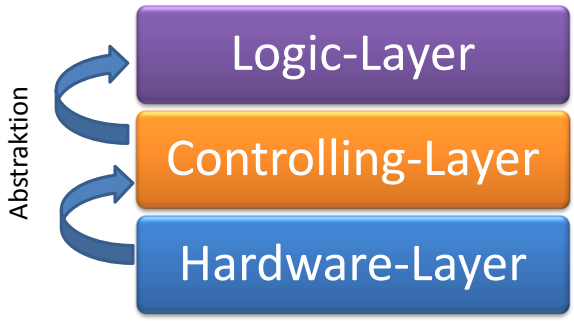

Abbildung 5-21 Schichten Konzept

Der Hardware-Layer ist mit der Hardware der MCU stark gekoppelt. Es soll erreicht werden, dass nur dieser Layer ausgetauscht werden muss, wenn ein anderer MCU Typ verwendet wird. Daraus folgt auch, wie beim OSI Modell, das Funktionen nur auf ihre jeweiligen nachbarschichten zugreifen dürfen. Dafür ist eine möglichst lose Koppelung zwischen den Schichten nötig.

Als weiteres Konzept soll der komplette Bein-Kontroller Maximalzeiten garantieren. Die erste

Schlussfolgerung, wäre komplett auf Interrupts zu verzichten. Dabei werden zwei Arten von Interrupts unterschieden. **Erstens**: Der ereignisgetriebene Interrupt, der den normalen Programmablauf zu einem *nicht vorhersagbaren Zeitpunkt* unterbricht. In diesem System ist nicht vorhersagbar, wie oft dieser Interrupt ausgelöst wird. Eine Interrupt-Flut würde die RTX verhindern. Es wäre zwar möglich, die auslösenden Ereignisse zu begrenzen, aber ein mechanischer Kabelbruch könnte immer noch eine Interrupt-Flut auslösen. Beispielsweise wäre beim Erreichen eines bestimmten Winkels einen Interrupt auszulösen, ein Ansatz und würde eine sehr hohe Präzision der Mechanik ermöglichen. Nur würde auch jedes kleine Wackeln um den Zielwinkel der Mechanik immer wieder diesen Interrupt auslösen. Aus diesen Gründen scheiden ereignisgetriebene Interrupts für dieses Konzept aus. **Zweitens**: Der *zeitlich periodisch* auftretende Interrupt. Dieser Interrupt ist vorhersagbar und die Bearbeitungszeit der Interrupt-Routine kann in der Berechnung der maximalen Reaktionszeit des Systems hinzu addiert werden. Hat die Interrupt-Quelle eine absolute Frequenz des Auslösens, kann keine Interrupt-Flut auftreten. Es kann immer noch nicht genau vorhergesagt werden, wo dieser Interrupt den Programmablauf unterbricht, aber es kann eine maximale Laufzeit eines Super-Loop Durchlaufs angegeben werden. Daraus folgernd kann der periodisch / zyklisch auftretende Interrupt genutzt werden[17].

Anmerkung:

Theoretisch wäre ein komplett Interrupt getriebenes System denkbar. Es müsste zu jedem Ereignis ein geeigneter Hardware-Interrupt gefunden werden. Es wäre aber nicht möglich, garantierte Reaktionszeiten des Systems anzugeben. Es wäre nur möglich, eine mittlere Reaktionszeit anzugeben. Zudem wäre das System unzuverlässig bei mechanischen Defekten wie oben beschrieben. Es würde auch sehr schwer werden, mechanische Ungenauigkeiten (z.B. Spiel in den Antrieben) zu kompensieren, da die Prioritätenliste der AVR Serie absolut und fest durch die Hardware vorgegeben ist. Bei geänderten Einflüssen von außen, kann nur schwer eine Interrupt-Routine zur Laufzeit gewechselt werden. Erfasste Werte und Reaktionen darauf beeinflussen sich gegenseitig. Durch rekursiv veränderte Werte, kann es zu Anomalien in den Automaten kommen. Die Grundlage für den Eintritt in einen Zustand, könnte sich zur Laufzeit des Automaten ändern. Vorteile wären im Normalbetrieb eine schnelle Reaktion, da keine Register und Funktionen auf den negativen Fall geprüft werden. Die Modellierung der höheren Logik und Kapselung nach physikalischen Bedingungen wäre erheblich schwerer. Aus diesen Gründen wurde dieses Konzept verworfen.

Die Anbindung an ein Bussystem, die physikalische Schnittstelle zum externen Hauptkontroller (Intel Atom), setzt einige Implementierungsdetails voraus. Konzeptionell widersprechen sich die Anforderungen von RTX und Ethernet. Aus diesem Grund wird die Ethernet Schnittstelle in gesonderten Abschnitten (5.3.3.2, 5.3.4.4.12) erörtert.

5.3.3.1 Laufzeitsystem

In diesem Abschnitt wird erläutert, wie die obigen Konzepte auf einer MCU ohne RTOS umgesetzt wurden. Ein RTOS für AVR MCUs wäre denkbar, bietet aber kaum Vorteile. Bei der Verwendung eines RTOS würde auch der ungenutzte Overhead Ressourcen der MCU belegen. Besonders betroffen wäre der verfügbare RAM in der MCU. Der RAM ist aber in dieser Entwicklung besonders kritisch. In Abschnitt 5.3.4 wird dies genauer erläutert.

[17] Im Grundkonzept zu diesem Laufsystem war dies nicht vorgesehen. Durch eine relativ lange Rechenzeit der inversen Kinematik, kam es zu Problemen bei der Glättung der Messwerte.

Um ein funktionsfähiges Multitasking zu erreichen, darf nach Eintritt in jeden Kontroller nur ein aktueller Zustand bearbeitet werden. Auch bei einem Wechsel des Zustands muss der Kontroller verlassen werden. Das muss genauso für hierarchische untergeordnete Automaten gelten. Nach dem strikten Muster: prüfen → ggf. bearbeiten → verlassen.
Durch dieses Konzept sind die Automaten immer synchronisiert. Jeder Automat kann pro Super-Loop nur einen Zustand ändern. Damit können Automaten realisiert werden, die aufeinander reagieren aber getrennt implementiert wurden. Die beteiligten Kontroller können den Zustand eines anderen Kontrollers nicht verpassen. Weiterhin kann damit die Tiefe der hierarchischen Automaten reduziert werden, was ein zu starkes Anwachsen des Stacks im RAM entgegenwirkt. Wie in *Abbildung 5-22* kann man sich die eigenständigen aber gekoppelten Automaten bildlich als Zahnräder vorstellen.

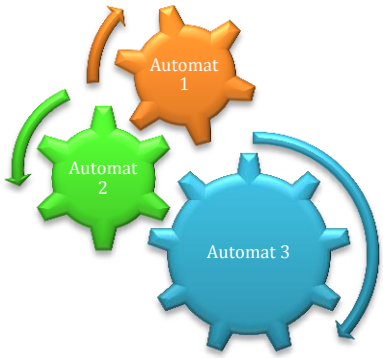

Abbildung 5-22 Synchronisation der Kontroller/Automaten

Jeder Zahn stellt einen Zustand des Automaten dar. Pro Super-Loop Durchlauf wird jeder Automat vom Antriebszahnrad weitergetrieben. Die Analogie *verzahnt* ist bildlicher als synchronisiert und kennzeichnet die Beziehung der Automaten zueinander. Zweck dieser Betrachtung ist das Abstrahieren von Automaten / Kontrollern und der physikalische Hintergrund kann getrennt in einzelne Kontroller abgebildet werden. Beispielsweise wurde die Drehmomentkontrolle eigenständig implementiert, wird aber durch den Zustand des Motorkontrollers getrieben und kann in diesen direkt eingreifen. Natürlich ist nicht mit jedem Super-Loop Durchlauf ein Zustandswechsel zwingend. Hier weicht die Analogie vom realen Verhalten ab.

5.3.3.2 MCU Interface, Ethernet Anbindung

Das Bein-System soll mit dem externen Hauptkontroller kommunizieren. Die Vorteile, Ethernet als Bussystem zu verwenden, stellen hohe Anforderungen an das aufgestellte Software Konzept. Alle untersuchten Hardware TCP/IP Stacks benötigen einen Interrupt. Jede untersuchte API setzt zwingend eine Bearbeitung mit einem Interrupt voraus. Damit treten asynchrone, ereignisgetriebene Interrupts auf (5.3.3). Für dieses System könnten keine garantierte Reaktionszeit angegeben werden. Das System kann nicht vor „Babbling Idiots“ [Jew07] geschützt werden. Aus diesem Grund wird für die Bearbeitung der Ethernet Schnittstelle eine zweite MCU eingesetzt. Der Einsatz von externer Hardware für einen sog. „Bus-Guardian“ [Jew07], wird auch im FlexRay [Jew07] Bussystemen empfohlen. Diese extra MCU mit Ethernet Schnittstelle (Ethernet-MCU, 4.2.5.2), arbeitet als Bus-Filter. Der Inhalt von ein- und ausgehender Nachricht wird geprüft. Nachrichteninhalte, die einer Whitelist zugeordnet werden können, werden übermittelt. Andere werden verworfen. Die Last des

Bussystems bleibt in der Ethernet-MCU.
Die Verbindung der Bein-MCU und der Ethernet-MCU erfolgt über die Hardware USART Schnittstelle der AVR Kontroller. Die Last der USART Schnittstelle wird in den verwendeten MCU Typen komplett in Hardware gekapselt [ATM101]. Die Schnittstelle hat für Ein- und Ausgang jeweils einen Hardware Puffer von einem Byte. Damit kann eine ein Byte lange Nachricht asynchron empfangen und gesendet werden, ohne Last auf dem MCU-Kern (asynchron zum Kern). Um dies zu ermöglichen wird ein eigenes Übertragungsprotokoll implementiert. Der Datenfluss wird durch RTS und CTS gesteuert. Nach jedem Datenpakt werden die Pins vom Empfänger wechselseitig beschaltet. Die Implementierung der „avr-library“ [ATM11] ist dafür nicht geeignet, da sie für den Transport größerer Nachrichten optimiert wurde.
Andere Schnittstellen für die MCU-zu-MCU Übertragung bieten nur eine Hardware*unterstützung* an. Beispielsweise besitzt die I^2C (TWI) Schnittstelle keinen Hardware Puffer [ATM101]. Der Datentransport ist die Last des MCU-Kerns.
Durch den Hintergrund der Ausfallsicherheit wird die Verwendung von zusätzlicher Hardware nötig. Im störungsfreien Betrieb, des Bein-Kontrollers, wäre die Auslagerung auf eine weitere MCU nicht zu vertreten.
Die Ethernet-MCU wird im Abschnitt 5.3.4.4.13 erörtert. Die USART Schnittstelle im Bein Kontroller wird im Abschnitt 5.3.4.4.12 erläutert.

5.3.3.3 Identifizierung der Kontroller

Das System wird nun unter den oben aufgestellten Forderungen und Konzepten auf einzelne Kontroller hin untersucht. Die meisten Kontroller können einem Layer zugeordnet werden. Sie werden in der Detailmodellierung genauer betrachtet.

Hardware-Layer

Im Hardware-Layer wird kein Kontroller benötigt. Hier sind kleine asynchrone Automaten ausreichend. Beispielsweise um ungeregelte Bewegungen der Aktoren durchzuführen.

Control-Layer

Hier gibt es drei Kontroller. Der erste Kontroller (*Rotation-Ctrl**) soll die einzelnen Glieder (Aktoren) in einen von der höheren Logik bestimmten Winkel fahren. Er soll den Ist-Zustand mit dem Soll-Zustand vergleich und entsprechend reagieren. Dieser Kontroller soll auch die meisten der in Kapitel 5.3.1.11 aufgestellten Forderungen erfüllen.
Der zweite Kontroller überwacht das Drehmoment (*Torque-Ctrl**) und detektiert damit Kollisionen der Aktoren mit realen Objekten. Er soll entsprechend in den ersten Kontroller eingreifen können. Der dritte Kontroller (*Speed-Ctrl**) überwacht die Geschwindigkeit der einzelnen Aktoren und greift ein, wenn sich der Aktor vor Erreichen des Zielwinkels nicht mehr bewegt und *keine* Kollision vorliegt. Dies ist der Fall, wenn die gewählte Geschwindigkeit ein zu geringes Drehmoment erzeugt um den Aktor gegen einen physikalischen Widerstand (z.B. Schwerkraft) zu bewegen. Auch dieser Kontroller ist mit dem ersten und dem zweiten Kontroller schwach gekoppelt.
Die obigen Kontroller bilden eine hierarchische Abhängigkeit, können aber aufgrund der Konzepte aus Kapitel 5.3.3.1 separat betrachtet werden.

Logic-Layer

Dieser Layer wird von einem Kontroller (*Motion-Ctrl**) gesteuert, der den komplexen Bewegungsablauf in einzelne Punkt-zu-Punkt Bewegungen transformiert. Er wird von spezialisierten Kontrollern

unterstützt, die der "*Motion Ctrl.**" hierarchisch untergeordnet sind. Durch die Abstraktion in diesem Layer fließen hier keine physikalischen Größen mehr ein. Dies soll der Control-Layer kapseln. Weiterhin bietet der Logic-Layer eine Schnittstelle zur Synchronisation der Beine untereinander an.

Gekapselt von diesen Kontrollern gibt es den Kommunikation Kontroller (*Com-Ctrl**). Dieser Automat überwacht die Nachrichten an der Schnittstelle nach außen und bereitet die Nachrichten für den „*Motion-Ctrl**“ Kontroller auf. Auch abgehende Nachrichten werden hier behandelt. Die Hauptkomponenten mit ihrer Platzierung im System werden in *Abbildung 5-23* dargestellt.

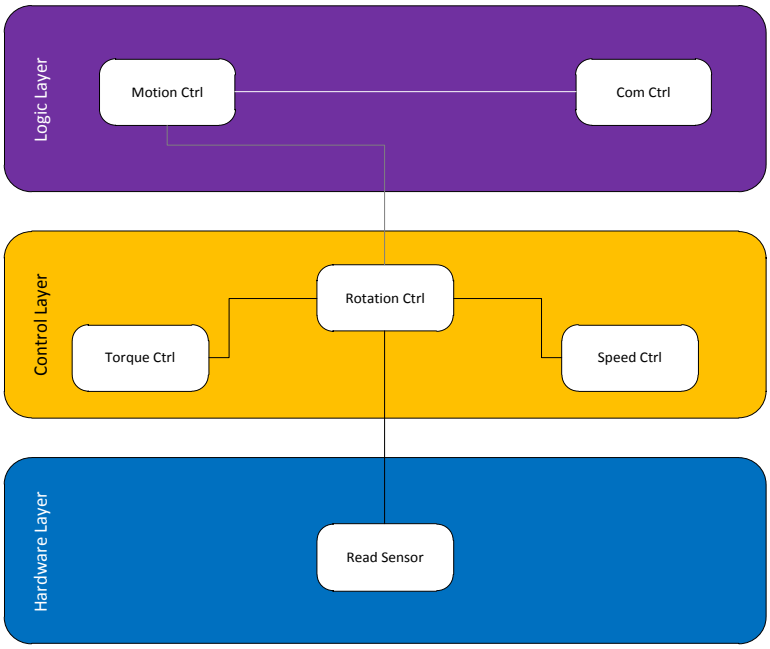

Abbildung 5-23 vereinfachte Layer Darstellung

**Das „Handbook of Robotics* [Sic08]“ schlägt teilweise andere Bezeichnungen vor. Die dortigen Kontroller basieren auf einem anderen Prinzip. Durch eine andere Namensgebung wird hier deutlich gemacht, dass es sich um abgewandelte oder andere Konzepte handelt. Besonders die Bezeichnungen „Motion Control“ und „Motion Planning“ verändern in der Literatur, je nach Kontext die Bedeutung. Hier ist, in Bezug auf das Kapitel, die Bewegung der Aktoren gemeint.*

5.3.3.4 Sicherheitskonzept

Das Laufsystem soll in einem autonomen Roboter eingesetzt werden, der in für Menschen gefährlichen Umgebungen agiert. Im Fehlerfall kann das System in der Regel nicht manuell zurückgesetzt werden. Es müssen Sicherheitssysteme konzipiert werden. Sicherheit bedeutet hier nicht die virtuelle Abwehr eines virtuellen Angreifers, sondern die Reaktion auf Defekte und unvorhersehbare Ereignisse.

5.3.3.4.1 Sicherheitsfunktionen

Die erste Sicherheitsfunktion ist bereits in der Hardware der MCU integriert. Der *Hardware Watchdog* (HW Watchdog) wird nur aus der Software initialisiert. Er arbeitet unabhängig vom Kern der MCU. Wird in einem bestimmten Zeitraum der interne Timer nicht zurückgestellt, erfolgt der Neustart der MCU. In diesem System ist der HW Watchdog für unvorhersehbare Fälle vorgesehen.

Dieses Konzept löst extreme Probleme durch Programmfehler oder Speicherüberläufe und wird im Normalbetrieb nie ausgelöst.

Software Watchdog und Fehlerbehandlung:
Ein weiteres Sicherheitskonzept ist die Überwachung der Kontroller durch eine Software. Dieser *Software Watchdog* basiert auf den Konzepten des HW Watchdogs und dem Observer-Pattern. Der Software Watchdog (SW Watchdog) erhält einen eigenen Hardware Timer (HW Timer). Wird eine Bewegung der Aktoren eingeleitet, wird dieser Timer mit dem SW Watchdog gestartet. Jeder Kontroller (Automat) ist in eine Liste im SW Watchdog eingetragen. Siehe *Abbildung 5-24*. Dieser Eintrag beinhaltet für jeden Automaten eine maximale Bearbeitungszeit. Der SW Watchdog überprüft jeden Automaten, ob ein Zustandswechsel stattgefunden hat. Ist dies der Fall, wird die individuelle Zeit mit der aktuellen Zeit des HW Timers überschrieben. Ist dies nicht der Fall, prüft der SW Watchdog ob die in der Liste eingetragene maximale Zeit überschritten wurde. Trifft dies zu, wird der Automat in den Anfangszustand versetzt und je nach Bedarf wird ein Zähler im SW Watchdog inkrementiert. Im Vergleich zum HW Watchdog unterscheidet sich das Konzept dahingehend, dass nicht der Timer bei Erreichen eines Kontrollpunkts auf null gesetzt wird, sondern die letzte Zeit mit der aktuellen Zeit überschreiben wird. Dadurch wird erreicht, dass mit *einem* HW Timer mehrere Kontroller individuell überwacht werden können und jeder Kontroller ein eigenes „Fehlerverhalten“ haben kann.
Auslöser für ein „Fehlerverhalten“ muss *nicht* zwingend ein Fehler des Automaten / Kontrollers sein. Der SW Watchdog wird auch dazu genutzt, um reale zeitliche Abläufe abzubilden. Er synchronisiert an bestimmten Punkten die Systemzeit mit der realen Zeit für reale Abläufe. Beispielsweise neigt das System, mit absichtlich provoziertem mechanischem Spiel, zum Aufschwingen der Aktoren beim Erreichen einer Endposition. Der SW Watchdog veranlasst das System dazu, nur einmal pro Sekunde die Endposition der Aktoren zu überprüfen. Konkret wird forciert, dass die maximale Bearbeitungszeit des Kontrollers überschritten wird. Da ein Schwingen der Mechanik ca. 300-500ms anhält, wird einfach und wirksam ein Aufschaukeln ohne Fuzzy Logic mit dem SW Watchdog verhindert.
Auch das *Observer Pattern* [Hus07] wurde den Bedingungen angepasst. Üblicherweise würde sich der *Observer* den Zustand des *Subjects* merken. Um Speicherplatz im RAM zu sparen und den aktuellen Zustand nicht zweimal zu halten, hält jeder Kontroller ein zusätzliches „`stateChange`“ Bit. Dieses Bit wird beim Verlassen des jeweiligen Kontrollers auf `true` oder `false` gesetzt. Der SW Watchdog fragt nur dieses Bit, in jedem Kontroller, ab. Damit wird geprüft, ob im Kontroller ein Zustandswechsel stattgefunden hat. In *Abbildung 5-24* wird der vereinfachte SW Watchdog mit fiktiven Kontrollern dargestellt.
Der SW Watchdog überwacht auch seine eigene maximale Laufzeit, was in der *Abbildung 5-24* nicht dargestellt wurde. In der aktuellen Konfiguration terminiert der SW Watchdog, nach 15 Sekunden, sich selbst. Dadurch kann effizient ein mechanischer Defekt erkannt werden. Eine Bewegung, mit maximalem Weg, benötigt weniger als drei Sekunden. Ist der Aktor nach 15 Sekunden noch nicht angekommen und es gab in dieser Zeitspanne keine anderen Vorkommnisse (z.B. eine Kollision), kann auf einen Defekt geschlossen werden. Nun können geeignet Prüfroutinen eingeleitet werden, um den Defekt einzugrenzen. Wird beispielsweise bei einer Bewegung kein Drehmoment mehr gemessen, kann nur der Antrieb, die Verkabelung oder direkt der Output-Treiber defekt sein.

Im gesamt Konzept wird der SW Watchdog immer aus dem Super-Loop aufgerufen, unabhängig von seiner Nutzung. Die eigenen Daten des SW Watchdog sind gekapselt und er kann nur durch eine spezielle Funktion von außen gestartet werden. Jede Bewegung der Aktoren startet den SW-Watchdog neu. Damit ist sichergestellt, dass sich überschneidende Aufrufe nicht zu unerwünschten Ergebnissen führen.

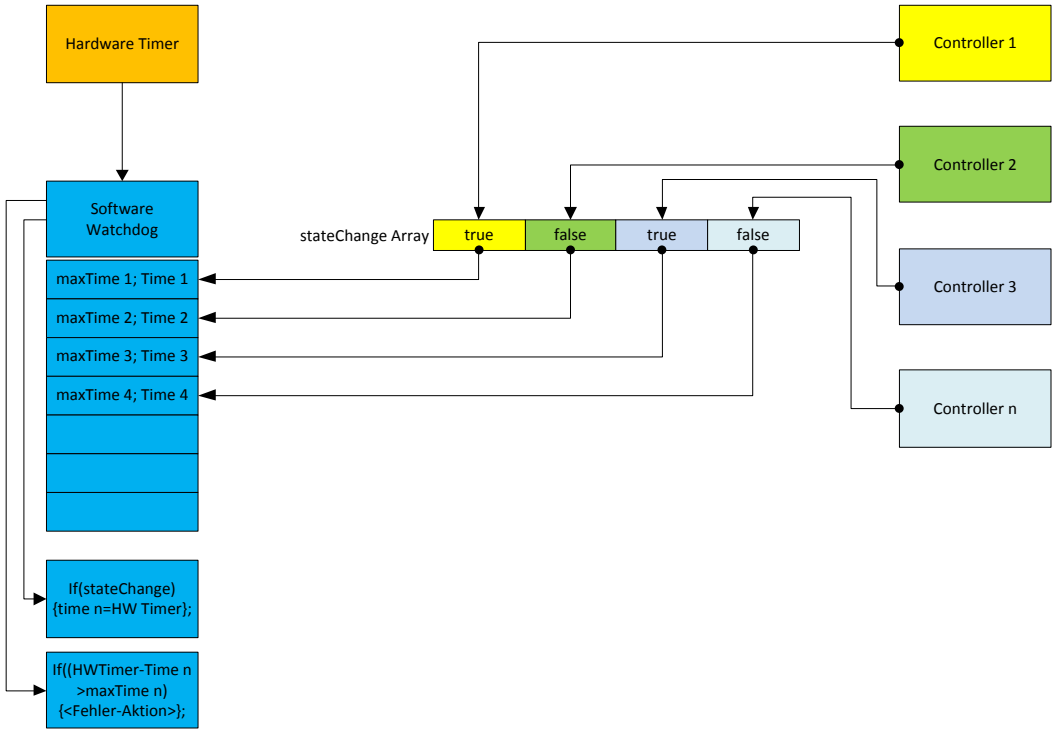

Abbildung 5-24 Konzept Software Watchdog

Der SW Watchdog verletzt das angestrebte OSI ähnliche Layer-Konzept. Er greift über mehrere Schichten, um zentral alle Kontroller zu überwachen. Für diese Betrachtung ist ein nicht so konsequentes Konzept der Layer sinnvoll, wie beispielsweise die Implementierung des TCP/IP Modells (RFC 1122/1123). Damit erweitert sich das Layer Diagramm wie in *Abbildung 5-25* dargestellt, um den senkrecht verlaufenden Security Layer.

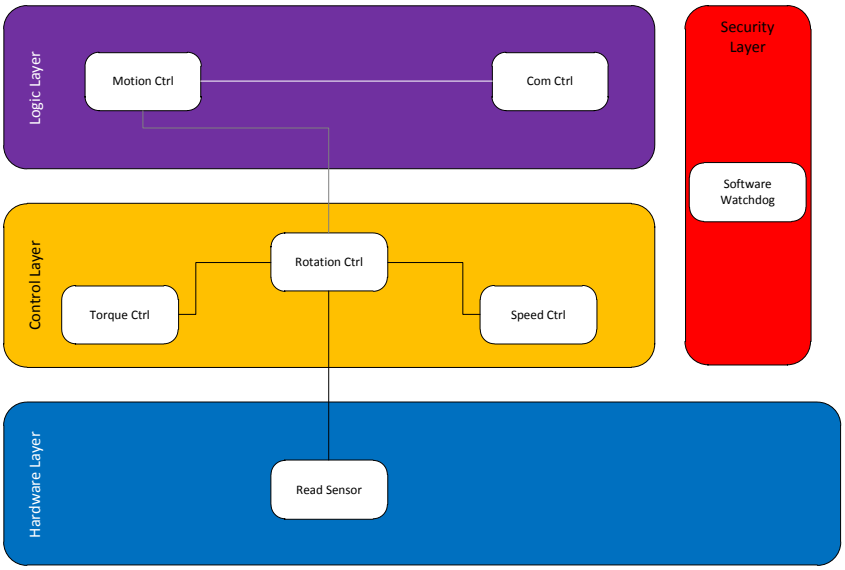

Abbildung 5-25 Layer Darstellung mit Security Layer

Der Security Layer könnte auch zwischen Logic Layer und Control Layer liegen. Dann würde das Konzept aber nur mit dieser Konstellation der Layer sinnvoll anzuordnen sein. Das Konzept würde mit drei Layern, in denen Kontroller liegen, strukturell kein sauberes Konzept bilden.

5.3.3.5 Synchronisation

Das Laufsystem soll den Roboter-Torso über den Boden bewegen (2.5). Dazu ist es nötig die beteiligten Beine zu synchronisieren. Würden die Beine asynchron bewegt werden, würde der ganze Roboter keinem Pfad folgen können. Es kann nicht davon ausgegangen werden, dass jeder Antrieb die gleiche Umdrehungszahl bei gleicher elektrischer Spannung liefert[18].

Der externe Hauptkontroller wird als Koordinator für die Synchronisation verwendet. Nur der externe Hauptkontroller (Koordinator) hält die Information, welche Beine an einer Bewegung beteiligt sind. Wird eine gemeinsame (synchronisierte) Bewegung vom Koordinator eingeleitet, werden die beteiligten Beine auf die Startposition gefahren. Ab hier wird die synchronisierte Bewegung in Zwischenschritte zerlegt. Diese *Zwischenschritte* haben in jedem Bein dieselbe Länge. Der Koordinator wartet auf die Nachricht, dass die Beine den Startpunkt erreicht haben. Sind alle Nachrichten der beteiligten Beine eingetroffen, sendet der Koordinator Start-Nachrichten an die Beine. Die Beine setzten ihre Teilbewegung (5.3.4.4.4) zum nächsten Zwischenschritt fort. Der Koordinator wartet wieder auf die Nachrichten der beteiligten Beine. Der Vorgang wiederholt sich an allen Zwischenpunkten bis die Endposition erreicht wird.

Damit das Laufsystem um Kurven laufen kann, haben die synchronisierten Bewegungen nicht die gleiche Länge. Dadurch haben die Bewegungen auch nicht die gleiche Anzahl an Zwischenschritten und damit auch nicht die gleiche Anzahl an Nachrichten. Aus diesem Grund sind die Nachrichten anonym. Es gibt keinen Index an dem der Zwischenschritt identifiziert werden kann. Der Koordinator kann damit einfacher realisiert werden. Die Strecke auf der Außenseite einer zu laufenden Kurve ist

[18] Bedingt durch Produktionsschwankungen der Antriebe und durch das Einwirken von unterschiedlichen physikalischen Kräften auf die Beine.

länger und hat damit mehr Zwischenpunkte. In dieser Situation ist es Aufgabe des Koordinators, dem Bein an der Außenseite mehr Zwischenpunkte zu bestätigen, als dem Bein auf der Innenseite der Kurve.
Die Realisierung des externen Hauptkontrollers ist nicht mehr Bestandteil dieser Arbeit. Im Abschnitt (5.4) wird dieser skizziert und die Ansatzpunkte für seine Entwicklung erläutert.

5.3.3.6 Ablaufbeschreibung des Systems

In diesem Abschnitt werden vor der Detailmodellierung die Abläufe in einer Top-Down Betrachtung *vereinfacht* beschrieben. Es wird ein störungsfreier Ablauf beschreiben, der eine komplexe Bewegung ausführt. Die einzelnen Schichten werden mit ①Logic-Layer, ②Control-Layer und ③ für den Hardware-Layer gekennzeichnet. Der Ablauf wird in *Abbildung 5-26* dargestellt. Eine Übersicht des Logic-Layer wird in *Abbildung 5-31*, Seite 99 dargestellt. Der Control-Layer wird in *Abbildung 5-28*, Seite 89 veranschaulicht.

Das komplette Beinsystem befindet sich im Ruhezustand (idle) und wartet auf eingehende Anweisungen vom externen Hauptkontroller (Intel Atom®, *Abbildung 4-1*, Seite 37). Der Hauptkontroller sendet über Ethernet die *Anweisung „DoStep“*(*Tabelle 5-4*), mit entsprechenden Koordinaten an das Beinsystem. Die Ethernet-MCU (5.3.4.4.13) überprüft anhand einer Whitelist (5.3.3.2), ob es sich um eine korrekte Anweisung handelt. Ist dies der Fall, werden die Koordinaten für die Bewegung in Datenpakete zerlegt(5.3.4.4.12). Die Anweisung wird als Header mit den Koordinaten als Datenpakte, an die Bein-MCU gesendet.
Die Bein-MCU empfängt diese Nachricht über den „*Com-Ctrl.* ①“ (5.3.4.4.12) Kontroller, der den „*Motion-Ctrl.* ①“ (5.3.4.4.2) Kontroller in den Startzustand setzt. Der „*Motion-Ctrl.* ①“ Kontroller verzweigt entsprechend dem Header (Anweisung) in Subautomaten, die diese Anweisung bearbeiten. In diesem Fall verzweigt die „*Motion Ctrl.* ①“ in den *Subautomaten „DoStep*①“ (5.3.4.4.3), der die Anweisung bearbeitet. Dieser Subautomat zerlegt den Bewegungsablauf in Einzelbewegungen. In der ersten Einzelbewegung wird das Bein an den Körper gezogen (Verfahren aus „*CalcMinTorsoDistance*“ 5.3.4.4.10, Kinematik 5.3.4.4.9 und inverser Kinematik 5.3.4.4.8, ①). Diese errechnete Koordinate wird an den Control-Layer übermittelt und damit wird eine neue Einzelbewegung eingeleitet. Die Koordinate wird in Winkel transformiert(5.3.4.4.8, Umrechnung in ADC-Werte) und in die Datenstruktur②(5.3.4.2) des Control-Layers kopiert. Der Logic-Layer startet alle beteiligten Kontroller und zudem die Überwachung des realen zeitlichen Ablaufs durch den SW-Watchdog(5.3.3.4).
Im Control-Layer wird die Einzelbewegung vom „*Rotation Ctrl.* ②“ (5.3.4.3.1) Kontroller geprüft, ob die Winkel von der Mechanik erreicht werden können. Ist dies der Fall, werden die Aktoren in die entsprechenden Winkel gefahren. Dabei wird die „*Rotation Ctrl.* ②“ von der Kollisionskontrolle②(5.3.4.3.3) und der Geschwindigkeitskontrolle② (5.3.4.3.4) überwacht. Werden die Winkel ohne Störung (z.B. Kollision, Defekte) erreicht, versetzt sich der „*Rotation-Ctrl.* ②“ Kontroller in den Idle Zustand. Die Kollisionskontrolle② und die Geschwindigkeitskontrolle② reagieren auf diesen Zustand und versetzten sich ebenfalls in den Idle Zustand.
Der Subautomat „*DoStep*①“(5.3.4.4.3) hat die „*Rotation Ctrl.* ①“ während der Einzelbewegung der Aktoren überwacht und reagiert auf den Zustand Idle. „*DoStep*①“ leitet weitere Einzelbewegungen nach dem obigen Schema ein, bis sich der Fuß *über* der Startposition, für eine Einzelbewegung mit

Bodenkontakt, befindet. Der Fuß wird zur Startposition gefahren. Dabei beauftragt der Kontroller „*DoStep*①“ den Kontroller „*Touch Ctrl.* ①“ (5.3.4.4.11), der die Zehe des Fußes beobachtet um einen sicheren Bodenkontakt herzustellen.
Für diese Einzelbewegung werden die Aktoren wieder mit der „*Rotation Ctrl.* ②“ gefahren. Dabei werden die Zustände der Zehen überwacht, bis ein Kontakt der Zehen mit dem Boden detektiert wurde. Gelingt dies nicht, wird die Bewegung in weitere Einzelbewegungen durch den „*Touch Ctrl.*①“ Kontroller zerlegt, der wiederum neue Einzelbewegungen im Control-Layer beauftragt. Der Kontroller „*DoStep*①“ wartet auf den Erfolg der „*Touch Ctrl.* ①“[19].
Sobald der Fuß sicher platziert wurde, sendet die „*Touch Ctrl.* ①“ die erreichten Koordinate an den externen Hauptkontroller, die mit der Funktion „*ADC2Point*①“(5.3.4.4.9) aus den realen Winkeln errechnet wird und beendet seinen Auftrag. Das Senden wird vom „*Com-Ctrl.* ①“(5.3.4.4.12) Kontroller bearbeitet und ähnelt dem Empfang von Nachrichten in umgekehrter Reihenfolge. Die vom externen Hauptkontroller geplante Koordinate wird durch die tatsächlich erreichte Koordinate in der Datenstruktur des Logic-Layers ersetzt.
Der „*DoStep*①“ Subautomat reagiert auf den Erfolg der „*TouchCtrl.* ①“ und beauftragt den Kontroller „*LineMover*①“ (5.3.4.4.4) mit der weiteren Bearbeitung des Bewegungsauflaufs. Der „*LineMover*①“ zerlegt die Einzelbewegung am Boden in weitere kleine Zwischenbewegungen (5.3.4.4.7). Der „*LineMover*①“ sendet eine Nachricht an den externen Hauptkontroller, dass dieses Bein bereit ist für eine synchronisierte Teilbewegung. Der externe Hauptkontroller fungiert jetzt als Koordinator und sammelt die Meldungen der Beine, die an der synchronisierten Bewegung beteiligt sind. Sobald alle Meldungen von den Beinen eingegangen sind, sendet er den Startbefehl.
Der „*LineMover*①“ wartet darauf, dass die „*Com-Ctrl.* ①“ den Startbefehl empfangen hat. Ist dieser eingegangen, beauftragt der „*LineMover*①“ den Control-Layer mit der Bewegung für eine Zwischenbewegung. Dabei verwendet der „*LineMove*①“ die „*BruteForceRotationCtrl.* ②“(5.3.4.3.2) für die Bewegung und nicht die „*RotationCtrl.* ②“, da dieser speziell für diese Aufgabe modelliert wurde. Während die Zwischenbewegung real abläuft, berechnet der „*LineMover*①“ die Winkel für den nächsten Zwischenschritt. Ist der vorherige Zwischenschritt beendet, sendet der „*LineMove*①“ wieder die Synchronisierungsnachricht an den externen Hauptkontroller und wartet auf die Bestätigung. Wird diese Empfangen, schreibt der „*LineMover*①“ die vorausberechneten Winkel in die Datenstruktur des Control-Layers und der Vorgang beginnt von neuem bis der Endpunkt der *Anweisung* „*DoStep*“ erreicht wurde.
Sobald der Endpunkt erreicht wurde, nimmt der „*LineMover*①“ den Zustand Idle an. Der Subautomat „*DoStep*①“ reagiert darauf und versetzt sich in den Zustand Idle. Die „*Motion-Ctrl.* ①“ erkennt dies und sendet an den externen Hauptkontroller die Nachricht über den Erfolg und die Beendigung der *Anweisung* „*DoStep*“. Der „*Motion-Ctrl.* ①“ Kontroller setzt alle Kontroller, Automaten und sich selbst, sowie den SW-Watchdog in den Ruhezustand. Ausgenommen hiervon ist die „*Com-Ctrl*. ①“, da sie weiterhin aktiv bleiben muss, um neue Befehle entgegenzunehmen. Die Bearbeitung der Anweisung „*DoStep*“ ist damit beendet.
Diese Darstellung berücksichtigt keine Störungen und Fehler. Die Behandlung dieser wird in den einzelnen Kontrollern und Funktionen in der Detailmodellierung erläutert.

[19] In der Realisierung wird diese Aufgabe vom „LineMover“ bearbeitet, der die „TouchCtrl.“ beauftragt. Die Rückmeldung wird hierarchisch von der „TouchCtrl.“ über den „LineMover“ zum „DoStep“ Kontroller gereicht.

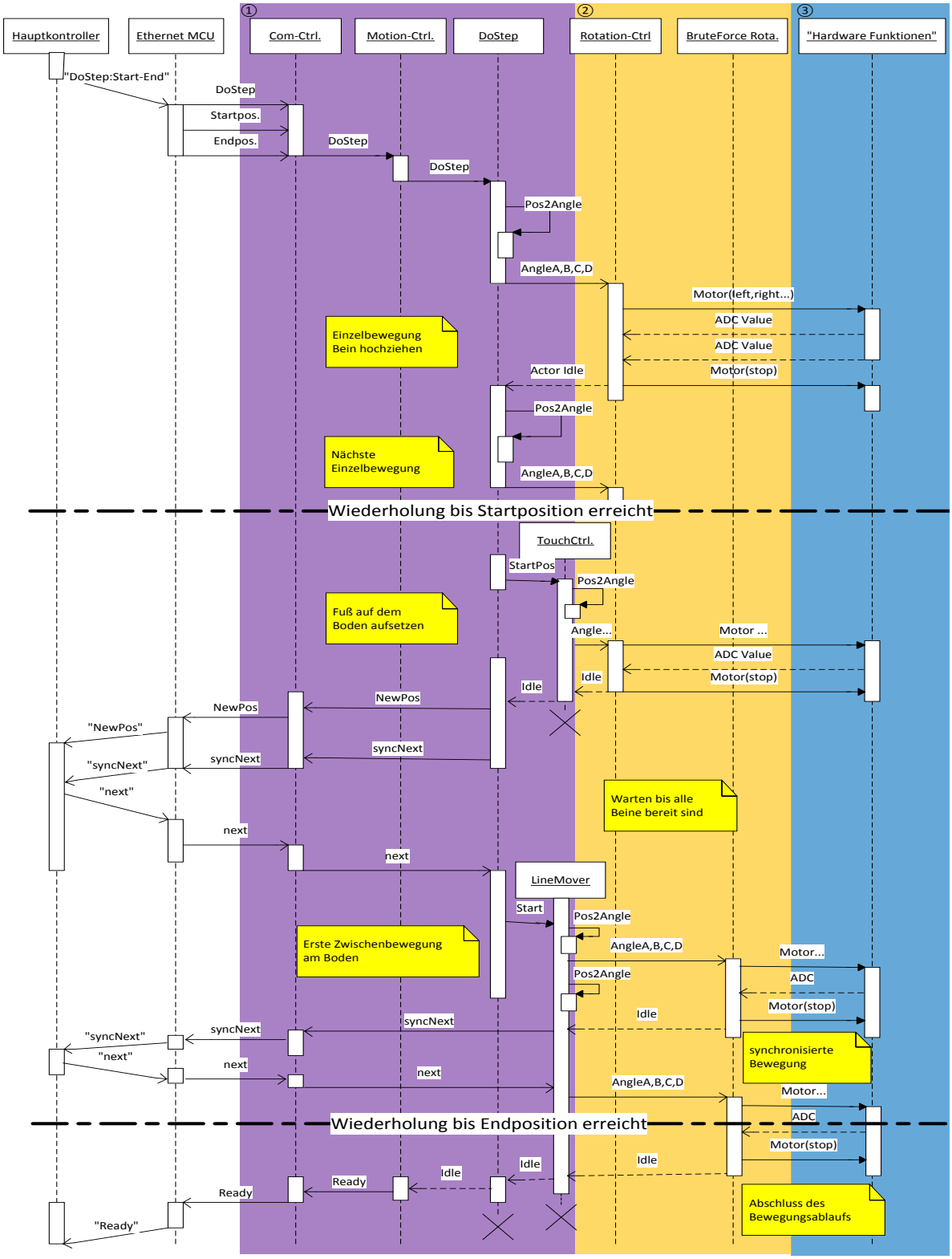

Abbildung 5-26 Vereinfachter Ablauf "DoStep"

5.3.4 Detailmodellierung

Die Detailmodellierung erfolgt in einer Bottom-Up Betrachtung, da für höhere Funktionen erst einige hardwarenahe Implementierungsdetails erläutert werden. Für dieses System wurde ein eigener Coding Guide verwendet, weil das vorher aufgestellte Konzept und die Beschränkungen einer MCU einige Maßnahmen erfordern.

Coding Guide:

Der ATMEGA 128® verfügt nur über 4kByte SRAM. Die Rechnungen mit Winkelfunktionen auf einem 8 Bit System benötigen viel Stack-Speicher. Die AVR math.lib, unter Verwendung von libm.a [ATM11], verwendet nur den double Datentyp (64 Bit). Das Konzept verlangt ständig den Zustand fast aller Kontroller/Automaten zu halten. Es wird erforderlich, sparsam mit der RAM Belegung umzugehen. Da es aber einige zeitkritische Abläufe gibt, wie beispielsweise die IK, muss sehr genau abgewogen werden, wie und wo Speicher belegt oder der geforderte Wert errechnet wird.

Es werden viele Adressen und Koordinaten errechnet anstatt sie zu speichern. Aufgrund des relativ großen Code Speichers (Flash) kann dieser stärker genutzt werden um die RAM Auslastung einzuschränken.

Das allgemein gültige Vorgehen, *keine globalen Variablen* zu verwendet, kann an einigen Stellen nicht ohne hohe Performanceverluste umgesetzt werden. Die 8Bit MCU verwendet 16Bit Adressen die immer von der ALU auf dem Bus zusammengesetzt werden müssen. Einige Automaten rufen viele Funktionen auf. In der Summe der Aufrufe würde die Performance vermindert werden. Trotz des Vorteils von gekapselten Daten wird auf ein globales Array zugegriffen. Durch ausdrucksstarke Namen wird eine versehentliche Nutzung vermieden.

Permanente Daten werden auf dem Heap angelegt, um eine RAM Fragmentierung zu verhindern, die von einer MCU nicht beseitigt werden kann, da es weder eine MMU noch andere geeignete Mittel gibt.

Die meisten Kontroller können von realen Ereignissen (z.B. Kollisionen) abgebrochen werden[20]. Durch das Konzept aus Kapitel 5.3.3.1 kann nicht garantiert werden, dass zum Bereinigen des Speichers an die Ausgangsstelle zurückgekehrt wird. Dadurch ist vom Einsatz der Funktion „`malloc`“ abgesehen worden. Es würde erheblich das Sicherheitskonzept durch ein Speicherleck gefährden. Alle Variablen und ihre Größe stehen vorher fest und es gibt keine dynamischen Größenveränderungen der verwendeten Daten.

Funktionen habe eine garantierte Laufzeit. Es können nur numerische und rekursive Verfahren verwendet werden, wenn sie (als Funktion) eine garantierte Laufzeit haben. Dadurch ist jedes Verfahren mit einfachsten Mitteln umgesetzt worden. Der SW Watchdog(5.3.3.4.1) überwacht keine Funktionen, da es in diesem Konzept nicht nötig ist.

[20] Beispielsweise werden bei einer Kollision die Kontroller (von außen) durch die Kollisionskontrolle in den Idle Zustand versetzt. Durch das Speicherkonzept kann der Ausgangszustand nicht rekonstruiert werden.

5.3.4.1 Hardware-Layer

Es müssen Funktionen implementiert werden, die rudimentäre Aktionen wie Motor rechts, links und stopp auf Register bijektiv abbilden. Die von den Motortreibern geforderten Kommandos (4.2.3) bzw. Pin-Zustände werden als symbolische Konstante binär festgelegt (*Tabelle 5-3*).

_stop	_left	_right	_hold
0b00	0b01	0b10	0b11

Tabelle 5-3 Binäre Treiberbefehle

Dadurch, dass die betreffenden Pins immer paarweise nebeneinander gelegt wurden (4.2.5.1), kann das Kommando (_left, _right usw.) direkt verodert an das entsprechende Port-Register geschoben (*shift*) werden.

Die obige Vereinfachung ermöglicht es, einige simple Grundfunktionen zu implementieren. Als Beispiel wird die Funktion „`setMotor(Actor,command,speed)`“ genauer beschrieben. Die Funktion benötigt als ersten Parameter einen Aktor, der als symbolische Konstante mit `_MotorA 0`, `_MotorB 1` usw. festgelegt wurde. Das „`command`“ ist das Kommando für eine Aktion des Motortreibers, respektive des Antriebs (z.B. `_left`). Das Argument „`speed`“ von 0-255 bildet die zu erzeugende elektrische Spannung von 0-12 V des Motortreibers ab. Realisiert wird dies über phasenkorrekte PWM in den OCRn Registern der MCU. Für die Kommandos „`_left`“ und „`_right`“ ist der Wert von „`speed`“ das Äquivalent zur Geschwindigkeit des Aktors. Da aber andere Faktoren wie z.B. die Schwerkraft auf den Aktor einwirken, ist „`speed`“ nicht als absolute Geschwindigkeit zu verstehen. Es kann *nicht* allgemein daraus geschlossen werden, das der Wert „`speed=x`“ bedeutet $\frac{x \cdot Winkelgrad}{sec}$.
Das Kommando „`_hold`“ bildet „`speed`“ als stärke der Haltefunktion ab. Das Bedeutet, je höher der Wert „`speed`“ in Kombination mit „`_hold`“ ist, desto stärker wird der Motor aktiv gebremst bzw. gehalten. Für das Kommando „`_stop`“ ist der „`speed`“ Wert nicht relevant. Bei „`_stop`“ wird die anliegende Spannung am Motor abgeschaltet. Es ist aber semantisch eindeutiger den Wert auf null zu setzten, damit deutlich wird, dass der Motor nicht aktiv sein soll.

Um möglichst wenig RAM zu belegen, wurden einige Hilfsfunktionen ergänzt. Die übergeordneten Kontroller speichern nicht, welche Befehle sie selbst an die Aktoren gesendet haben. An einigen Stellen ist es aber nötig, Entscheidungen aufgrund dieser Daten zu treffen. Diese Daten sind in der HW bzw. in den Registern gespeichert. Um das Layer Konzept nicht zu verletzen, werden durch die Hilfsfunktionen die Register abgefragt. Beispielsweise gibt die Funktion „`getSpeed(Actor)`“ die anliegende relative Geschwindigkeit zurück.

Die Abfrage der analogen Sensoren wird komplett von der Interrupt-Routine erledigt und bedarf keiner weiteren Funktionen. Die digitalen Sensoren bzw. die Drucktaster im Fuß werden durch die Funktion „`getTouch`“ abgefragt. Um unterscheiden zu können, welcher Fußkontakt ausgelöst wurde, gibt diese Funktion binär den Zustand der Drucktaster als uint8 zurück. Verwendet wird dies aber erst im Logic-Layer(5.3.4.4.11).
Die Serielle Schnittstelle wurde mit einer Datenübertragungsrate von 115.2 kBaud (in dieser Konfiguration auch kBit/s) konfiguriert, da die Übertragungsstrecke wenige Zentimeter beträgt. Zur

Flusskontrolle werden CTS und RTS Pins definiert. Dies wird vom „*Com-Ctrl.*“ (5.3.4.4.12) Kontroller genutzt.

5.3.4.2 Datenstruktur im Hardware- und Control Layer

Die in Abschnitten 5.3.4 erläuterten Bedingungen haben direkte Auswirkungen auf die Implementierung der Datenstruktur. Um nicht zu viele Referenzen und globale Variablen zu erzeugen, wird eine einzelne Datenstruktur angelegt. Diese Datenstruktur wird gemeinsam von allen Kontrollern und Hilfsfunktionen in diesen beiden Schichten (Hardware- und Control-Layer) genutzt. Werte die in Funktionen gekapselt werden können, werden dort statisch (mit static) angelegt.
Ausgewählt werden die Elemente durch die Häufigkeit ihrer Nutzung. Auch die Koppelung der Daten ist ein Kriterium. Häufige Zugriffe erfolgen auf die von den Sensoren erfassten Winkel und die Zielwinkel, auf die die Aktoren gedreht werden sollen. Auch das Drehmoment ist mit vielen Funktionen gekoppelt. Da dieses Laufsystem pro Bein vier Antriebe verwaltet, bietet sich ein zweidimensionales globales Array an. Dieses Array befindet sich logisch gesehen im Control-Layer, da die stärkste Kopplung zu den Daten durch die „*Rotation-Ctrl“* stattfindet.
Gefüllt wird dieses Array vom *Logic Layer* mit Zielwinkeln und Kontroll-Flags (Steuerbits). Der Hardware-Layer liefert einen hohen Input durch die Sensoren. Der Control-Layer verarbeitet diese Daten. Das in *Abbildung 5-27* dargestellte Array hält fast alle benötigten permanenten Daten. Die hier festgelegte Reihenfolge, der Zeilen, wird im kompletten System eingehalten. Es werden symbolischen Konstanten bzw. Makros benutzt. Die Angabe „_MotorA“, „_MotorB,…“ wird immer benutzt um die Spalten 0,1,… anzusprechen.

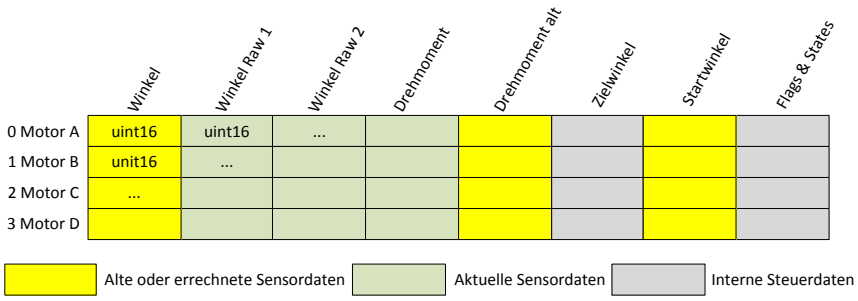

Abbildung 5-27 Control-Layer Array

Die Daten wurden nicht als Struktur (durch struct) angelegt, um einen einfachen Zugriff über den Zeilenindex auf alle Daten des jeweiligen Aktors zu ermöglichen. Damit wird die Detailmodellierung des Kontrollers „*Rotation-Ctrl*“ (5.3.4.3) vereinfacht und es kann ein Programmcode für unterschiedliche Aktoren genutzt werden.
In der Spalte „Flags & States“ werden alle Zustände der einzelnen Kontroller in *einem* unsigned Integer pro Antrieb zusammengefasst. Damit werden Ressourcen an verfügbarem RAM geschont. Aktor abhängige Boolean Werte werden im gleichen Element zusammengefasst. Damit wird erreicht, dass diese Einzeldaten über denselben Spaltenindex verfügbar sind. Aber der Verzicht auf eine Struktur (Bündelung verschiedener Datentypen) wirkt sich negativ auf die Handhabung aus. Es wird mit Masken und Bitmanipulation auf die einzelnen Bits zugegriffen. Aus anderen Schichten wird durch Hilfsfunktionen zugegriffen. Damit bleibt die Lesbarkeit des Codes erhalten.

5.3.4.3 Detailmodellierung Control-Layer

Der Control-Layer ermöglicht es, die Aktoren auf vorgegebene Winkel zu drehen. Diese Winkel werden von der *höheren Schicht (Logic-Layer 5.3.4.4) vorgegeben*. Weiterhin sollen eine Kollisionskontrolle und eine Geschwindigkeitskontrolle, *die realen Einflüsse* zu höheren Schichten weitestgehend kapseln.

In dieser Schicht werden die Zielwinkel und Messdaten in ADC-Werten (Analog-Digital-Converter) gehalten und verarbeitet. Ein ADC Wert beschreibt den Wert, wie er von der Wandler Hardware abgebildet wird. Die Zielwinkel wurden im *Logic-Layer* in diese Normung umgerechnet.

Ablauf

Es wird nun ein störungsfreier Ablauf beschrieben. Detaillierte Verhaltensweisen der Kontroller werden in den jeweiligen Abschnitten beschrieben.

Der hierarchisch höchste Kontroller, in dieser Schicht, ist die „*RotationCtrl*“(5.3.4.3.1). Die Kontroller „*TorqueCtrl*“(5.3.4.3.3) und „*SpeedCtrl*“(5.3.4.3.4) überwachen die Bearbeitung. Die drei Kontroller (RotationCtrl, TorqueCtrl, SpeedCtrl) werden immer aus dem Super-Loop aufgerufen und werden erst aktiv, wenn ein Auftrag des Logic-Layers vorliegt. Dieser Auftrag wird über die Kontroller-Zustände in der Datenstruktur (5.3.4.2) manipuliert. Soll der Kontroller einen Auftrag ausführen, wird jeder Kontroller in den Startzustand, durch die Hilfsfunktion „setActorStart“, gesetzt. Diese Hilfsfunktion kopiert auch die Zielwinkel für eine Bewegung in die Datenstruktur 5.3.4.2. Die Daten werden kopiert, um sie einfacher in dieser Schicht zu kapseln.

Ist die „RotationCtrl“ gestartet, steuert diese die Aktoren auf die Zielwinkel. Sind die Zielwinkel erreicht worden, versetzten sich die Kontroller selbst in den Idle Zustand und warten auf neue Aufträge. Über die Hilfsfunktion „areAllActorsIdle“ fragt ein Kontroller[21], des Logic-Layer, permanent die „RotationCtrl“ ab. Der Logic-Layer(5.3.4.4) stellt damit fest, wann ein Auftrag abgearbeitet wurde und kann ggf. den nächsten Auftrag einleiten.

Abbildung 5-28 zeigt symbolisch den Zugriff auf die Kontroller und deren Funktionen im Control Layer.

[21] Im Logic-Layer werden für bestimmte Aufgaben spezialisierte Kontroller eingesetzt. Nähe Details werden im Logic-Layer erläutert.

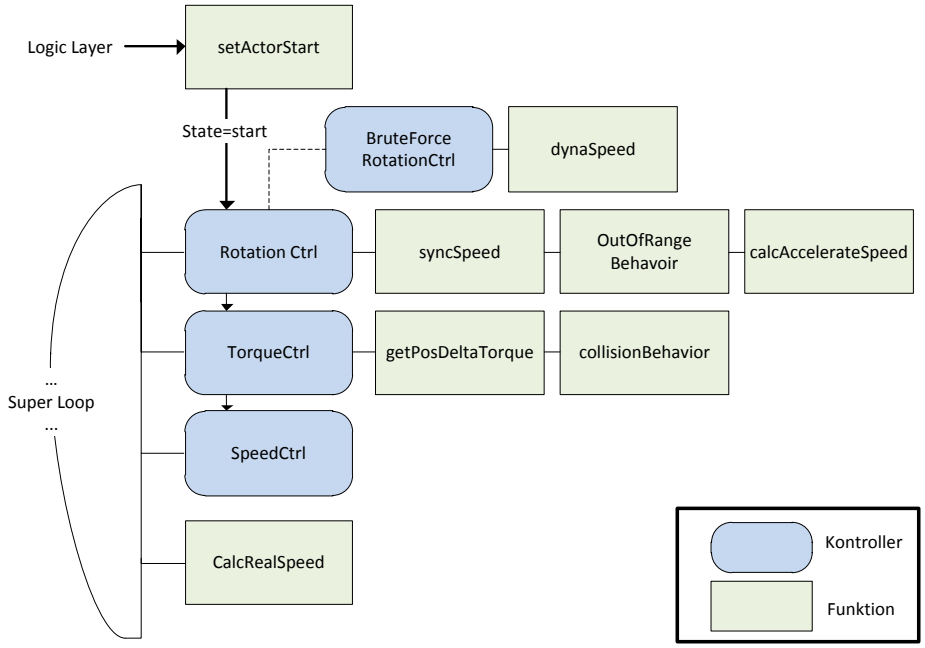

Abbildung 5-28 Kontroller und Hilfsfunktionen im Control-Layer

5.3.4.3.1 Rotation Ctrl

Ein komplexer Kontroller ist in der Funktion „*RotateActor(speed)*“ integriert. Diese Funktion enthält den Automaten, aus Kapitel 5.3.3.3 der als „Rotation Ctrl“ bezeichnet wird. Der Übergabewert „speed“ bezeichnet die relative Geschwindigkeit des jeweiligen Aktors.

Der Kontroller ist in *Abbildung 5-29*, die Hilfsfunktionen in *Abbildung 5-30* dargestellt. In den Zuständen werden die realen Hintergründe erläutert.

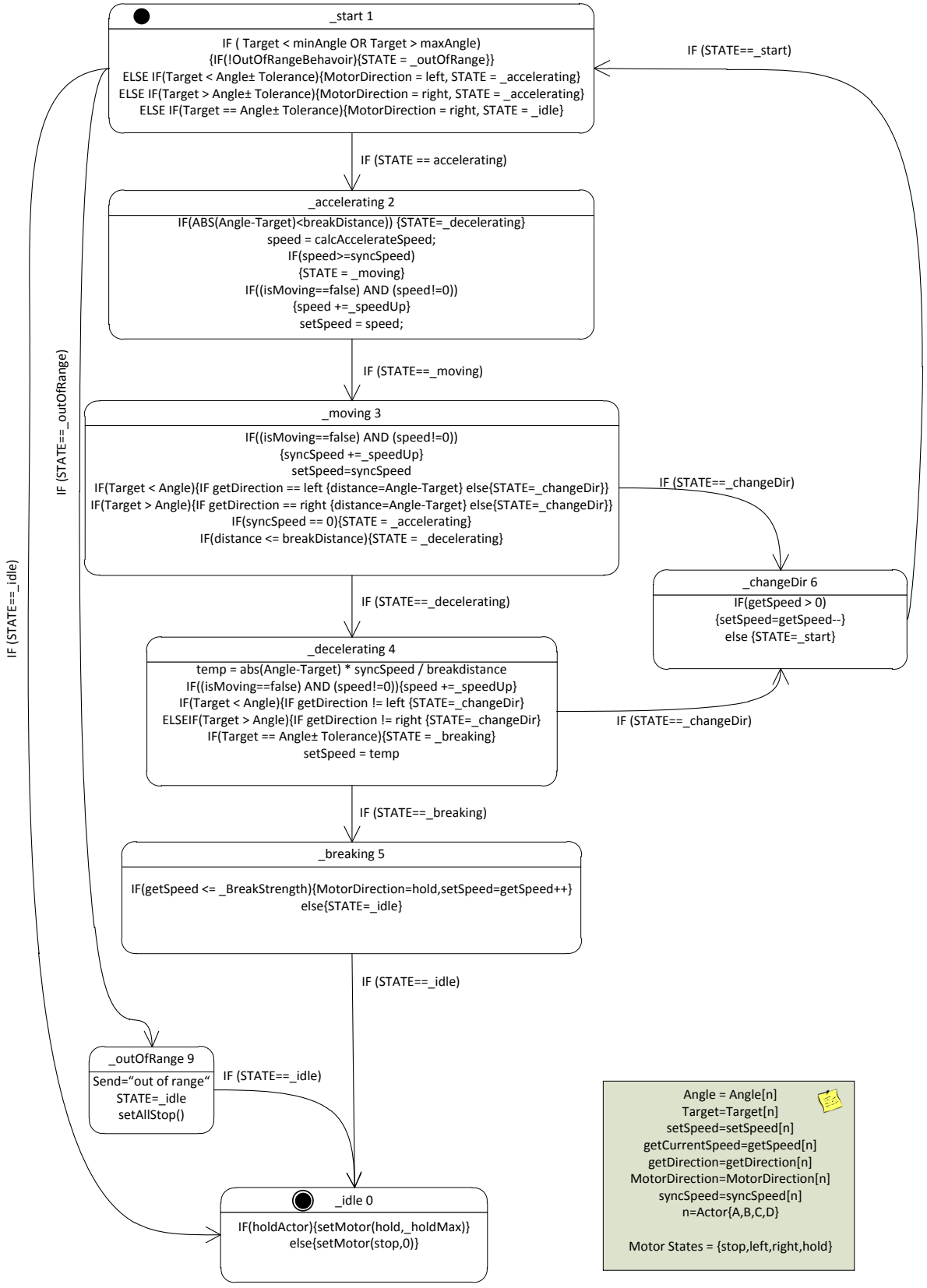

Abbildung 5-29 RotationCtrl Automat

Abbildung 5-30 Hilfsfunktionen RotateCtrl

„syncSpeed“

Bevor der eigentliche Kontroller aufgerufen wird, wird mit der Hilfsfunktion „syncSpeed“ (*Abbildung 5-30*), ein individueller Geschwindigkeitswert für jeden Aktor berechnet. Mit diesem Geschwindigkeitswert wird die Drehzahl des Motors manipuliert. Diese Hilfsfunktion bezieht alle Aktoren in den Algorithmus mit ein. Durch die Anpassung der Geschwindigkeiten wird erreicht, dass die Aktoren zeitgleich ihren Zielwinkel erreichen. Es werden alle Aktoren in eine Beziehung zur maximal angegebenen Geschwindigkeit und des noch zurückzulegenden Abstands bis zum Ziel gesetzt.

Konkret wird interaktiv der maximale Abstand/Winkel Θ gesucht (*Abbildung 5-30*, *syncSpeed*). Dieses Θ dient als Faktor für die Berechnung von „syncSpeed“, um mit einem zweiten Durchlauf, jeweils jedem Aktor die individuelle Geschwindigkeit zuzuordnen. Diese einzelnen Geschwindigkeiten sind nur zum jeweiligen Zeitpunkt gültig und setzen ein physikalisch *ideales* Verhalten der Aktoren voraus. Da aber diese Berechnung bei jedem Aufruf des Kontrollers erfolgt und dies in schneller Folge, ergibt sich eine gute Annäherung an das reale Verhalten der Aktoren. Arbeitet beispielsweise ein Antrieb gegen die Schwerkraft und der zweite Antrieb mit ihr, nähert sich der „fallende“ Aktor schneller dem Zielwinkel als berechnet. Da aber dieser zu schnelle Aktor, schneller seine Distanz zum Ziel verringert, wird dieser im nächsten durchlauf langsamer angesteuert. Bei der Betrachtung der Gleichung ist sogar das Ergebnis Null möglich. Dies ist erwünscht, da dann der Aktor angehalten wird, bis die anderen Aktoren sich soweit genähert haben um gleichzeitig an den jeweiligen Zielwinkeln anzukommen. Eine Prüfung auf Division durch Null, wird in *Abbildung 5-30* „syncSpeed“ nicht dargestellt.

In der Realisierung liefert diese Annäherung ein gutes Verhalten. Hier wäre auch eine Funktion denkbar, die Aufgrund der Lage des Aktor berechnet, wie hoch ein Korrekturfaktor sein müsste. Aber durch die vielen Einflüsse, die in der Realität auf das komplette Bein einwirken, wäre dies schwer umsetzbar. Da diese Einflüsse selten vorher bekannt sind, *ist hier das Reagieren besser als das Planen*.

Kontroller

Die *Abbildung 5-29* zeigt nur einen Kontroller/Automaten. Tatsächlich werden alle Antriebe (pro Super-Loop) iterativ durchlaufen. In dieser Konfiguration sind es vier „pseudoparallel“ ablaufende Kontroller, die jeweils ihren eigenen Zustand, mit dazugehörigen Daten, haben. Konkret bedeutet dies, dass jeder Antrieb eine eigene Zustandsbehandlung erhält, aber der Code nur einmal implementiert wurde.

Über den indizierten Zugriff auf das globale Control Layer Array (*Abbildung 5-27*) stehen alle benötigten Daten für den jeweiligen Aktor bereit. Der Nachteil dieser Methode ist, dass ausgehend vom Array Index, jede Speicheradresse berechnet werden muss. Der Compiler hat keine Möglichkeit eine feste Adresse im Assembler Code festzulegen[22]. Um diesen Umstand zu berücksichtigen, wird vor dem Eintritt in den Kontroller der Pointer auf das Element temporär zwischengespeichert.

Jeder Aktor wird vom Kontroller gleich behandelt und unterscheidet sich nur durch seine individuellen Daten. In folgendem Text ist immer die Abfolge aller Aktoren gemeint.

Zustand _Idle 0 State:

Der Kontroller befindet sich im Ruhezustand „idle 0“. Hier wird aus Sicherheitsgründen der Motor immer auf „_stop“ geschaltet. Der Kontroller kann auch bei abweichenden Soll- und Istwerten diesen Zustand nicht allein verlassen.

Zustand _start 1:

Von außen wird der Kontroller gestartet bzw. eine Bewegung eingeleitet. Durch die Schnittstellen-funktion „setActorsStart(*target)“ wird der Kontroller (alle *n* Kontroller) in den „_start 1“ Zustand versetzt. Das übergebene Array „target“ enthält die Zielwinkel, wie vorher vereinbart, in ADC Werten. Die Hilfsfunktion kopiert die Daten aus dem „target“ Array in das globale Array (*Abbildung 5-27*). Durch das Kopieren werden die Daten zu den höheren Schichten gekapselt. Damit wird verhindert, dass Anomalien in den Kontrollern auftreten wenn diese manipuliert werden. Weiterhin werden noch einige Steuerbits gesetzt, die aber erst in anderen Zuständen und Kontrollern benötigt und dort behandelt werden.

Im Zustand „_start 1“ werden die übermittelten Zielwinkel überprüft. Hierzu wurde ein zusätzliches Array direkt im Flash-Speicher angelegt, in dem die empirisch ermittelten mechanischen Extremwerte gespeichert sind (5.3.5.1). Diese Extremwerte sind die jeweils mechanisch zu erreichenden maximalen und minimalen Winkel der einzelnen Glieder. Auch dieses Array folgt dem Spaltenindex, wie das globalen Array in *Abbildung 5-27*, und wird über die symbolische Konstante „_MotorA,B,C,D“ angesprochen. Ergibt die Überprüfung eine nicht mechanisch erreichbare Position, wird eine zweite Prüfung der Zielwinkel eingeleitet, die die Forderung der Selbstkorrektur (5.3.2 c) erfüllt. Hierzu wird die Hilfsfunktion „OutOfRangeBehavoir(Actor)“ *Abbildung 5-30* aufgerufen. Es wird gegen einen, zuvor festgelegten, Toleranzwert geprüft. Liegt der Zielwinkel innerhalb dieser Grenzen, wird der Wert auf das jeweilige Extrem korrigiert. Ergibt diese Prüfung eine zu große Abweichung, werden *alle* Aktoren in den Zustand „outOfRange 9“ gesetzt und die Mechanik gestoppt. Es wird eine Fehlernachricht (Exception) an den externen Hauptkontroller (Intel ATOM) gesendet(5.3.2i). Der Befehl wird im Endeffekt von der Mechanik ignoriert, da große Abweichungen nur durch einen Fehler des externen Hauptkontrollers entstehen können. Im Gegensatz dazu können kleine Abweichungen, wie sie von der Toleranzgrenze berücksichtigt werden, durch Rundungsfehler entstehen und werden

[22] Es wäre möglich den Compiler dazu zu zwingen, den Code n-Mal hintereinander im Flash-Speicher anzulegen, um festen Adressen zu erhalten. Diese Möglichkeit wurde nicht weiter untersucht.

damit ohne Fehlermeldung korrigiert.
Nach der obigen Prüfung wird entschieden, in welche Richtung der Aktor bewegt werden soll und in den Zustand „_accelerating 2“ gewechselt. Um verkettete Bewegungen möglichst flüssig ineinander übergehen zu lassen, wird nur die Richtung an den Antrieb gesendet. In der Implementierung wird zusätzlich auf einen Richtungswechsel geprüft.
Ist der Zielwinkel bereits erreicht, wird in den Zustand „_idle 0“ gewechselt.
Zustand _accelerating 2:
Das gesamte Bein verteilt eine Masse von ca. 5kg auf einen Hebelarm von insgesamt 60cm. Die Antriebe müssen langsam und gleichmäßig auf die übergebene, relative Zielgeschwindigkeit beschleunigt werden. Zusätzlich muss verhindert werden, dass ein zu starkes Anlaufmoment der Antriebe als Kollision ausgewertet wird (5.3.4.3.3). Dies wird mit zunehmendem Verschleiß und Spiel der Antreibe kritischer. Der Aktor wird immer in diesen Kontroller-Zustand gezwungen.
Geprüft wird der *noch zurückzulegende Weg* bzw. Winkel. Unterschreitet der Aktor diese feste Distanz, wird der Zustand auf „_decelerating 4“ gesetzt. Diese untere Schranke wurde experimentell ermittelt[23] (als _BreakDistance bezeichnet).
Im negativen Fall wird ein Geschwindigkeitswert in Abhängigkeit vom Startwinkel errechnet. Die Hilfsfunktion „calcAccelerateSpeed“ (*Abbildung 5-30*) errechnet proportional zur Distanz vom Startpunkt, einen ansteigenden Geschwindigkeitswert der an den Motor übermittelt wird. Es wird wieder ein idealisiertes Verhältnis zwischen Weg und Geschwindigkeit angewendet. Im Gegensatz zu „*syncSpeed*“, wird nicht die *Restdistanz,* sondern der schon zurückgelegte Weg im Algorithmus benutzt. Als maximale Geschwindigkeit wird der jeweilige Rückgabewert von „*syncSpeed*“ verwendet. Dadurch dass in beiden Gleichungen mit dem Verhältnis zwischen Weg und Geschwindigkeit gerechnet wird, übt die Hilfsfunktion „calcAccelerateSpeed“ nur einen korrigierenden Faktor aus.
Liegt bereits die Zielgeschwindigkeit am Antrieb an, wird der Algorithmus übersprungen, um zu verhindern, dass der Antrieb unnötig gebremst wird. Dieser Fall tritt ein, wenn der Aktor bereits von einer vorherigen Bewegung beschleunigt wurde.
Zusätzlich wird geprüft, ob „*syncSpeed*“ den Aktor in den Stillstand versetzt hat. Trifft dies zu, wird der Startwinkel mit dem aktuellen Winkel überschrieben. Damit wird erreicht, dass der Antrieb nach einem Stillstand wieder langsam angefahren wird.
Anschließend wird der Antrieb auf den ermittelten Geschwindigkeitswert gesetzt.
Geprüft wird jetzt, ob das Steuerbit „isMoving*“ *nicht* gesetzt wurde und die Geschwindigkeit *nicht* auf Null gesetzt wurde. Treffen beide Bedingungen zu, wird die Geschwindigkeit zusätzlich inkrementiert. Es wird geprüft, ob die gesetzte Geschwindigkeit zu einer realen Bewegung des Aktors führt. In einigen Situationen erzeugt der Antrieb bei der gesetzten Geschwindigkeit zu wenig Drehmoment, um den Aktor zu bewegen. Durch schrittweises inkrementieren der Geschwindigkeit wird der Totpunkt des Antriebs überwunden. Die Geschwindigkeit kann aber nicht unkontrolliert wachsen, da sie im nächsten Super-Loop wieder durch „s*yncSpeed*“ angepasst wird.
Ist die Zielgeschwindigkeit aus „*syncSpeed*“ erreicht, wird der Zustand auf *„_moving 3*“ gesetzt.

[23] Der Wert wurde in Versuchen ermittelt, in dem er solange angepasst wurde, bis die Antriebe nicht mehr über den Zielpunkt fahren. Dies wurde in allen Winkellagen und mit absichtlich erhöhtem Spiel getestet. Aufgrund der verketteten Glieder und der daraus resultierenden veränderlichen Masse am Hebelarm, haben die Gleichungen aus der Fachliteratur eine zu hohe Laufzeit für eine MCU.

**Das Steuerbit „isMoving“ wird von der Funktion „CalcRealSpeed“(5.3.4.3.5) manipuliert und bildet die reale Geschwindigkeit als Boolean ab. Diese Abstraktion der Geschwindigkeit zu einem booleschem Wert ist für dieses System ausreichend.*

Zustand moving 3:

Es wird wieder die reale Geschwindigkeit geprüft. Wie im Zustand „*accelerating 2*“ wird ggf. die zu setzende, relative Geschwindigkeit korrigiert. Im Anschluss wird die Drehrichtung überprüft. Anhand des Zielwinkels und dem aktuellen Winkel wird ermittelt, ob der Aktor schon über den Zielwinkel gefahren ist. Trifft dies zu wird der Zustand auf „_changeDir 6“ gesetzt. Dieser Fall tritt auch ein, wenn ein anderer Kontroller (z.B. die Kollisionskontrolle 5.3.4.3.3) die Zielwinkel ändert.

Wurde der Aktor durch „*syncSpeed*“ angehalten, wird der Zustand auf „accelerating 2“ zurückgesetzt. Der Antrieb wird dadurch wieder langsam beschleunigt.

Wird die Distanz zwischen Zielwinkel und dem aktuellen Winkel kleiner als der feste Wert „*_BreakDistance*“, wechselt der Zustand auf „_decelerating 4“.

Danach wird die relative Geschwindigkeit auf den Antrieb gesetzt.

Zustand decelerating 4:

Damit das Getriebe nicht zu stark mechanisch belastet wird, reduziert dieser Zustand die relative Geschwindigkeit proportional zur Winkeldistanz zum Ziel (*Abbildung 5-29*, Zustand 4). Als maximale Geschwindigkeit wird der Rückgabewert von „*syncSpeed*“ verwendet.

Wie in den Zuständen "accelerating 2" und "moving 3" wird der Geschwindigkeitswert nach oben korrigiert, wenn sich der Aktor, aufgrund zu kleiner Werte und folglich nicht ausreichender elektrischer Spannung, nicht real bewegt. Ein einfacher Minimalwert ist nicht ausreichend, da er abhängig vom Winkel wäre.

In diesem Zustand wird die Drehrichtung, wie im Zustand „moving 3“, überprüft. Tritt dieser Fall ein wird in den Zustand „_changeDir 6“ gewechselt.

Nun wird auf Erreichen des Zielwinkels geprüft. Der Zielwinkel wird durch einen festen Toleranzwert erweitert. Dieser Toleranzwert unterdrückt das elektrische Rauschen der AD-Wandler. Ist die Bedingung erfüllt, wird der Zustand auf „_breaking 5“ geändert.

Die berechnete relative Geschwindigkeit wird an den Antrieb übertragen.

Zustand breaking 5:

Der Motor wird hier immer, ohne Prüfung, auf „_hold“(5.3.4.1) geschaltet und ausgehend von der anliegenden relativen Geschwindigkeit auf maximale Stärke der Haltefunktion inkrementiert.

Ist dieser Maximalwert erreicht, wird in den Endzustand „_idle 0“ gewechselt. Für diesen Aktor ist damit der Bewegung beendet.

Zustand changeDir 6:

In diesem Zustand muss ein Mittelweg zwischen schneller Reaktion und dem Schonen der Antriebe gefunden werden. Dieser Zustand fängt mögliche Fehlfunktionen ab, die stets durch einen mechanischen Defekt ausgelöst werden. Im Normalbetrieb, ohne Defekte, wird dieser Zustand aktiv von der Kollisionskontrolle (5.3.4.3.3) genutzt, um schnell zu reagieren.

Als Ausgangswert wird die anliegende relative Geschwindigkeit schrittweise auf Null dekrementiert. Durch die Massenträgheit wird der Aktor nicht vollständig zum Stillstand gebracht. Der Motor wird aber durch den resultierenden Spannungsabfall soweit verzögert, dass nur noch eine geringe Drehzahl real anliegt.

Ist die *relative* Geschwindigkeit gleich Null, wird der Zustand in „_start 1“ geändert. Damit wird der gesamte Durchlauf des Kontrollers wiederholt. Der Kontroller kann damit sicher auf Richtungs-

änderungen reagieren, da er nochmal durchlaufen wird.

Zustand outOfRange 9:

Es werden alle Aktoren gestoppt. Eine weitere Bearbeitung der Zielwinkel ist nicht möglich. Es wird die Forderung nach Selbstschutz der Mechanik (5.3.2d) erfüllt. Eine Fehlernachricht wird an den externen Hauptkontroller gesendet. Der Zustand wird auf „_idle 0“ gesetzt und damit in den Endzustand. Die Zielwinkel werden ignoriert.

Zustand SensorFault 7: (nicht dargestellt)

Um die *Abbildung 5-29* übersichtlich zu halten ist dieser Zustand nicht dargestellt. Aus jedem Zustand wird die Hilfsfunktion „SensorFaultCheck(Actor)“ aufgerufen. In dieser Hilfsfunktion werden die Messwerte (ADC Wert) der Winkelsensoren auf extreme Werte geprüft. Ohne Defekte, treten diese Extremwerte nicht auf. Es gibt zwei Fälle, die einem realen Defekt entsprechen.

Fällt der Messwert auf Null (±Toleranz), ist die Verbindung zum Sensor unterbrochen.

Erreicht der Messwert ADC max. (±Toleranz), ist der Sensor direkt die Ursache. Entweder wurde der Magnet des Magnetfolienpotentiometers (4.2.4.1) entfernt oder das Magnetfolienpotentiometer ist mechanisch Defekt.

Andere Fälle, wie Kurzschlüsse, werden nicht abgedeckt, da die Ursache nicht genau vom System eingegrenzt werden kann. Der Toleranzwert liegt leicht über den realen Rauschwerten des ADC. Trifft eine Fallunterscheidung zu, werden alle Kontroller (alle Aktoren) in den Zustand „SensorFault“ versetzt und alle Antriebe spannungsfrei geschaltet. Der Kontroller kann diesen Zustand nicht selbständig verlassen. Dieser Zustand wird vom Security Layer (5.3.4.5) bearbeitet.

StateChangeBit:

Beim Eintritt in jeden Automaten (Kontroller) wird temporär der Kontroller-Zustand gespeichert. Bei jedem Verlassen des Kontrollers wird geprüft, ob ein Zustandswechsel stattgefunden hat. Ist dies der Fall, wird das Bit auf `true` gesetzt. Dieses „*stateChange*“ Bit wird vom SW Watchdog (5.3.3.4.1) abgefragt. Beim Verlassenen des Kontrollers, wird in beiden Fällen der temporäre Eintrittszustand gelöscht.

5.3.4.3.2 BruteForceRotationCtrl

Dieser Kontroller wird genutzt, um verkettete Bewegungen auf einer geraden Line zu realisieren. Führt der Roboter einen Schritt (5.3.4.4.3) aus, wird in einem Bewegungsabschnitt der Roboter-Torso bewegt. In diesem Bewegungsabschnitt ist das Bein auf dem Boden und führt eine Bewegung auf einer geraden Line aus.

Würde das Bein nur über zwei Punkte vom Startpunkt der Bewegung bis zum Endpunkt bewegt werden, ergäbe der Bewegungsablauf immer einen Bogen. Dies hängt mit dem Aufbau eines Beins zusammen (auch in der Natur). Jedes Bein wird über Achsen bewegt und beschreibt immer einen Kreisabschnitt. Der Roboter hätte einen wankenden Gang.

Um dieses Problem im Detail zu lösen, werden auf der abzufahrenden geraden Linie so viele Punkte wie möglich als Zwischenpunkte auf die Line gesetzt und wird in Folge angefahren. Dadurch ergeben sich sehr kurze Einzelbewegungen. Der Kontroller „*Rotation Ctrl*“ (5.3.4.3.1) kann dies nicht leisten, denn er ist auf Schonung der Antriebe modelliert worden. Die Bewegung wäre zu langsam, da die Wegstrecke der obigen Zwischenpunkte zu klein ist. Diese kurzen Bewegungen würde nur im Zustand

„*decelerating 4*“ ausgeführt werden. Aus diesem Grund wird ein spezialisierter Kontroller benutzt. Dieser Kontroller hat vier Zustände, die von der Bedeutung her äquivalent zum „Rotation Ctrl“ Kontroller (5.3.4.3.1) sind. Beide nutzen dieselben Daten. Aus jedem Zustand des "BruteForce-RotationCtrl" kann zur Laufzeit in den "Rotation Ctrl" gewechselt werden. Die Nutzung wird im Logic-Layer (Kontroller „*DoStep*“ 5.3.4.4.3) erläutert.

Zustand _idle:

Hier wartet der Kontroller und schaltet die Aktoren auf das Kommando „_hold“. Er kann diesen Zustand nicht selbständig verlassen.

Zustand _start:

Es erfolgt das gleiche Verhalten wie im Zustand „_start 1“ der „*Rotation Ctrl*“ (5.3.4.3.1). Die relative Geschwindigkeit der Aktoren wird mit der Hilfsfunktion „*dynaSpeed*“ angepasst.

Zustand _moving:

In diesem Zustand wird geprüft, ob der Aktor über den Zielwinkel gefahren ist. Ist dies der Fall wechselt der Zustand auf „_idle“. Weiterhin wir geprüft, ob das Ziel erreicht wurde. Trifft dies zu, wird in den Zustand „_idle“ gewechselt.

Zustand „ outOfRange“:

Dieser Zustand entspricht dem Verhalten des äquivalenten Zustands der „*Rotation Ctrl*“ (5.3.4.3.1).

StateChange Bit:

Wird der Kontroller verlassen, erfolgt eine Prüfung, ob ein Zustandswechsel stattgefunden hat. Das Verhalten entspricht dem in der „*Rotation Ctrl*“ (5.3.4.3.1).

Hilfsfunktion „*dynaSpeed*“:

Dieser Kontroller (BruteForceRotationCtrl) korrigiert ein Überfahren der Zielwinkel nicht, weil die Korrektur zu viel Zeit benötigt. Da die Antriebe eine steile Drehmomentkurve habe, wird ein anderes Verfahren für die relative Geschwindigkeitskorrektur benutzt. Wird der Zielwinkel genau getroffen, wird die relative Geschwindigkeit inkrementiert. (Der OCR Wert der Pulsweitenmodulation wird genau um 1 erhöht). Wird der Zielwinkel um mehr als einen Winkelgrad überfahren, wird die relative Geschwindigkeit dekrementiert. (Der OCR Wert der Pulsweitenmodulation wird genau um 5 verringert). Dadurch dass ein Überfahren der Zielwinkel vom Algorithmus stärker „bestraft“ wird, kommt es nicht zu einer Rückkopplung der relativen Geschwindigkeit. Die Berechnung erfolgt nur im Zustand „_start“. Dies wirkt sich zusätzlich wie ein Delay auf einen Regler aus.

5.3.4.3.3 Kollisionskontrolle

Die Kollisionskontrolle soll es ermöglichen, eine Kollision zu erkennen. Bedingt durch das USAR Szenario dürfen die Beine nicht auf kleine Kollisionen reagieren. Das System muss durchsetzungsfähig bleiben. Beispielsweise soll auf kleine Äste *nicht* reagiert werden.

Dies wird in diesem eigenständigen Kontroller realisiert. Dieser Kontroller wird im Kapitel 5.3.3.3 als „*TorqueCtrl*.“ bezeichnet. Er setzt das Grundprinzip der „Pitch Detection“ um, wie sie in der Application Note AVR480 [ATM06] der Firma ATMEL beschrieben wird. Das „Anti Pitch System“ wird eingesetzt, um Verletzungen durch elektrische Fensterheber in Fahrzeugen zu verhindern.

Im Roboterbein wird auf einen Grenzwert reagiert. Dieser Grenzwert bildet den Drehmoment*anstieg* in einem festen Zeitraum ab. Es wird nur die Steigung des Drehmoments beobachtet, nicht der (eigentliche) Wert selbst. Es wird nur auf die Größe des Drehmoment-Peaks reagiert.

Der Kontroller besteht aus drei Zuständen und überwacht iterativ alle Aktoren individuell. Es handelt sich auch hier um *n* Kontroller, wie in der *„Rotation-Ctrl“* (5.3.4.3.1).

Zustand _idleTCtrl:

Im „_idleTCtrl“ Zustand wird auf die Zustände *„_moving 3*“ und *„_decelerating 4*“ der *„Rotation-Ctrl“* (5.3.4.3.1) gewartet. Tritt einer dieser Zustände ein, wird in den Zustand „_startTCtrl“ gewechselt. Es wird nicht bei der Beschleunigung auf eine Kollision reagiert. Das Verhalten des Drehmoment-anstiegs, ist mit Spiel in den Antrieben nicht von einer realen Kollision zu unterschieden. Hier steht nur die unten beschriebe *Sicherheitsfunktion* zur Verfügung.

Zustand _startTCtrl:

Wird ein experimentell ermittelter Drehmomentwert[24] *„deltaTorqueMax*“ eines Antriebs überschritten, wird die Hilfsfunktion *„getPosDeltaTorque(Actor)“* aufgerufen. Hier wird geprüft, ob der aktuelle Drehmomentwert großer ist als der vorherige Drehmomentwert. Ist dies der Fall, wird die betragsmäßige Differenz zwischen aktuellem und vorherigem Drehmoment zurückgegeben. Andernfalls wird eine Null zurückgegeben.

Ist nach der Prüfung der Rückgabewert größer als der feste Wert „deltaTorqueMax“, wird ein „Kollisionsverhalten“ eingeleitet. Der Zustand dieses Kontrollers wird auf „_waitTCtrl“ geändert. Andernfalls wird der Zustand auf „_idleTCtrl“ gesetzt. Dieser künstliche Zustandswechsel dient als „Keep Alive“ für den SW-Watchdog 5.3.3.4.1 und signalisiert ihm, dass der Kontroller noch aktiv ist.

Zustand „_waitTCtrl“:

Dieser Zustand hat nur die Aufgabe zu warten und somit ein Fehlverhalten zu provozieren. Der Kontroller ist nicht in der Lage, diesen Zustand selbstständig zu verlassen. Die weitere Behandlung des Zustands erfolgt durch den SW-Watchdog.

Kollisionsverhalten:

Die Routine „collision Behavior“ wird eingeleitet, wenn eine Kollision detektiert wurde. Es soll der natürliche Reflex des Zurückweichens nachgebildet werden.

Realisiert wird dies durch das Setzen neuer Zielwinkel[25] für alle Aktoren, die 10% entgegen dem Kollisionswinkel liegen. Dadurch dass es sich um eine Gliederkette handelt, muss jeder Aktor, für jede Richtung, ein gesondertes Verhalten bekommen. Dies ist besonders wichtig für Bewegungen nahe am Roboter-Torso, da es hier leicht zu Verletzungen eines Menschen kommen kann. Es werden bei vier Antrieben sechszehn Fallunterscheidungen, mit jeweils einer Reaktion für jeden Antrieb benötigt.

Sicherheitsfunktion:

Als Sicherheitsfunktion wird das absolute Drehmomentmaximum jedes Antriebs überwacht. Wird dieses überschritten, wird auch das Kollisionsverhalten eingeleitet.

StateChange Bit:

Wird der Kontroller verlassen, erfolgt eine Prüfung, ob ein Zustandswechsel stattgefunden hat. Auch dieser Kontroller hat ein eigenes „stateChange“ Bit. Das Verhalten entspricht dem in der *„Rotation Ctrl*“ (5.3.4.3.1).

[24] Der Drehmomentwert wurde am Roboterbein mit realen Kollisionen ermittelt. Dieses Vorgehen wird in der AVR 480 Applikation Note [ATM06] vorgeschlagen.

[25] Es wird der Zustand „_changeDir 6“ in der „Rotation Ctrl“ (5.3.4.3.1) aktiv genutzt.

5.3.4.3.4 SpeedCtrl

Die "*SpeedCtrl*" arbeitet nicht wie die vorherigen Kontroller mit einem Automaten, sondern ist als Sicherheitsfunktion konzipiert, für den Fall, dass die anderen Maßnahmen einen Totpunkt nicht erkennen. Dies ist nicht zu verwechseln mit den Fällen in denen eine relative Geschwindigkeitskorrektur nötig ist, was von "*Rotation Ctrl*" 5.3.4.3.1 abgedeckt wird. "*SpeedCtrl*" überwacht, zusätzlich zur "*TorqueCtrl*", das Drehmoment aller Antriebe. Liegt *keine* Kollision vor ("*TorqueCtrl*" 5.3.4.3.3) und weicht das Drehmoment von einem experimentell ermittelten Normalwert ab, wird die relative Geschwindigkeit erhöht. Dieser Normalwert ist das maximale Drehmoment, das im Normalbetrieb auftritt.
Diese Funktion hält keine permanenten Daten. Damit wirkt sich die Korrektur nur kurz auf den Antrieb aus und überwindet damit einen möglichen Totpunkt der Antriebe.

5.3.4.3.5 Funktion „CalcRealSpeed“

Diese Funktion beobachtet, ob sich die Aktoren in der Realität bewegen, da im gesamten System mit relativen (gewünschten) Geschwindigkeiten gearbeitet wird. Dies wurde im Kapitel 5.3.4.1 Hardware Layer bereits erläutert. Es wird die Winkelgradveränderung pro Zeiteinheit betrachtet.
Diese Funktion hat als reale Zeitbasis einen Timer. Der Timer gibt die Funktion nur alle 100ms zur Bearbeitung frei. Sie hat eine gekapselte permanente Datenstruktur, in der 100ms alte Werte der Winkelsensoren gespeichert werden. Wird in die Funktion zur Bearbeitung eingetreten, wird der alte Winkel mit dem aktuellem Winkel verglichen. Weicht dieser vom 100ms alten Wert ab, wird das „isMoving“ Bit in der Datenstruktur 5.3.4.2 gesetzt. In diesem Fall hat sich der Aktor real bewegt. Für dieses System ist es ausreichend, als Boolean zu erfassen ob sich der Aktor bewegt.
„*CalcRealSpeed*“ nutzt keinen Interrupt. Eine präzise Zeitmessung ist hier nicht nötig, da von den aufrufenden Funktionen nur der boolesche Wert betrachtet wird.

5.3.4.4 Detailmodellierung Logic-Layer

In dieser Schicht (Layer) werden die einzelnen Aktoren zum Bein abstrahiert. Das Bein wird nur noch als Einheit betrachtet. Diese Schicht ermöglicht es, Kommandos (Aufträge) vom externen Hauptkontroller zu verarbeiten. Die externen Aufträge werden hier in Einzelbewegungen zerlegt. In diesem Layer wird mit normierten metrischen Koordinaten gearbeitet, die in Winkel bzw. in systeminterne ADC-Werte (5.3.4.3) transformiert werden.

Da diese Schicht direkt mit dem externen Hauptkontroller kommuniziert, ist dieser Layer relativ offen und transparent für eine Weiterentwicklung. Besonders der Kontroller „*MotionCtrl*.“ (5.3.4.4.2) ist ein zentraler Punkt für Erweiterungen. Der Roboter AMEE wird ständig weiterentwickelt. Dadurch können nur Annahmen getroffen werden, welche Anforderungen der externe Hauptkontroller stellen wird.

Alle Kontroller in dieser Schicht bieten die Unterstützung für den SW-Watchdog (5.3.3.4.1, „stateChange“ Bit) an. Die Funktionsweise im Kontroller wurde in der „*RotationCtrl*.“ (5.3.4.3.1) erläutert.

Die Kontroller und die wichtigsten Funktionen werden in *Abbildung 5-31* schematisch dargestellt.

Ablauf:

Der „*MotionCtrl*.“ (5.3.4.4.2) Kontroller wartet auf eingehende Aufträge. Wird ein Auftrag vom „*Com-Ctrl.*“ (5.3.4.4.12) Kontroller empfangen, setzt dieser die Nachricht aus dem Header und den Datenpaketen zusammen. Hat der „*Com-Ctrl*.“ Kontroller diese Aufgabe abgearbeitet, reagiert der „*Motion-Ctrl.*“ Kontroller darauf und kopiert die empfangenen Daten in die Datenstruktur(5.3.4.4.1). Durch den Header der Nachricht verzweigt der „*Motion Ctrl.*“ Kontroller in betreffende Zustände und beauftragt spezialisierte Kontroller mit der Bearbeitung des Auftrags. Ist der Auftrag abgeschlossen oder ist ein Fehler aufgetreten, sendet die „*Motion-Ctrl*.“ eine entsprechende Nachricht an den Hauptkontroller. Der „*Motion-Ctrl*.“ Kontroller wartet wieder auf neue Aufträge.

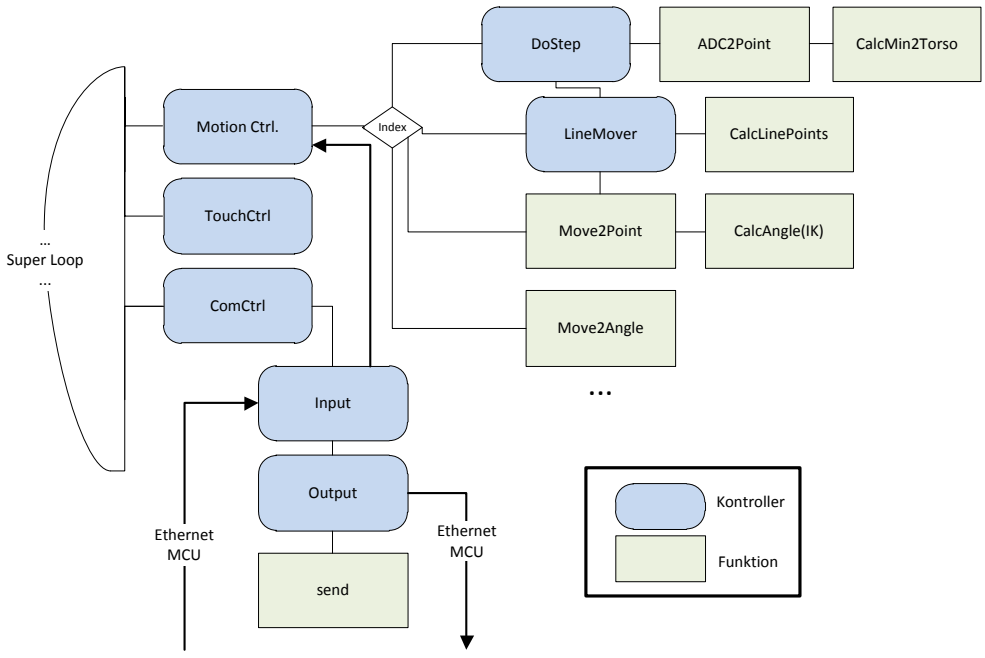

Abbildung 5-31 Kontroller und Funktionen im Logic-Layer

5.3.4.4.1 Datenstruktur im Logic-Layer

Die Datenstruktur im Logic-Layer wird mit symbolischen Namen angesprochen. Im Logic-Layer werden nicht die einzelnen Aktoren betrachtet. Deshalb ist keine iterativ durchsuchbare Datenstruktur (5.3.4.2), wie im Control-Layer, nötig. Es können verschiede Datentypen zusammengefasst werden. Damit ist auch die Erweiterbarkeit einfacher gegeben.
Auch in diesem Layer reagieren die Kontroller aufeinander. Es werden alle Zustände der Kontroller in einer gemeinsamen Datenstruktur permanent gehalten. Auch Steuerbits, die im ganzen Logic-Layer benötigt werden, sind hier zentral gebündelt.
Start- und Endkoordinaten werden permanent im Datentyp double gehalten. Es werden weiterhin permanent die Zielwinkel der Aktoren gehalten. Dies ist für die Verwendung im Kontroller „*LineMover*“ (5.3.4.4.4) nötig. Zudem können, durch die permanente Speicherung, Daten wiederhergestellt werden. Da bei einer Kollision die Zielwinkel im Control-Layer (5.3.4.3), durch das Kollisionsverhalten(5.3.4.3.3) überschreiben werden.
Kontroller interne Daten werden in eigenen statischen Datenstrukturen gehalten.

5.3.4.4.2 Kontroller Motion Ctrl.

Der Motion Kontroller ist in einer Top-Down Betrachtung, der höchste Kontroller. Er verarbeitet Kommandos des externen Hauptkontrollers.
Im Idle Zustand wird das Steuerbit „*newMessage*“ geprüft, das vom „*Com-Ctrl*.“ (5.3.4.4.12) Kontroller gesetzt wird. Ist dies der Fall, wird der Header der Nachricht gelesen. Dieser Header bildet einen Index (*Tabelle 5-4*) für Aufträge ab, mit dem in den entsprechenden Zustand gewechselt wird. Im jeweiligen Zustand, wird das „*newMessage*“ Steuerbit zurückgesetzt. Benötigt der Befehl zusätzliche Daten, werden diese aus der Datenstruktur des „*Com-Ctrl*.“ (5.3.4.4.12) Kontrollers kopiert. Danach werden die entsprechenden Subkontroller beauftragt. Der „Motion-Ctrl.“ Kontroller überwacht den Subkontroller und wartet (indirekt) auf den Abschluss des Auftrags. Dies wird über den Zustand der Subkontroller realisiert. Im Logic-Layer nehmen auch im Fehlerfall die Kontroller immer den Zustand Idle an. Ist der Auftrag abgearbeitet, sendet der „Motion-Ctrl.“ Kontroller die Nachricht „7,ready“ (*Tabelle 5-5*) an den externen Hauptkontroller.

Ablauf:

Da der Ablauf immer identisch bleibt, folgend ein Beispiel mit dem komplexen Befehl „7, DoStep“ aus der *Tabelle 5-4*.
Der „*Motion-Ctrl.*“ Kontroller reagiert auf das Steuerbit „*newMessage*“, woraufhin er in die Anweisung „7, DoStep“ verzweigt und das „*newMessage*“-Steuerbit zurücksetz. Der Start- und Endpunkt wird in die Datenstruktur (5.3.4.2) des Logic-Layers kopiert. Die „*Motion-Ctrl.*“ setzt den Kontroller „*DoStep*“ (5.3.4.4.3) in den Startzustand. Jetzt überwacht die „*Motion-Ctrl.*“ nur noch den Zustand des „*DoStep*“ Kontroller. Ist „*DoStep*“ in den Idle Zustand gewechselt, sendet die „*Motion-Ctrl.*“ die Nachricht „7, ready“ an den externen Hauptkontroller. Der „*Motion-Ctrl.*“ Kontroller wechselt in den Zustand Idle. Der Auftrag wurde abgearbeitet.

Header Index	Befehl	Bedeutung	Werte	Datentyp	Datenpakete ohne Header
0	idle	Es liegt kein Auftrag vor	keine		
1	stop	Alle Aufträge werden abgebrochen	keine		
2	next	Synchronisationsmeldung, Aktion wird fortgesetzt	keine		
3	Move2Point	Es wird eine Koordinate ohne Bodenkontakt angefahren	3	16Bit	6
4	Move2PointTouch	Es wird eine Koordinate mit Bodenkontakt angefahren	3	16Bit	6
5	Move2Angle	Es werden Winkel zwischen den Aktoren angefahren	4	16Bit	8
6	Move2AngleTouch	Es werden Winkel zwischen den Aktoren mit Bodenkontakt angefahren	4	16Bit	8
7	DoStep	Es wird ein Schritt vom Start zum Endpunkt gefahren	6	16Bit	12
8	LineMove	Es wird eine lineare Bewegung auf einer Geraden gefahren	6	16Bit	12
9	LineMoveTouch	Es wird eine Gerade gefahren mit dem Anfangspunkt als Bodenkontakt	6	16Bit	12
10	speed	Maximale Geschwindigkeit der folgenden Aufträge	1	8Bit	1
11	saveWalk	Ändert den Laufmodus	1	8Bit	1
…					
255					

Tabelle 5-4 Anweisungen (Input)

Header Index	Nachricht	Bedeutung	Werte	Datentyp	Datenpakete ohne Header
0	idle	es liegt keine Nachricht vor	keine		
1	OutOfRange	Die Koordinaten / Winkel können nicht erreicht werden	keine		
2	Collision	Die Kollision(auf Aktor n) konnte nicht behoben werden	1	8Bit	1
3	SensorFault	Ein Sensor n hat versagt	1	8Bit	1
4	Error	Allgemeiner Fehler mit Index n	1	8Bit	1
5	Found	Position an dem der Fuß sicher platziert werden konnte	3	16Bit	6
6	syncNext	Es wurde eine zu synchronisierende Position erreicht	keine		
7	ready	Der letzte Auftrag wurde abgearbeitet.	keine		
…					
255					

Tabelle 5-5 Antworten (Output)

Eine Sonderstellung haben die Befehle „1, stop“ und „2, next“ aus *Tabelle 5-4*. Der Befehl „stop“ wird direkt im *„Com-Ctrl.“* (5.3.4.4.12) Kontroller bearbeitet. Der *„Motion-Ctrl.“* Kontroller reagiert in jedem Zustand auf den „stop“ Befehl und versetzt sich selbst in den Idle Zustand. Der „next“ Befehl wird vom *„Motion-Ctrl.“* Kontroller ignoriert. Für beide Anweisungen werden die entsprechenden Steuerbits direkt vom *„Com-Ctrl.“* (5.3.4.4.12) Kontroller gesetzt.

5.3.4.4.3 Kontroller DoStep

Der Kontroller *„DoStep“* führt den Bewegungsablauf eines Schrittes aus. Er wird vom *„MotionCtrl.“* (5.3.4.4.2) Kontroller beauftragt. Der Kontroller zerlegt den Bewegungsablauf in Einzelbewegungen und beauftragt andere Kontroller.

Ablauf

Erfolgt der Auftrag für einen Schritt, müssen der Start- und Endpunkt, als normierte Koordinaten, in der Datenstruktur(5.3.4.4.1) des Logic-Layers vorliegen. Der Endeffektor (Fuß) wird von der aktuellen Position zum Torso bewegt. Diese Bewegung erfolgt senkrecht nach oben auf der Y-Achse (*Abbildung 5-4 Seitenansicht Beinmodell*). Danach wird der Fuß auf selber Höhe (Y-Achse) nach vorn bewegt (X-Achse, Richtung X+). Die X-Position ist die halbe Strecke zwischen der aktuellen Position und dem anzufahrendem Startpunkt. Danach wird der Fuß wieder auf selber Y-Höhe *über* den Startpunkt gefahren. Es wird einer Bewegung über drei Punkte ausgeführt. Ab hier übernimmt der „*LineMover*"(5.3.4.4.4) Kontroller die weitere Bearbeitung. Der Ablauf ist abgeschlossen.
Die Bewegung über drei Punkte könnte auch über einen Spline erfolgen. Die Tests mit dem realen Bein haben aber ergeben, dass eine Bewegung über drei Punkte ein fast identisches Verhalten liefert. Die Berechnung über einen Spline wurde aufgrund der höheren Last nicht angewendet.
Das Verhalten dieses Kontroller (*DoStep*) kann über ein Steuerbit (*safeWalk*) beeinflusst werden. Ist das „safeWalk" Bit gesetzt, wird das Bein so nah wie möglich am Roboter-Torso bewegt („CalcMinTorsoDistance" 5.3.4.4.10). Der Roboter erhält damit die maximal mögliche Bodenfreiheit. Ist das „*safeWalk*" Bit nicht gesetzt, wird der Fuß, wenn möglich, nur 100mm angehoben. Dadurch wird der Bewegungsablauf minimal beschleunigt. Dies dient der *Ästhetik* und ist, wissenschaftlich betrachtet, unnötig.
Am Ende der Bewegung über drei Punkte erfolgt eine Synchronisation mit dem externen Hauptkontroller. Dies ist nötig, um z.B. den „Fast Static Walk" (2.5.2) zu ermöglichen.
Da die Bewegung über drei Punkte schneller ist als die synchronisierte Bewegung am Boden, muss das Bein in einer Position ohne Bodenkontakt *warten,* bis die Bewegung abgeschlossen ist bzw. der externe Kontroller die weitere Bearbeitung frei gibt. Aus Sicht dieses Beins bewegt sich der Boden unter dem Fuß, bis die Bewegung der anderen Beine abgeschlossen ist.

Dieser Kontroller besteht aus sechs Zuständen.
Zustand idle:
Der „DoStep" Kontroller wird von der „MotionCtrl" (5.3.4.4.2) in den Zustand „start" gesetzt.
Zustand start:
In diesem Zustand werden die Koordinaten aus den aktuell anliegenden Winkeln mit der Funktion „*ADC2Point*"(5.3.4.4.9) ermittelt. Diese Koordinaten und der aktuell anliegende Winkel der Achse B (*Abbildung 5-1*) werden der Funktion „*CalcMinTorsoDistance*" (5.3.4.4.10) übergeben. Es werden die Koordinaten manipuliert. Es wird mit der Distanz zwischen Roboter-Torso und Boden geprüft, ob eine Bewegung überhaupt möglich ist. Ist dies nicht der Fall, erfolgt eine Meldung an den externen Hauptkontroller. Der Kontroller geht in den Zustand „*idle*".
Ist das „*safeWalk*" Bit gesetzt, wird das Bein auf die vorher errechneten Koordinaten gefahren. Ist das „*safeWalk*" Bit nicht gesetzt, wird die Y-Achse nur um 100mm nach oben korrigiert (Richtung Y+). Der Auftrag für eine Bewegung wird in beiden obigen Fällen mit der Funktion „*Move2Point*" (5.3.4.4.5) eingeleitet. Der Zustand des Kontrollers wird auf „*halfWay*" gesetzt.
Im Detail wird für die Berechnung die Datenstruktur(5.3.4.4.1) der MotionCtrl. genutzt. Die eigentlichen Startkoordinaten werden zwischengespeichert und nach der Berechnung in die Datenstruktur zurückkopiert.

Zustand halfWay:
Es wird geprüft, ob das Bein die Bewegung abgeschlossen hat. Dies erfolgt über den Zustand des „*RotationCtrl*." (5.3.4.3.1) Kontrollers im Control-Layer.
Ist die Bewegung abgeschlossen, wird die Strecke zwischen aktuellem Punkt und dem Startpunkt berechnet. Es wird die Strecke der X und Z-Achse betrachtet und durch zwei dividiert, um die Koordinate auf halben Weg zu ermitteln. Die Y-Koordinate wird nicht verändert, damit der Fuß auf selber Höher bleibt.
Diese Koordinaten werden der Funktion „*ADC2Point*"(5.3.4.4.9) übergeben[26]. Die Bewegung wird wieder mit der Funktion „*Move2Point*" (5.3.4.4.5) eingeleitet.
Der Fuß bewegt sich jetzt auf die Position, die zwischen dem aktuellem Wert und dem Startpunkt liegt. Der Kontroller wechselt in den Zustand „*moveOver*".
Zustand moveOver:
Es wird wieder geprüft, ob das Bein die Position aus dem vorherigem Zustand erreicht hat. Tritt der Fall ein, wird der Control-Layer(5.3.4.3) beauftragt auf die Startposition zu fahren. Der Ablauf ist äquivalent zum Zustand „halfWay". Nur wird auf die X und Z Koordinate der Startposition gefahren. Die Y Koordinate bleibt unverändert. Es wird in den Zustand „*waitFirstSync*" gewechselt.
Zustand waitFirstSync:
Hat das Bein die Koordinaten aus dem vorherigen Zustand erreicht, wird die Nachricht „*syncReady*" an den externen Hauptkontroller gesendet. Erfolgt die Antwort „next" vom externen Hauptkontroller, wird der Zustand des „*LineMover*"(5.3.4.4.4) Kontrollers auf „start" gesetzt. Der Zustand dieses (*DoStep*) Kontrollers wechselt auf „*waitLineMover*".

Zustand waitLineMover:
Da die Bearbeitung jetzt vom hierarchisch untergeordneten „*LineMover*"(5.3.4.4.4) Kontroller übernommen wird, kann hier auf seine Beendigung gewartet werden. Es wird nur der Zustand des „*LineMovers*" beobachtet und auf seinen „idle" Zustand reagiert. Tritt dieser Zustand ein, wird auch der „*DoStep*" Kontroller in den Zustand „idle" gesetzt. Der Bewegungsablauf eines Schrittes ist damit beendet.
Es wird in diesem Zustand gewartet, da der Kontroller „DoStep" hierarchisch dem „MotionCtrl" (5.3.4.4.2) Kontroller untergeordnet ist und dieser am Idle Zustand die Beendigung des Bewegungsablaufs erkennt.

5.3.4.4.4 Kontroller LineMover
Der „*LineMover*" Kontroller wird genutzt, um Bewegungen auf einer geraden Line auszuführen (5.3.4.3.2). Er wird vom „*DoStep*"(5.3.4.4.3) Kontroller benutzt oder direkt von der „*MotionCtrl*". Er synchronisiert mit Hilfe des externen Hauptkontrollers die Beine des Laufsystems(5.3.3.5). Es müssen Start- und Endkoordinaten in der Datenstruktur(5.3.4.4.1) des Logic-Layers vorliegen. Diese beiden Koordinaten geben die Linie der Bewegung an. Durch Start- und Endkoordinate kann die Richtung der Bewegung bestimmt werden. Für eine Laufbewegung, die den Roboter-Torso über den Boden zieht

[26] Die vorherigen Koordinaten werden nicht gehalten. Die Koordinaten werden errechnet, da dieser Ablauf nicht zeitkritische ist.

oder schiebt, sind nur Linien sinnvoll[27], die auf den gemeinsamen Nullpunkt der X und Z Achse (5.3.1.5, *Abbildung 5-2*) zeigen.

Der „*LineMover*“ Kontroller besteht aus fünf Zustanden. Er wird vom aufrufenden Kontroller in den Startzustand versetzt.

Zustand „start“:

Der Endeffektor(Fuß) wird mit der Funktion „*Move2Point*“(5.3.4.4.5) zum Startpunkt bewegt. Es wird das Steuerbit „*touch*“ gesetzt, damit der Fuß sicher platziert wird. Der Kontroller „*TouchCtrl*.“ (5.3.4.4.11) wird dafür in den Startzustand gesetzt. Der Zustand des „*LineMover*“ Kontrollers wechselt in den Zustand „*wait*“.

Zustand „wait“:

In diesem Zustand wird zuerst geprüft, ob das Bein den Startpunkt erreicht hat. Ist dies der Fall, wird die Nachricht „*sync ready*“ an den externen Hauptkontroller gesendet.

Danach wird geprüft, ob der externe Hauptkontroller die Nachricht „*next*“ gesendet hat. Realisiert wird der Empfang durch den „*Com-Ctrl.*“ Kontroller(5.3.4.4.12), der das Steuerbit „*syncNext*“ auf *true* setzt.

Sind die beiden obigen Bedingungen erfüllt, wird mit der Funktion „*CalcLinePoints*“ (5.3.4.4.7) der erste Zwischenpunkt berechnet. Dieser Zwischenpunkt wird mit dem Aufruf der Funktion „*Move2Point*“ (5.3.4.4.5) als Bewegung in Auftrag gegeben. Durch das setzen eines Steuerbits in der Datenstruktur des „*Control-Layers*“ (5.3.4.2) wird veranlasst, dass die Bewegung mit dem „*BruteForceRotationCtrl*“ (5.3.4.3.2) Kontroller bearbeitet wird.

Der Zustand dieses Kontrollers (LineMover) wird auf „*calc*“ gesetzt.

Ist der Endpunkt schon erreicht worden (5.3.4.4.7), wechselt der Zustand auf „*idle*“. Die synchronisierte Bewegung ist dann beendet.

Zustand calc:

Der Kontroller kann diesen Zustand nur erreichen, wenn dem „*Control-Layer*“ (5.3.4.2) vorher ein Auftrag zu einer Bewegung übergeben wurde. Das Bein befindet sich jetzt in einer realen Bewegung. Diese Zeit wird genutzt, um den nächsten Zwischenpunkt zu berechnen.

Der nächste Zwischenpunkt wird mit der Funktion „*CalcLinePoints*“ (5.3.4.4.7) berechnet. Ist der Endpunkt erreicht, wird in den Zustand „wait“ gewechselt, um die Bewegung zu beenden.
Ist der Endpunkt, des kompletten Bewegungsablaufs, noch nicht erreicht worden, wird dieser Zwischenpunkt, direkt mit der Funktion „*CalcAngle*“ (5.3.4.4.8) in ADC-Werte (5.3.4.4.8, Umrechnung in ADC-Werte), umgerechnet. Die resultierenden ADC-Werte (Winkel) werden in einem temporären Array zwischengespeichert. Der Zustand des Kontrollers wechselt auf „moving“.

Ist die Koordinate nicht erreichbar, wird die Bewegung abgebrochen und es wird eine Fehlernachricht an den externen Kontroller gesendet. In diesem Fall geht der Kontroller in den Zustand „idle“.

Anmerkung: Wird vom Zustand „calc“ in den Zustand „move“ gewechselt, sind maximal 1,5ms vergangen (5.3.5.4). Die reale Bewegung sollte mindestens 1,5ms dauern. Die Aktoren würden sonst den Zielwinkel überfahren, da die MCU, durch die hohe Last der IK, keine anderen Kontroller bearbeitet. Ein interruptgesteuerter Control-Layer würde zwar das Überfahren der Zielwinkel

[27] Die Füße sind nur bedingt drehbar. Aber um andere Bewegungsabläufe zu ermöglichen, wird dies nicht geprüft. Andere Bewegungsabläufe können hier Beispielsweise das Hochziehen an einem Hindernis sein. Die Beine würden eine Bewegung ausführen, die nicht zum Nullpunkt der X, Z-Achsen zeigt.

verhindern. Aber es müsste trotzdem, auf das Beenden der Berechnung gewartet werden, bis der nächste Zwischenpunkt angefahren werden kann. Es käme zu einer Unterbrechung der flüssigen Bewegung.
In dieser Realisierung wird kein Interupt genutzt, da das oben beschriebene Überfahren der Zielwinkel minimal ist und die Richtung in den meisten Fällen logisch richtig ist. Die Bewegung ist zwar nicht so präzise, aber flüssiger. Damit wird der Roboter-Torso ruhiger bewegt.
Die Zeit, die für eine Teilbewegung benötigt wird, kann mit der Anzahl der Zwischenpunkte der Funktion „CalcLinePoints"(5.3.4.4.7) beeinflusst werden.
Zustand moving:
Bei Erreichen dieses Zustands ist das Bein immer noch in Bewegung. (s*iehe oben)*
Es wird geprüft, ob die Bewegung abgeschlossen ist. Trifft dies zu, wird die Nachricht „*sync ready*" an den externen Hauptkontroller gesendet und geprüft, ob die Antwort „*next*" vom externen Hauptkontroller eingetroffen ist.
Sind die Bedingungen erfüllt, werden die im Zustand „*calc*", vorausberechneten Koordinaten vom temporären Array in die Datenstruktur(5.3.4.2) des „Control-Layers" kopiert. Es wird eine neue Bewegung beauftragt.
Eine Überprüfung der Fußkontakte und einer damit einhergehenden Sturzkontrolle obliegt dem externen Hauptkontroller. An dieser Stelle würde dies nur zu einer Überempfindlichkeit auf kleinere Störungen führen.
Der Zustand wechselt auf „*calc*". Dieser Ablauf wird solange ausgeführt, bis alle Zwischenpunkte bearbeitet wurden.
Zustand idle:
Dies ist der Endzustand. Hier erfolgen keine Aktionen.
Im Detail, wird bei jeder synchronisierten Bewegung, in den Zuständen „*wait*" und „*moving*", das Steuerbit „*syncNext*" bearbeitet. Wird der Control-Layer beauftragt, eine Bewegung durchzuführen, wird auch das Steuerbit „*syncNext*" auf „*false*" zurückgesetzt.

5.3.4.4.5 Funktion Move2Point

Diese Funktion bewegt die Aktoren an eine übergebene Koordinate. Das Steuerbit „*touch*" aus der Datenstruktur(5.3.4.4.1) gibt an, ob die Koordinate auf einen sicheren Stand überprüft werden soll. Siehe hierzu „*TouchCtrl*."(5.3.4.4.11).
Beim Aufruf von „*Move2Point*" wird die Funktion „*CalcAngle*" (5.3.4.4.8) aufgerufen. Die zurückgegebenen ADC-Werte (5.3.4.4.8, Umrechnung in ADC-Werte) werden an die Datenstruktur(5.3.4.2) im Control-Layer übergeben. Es erfolgt der Ablauf wie im Abschnitt 5.3.4.3 unter „Ablauf" beschrieben. Sind alle Aktoren an der Zielkoordinate angekommen, erfolgt eine Meldung an den externen Hauptkontroller.
Sind die Koordinaten nicht erreichbar, wird der Befehl ignoriert. Es wird eine Exception (Nachricht) an den externen Hauptkontroller gesendet. Die inverse Kinematik („*CalcAngle*" (5.3.4.4.8)) prüft, ob die Koordinaten überhaupt erreicht werden können.

5.3.4.4.6 Funktion Move2Angle

Mit dieser Funktion werden die Aktoren auf die übergebenen Winkel gedreht. Damit wird es dem externen Hauptkontroller ermöglicht, direkt Winkel mit den Aktoren anzufahren. Diese Funktion ist eine Schnittstelle zum Control-Layer(5.3.4.3). Um das System offen für Erweiterungen zu halten, kann mit dieser Funktion der gesamte Logic-Layer vom externen Hauptkontroller übergangen werden. Nach dem Aufruf der Funktion, werden die Winkel in ADC-Werte überführt (5.3.4.4.8, Umrechnung in ADC-Werte). Die ADC-Werte werden an die Datenstruktur (5.3.4.2) im Control-Layer übergeben. Der SW-Watchog(5.3.3.4.1) wird gestartet. Es erfolgt der Ablauf wie im Abschnitt 5.3.4.3 unter „Ablauf“ beschrieben. Sind alle Aktoren am Zielwinkel angekommen, erfolgt eine Meldung an den externen Hauptkontroller.

5.3.4.4.7 Funktion CalcLinePoints

Diese Funktion berechnet auf der Strecke vom Startpunkt bis zum Endpunkt Zwischenpunkte. Diese Strecke stellt eine Gerade im Raum dar und wird für eine Bewegung auf einer geraden Line benötigt (5.3.4.3.2).

Beim Aufruf der Funktion wird ein Pointer auf die Datenstruktur (5.3.4.4.1) des Logic-Layers übergeben. Die Berechnung der Zwischenpunkte erfolgt vektoriell und wurde auf die Anforderungen vereinfacht. Als erstes wird die Strecke zwischen Start- und Endpunkt berechnet.

$l = \sqrt{(x_{end} - x_{start})^2 + (y_{end} - y_{start})^2 + (z_{end} - z_{start})^2}$. Diese Länge l wird durch eine feste *minimale Schrittweite* dividiert und als Faktor *f* bezeichnet. Die *minimale Schrittweite* (Länge einer Zwischenbewegung) wurde experimentell ermittelt. f stellt die maximal mögliche Anzahl der Zwischenschritte dar. Diese Zwischenschritte werden als Folge [1,*f*] betrachtet und werden als β_n bezeichnet. Mit dem Faktor f und β_n wird ein Skalar λ errechnet. $\lambda = \frac{\beta_n}{f}$.

Damit können dann die Zwischenpunkte mit folgender Gleichung berechnet werden:

$$\begin{matrix} x_{temp} \\ y_{temp} \\ z_{temp} \end{matrix} \begin{matrix} (\lambda \cdot (x_{end} - x_{start})) + x_{start} \\ = (\lambda \cdot (y_{end} - y_{start})) + y_{start} \\ (\lambda \cdot (z_{end} - z_{start})) + z_{start} \end{matrix}$$

Im Detail wird in der Funktion bei jedem Aufruf die Variable β_n inkrementiert. Damit gibt die Funktion bei jedem Aufruf den nächsten Zwischenpunkt auf einer geraden Linie zurück. Abbruchbedingung ist $\beta_n >= f$. Dies stellt den letzten Zwischenpunkt dar, und die Folge β_n ist am Endpunkt angekommen. Der Boolean „*hasNext*“ in der Datenstruktur (5.3.4.4.1) im Logic-Layer wird auf *false* gesetzt und vom „*LineMover*“ (5.3.4.4.4) genutzt.

5.3.4.4.8 Funktion CalcAngle (inverse Kinematik)

Die inverse Kinematik aus Kapitel 5.3.1.1 errechnet die Winkel mit einem idealisierten Modell. Es werden nur die Achsmittelpunkte betrachtet. Die IK muss um den bauartbedingten Versatz der Mechanik erweitert werden. In *Abbildung 5-32* werden die einzelnen Längen dargestellt und benannt.

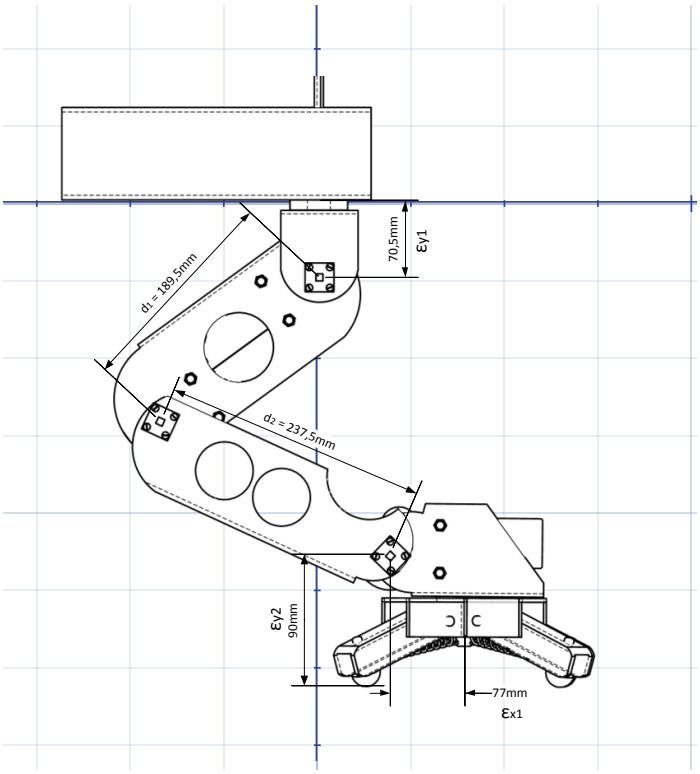

Abbildung 5-32 Versatz der Achsen

Der Funktionsaufruf erfolgt mit der Übergabe der Zielkoordinaten-Pointer und einem Pointer auf ein Array, in dem die Zielwinkel abgelegt werden. Die Koordinate Y kann sofort um den mechanischen Versatz korrigiert werden. Dabei erfolgt eine Fallunterscheidung, ob die Y-Koordinate über oder unter dem Fixpunkt liegt.

$$y = \begin{cases} y + \epsilon_{y1} + \epsilon_{y2} \, f\ddot{u}r \, y < 0 \\ (y - \epsilon_{y1}) + \epsilon_{y2} \, f\ddot{u}r \, y \geq 0 \end{cases}$$

Durch das Abtrennen des 3D Anteils (5.3.1.5), erfolgt eine Transformation zu einem 2D Modell in der Seitenansicht. Die Füßlänge ε_{x1} wird im Raum gedreht. Durch die zweidimensionale Abbildung verkürzt sich die Fußlänge in Abhängigkeit vom Winkel der Achse A. Berechnet wird der Versatz mit: $x = x - \left(\epsilon_{x1} \cdot \sin\left(\left(\frac{\pi}{2}\right) - |\theta_0|\right)\right)$.

Dadurch dass der Fuß immer nach vorn zeigen soll, wird die Korrektur der Z-Koordinate mit vier Fällen unterschieden. Damit die Korrektur nachvollziehbar bleibt, wird vernachlässigt, dass sich negativen Vorzeichen der Koordinaten durchsetzten. Für genauere Details siehe (5.3.1.5). Jede Fallunterscheidung stellt einen Sektor der atan2 Funktion dar.

$$z = \begin{cases} z - |\epsilon_{x1} \cdot \sin(\theta_0)| \, f\ddot{u}r \, x \geq 0 \, und \, z > 0 \\ z + |\epsilon_{x1} \cdot \sin(\theta_0)| \, f\ddot{u}r \, x \geq 0 \, und \, z < 0 \end{cases} \text{ und } z = \begin{cases} z + |\epsilon_{x1} \cdot \sin(\theta_0)| \, f\ddot{u}r \, x < 0 \, und \, z > 0 \\ z - |\epsilon_{x1} \cdot \sin(\theta_0)| \, f\ddot{u}r \, x < 0 \, und \, z < 0 \end{cases}$$

Der Fall *z=0* wird nicht bearbeitet, da eine Korrektur der Z-Koordinate in diesem Fall unnötig ist. Die Fußlänge wurde nicht gedreht, damit besteht kein Grund *z* zu korrigieren.

Umrechnung in ADC-Werte(5.3.4.3):
Die Winkelsensoren haben ein lineares Verhalten. Damit können die Winkel mit einer einfachen Geradengleichung in ADC-Werte überführt werden. Es wird die Gleichung $y = ax + b$ verwendet. Wobei *y* den ADC-Wert darstellt und *x* den Winkel im Bogenmaß. Die Faktoren *a* und *b* wurden mit einer linearen Regression[28] ermittelt. Die Grunddaten wurden beim Kalibrieren(5.3.5.1) ermittelt. Die Faktoren *a*, *b* werden als symbolischer Wert für jeden Winkelsensor implementiert. Inkrementell werden alle ermittelten Winkel in ADC-Werte umgewandelt und im übergebenen Winkel-Array gespeichert.

Mehr Korrekturen sind für die IK nicht nötig, und sie kann wie in Abschnitt 5.3.1.9 implementiert werden.

5.3.4.4.9 Funktion ADC2Point (Kinematik)

Die Kinematik berechnet aus den Winkeln der Aktoren (Glieder) zueinander einen normierten metrischen Punkt im Raum. Diese Funktion ist die Umkehrung der inversen Kinematik. Begründet wird die Notwendigkeit dieser Funktion in Abschnitt 5.3.4.4.11 und 5.3.4.4.2. Es wird, wieder ein analytisches Verfahren verwendet. Geschwindigkeitsvorteile gegenüber anderen Verfahren (z.B. DH) sind minimal. Im System wird diese Funktion nur in zeitunkritischen Situationen benötigt.
Die Winkel der einzelnen Aktoren sind im Control Layer 5.3.4.3 in ADC-Werten gespeichert. Diese Werte werden in Winkel mit einer linearen Funktion umgerechnet. Grundlage ist die Geradengleichung $y = ax + b$. Der ADC-Wert wird durch y beschreiben. Der Winkel wird mit x beschreiben. Die beiden Faktoren a und b wurden mit einer linearen Regression bestimmt. Siehe Abschnitt 5.3.4.4.8. Umgerechnet werden die ADC-Werte zu Winkeln mit $x = \frac{y-b}{a}, a \neq 0$. Der Fall a=0 tritt nicht ein, da der minimale Winkelsensorwert bauartbedingt nie Null sein kann[29]. Der Winkel der Achse A (Schulter, Drehung des Beins) muss gesondert behandelt werden. Durch Rundungsfehler und die Toleranzgrenzen, beim Anfahren einer Position, wird in den meisten Fällen nicht der Winkel 0° errechnet, obwohl zuvor der Winkel 0° angefahren wurde. Der Fuß würde bei Nachkomma-Rundungsfehlern im falschen Sektor berechnet werden. Siehe hierzu „5.3.1.5 Abtrennen des 3D Anteils“. Es werden alle ADC-Werte, die innerhalb der Toleranzen (Rauschen) um Null liegen, als absolut Null behandelt.
Durch die obigen Gleichungen liegen die ADC-Werte im Bogenmaß als Winkel vor.

[28] Durch das lineare Verhalten der Winkelsensoren ist eine lineare Regression unnötig. Bei sehr vielen Werten ist aber nicht sofort ersichtlich, ob sich die Sensoren tatsächlich linear Verhalten.
[29] Nur ein defekter Winkelsensor könnte Null liefern. Dies wird in der „Rotation Ctrl.“(5.3.4.3.1) abgefangen. Zudem wird auf dem AVR X/0 als NaN abgebildet, was zum maximalen Wert des Datentyps führt. Der resultierende Winkel ist damit außerhalb der Reichweite.

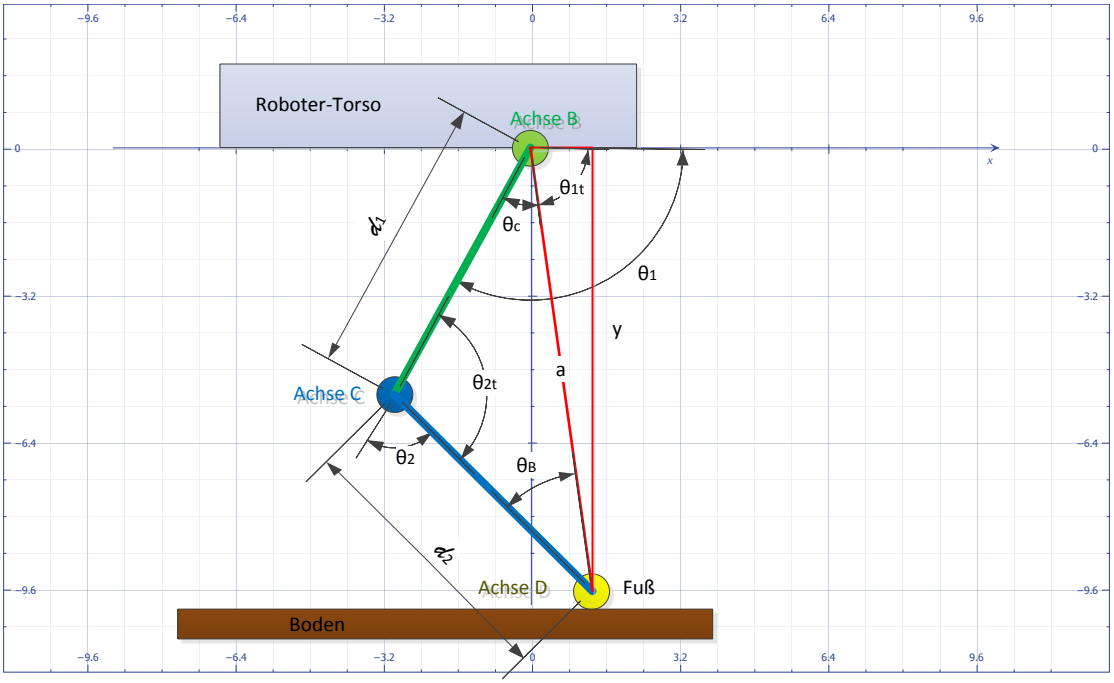

Abbildung 5-33 Berechnung der Position y

In dieser Herleitung wird analog zur IK (5.3.1) gearbeitet, es gilt auch diese Normierung für Winkel und Koordinaten. Aus diesem Grund werden nur kurz die Gleichungen aufgezeigt.

Berechnung der Koordinate Y

Gesucht wird die Koordinate y. Als erstes wird das Dreieck Achse B, C, D betrachtet. Gesucht wird dazu die Länge *a* in *Abbildung 5-33*. Die Länge *a* wird berechnet mit:

$$a = \sqrt{-2 \cdot d_1 \cdot d_2 \cdot \cos(\pi - \theta_2) + {d_1}^2 + {d_2}^2}.$$

Der Winkel θ_B wird benötigt, um den Winkel der Drehung des Dreiecks BCD zu erhalten:

$$\theta_B = \cos^{-1}\left(\frac{a^2 - {d_1}^2 + {d_2}^2}{2 \cdot a \cdot d_2}\right)$$

Der Winkel θ_c ergibt sich damit aus $\theta_c = \pi - (\pi - \theta_c) - \theta_B$.

Als Hilfsvariable wird noch $\theta_{1t} = \theta_1 - \theta_c$ eingeführt. Damit kann folgender Algorithmus angewendet werden:

$$x = \begin{cases} 0 \; für \; \theta_{1t} = \frac{\pi}{2} \\ a \cdot \sin\left(\left(\frac{\pi}{2}\right) - (\theta_{1t})\right) für \; \theta_{1t} < \frac{\pi}{2} \\ (a \cdot \sin(\left(\frac{\pi}{2}\right) - (\pi - \theta_{1t}))) \cdot (-1) \; für \;\; \theta_{1t} > \frac{\pi}{2} \end{cases} \quad , \quad y = \begin{cases} a \; für \; \theta_{1t} = \frac{\pi}{2} \\ a \cdot \sin(\theta_{1t}) \; für \; \theta_{1t} < \frac{\pi}{2} \\ a \cdot \sin(\pi - \theta_{1t}) \; für \;\; \theta_{1t} > \frac{\pi}{2} \end{cases}$$

Der Versatz der Koordinaten *y* kann einfach mit $y = \begin{cases} y - \epsilon_{y1} - \epsilon_{y2} \; für \; y < 0 \\ y + \epsilon_{y1} - \epsilon_{y2} \; für \; y \geq 0 \end{cases}$ berechnet werden.

Siehe hierzu *Abbildung 5-32*.

Berechnung der Koordinate Z

Von der Seite betrachtet ist der Versatz der Koordinate X abhängig vom Winkel der Drehung um Achse A (θ_0, siehe *Abbildung 5-2*). Um mit einem rechtwinkeligen Dreieck zu arbeiten, wird erst die Koordinate Z berechnet. Hierzu wird die X Koordinate ohne Versatz verwendet. $z = (x \cdot \sin(\theta_0))$ Der Versatz der Z Koordinate durch die Fußlänge wird dann mit $z = z + \epsilon_{x1} \cdot \sin(\theta_0)$ berechnet. Da die Vorzeichen von negativen Winkeln sich hier durchsetzen, ist keine Fallunterscheidung nötig. Diese Gleichung ist nur für dieses System gültig, da der Fuß *immer* nach vorn (X+) zeigt.

Drehung und Korrektur der Koordinate X

Analog zur IK muss auch die Koordinate X mit dem Winkel der Achse A korrigiert werden.

$$x = x \cdot \sin\left(\left(\frac{\pi}{2}\right) - \theta_0\right)$$

Nun wird die X Koordinate um die Fußlänge korrigiert. $x = x + \epsilon_{x1} \cdot \sin\left(\left(\frac{\pi}{2}\right) - \theta_0\right)$.

Damit ist die Berechnung abgeschlossen. Im Detail wird beim Funktionsaufruf ein Pointer auf ein Array mit drei Elementen übergeben. Die Funktion füllt dieses Array mit den Koordinaten. Es werden immer die aktuell anlegenden Winkel an den Achsen berechnet.

5.3.4.4.10 Funktion CalcMinTorsoDistance

Im Kontroller „*DoStep*“ (5.3.4.4.3) wird das Bein so nah wie möglich an den Roboter Torso gezogen, um die maximal mögliche Bodenfreiheit zu erhalten. Dies ist in extrem unebenem Gelände nötig. Dieser Funktion wird ein Pointer auf Koordinaten und ein Pointer auf den Winkel der Achse B, in ADC Werten, übergeben. Diese Koordinaten werden je nach Situation vom „*DoStep*“ Kontroller bestimmt. Für den Winkel wird nicht in jeder Situation der aktuelle anliegende Winkel B genutzt. Siehe „*DoStep*“ (5.3.4.4.3).

Nach Eintritt in die Funktion „*calcMinTorsoDistance*“ wird der ADC-Wert in einen Winkel, wie in der Funktion „*ADC2Point*“ 5.3.4.4.9, überführt. Um die Distanz zwischen Achse C (Knie *Abbildung 5-4*) und dem Roboter-Torso zu ermitteln, wird das rechtwinklige Dreieck mit den Winkeln B und C betrachtet.

Die Distanz wird berechnet mit: $distance = \begin{cases} d_1 \cdot \sin(\pi - \theta_2) \; f\ddot{u}r \; \theta_2 > \frac{\pi}{2} \\ d_1 \cdot \sin(\theta_2) \; f\ddot{u}r \; \theta_2 \leq \frac{\pi}{2} \end{cases}$

Jetzt wird der Wert *distance,* um den Versatz der Mechanik korrigiert, da vom Achsmittelpunkt gerechnet wurde. Die Korrektur der *y* Koordinate wird mit $y = \begin{cases} y + distance - \epsilon_{ActorY} \; f\ddot{u}r \; y < 0 \\ y - distance + \epsilon_{ActorY} \; f\ddot{u}r \; y \geq 0 \end{cases}$ berechnet.

Der Wert ϵ_{ActorY} ist die Dicke des Aktors B (Oberschenkel) minus dem Versatz der Achse C zur hinteren Außenkante. Dies bezieht sich auf die Seitenansicht.

Die Koordinate Y ist jetzt manipuliert.

Werden Beispielsweise die zurzeit aktuellen Koordinaten mit dieser Funktion manipuliert, wird ein Punkt senkrecht über der aktuellen Position zurückgegeben. Dieser Punkt ist so nah wie mechanisch möglich am Roboter Torso.

5.3.4.4.11 Kontroller TouchCtrl.

Dieser Kontroller manipuliert die Koordinaten solange, bis ein sicherer Stand des Fußes gefunden wurde. Eingesetzt wird diese Funktion in Situationen, in denen eine Koordinate mit Bodenkontakt gefordert wird. Die „*TouchCtrl*“ befindet sich noch im *Experimentierstadium*.

Der Kontroller „*TouchCtrl*.“ wird immer aus dem Super-Loop aufgerufen. In diesen Kontroller kann nur alle 10ms eingetreten werden[30]. Dies wird über einen Timer realisiert. Es wird der Timer des „*Com-Ctrl.*“(5.3.4.4.12) Kontrollers genutzt. Der „*TouchCtrl*.“ Kontroller befindet sich solange im Zustand „*idle*“ bis er von der „*MotionCtrl*“ (5.3.4.4.2) in den Zustand „*start*“ versetzt wird. Es wird der Zustand der Fußkontakte überwacht. Jede Einzelbewegung wird über die Funktion „*Move2Point*“ (5.3.4.4.5) realisiert.

Zustand „idle“:

Hier wird indirekt auf den „start“ Zustand gewartet.

Zustand „start“:

Es werden zwei Fälle überwacht.

Erstens: Die „*TouchCtrl.*“ beobachtet den Zustand der „*RotationCtrl.*“ (5.3.4.3.1). Ist die Bewegung beendet worden, ohne dass die Fußkontakte ausgelöst wurden, wird der Kontroller in den Zustand „correction“ gesetzt. In der Realität bedeutet dieser Zustand, dass die Koordinate zu hoch war, um den Boden zu erreichen.

Zweitens: Bei einer Bewegung wird irgendein Fußkontakt ausgelöst.

Nun wird geprüft, ob ein sicherer Stand erreicht wurde. Sind mindestens drei Fußkontakte ausgelöst worden, werden die aktuellen Winkel, der Winkelsensoren, an die „*RotationCtrl*.“ (5.3.4.3.1) als Zielwinkel gesetzt. Durch das Überschreiben der Zielwinkel in der Datenstruktur(5.3.4.2) des Control-Layers wird die Bewegung normal beendet. Da drei Fußkontakte ausgelöst wurden, kann davon ausgegangen werden, dass der Fuß genügend halt findet. Bauartbedingt muss der Fuß damit sicher stehen. Der Kontroller wird in den Zustand „*idle*“ gesetzt. Die Rückgabewerte der Funktion „*ADC2Point*“(5.3.4.4.9) wird an den externen Hauptkontroller übermittelt.

Werden weniger als drei Fußkontakte ausgelöst, wird der Kontroller in den Zustand „unsafe“ gesetzt.

Zustand correction:

Der Fuß hat noch keinen Bodenkontakt. Es wird eine neue Bewegung eingeleitet. Die alten Zielkoordinaten werden um 100mm* nach unten korrigiert (Y-Achse). Der Kontroller wird in den Zustand „start“ gesetzt. Dies wird solange wiederholt bis der maximale Aktionsradius des Beins erreicht wurde. Da die Koordinaten immer mit der Funktion „*CalcAngle*“(5.3.4.4.8) berechnet werden, erfolgt nach dem Überschreiten des maximalen Aktionsradius, eine Meldung an den externen Hauptkontroller. Damit werden die Koordinaten von diesem System ignoriert. Weitere Versuche werden abgebrochen. Tritt dies real ein, steht der Roboter an einem Abgrund, oder er versucht sein Fuß in ein tiefes Loch zu setzten. In beiden Fällen ist es besser, das weitere Verhalten im externen Hauptkontroller zu bearbeiten.

Zustand unsafe:

Es wird hier unterschieden welche Fußkontakte ausgelöst wurden.

[30] Die Zehen am Fuß sind weich gelagert. Es tritt immer eine kleine Verzögerung ein, bis die Gummipuffer(3.3.3) nachgegeben haben und der Kontakt im Fußschalter geschlossen wurde. Aus diesem Grund werden die Fußschalter maximal alle 10ms geprüft. Eine schnellere Prüfung führt zu fehlerhaften Entscheidungen des Kontrollers. Die Taster sind nicht entprellt.

Wurde *ein* Fußkontakt ausgelöst, wird die Zielkoordinate um 25mm* tiefer gesetzt. Der Fuß wird dabei um 30° Winkelgrad* gedreht. Die Drehrichtung ist davon abhängig, ob der Kontakt vor oder hinter dem Fußmittelpunkt ausgelöst wurde. Im Detail werden mit der Funktion *„CalcAngle“* (5.3.4.4.8) die Winkel der neuen Zielkoordinate berechnet. Der Winkel des Fußes (Achse D) wird mit 30° überschreiben, da die inverse Kinematik, den Fuß, parallel zum Roboter-Torso berechnet. Der schon ausgelöste Fußkontakt wird temporär gespeichert. Der Kontroller wird in den Zustand „*unsafemove*“ gesetzt.

Wurden *zwei* Fußkontakte ausgelöst, wird unterschieden, wo diese liegen. Liegt ein Fußkontakt vor dem Fußmittelpunkt und der andere dahinter(betrachtet in der Seitenansicht), kann eigentlich kein sicherer Halt hergestellt werden. In der Realität kann es aber durchaus vorkommen, das nur einige Millimeter fehlen, um noch einen dritten Fußkontakt auszulösen. Aus diesem Grund wird die Zielkoordinate trotzdem um 10mm* tiefer gesetzt. Es wird noch einmal Kraft auf den Fuß ausgeübt. Der Zustand des Kontrollers wird auf „*correctionFail*“ gesetzt.

Wurden zwei Fußkontakte ausgelöst, die beide vor oder hinter dem Fußmittelpunkt liegen, wird wie beim Auslösen eines Fußkontakts reagiert. Die schon ausgelösten Fußkontakte werden temporär gespeichert. Der Kontroller wird in den Zustand „*unsafemove*“ gesetzt.

Zustand *unsafemove*:

Es wird geprüft, ob sich die ausgelösten Fußkontakte geändert haben. Verglichen werden die zuvor temporär gespeicherten Fußkontakte und die aktuellen ausgelösten Fußkontakten. Wird die (im Zustand „*unsafe*“) geänderte Koordinate erreicht, ohne dass sich die ausgelösten Fußkontakte geändert haben, wird dieser Versuch abgebrochen. Der Kontroller geht in den Zustand „*fail*“.

Werden mindestens *drei* Fußkontakte ausgelöst, werden die aktuell gemessenen Winkel, der Winkelsensoren, in die Datenstruktur des „*Control-Layers*“(5.3.4.2) als Zielwinkel geschrieben. Die Bewegung wird damit beendet. Der Kontroller wird in den Zustand „*idle*“ gesetzt. Die Rückgabewerte der Funktion „*ADC2Point*“(5.3.4.4.9) wird an den externen Hauptkontroller übermittelt.

Werden auch in diesem Zustand nur zwei Fußkontakte ausgelöst, wird in den Zustand „*fail*“ gewechselt. Eine weitere Bearbeitung würde zu viel Zeit kosten.

Zustand correctionFail:

Es wird geprüft ob mindestens *drei* Fußkontakte ausgelöst wurden. Ist dies der Fall werden die aktuell gemessenen Winkel, an den Achsen, in die Datenstruktur des Control-Layers wieder als Zielwinkel geschrieben[31]. Die Bewegung ist beendet. Der Kontroller geht in den Zustand „*idle*“. Die Rückgabewerte der Funktion „*ADC2Point*“(5.3.4.4.9) wird an den externen Hauptkontroller übermittelt.

Werden weniger als drei Fußkontakte ausgelöst, wird der Kontroller auf Zustand „fail“ gesetzt.

Zustand fail:

Der Zustand „*fail*“ startet den Subkontroller „SearchSafe“. Die Abläufe entsprechen einigen Zuständen des Kontrollers „*DoStep*“(5.3.4.4.3). Deshalb wird an dieser Stelle nur der Ablauf ohne Details erläutert. Dieser Subkontroller „*SearchSafe*“ befindet sich noch in der Testphase und ist für die Weiterentwicklung vorgesehen.

Beim Eintritt wird ein Zähler „countFail“ inkrementiert. Der Fuß wird 50mm* nach oben gezogen. Danach wird der Fuß 10mm* nach hinten (aus der Seitenansicht) gefahren. Der Ablauf der

[31] Damit wird erreicht, dass mögliche Folgebewegungen mit der richtigen Höhe ausgeführt werden.

„*TouchCtrl*.“ wird neu gestartet. Der Fuß wird wieder auf den Boden gesetzt.
Es wird versucht, einen sicheren Stand für den Fuß zu finden. Dieser Ablauf wird drei Mal wiederholt. Wird auch nach dem dritten Versuch keine sichere Position gefunden, erfolgt eine Meldung an den externen Hauptkontroller. Das Bein bleibt auf der letzten Position. Das MCU System bricht alle laufenden Bewegungen ab und wartet auf neue Befehle des externen Hauptkontrollers.
Anmerkung: An dieser Stelle wird die Erörterung des Themas nicht weiter vertieft. Es müssen erst Erkenntnisse über die Leistungsfähigkeit des externen Hauptkontrollers vorliegen. Dieser externe Hauptkontroller existiert zurzeit noch nicht. Einige Tests der „TouchCtrl.“ werden im Anhang (9.3)dargestellt.

*Diese Werte sind geschätzte Werte, wie sie in ersten Versuchen verwendet wurden. Ob diese Werte auch im realen Gelände zufriedenstellende Ergebnisse liefern, muss mit der kompletten Hardware (alle vier Beine) getestet werden. Zurzeit gibt es nur ein Bein für die Tests.

5.3.4.4.12 Kontroller Com-Ctrl.

Der „*Com-Ctrl.*“ Kontroller verarbeitet ein- und ausgehende Nachrichten. Diese Nachrichten werden von der Hardware der seriellen RS232 Schnittstelle in einem 8Bit Schieberegister gepuffert.
Diese Nachrichten werden von der Ethernet-MCU(5.3.4.4.13) vorverarbeitet. Die Nachricht besteht aus einem 8Bit langem Header und mehreren 8Bit langen Datenpaketen. Der Header bildet eine Anweisung ab und es können 256 Anweisungen unterschieden werden. Der „*MotionCtrl*.“(5.3.4.4.2) Kontroller verarbeitet diese Anweisungen. Die Datenpakete werden immer in 8Bit Pakete zerlegt. Um die Transferrate zu erhöhen, ist der maximale Datentyp signed Integer (16Bit). Da das System intern Koordinaten und Winkel mit 64 Bit verarbeitet, werden diese konvertiert. Die Koordinaten stellen reale metrische Dimensionen dar und der Aktionsradius ist mechanisch begrenzt. Die Koordinaten werden mit 10^2 multipliziert. Nur der ganzzahlige Wert, wird verwendet. Damit ist die Genauigkeit, bei dieser Normung, auf zwei Nachkommastellen begrenzt. Durch die Normung der Werte in Millimeter, ist dies völlig ausreichend. Das Maximum von signed Integer wird mit zu erreichenden Koordinaten nicht überschritten. Mit den Winkeln im Bogenmaß wird gleich verfahren. Nur werden diese mit 10^5 multipliziert. Dies ist kleiner als die Auflösung des ADC. Diese 16Bit Werte werden in zwei 8Bit Datenpakete zerlegt.
Der Kontroller wird immer aus dem Super-Loop gestartet. Er verfügt über eine eigne gekapselte Datenstruktur. Der Kontroller ist in zwei asynchrone Automaten aufgeteilt. Ein Input- und einen Output-Automaten.

Input-Automat

Der Input-Automat prüft ein Hardware-Register (RX-Register) der Seriellen Schnittstelle, ob ein vollständiger 8 Bit Wert im Puffer eingetroffen ist. Ist dies der Fall, wird dieser Wert zwischengespeichert. Beim ersten Wert muss es sich um einen Header handeln. Dieser Header verzweigt durch eine Fallunterscheidung, wie in Tabelle (*Tabelle 5-4*) dargestellt. Es wird ein kleiner Unterautomat gestartet, der je nach Headerinhalt weitere Datenpakete verarbeitet. Am Header wird erkannt, wieviele Datenpakte noch folgen und um welchen Datentyp es sich handelt. Ist eine vollständige Anweisung eingetroffen, wird ein Steuerbit gesetzt. Dieses Steuerbit signalisiert der „*MotionCtrl*.“ (5.3.4.4.2), dass eine neue Anweisung vorliegt. Dieses Steuerbit wird zurückgesetzt, wenn die „*MotionCtrl*.“ die Daten lesen will. Trifft eine neue Nachricht ein, bevor die „*MotionCtrl*.“ die Daten gelesen hat, wird die alte Nachricht komplett überschrieben. So ist sichergestellt, dass bei

sich überschneidenden Anweisungen, kein Datenstau entsteht.

Anmerkung: Bei störungsfreiem Betrieb kann kein Datenstau entstehen, da der externe Hauptkontroller immer über den Abschluss eines Auftrags oder über einen Fehler informiert wird.

Die Nachrichten / Anweisung „*stop*“ und „*next*“ (*Tabelle 5-4*) werden immer behandelt. Bei diesen beiden Anweisungen manipuliert der „*Com-Ctrl.*“ Kontroller die Datenstruktur des Logic-Layers direkt. Bei der Anweisung „*stop*“, stoppt der Kontroller selbst die Antriebe und setzt alle Kontroller (auch sich selbst) in den Idle Zustand.

Um einen Stillstand dieses Input Automaten zu verhindern, wird der Timer 0 genutzt. Trifft nach 10ms *kein* neues Datenpaket ein, wird der Input-Automat neu gestartet und alle empfangenen Daten werden damit verworfen. Um die Implementierung zu vereinfachen, wird der Input-Automat auch zurückgesetzt, wenn keine weiteren Datenpakete eintreffen. Damit kann der Timer 0, auch von der „*TouchCtrl*.“ (5.3.4.4.11) genutzt werden.

Output-Automat

Der Output-Automat bietet eine (Software) Schnittstelle über die Funktion „*send*“ an. Beim Aufruf dieser Funktion wird ein 8Bit langer Header übergeben. Es wird geprüft, ob der hardwareseitige Sendepuffer leer ist. Ist dies der Fall, wird genau wie im Input-Automaten verfahren. Bei Koordinaten werden beispielsweise die aktuell an den Sensoren anliegenden Werte der jeweiligen Datenstruktur verwendet. Datenstaus werden durch einen Ringpuffer mit vier 8 Bit Elementen vermieden. Dieser Ringpuffer ist im fehlerfreien Betrieb ausreichend. Jeder Nachricht aus diesem System (MCU-Bein-Kontroller) hängt mit realen Ereignissen zusammen. Es können keine Ereignisse in schneller Abfolge auftreten. Im System gibt es nur den Fall, dass gleichzeitig die Koordinaten des Fußes (TouchCtrl. 5.3.4.4.11) und die Aufforderung zum Synchronisieren gesendet werden.

5.3.4.4.13 Ethernet MCU

Die Ethernet-MCU (5.3.3.2) ist das Hardware-Interface zwischen der Bein-MCU und dem externen Hauptkontroller. Die Software in der Ethernet-MCU ist in zwei Teile getrennt.

Ethernet-MCU zu Bein-MCU

Die Hardware Schnittstelle zwischen Bein-MCU und Ethernet-MCU wird in Abschnitt 5.3.3.2 beschrieben. Die Software der Ethernet-MCU arbeitet mit denselben Verfahren, wie der „*Com-Ctrl.*“ Kontroller (5.3.4.4.12) im Logic-Layer. Die Whitelist zur Filterung der Anweisungen entspricht der Fallunterscheidung per Header (5.3.4.4.12). Anweisungen, die nicht durch diese Fallunterscheidung abgebildet werden, werden ignoriert.

Ethernet-MCU zu Ethernet (Bussystem)

Dieser Teil ist die Schnittstelle zwischen dem Bussystem des Roboters und der Ethernet-MCU. Der verwendete „Hardware TCP/IP Stack“ (4.2.2) bildet das TCP/IP Protokoll in Hardware inkl. des Application Layers ab. Der „HW TCP/IP Stack“ (W3100A) wurde über das „Intel MCU Interface“ verbunden. Dadurch werden die Register und Puffer des W3100A am Adressbus der MCU abgebildet. Die Adressen liegen über dem internen RAM der MCU. Das Verhalten der Register des W3100A entspricht der internen MCU Hardware. Durch die Anbindung des W3100A an den Adressbus kann auf den Puffer direkt zugegriffen werden. Es wird mit voller Busbreite des MCU internen Busses zugegriffen. Bei einer mögliche I^2C Anbindung des W3100A würden alle Daten und Steuer-anweisungen bit-weise ausgetauscht werden. Die Übertragungsrate ist im I^2C Modus um den Faktor vier kleiner als die Anbindung am Adressbus [Wiz06].

Für die Software Schnittstelle zum W3100A wurde die mitgelieferte API [Wiz06] des Herstellers

verwendet. Diese API abstrahiert den Zugriff auf den W3100A auf wenige generische Funktionen für das Senden und Empfangen von Nachrichten per TCP/IP.
Im Bussystem des Roboters werden die Nachrichten per HTTP übertragen. Würden die Nachrichten per UDP übertragen werden, müsste ein eigenes Verfahren implementiert werden, um z.B. defekte Ethernet-Kabel zu erkennen. Die nötige Implementierung für das HTTP-Protokoll ist in der API und Hardware des W3100A gekapselt.
Anmerkung: Die Designentscheidung für das HTTP-Protokoll wurde auch durch einen Hardware Design Fehler im W3100A beeinflusst. Laut Errata des Datenblatts werden nicht alle UDP Nachrichten zuverlässig empfangen. Ein Workaround des Herstellers ist es, den W3100A ständig mit leeren Anweisungen anzusprechen, was sich aber negativ auf die Reaktionszeiten der MCU auswirken würde.
Nachrichten werden im Bussystem des Roboters decodiert übertragen (als Klartext String). Dies ermöglicht eine einfachere Weiterentwicklung. Zudem sind die ausgetauschten Nachrichten kleiner als der interne Puffer des W3100A, wodurch sich keine Geschwindigkeitsnachteile ergeben. Durch den relativ hohen Datendurchsatz von Fast-Ethernet kann die Laufzeit und der Overhead der Daten im Bussystem vernachlässigt werden.
Jede Nachricht besteht aus dem systeminternen Header und den Datenpakten (5.3.4.4.12). Es wird der Doppelpunkt als Trennzeichen verwendet. Eine Nachricht auf dem Bussystem des Roboters hat folgendes Schema: „*http://ip-Adresse/Header: Datenpaket 1: ...: Datenpaket n*“.
Beispielhafter und vereinfachter Ablauf
Vom externen Hauptkontroller wird die Nachricht „http://192.168.0.xxx:3550/Move2Point:X:Y:Z“ gesendet. Der W3100A empfängt diese Nachricht und filtert nach IP Adresse und Port. Treffen die Daten zu, wird die Nachricht durch ARP bestätigt. Der W3100A löst einen Interrupt in der Ethernet-MCU aus. Im Puffer des W3100A liegen die Daten „Move2Point:X:Y:Z“ vor. Der Header wird abgetrennt und mit der Fallunterscheidung, der möglichen Befehle, verglichen (Whitelist). Wird keine Anweisung mit dem Headerinhalt gefunden, wird die Nachricht ignoriert und der Puffer des W3100A gelöscht. Wird ein Fall gefunden, erfolgt die Zerlegung in Datenpakte, äquivalent zum Verfahren im „*Com-Ctrl.“* (5.3.4.4.12). Die Datenpakete werden von der Ethernet-MCU zur Bein-MCU per USART übermittelt.
Während der Übermittlung zur Bein-MCU werden nur die Anweisungen „stop“ und „next“ bearbeitet. Andere Anweisungen überschreiben den Puffer im W3100A, um einen Überlauf oder Datenstau zu vermeiden.
Für das Senden von Nachrichten wird das Verfahren in umgekehrter Reihenfolge angewendet.

5.3.4.5 Security Layer

Hardware Watchdog(5.3.3.4.1): Der HW Watchdog wird pro Super-Loop zurückgesetzt. Er wird auf einen Auslösezeitraum von einer Sekunde gesetzt.
Software Watchdog(5.3.3.4.1): In den SW Watchdog werden alle Kontroller, aus allen Layern, eingehängt. Alle Kontroller haben eine Auslösespielraum von einer Sekunde. Ausgenommen sind hier die In- und Output-Automaten des „*Com-Ctrl.*“ (5.3.4.4.12) Kontrollers. Diese laufen asynchron zum restlichen System. Der SW Watchdog selbst hat einen Auslösezeitraum von 15 Sekunden, da jede Bewegung innerhalb dieser Zeit abgeschlossen sein muss.
Kollisionen(5.3.4.3.3): Kollisionen werden zuerst vom SW-Watchdog behandelt. Tritt der Zustand einer Kollision ein, wird ein Zähler inkrementiert. Das System reagiert mit dem Kollisionsverhalten. Nach einer Sekunde werden die alten Zielkoordinaten wiederholt angefahren. Dieser Vorgang wird

dreimal wiederholt. Tritt beim dritten Versuch wieder eine Kollision auf, wird die Nachricht „*Collision*“ (*Tabelle 5-5*) an den externen Hauptkontroller gesendet und die Aktion abgebrochen. Das komplette System geht in den Zustand idle.
Der SW Watchdog, wird in jedem Durchlauf des Super-Loops aufgerufen.

5.3.5 Implementierung und Tests

In diesem Abschnitt werden Reaktionszeiten gemessen und protokolliert. Betrachtungen der einzelnen Funktionen werden mit dem AVR Simulation des AVR Studio 5[32] gemessen. Um eine fehlerhafte Zeitmessung im Simulator weitestgehend auszuschließen, wurden die ermittelnden Zeiten abgeglichen. Dazu wurden die realen Benchmark-Zeiten aus Abschnitt 5.3.1.11 mit den simulierten Zeiten verglichen. Die ermittelte Differenz liegt im Mittel bei 5%. Diese Differenz kann auf die menschliche Reaktionszeit zurückgeführt werden. Die Simulation des AVR-Studios wird als korrekt betrachtet. Die Bein-MCU wurde in diesem Test mit 18.432 MHz betrieben.

5.3.5.1 Kalibrieren

Damit das System einem ADC-Wert einen Winkel zuordnen kann, muss das System kalibriert werden. Für die Kalibrierung wurde ein Trunk bei der Versionierung erstellt. Diese Version bildet jede Motorfunktion mit Tastern des STK500 ab. Damit kann jeder Motor manuell gefahren werden. Die Daten der Sensoren, werden per RS232 auf ein Hostsystem übertragen. Als erstes wurden die Extreme der Mechanik angefahren. Die resultierenden Werte sind die Vergleichswerte für die Selbstschutzfunktion in der „*Rotation-Ctrl.*“ (5.3.4.3.1, Prüfung auf „OutOfRange“).
Um die Winkel abzugleichen, wurde jeder Aktor in mehrere Winkel verfahren und die ADC-Werte aus den Sensoren erfasst. Für jeden Winkelsensor wurde mit einer linearen Regression eine individuelle Gleichung aufgestellt. Die Faktoren der Geradengleichung (5.3.4.4.8, ADC-Werte) wurden im System abgelegt. Die Winkelsensoren verhalten sich, wie vom Hersteller beschrieben, linear. Die nötigen Winkel wurden mit einem Winkelmesser und einem Lot ermittelt.

5.3.5.2 Initialisierung der Hardware

Nach dem Start des Systems wird die Hardware und die Interrupt Routine initialisiert. Das gesamte System wartet danach 50ms. Der TCP/IP Stack benötigt laut Datenblatt [WIZ08] 10ms, bis er angesprochen werden kann. Der MCU-Beinkontroller wartet 50ms, um genügend Messwerte zu erfassen, siehe Abschnitt 5.3.5.3. In dieser Wartezeit werden keine Anweisungen per Ethernet oder interne Anweisungen (zwischen Bein-MCU und Ethernet-MCU) verarbeitet. Nachrichten in diesem Zeitraum werden ignoriert. Alle Puffer werden nach dieser Wartezeit gelöscht.

5.3.5.3 ADC

Die erfassten Werte der Winkelsensoren müssen gesondert behandelt werden. Werden die Aktoren unter 30% der maximalen Spannung / Drehzahl gefahren, wird jeder 15. – 20. Messwert verfälscht. In der Messreihe sind periodische Ausreißer (Peaks) um ca. 10% nach oben im Messwert reproduzierbar.
Durch die Konstruktion (3.3.3) des Beins, müssen die Messleitungen über die Antriebe B und C verlegt werden. Die Messleitungen sind zusätzlich über die elektrische Masse des Systems

[32] Da die verwendete AVR-Studio Version sich zum Zeitpunkt dieser Arbeit noch im Beta Stadium befindet, wurden die Zeiten des Simulators überprüft.

abgeschirmt. Zudem wurden die Messleitungen getrennt von den Stromversorgungsleitungen der Antriebe verlegt. Bei langsamen Bewegungen der Aktoren wird anscheinend[33] die PWM der Treiber in die Messleitungen induziert. Dieser Effekt ist besonders kritisch, da er immer Auftritt wenn die Antriebe eine Position genau und damit langsam anfahren. Umgangen wird dieses Problem in der Interrupt Routine. Dort wird ein gleitender Mittelwert der erfassten Winkel gebildet. Damit können sich die Peaks in den Messwerten nicht durchsetzen. Auf komplexere Verfahren wurde zugunsten der Laufzeit verzichtet. Im Betrieb wirkt sich das langsame Durchsetzen einer Veränderung nicht negativ auf die Genauigkeit aus. Die Interrupt Routine wird alle 6,9 µs aufgerufen. Da die Interrupt Routine in Folge alle acht ADC Kanäle der MCU abfragt, wird der Einzelwert alle 55,2 µs aufgefrischt. Nach durchschnittlich zehn Durchläufen hat sich der reale Wert durchgesetzt. Dies ist mehr als ausreichend.

5.3.5.4 Reaktionszeiten

In diesem Abschnitt werden einige zeitkritische Routinen dargestellt und am Ende wird eine Reaktionszeit des kompletten Systems ermittelt.

Inverse Kinematik

Um die Laufzeit der inversen Kinematik realistisch zu ermitteln, wird ab dem Aufruf „*Move2Point*“ (5.3.4.4.5) gemessen. Es werden alle hierarchisch untergeordneten Funktionen mit berücksichtigt. Die Zeitmessung endet beim ersten Rücksprung in den Super-Loop. Ohne Unterbrechung der Interrupt Routine werden 706,93 µs benötigt. Im realen Betrieb mit Interrupt Routine werden 895,44µs benötigt, um einen dreidimensionalen Punkt in Winkel bzw. ADC Werte umzurechnen und die Daten in die Datenstruktur zu übertragen. Der Wert ohne Interrupt Routine ist kleiner als das Ergebnis im Benchmark in Kapitel 5.3.1.11. Diese Abweichung ist bedingt durch das Messverfahren des Benchmarks. Dort wurde die Zeit manuell gemessen.

Interrupt Routine

Die Interrupt-Routine wurde mit einem Zähler und mit festen Adressen implementiert. Die Interrupt Routine wird durch die ADC Logik des AVR alle 6,9 µs ausgelöst. Die Interrupt Routine hat eine Laufzeit von 1,84 µs.

Super-Loop

Für einen Super-Loop Durchlauf wird die maximale Reaktionszeit betrachtet. In dieser Betrachtung werden folgende Aktionen durchgeführt. Das System holt eine Nachricht ab und verarbeitet diese. Eine Nachricht wird gesendet. Die „*Rotation Ctrl*“(5.3.4.3.1) befinden sich im Zustand „*decelerating*“. Der SW-Watchdog(5.3.3.4.1) aktualisiert alle Timer. Es wird *kein* neuer Punkt berechnet. Die Laufzeit beträgt 633,98 µs.

Die zweite maximale Reaktionszeit wird gemessen während einer Bewegung auf einer Line. Der „*LineMover*“(5.3.4.4.4) ist aktiv. Die „*Rotation Ctrl*“(5.3.4.3.1) wechselt in den „*BruteForceRotation-*

[33] Warum nur die kurzen Impulse, bei langsamen Bewegungen, die Abschirmung durchschlagen, konnte noch nicht geklärt werden. Durch die PWM sind die Impulse der Leistungselektronik kürzer, aber nicht energetisch höher geladen. Demnach müsste der Effekt immer bei hoher Last an den Antrieben auftreten, was aber nicht der Fall ist.

Ctrl“ (5.3.4.3.2) Modus. Es wird zum Messzeitpunkt eine neue Koordinate berechnet. Die Laufzeit für diesen Wort Case Fall beträgt 1,3328 ms (1332,8 µs). Dieser Fall tritt einmal bei jedem Zwischenschritt (5.3.4.4.7) einer Bewegung auf einer geraden Line ein.

Empfang einer Nachricht

Diese Werte wurden mit einem Intel Atom 330 als externer Hauptkontroller ermittelt. Die Ethernet Kabellänge betrug 10 Meter und wurde über einen Switch geleitet. Das Messsystem im Hauptkontroller sendet das Kommando „*stop*“ an den MCU-Ethernet-Kontroller. Der MCU-Ethernet-Kontroller wurde für diesen Test verändert. Er prüft die empfangene Nachricht anhand der Whitelist (5.3.4.4.13) und sendet als Antwort „*ready*“ zurück. Das Messsystem im externen Hauptkontroller erfasst die Zeit zwischen Absenden der Nachricht bis zum Empfang der Antwort. Diese Zeit wird halbiert. Diese Betrachtung ist eine Annäherung, da Faktoren wie beispielsweise das .net Framework die Zeiten erhöhen. Es entspricht aber dem späteren Einsatz als komplettes System. Die maximale Laufzeit (inklusive Verarbeitung durch den Ethernet-Kontroller) beträgt 334 µs für eine Nachricht. Dieses Maximum gilt für alle Anweisungen aus *Tabelle 5-4* und *Tabelle 5-5*, da diese kürzer sind, als ein TCP Datenpaket und vollständig in den Hardware Puffer des W3100A (4.2.2) passen.

MCU-Ethernet-Kontroller zu MCU-Bein-Kontroller

Die MCU zu MCU Verbindung erfolgt über die Hardware USARTs mit 115.2kBit/Sekunde. Die Zeit für die Übermittlung eines Datenpakets (1 Byte) benötigt 69,4 µs. Bedingt durch die Hardware-USART ist diese Übertragungszeit nebenläufig zu den MCUs. Es kann in jedem Super-Loop Durchlauf genau ein Datenpaket gesendet und empfangen werden. Theoretisch betrachtet, benötigt jedes zehnte Datenpaket zwei Durchläufe des Super-Loops, da es beim Eintritt in den „Com-Ctrl.“ (5.3.4.4.12) noch nicht vollständig in den HW Puffer übertragen wurde.
Im einfachsten Fall (z.B. Anweisung „stop“, „next“, *Tabelle 5-4*) kann das System, in einem Super-Loop Durchlauf darauf reagieren. Im ungünstigsten Fall (z.B. Anweisung „DoStep“ mit 12 Datenpaketen, *Tabelle 5-4*) werden 13 bzw. 14 Super-Loop Durchläufe (8,241 ms) benötigt. Diese langen Anweisungen sind nicht zeitkritisch.

Reaktionszeit des Systems

Die zeitkritische Reaktionszeit des kompletten Systems wird für die Anweisungen „stop“ und „next“ benötigt. Im Fall der „stop“ Anweisung beträgt die Reaktionszeit 967,8 µs, bis die Antriebe stromlos geschaltet werden. Die maximale Reaktionszeit beträgt 1666,8 µs, wenn die „stop“ Anweisung während der Laufzeit der inversen Kinematik eintritt.
Für die Ermittlung der maximalen Reaktionszeit einer Synchronisation wird davon ausgegangen, dass das betrachtete Bein das langsamste ist. Die anderen Beine haben die Bewegung bereits abgeschlossen. Für das Senden der Nachricht „syncReady“ (*Tabelle 5-5*) werden maximal 967,8 µs benötigt. Der Reaktionszeit des externen Hauptkontrollers wird in dieser Betrachtung vernachlässigt. Der Empfang der Anweisung „next“ benötigt wiederum maximal 967,8 µs. Damit beträgt die maximale Reaktionszeit, für eine synchronisierte Bewegung 1935,6 µs pro Zwischenschritt.
Bei zeit*un*kritischen Anweisungen wird als Beispiel die Anweisung „DoStep“(*Tabelle 5-4*) betrachtet. Die maximale Reaktionszeit beträgt 8575 µs, bis die Motortreiber(4.2.3) den ersten Steuerbefehl erhalten, da inklusive Header 13 bzw. 14, 8Bit große Datenpakte übertragen werden.

Alle anderen Funktionen (z.B. Kollisionskontrolle) werden nicht näher betrachtet, da das Konzept (5.3.3.1) die Bearbeitung in maximal einem Super-Loop Durchlauf garantiert.

Anmerkung zu den maximalen Reaktionszeiten:

Die obige Betrachtung ist in einigen Punkten theoretisch und stellt die maximale Reaktionszeit dar. Im Betrieb des Systems verkürzen sich einige Zeiten, da beispielsweise das Senden und Empfangen von Nachrichten asynchron und nebenläufig zu den MCUs erfolgt.

5.3.5.5 Mechanik

Toleranzen

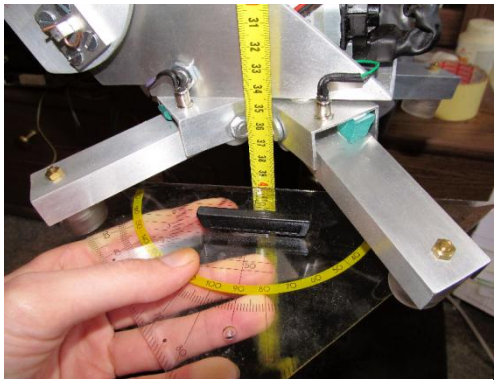

Abbildung 5-34 Position X=0 ,Y=-400, Z=0

Die Toleranzen in Kombination von Hard- und Software werden mit erhöhtem Spiel[34] der Aktoren gemessen. Dies simuliert einen Verschleiß der Antriebe bei hoher Laufleistung. Aus diesem Grund sind auch Zeitbetrachtungen der Mechanik relativ zum Spiel und werden nicht betrachtet.

Es wird die Position des Endeffektors betrachtet. Das heißt in der Draufsicht, der Mittelpunkt in X und Z-Richtung (*Abbildung 5-2*) des Fußes. In der Seitenansicht wird die Unterkante der belasteten Zehen, bei Druck-kontaktschluss (*Abbildung 3-12 Fußkontakte*), betrachtet. Die Orientierung des Koordinatensystems wird wie in Kapitel 5.3.1 beibehalten.

In Y-Richtung weicht ein angesteuerter Punkt um ±4mm ab. In der X-Richtung weichen die Koordinaten um maximal ±8mm ab. In Z-Richtung wird eine Genauigkeit von ±2mm erreicht, da in Z-Richtung keine Kräfte auf dem Bein lasten. Die Messungen wurden am frei hängenden Bein vorgenommen. Im belasteten Zustand des Beins, ergeben sich ähnliche Werte[35].

Antriebe

Trotz einer Untersetzung der Antriebe von 210:1 laufen die Antriebe bei einem langen Hebelarm zurück. Besonders betroffen ist der Antrieb B, da hier der ganze Hebelarm anliegt. Durch die Motortreiber Anweisung „hold“ (5.3.4.1) konnte dies kompensiert werden. Mehr dazu im Fazit (6.4).

Verhalten des Laufsystems

Für diese Arbeit stand aus Kostengründen nur ein Bein zur Verfügung. Um das Laufsystem zu testen, wurden die anderen Beine simuliert. Als Simulation wurde ein provisorischer externer Haupt-kontroller implementiert, der mit den oben gemessenen Reaktionszeiten antwortet. Die Reaktionszeiten wurden um ±10% mit einer Random Funktion variiert, damit ein annähernd realistisches Verhalten erzeugen wird. Es wurde das Verhalten des realen Beins beobachtet.

Trotz der langen Reaktionszeit bei synchronisierten Bewegungen (synchrone Bewegung auf einer geraden Line), ergaben sich nur kurze Verzögerungen. Die Bewegung kann aber nicht als vollkommen flüssig / gleitend bezeichnet werden. Ob sich dieses Verhalten störend auf eine optische

[34] Über die gesamte Länge des Beins, ergibt sich ein Spiel von ±55mm in X-Richtung.

[35] Der belastete Zustand bedeutet, dass das Bein frei auf dem Boden steht. Da ein Bein mit angedeutetem Torso kopflastig ist, musste es per Hand stabilisiert werden. Eine genaue Messung gestaltet sich dadurch schwierig.

Objekterkennung auswirkt, muss in der zukünftigen Weiterentwicklung ermittelt werden. Die Hauptursache sind die Verzögerungen durch die Synchronisationsnachrichten und die Laufzeit der inversen Kinematik. Der Abstand der Zwischenpunkten (5.3.4.4.7) kann auf 3mm reduziert werden. Unter 3mm kommt es zum sichtbaren Überfahren der Zielpunkte, da der mechanische Ablauf kürzer ist als die Reaktionszeit des Systems. Die inverse Kinematik und die Synchronisation haben eine höhere Laufzeit als die mechanische Bewegung von Zwischenpunkt zu Zwischenpunkt. Wird die Taktfrequenz der Bein-MCU auf 16 MHz abgesenkt, kann nur noch ein Abstand der Zwischenpunkte von 5mm als annähernd gleitend bezeichnet werden. Ab einem Abstand von 6mm sind leichte Bögen im Bewegungsablauf zu erkennen. Der Abstand von 6mm ist die maximale Schranke, bei dem eine gleitende Bewegung des Roboters möglich sein wird. Die Folgen daraus werden im Fazit (6) erörtert.

Kollisionserkennung

Die Kollisionskontrolle reagiert zuverlässig auf starre Gegenstände mit einer Masse von ca. 3kg. Durch die Wiederholung (5.3.4.5) der Aktion könne Gegenstände bis ca. 5kg beseitigt werden.
Weiterhin wurden Klemmtests mit Rundhölzern in verschiedenen Radien unternommen. Dazu wurde das Bein in die Parkposition in *Abbildung 3-11* gefahren und die Rundhölzer zwischen Aktor B und C geklemmt. Auf Rundhölzer bis 4mm Durchmesser wurde nicht reagiert und sie wurden zerbrochen. Ab ca. 5mm wird das Rundholz, durch die Wiederholung der Aktion wieder freigegeben. Das Laufsystem könnte sich damit im realen Umfeld gegen mittlere Störeinflüsse durchsetzten.

Abbildung 5-35 Kollisionstest

Bodenerkennung

Die Bodenerkennung („*Touch Ctrl.*“, 5.3.4.4.11) arbeitet unter Laborbedingungen zuverlässig. Auf realen unebenen Untergründen entstehen Situationen, die noch nicht korrekt erkannt werden. Wird die Fußmitte genau auf ein spitzes Objekt platziert, erfolgt keine Erkennung, da kein Fußkontakt ausgelöst wurde. Das Bein würde den Roboter bis zur maximalen Beinlänge anheben. Die Weiterentwicklung wird hinten angestellt bis geklärt ist, wie Leistungsfähig die optische Objekterkennung des kompletten Roboters sein wird. Eine andere Möglichkeit wäre, einen mittleren Fußkontakt zu montieren oder einen Abstandssensor im Fuß, wie beim Projekt ARAMIES (1.5.3). Weiterhin können auch die Drehmomentsensoren für die Bodenerkennung mit genutzt werden, wie beim Roboter ASIMO (1.5.1). Dies wird in einer späteren Version realisiert.

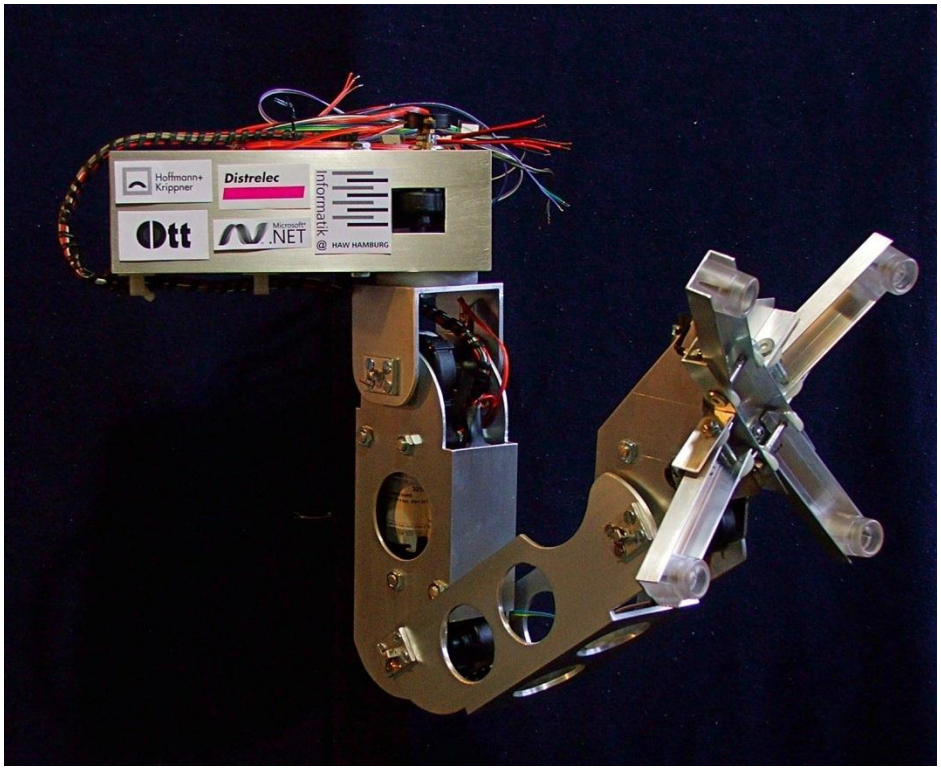

Abbildung 5-36 realisierter Fuß mit Tastern

Belastungstests

Die Implementierung wurde ca. 30 Stunden fehlerfrei betrieben. Das System hat Belastungstests mit ca. 1.000 Nachrichten pro Sekunde über 30 Minuten bestanden. Es wurden einige mechanische Defekte getestet und bestanden. Das System reagierte wie geplant auf Kabelbrüche.

5.4 Hauptkontroller

Das Projekt AMEE vereint viele Teilentwicklungen. Das Laufsystem ist nur ein kleiner Teil des Projektes. Damit auch diese Arbeit. Eine andere Arbeit zu diesem Thema, ist die Bachelorarbeit von Björn Bettzüche [Bet10]. In dieser Arbeit wurde die Verwendbarkeit des Microsoft Robotics Development Studio® (MS-RDS) mit Erfolg geprüft. Als Schlussfolgerung der Arbeit, wird (MS-RDS) für das Projekt AMEE eingesetzt. Diese Arbeit schließt an die Arbeit von Björn Bettzüche an. Der Hauptkontroller wird aus diesem Grund, hier nur skizziert.

5.4.1 Skizzierung des Hauptkontrollers

Egal wie der Hauptkontroller auch in der Weiterentwicklung modelliert wird, der Roboter muss einem Pfad folgen. Der Pfad wird als natürlicher Spline abgebildet, da der Roboter nur sehr langsam um rechte Winkel gedreht werden kann. Der Spline wird in Abschnitte zerlegt. Jeder Abschnitt hat die maximale Schrittlänge des Laufsystems. Diese Abschnitte werden in Punkte zerlegt, die alternierend rechts und links neben dem Spline liegen und die Anfangspunkte für die Schritte des Roboters darstellen. Aus Sicht des Laufsystems, ist der Hauptkontroller der Koordinator für synchronisierte Bewegungen. Es werden die Anfangspunkte für jeden Schritt (5.3.4.4.3) an das jeweilige Bein übermittelt (5.3.4.4.5). Die Bewegungen, bei dem die Füße auf dem Boden stehen, werden mit den beschriebenen Verfahren (5.3.3.5) synchronisiert. Der Roboter bewegt sich entlang des Splines.

Der Hauptkontroller muss auch entscheiden, welcher Laufmodus (2.5.1, 2.5.2) angewendet wird. Das Grundkonzept sieht vor, das der Roboter im „Fast Static Walk“ (2.5.2) beginnt. Treten mehr als drei Kollisionen pro 10 Schritten auf, wird in den Laufmodus „Save Walk“ (2.5.1) gewechselt. Ob die Objekterkennung des Hauptkontrollers dies frühzeitig erkennen kann, muss in der Weiterentwicklung ermittelt werden.

Die Entwicklung des Hauptkontrollers soll in weiteren Arbeiten des AMEE Teams fortgesetzt werden. Diese Arbeit endet mit seinen Betrachtungen an diesem Punkt.

6 Fazit

In diesem Kapitel wird das gesamte vierbeinige USAR-Laufsystem zusammengefasst betrachtet. Das Unterkapitel 6.1 fasst die Entwicklung kurz zusammen. Eine abschließende Bewertung findet im Kapitel 6.2 satt. In den Abschnitten 6.3 und 6.4 werden einige Punkte für die Weiterentwicklung aufgezeigt. Abschließend (6.5) werden kurz die Aussichten und das weitere Vorgehen für das Projekt AMEE vorgestellt.

6.1 Zusammenfassung

In dieser Arbeit wurde ein Roboter USAR-Laufsystem von den Grundzügen bis zur teilweisen Realisierung diskutiert. Es wurde ermittelt, dass ein statisches Laufsystem(2) für ein USAR-Szenario geeignet ist. Durch eine intensive Auseinandersetzung mit anderen Laufsystemen konnte auch ermittelt werden, dass ein dynamischer Läufer in fast allen Bereichen zu aufwendig und kostenintensiv für eine Low-Cost-Lösung wird.

Für die Konstruktion(3) der Beine wurde ein Simulationsmodell(3.1) des Laufsystems erstellt und schrittweise mit den ermittelten Komponenten konkretisiert. In Wechselwirkung mit der Simulation wurde ein Bein des Laufsystems real gefertigt(3.3.3). Durch die Auswahl von geeigneten Komponenten, wie beispielsweise die Winkelsensoren und die Antriebe, ist das Resultat der Konstruktion ein kompaktes und sehr kräftiges Bein. Zudem wurden einige grundlegende Konstruktionsdetails und Spezifikationen des kompletten Roboters ermittelt.

Der Aufbau der elektronischen Hardware(4) legt Details für ein dezentrales Laufsystem fest. Es wurde als verteiltes System konzipiert und teilt es in Bein-Subprozessoren auf (4.1). Hierbei waren die grundlegenden Erfahrungen des HRI (Honda Research Instituts) hilfreich. Für die Beine wurde eine Platine (4.2.5.3) entwickelt, die die Anbindung an das Ethernet-Bussystem des Roboters ermöglichen und zudem genügend Rechenleistung für ein rudimentäres, autonomes Verhalten jedes Beins bereitstellt.

Eng verknüpft mit der Hardware ist die Softwareentwicklung (5) des Laufsystems. Um die nötige Performance auf einer kostengünstigen MCU zu erzielen, wurde eine leicht wartbare inverse Kinematik entwickelt (5.3.1). Die gesamte Modellierung wurde den Einschränkungen einer 8-Bit MCU unterworfen (5.3.3.1). Das Softwaresystem wurde nach Abstraktionsgrad der Hardware in Schichten aufgeteilt (5.3.3.3). Die untere Schicht kapselt die Zugriffe auf die Hardware, enthält die Motorsteuerung und die Sensorerfassung(5.3.4.1). In der zweiten Schicht werden die Mechanik und die physikalischen Bedingungen gekapselt (5.3.4.3). Das ausführende Beinregelsystem wurde hier implementiert und eine integrierte Kollisionskontrolle setzt Konzepte der Anti-Pitch-Detection um (5.3.4.3.3). In der dritten Schicht wurden rudimentäre Verhaltensweisen implementiert (5.3.4.4). Hier werden komplexe Bewegungsabläufe koordiniert und in Einzelbewegungen zerlegt. Das Synchronisationssystem (5.3.3.5) der Beine wurde auch in die dritte Schicht integriert. Zudem wurde ein Sicherheitssystem entwickelt, das den gesamten Ablauf zeitlich überwacht und bei anormalem Verhalten und Defekten eingreift (5.3.3.4.1). Um die Betriebssicherheit im Störungsfall zu erhöhen, wurde eine separate MCU für die Kommunikationen mit dem Bussystem in das System integriert (5.3.3.2). Damit können maximale, garantierte Reaktionszeiten für das gesamte System angegeben werden (5.3.5.4). Im ganzen System werden keine komplexen Verfahren verwendet. In der Entwicklung befindet sich ein System, das es den Beinen ermöglicht, selbständig einen sicheren Stand auf dem Boden zu „ertasten“ (5.3.4.4.11). Dieses System wurde im realen Gelände getestet (9.3).

Abschließend wurde der externe Hauptkontroller skizziert, da dieser durch die Dezentralisierung des Laufsystems nur noch als einfacher Koordinator für die Synchronisation dient (5.4).

Es wurde ein funktionsfähiges Bein realisiert, das in der Weiterentwicklung des Laufsystems als Basis dient. Das System kann eine verschlissene Mechanik ausgleichen und die Kollisionskontrolle arbeitet zuverlässig. Die Kombination aus Schichten- und Sicherheitskonzept ergibt eine sehr robuste Lösung, die selbst absichtlich provozierte Implementierungsfehler kompensiert. Das System ist fehlertolerant und selbstkorrigierend. Es schützt sich weitestgehend selbst vor fehlerhaften Anweisungen. Es werden einige Defekte erkannt und darauf reagiert. Das System kann aber noch nicht auf alle Defekte richtig reagieren, was bedingt durch die vielen Kombinationsmöglichkeiten sehr aufwendig ist. Durch das Konzept der hierarchischen Automaten konnten die Problemstellungen erfolgreich in einzelne Automaten aufgeteilt und implementiert werden.
Mit der Aufteilung des Beinsystems in Ethernet- und Regelsystem konnte eine Entkoppelung von Realtime-Anforderungen und ereignisgetriebenen Interrupts erzielt werden.
Die Realisierung einer inversen Kinematik auf einer MCU mit integrierten Subsystemen ist erfolgreich. Der Ansatz, eine IK analytisch herzuleiten, ist aber nur unter den dargestellten Bedingungen sinnvoll. Sollen mehr Glieder gesteuert werden, sind die Herleitung und auch der resultierende Algorithmus aufwendiger als andere Verfahren.
Dieses System arbeitet mit einfachen Verfahren und verzichtet zugunsten der Laufzeit auf komplexe Verfahren.

6.2 Bewertung

Die Entwicklung, ein dezentral gesteuertes Laufsystem mit einfachen Mitteln und Algorithmen zu realisieren, kann bedingt als Erfolg bezeichnet werden. Unter *Laborbedingungen* ist die angestrebte Funktion gegeben. Durch die Simulation des Laufsystems mit nur einem realen Bein kann *keine stichhaltige Aussage* über die Funktion im *realen Gelände* getroffen werden.
Abschließend betrachtet hat diese Arbeit ihr Ziel erreicht und eine Teilentwicklung des Projekts AMEE vorangetrieben. *„Die Zielsetzung ist es, Erkenntnisse zu sammeln, um einen Roboter zu realisieren, der für einen realen Einsatz genutzt werden kann.“* (1.4)
Es wurden genügend Erkenntnisse gesammelt, um mit den folgenden Verbesserungen ein in der Realität einsatzfähiges Laufsystem für ein USAR Szenario zu realisieren.

6.3 Implementierung

Wird das gesamte System und der Zusammenhang aus Absatz (5.3.5.5, Verhalten des Laufsystems) betrachtet, ergibt dies die Schlussfolgerung: *Je höher die Rechenleistung der MCU ist, desto gleitender und ruhiger werden die Bewegungen.* Mit steigender Rechenleistung können auch die Zwischenschritte immer kleiner gewählt werden. Dies wird bei diesem System nur durch die Laufzeit der Synchronisationsnachrichten begrenzt. Das System liefert unter 16 MIPS (ATMEGA 128: MHz = MIPS [ATM101]) keine gleitenden Bewegungen mehr. Für diese Implementierung sind 16 MIPS, die untere Schranke. Die Laufzeit der kompletten Implementierung kann noch in vielen Punkten optimiert werden[36].

[36] Eine optimale Laufzeit war nicht der Entwicklungsschwerpunkt. Bedingt durch die geplante Weiterentwicklung des Systems wurden hier Wartbarkeit und Laufzeit abgewogen.

Der Kontroller „*Rotation Ctrl.*“(5.3.4.3.1) muss vereinfacht werden. Dort sollte mit festen Adressen gearbeitet werden. Dafür müsste auch der Aufbau der Datenstruktur (5.3.4.2) verändert werden. Der Kontroller „Touch Ctrl.“ (5.3.4.4.11) muss intensiver in realem Gelände geprüft und weiterentwickelt werden. Die jetzige Lösung kann nur als „Proof of Concept“ bezeichnet werden. Die Interrupt Routine für die Sensordatenerfassung, wird mit zu hoher Frequenz ausgelöst. Ein Auslösezeitraum von 500 µs ist ausreichend. Dafür müsste ein anderes Verfahren implementiert werden, das die Peaks (5.3.5.3) in den Sensordaten schneller filtern kann.
Der Datenaustausch zwischen dem gesamten Beinsystem und dem externen Hauptkontroller, per HTTP ist nicht optimal. Diese Designentscheidung wurde gewählt, weil die Last der Zustellprüfung von Nachrichten *nebenläufig zur MCU* im TCP/IP Hardware Stack erfolgt[37]. Hier muss genauer untersucht werden, ob Nachrichten per UDP und eigener *softwareseitiger* Nachrichtenbestätigung die Laufzeit in der MCU signifikant erhöhen.
Aus Kostengründen wurde auf eine genauere Betrachtung von Realtime-Ethernet verzichtet. Für den geplanten Einsatzzweck würden sich Systeme vom Automobilbaukonzern BMW [Jew07] anbieten, da sie die geforderten Umweltbedingungen erfüllen.

6.4 Mechanik und Hardware

Die Antriebe laufen trotz einer Untersetzung von 210:1 bedingt durch den langen Hebelarm zurück. Mit den jetzt verwendeten Getrieben muss das Bein aktiv mit der „hold“ Funktion (5.3.4.1), stabilisiert werden. Die Folge ist ein höherer Energiebedarf als geplant. Hier sollte mindestens eine Untersetzung von 400:1 verwendet werden. Bei der jetzigen Untersetzung von 210:1 ist auch der Geschwindigkeitsanstieg im Verhältnis zur anliegenden Spannung zu steil. Durch eine größere Untersetzung könnte das System langsamer und genauer die Zwischenschritte anfahren. Die Mindestspannung, um den Totpunkt des Motors zu überwinden, ergibt eine relativ schnelle Bewegung. Bei der jetzigen Untersetzung sind kleine Bewegungen (ca. 1° Winkelgrad) schwer zu kontrollieren. Durch die Massenträgheit des Ankers im Motor läuft dieser trotz „hold“(5.3.4.1) Anweisung zu lang nach. Bei gleichem Motor mit höherer Untersetzung kann der Antrieb schneller gestoppt werden.
Durch die aufgezeigten Weiterentwicklungsmöglichkeiten(6.3) werden die Komplexität und der Aufwand im Bein-Kontroller noch steigen. Beispielsweise muss die „TouchCtrl.“(5.3.4.4.11) noch mehr Fälle erkennen können, was ein komplexeres Verfahren nötig macht. Auch ein verbessertes Filterverfahren in der Sensordatenerfassung (6.3) benötigt wahrscheinlich ein komplexeres Verfahren. Weiterhin ist die Anbindung an das Bussystem des Bein-Kontrollers durch eine extra MCU nicht optimal, da der verwendete HW TCP/IP Stack im Vergleich zur restlichen Elektronik zu kostenintensiv ist. Um die Kosten und die maximalen Reaktionszeiten zu senken, sollte eine leistungsstärkere MCU mit integriertem TCP/IP HW-Stack gewählt werden. Beispielsweise bietet der MCU Typ AT32UC3C0128C[38] der Firma Atmel® die geforderten Leistungsreserven und eine integrierte Ethernet Schnittstelle. *Leider war dieser MCU Typ zu Beginn dieser Arbeit nicht verfügbar.* Aus diesen

[37] Das gesamte HTTP wird in der Hardware des TCP/IP Stack *ohne* MCU Last bearbeitet.
[38] Fast alle kritischen Punkte des verwendeten 8Bit MCU Typs werden mit dieser MCU umgangen. Die Ethernet Schnittstelle kann ohne Interrupt betrieben werden(5.3.3.2). *Persönliche Anmerkung: „Diese MCU wirkt, als wäre sie für dieses Projekt entwickelt worden. (32Bit Bus, FPU, Ethernet, 16 ADC Eingänge, 8 PWM Ausgänge, 66MHz, minimale Beschaltung, geringe Kosten)“*

Gründen wurde die MCU-Platine (4.2.5.3) nicht realisiert. Die im Kapitel 4.2.5 vorgestellte Hardware-Lösung erfüllt die Anforderungen zuverlässig, aber in der Weiterentwicklung wird die Leistungsgrenze der verwendeten MCU erreicht werden.

Die Zehen im Fuß (3.3.1) müssten einen Aktionsradius von mindestens 80° Winkelgrad nach unten erhalten. Es kommt im realen Gelände oft vor, dass ein Zeh auf einem Objekt aufliegt und die anderen Zehen zu weit vom Boden entfernt sind. Dazu müssten die Taster im Fuß durch Drucksensoren ausgetauscht werden. Aufgrund der belegten ADC Eingänge der MCU wurde diese Lösung bei der Realisierung verworfen. Durch den oben vorgeschlagenen MCU Typ wäre dies wieder möglich. Diese Änderung im Design würde auch die Weiterentwicklung des „Touch Ctrl.“(5.3.4.4.11) Kontrollers erleichtern.

6.5 Aussichten und Vision

Es soll die Vision des AMEE Projektes (1.4) realisiert werden. Nach dieser Arbeit werden die Verbesserungen umgesetzt. Es werden alle vier Beine mit neuen Antrieben und MCUs fertiggestellt. Dies ist eine Teamaufgabe und kann kaum von einem isolierten Entwickler geleistet werden. Parallel wird im Team des Projekts AMEE der Hauptkontroller entwickelt. Erst wenn diese Teilaufgaben realisiert wurden, kann das Laufsystem und Teile des Roboters im Gelände getestet werden. Auch könnten alternative Stromversorgungen, wie beispielsweise Brennstoffzellen, getestet werden.

Im Anhang 9.4 wird ein Artikel zitiert, der die Probleme im Massenmarkt für Roboter erläutert.
Demzufolge wird eine Vermarktung auf dem Consumer-Markt durch nicht standardisierte Komponenten behindert. Vielleicht ist der Ansatz in dieser Arbeit, alle Komponenten mit einer Grundlogik auszustatten eine mögliche Lösung des Problems. Mutmaßlich könnte es in der Weiterentwicklung gelingen, die beininterne Software so zu abstrahieren, dass nur noch ein Treiber für den Betrieb benötigt wird. Für die höhere Logik im Roboter ist es dann unerheblich, ob er mit Rädern, vier Beinen oder zwei Ketten ausgestattet ist. Vergleichbar mit einem Desktop-PC könnten die Komponenten nach Budget und Einsatzzweck modular zusammengestellt werden. Aber dies ist eine weit entfernte Vision.

7 Glossar

In diesem Glossar werden Begriffe auch aus dem Maschinenbau kurz erläutert. Diese Erläuterung ist nicht präzise und bezieht sich nur auf die vorliegende Arbeit.

A

anfahren
: In Bezug auf die Aktoren.
1. Eine Position *anfahren*. Den Aktor zum Zielwinkel drehen.
2. Einen Aktor *anfahren*. Der Aktor wird aus dem Stillstand beschleunigt.

G

gefahren
: In Bezug auf Aktoren.
Ein Aktor wird zu einer Position bewegt.

I

ISP
: In System Programming
Schnittstelle zur Programmierung einer MCU.

L

LQFP
: Bauteilgehäusebauform

S

Spiel
: In Bezug auf mechanisches *Spiel.*
Mindestens zwei mechanische Bauteile sitzen nicht fest zusammen. Sie haben *Spiel*.

STK500
: Entwicklungsplatine der Firma ATMEL

T

Totpunkt
: In Bezug auf Aktor, Antieb, Motor.
Die anliegende elektrische Spannung reicht nicht aus um den Aktor zu bewegen.

V

verfahren
: In Bezug auf Aktoren.
Ein Aktor wird von einer Position zu einer anderen Position bewegt.

verspannte Getriebe
: Die Getriebe haben *kein Spiel (siehe Spiel).* Die Zahnräder des Getriebes sind minimal zueinadner verdreht. *Sie sitzen auf (mechanischer) Spannung.*

8 Literaturverzeichnis

[ATM11] **ATMEL and avr-lib Developer. 2011.** *avr-libc-manual.pdf V1.7.1.* [PDF] s.l. : ATMEL, 2011.

[ATM06] **ATMEL Corp. 2006.** *Application Note AVR480 Anti Pitch System for Electrical Window.* [PDF] 2325 Orchard Parkway, San Jose, CA 95131, USA : ATMEL, 2006. 7559B-AVR-12/06.

[ATM101] **—. 2010.** *ATmega128A.* [PDF] 2325 Orchard Parkway, San Jose, CA 95131, USA : Atmel Corporation, 2010. 8151E–AVR–02/10.

[ATM10] **—. 2010.** *AVR042: AVR Hardware Design Considerations.* [PDF] 2325 Orchard Parkway, San Jose, CA 95131 , USA : ATMEL, 2010. 2521J-AVR-06/10.

[ATM02] **—. 2002.** *AVR180: External Brown-out Protection.* [PDF] 2325 Orchard Parkway, San Jose, CA 95131, USA : Atmel Corporation, 2002. 1051B–AVR–05/02.

[Bet10] **Bettzüche, Björn. 2010.** *Machbarkeitsprüfung zur Entwicklung von SW-Anwendungen mit MS-Robotics Developer Studio für das Robocup Rescue Szenario.* [PDF] s.l., Hamburg : HAW Hamburg, Technische Informatik, Juli 2010.

[Bos10] **Boston Dynamics Inc. 2010.** [Internet] 78 Fourth Avenue, Waltham, MA, 02451-7507, US : Boston Dynamics Inc., 2010. http://www.bostondynamics.com.

[Boy04] **Boyd, Stephen und Vandenberghe, Lieven. 2004.** *Convex Optimization – Boyd and Vandenberghe.* Cambridge, UK : Cambridge University Press, 2004. ISBN-13: 9780521833783 | ISBN-10: 0521833787.

[Bra07] **Brall Software GmbH. 2007.** *RN-VNH2 Dualmotor.* [PDF] 36200 Sontra, Germany : Brall Software GmbH, 2007. k.A..

[Bro86] **Brooks, Rodney A. 1986.** *A robust layered control system for a mobile robot.* [PDF] Massachusetts, USA : Journal of IEEE, 1986. ISSN: 0882-4967.

[Rod87] **—. 1987.** *Asynchronous distributed control system for a mobile robot.* [PDF] Massachusetts, USA : MIT, 1987.

[Bro87] **—. 1987.** *Planning is just a way of avoiding figuring out what to do next.* [PDF] Massachusetts, USA : Massachusetts Institute of Technology, 1987.

[Bru08] **Bruno Siciliano, Oussama Khatib (Eds.). 2008.** *Handbook of Robotics.* 80125 Napoli, Italy & CA 94305-9010, USA : Springer Science+Business Media, 2008. e-ISBN: 978-3-540-30301-5.

[Gro11] **Grote, Karl-Heinrich und Feldhusen, Jörg. 2011.** *Dubbel.* Berlin : Springer Verlag, 2011. ISBN 978-3-642-17305-9.

[Hit08] **Hitec RCD USA,LLC. 2008.** *Robonova-I User Manual V1.0.* [PDF] 12115 Paine St., Poway, CA 92064 : Hitec RCD USA,LLC., 2008. English Manual V1.00.

[Ant00] **Hoffman, Antony. 2000.** *Red Planet.* Warner Bros. Pictures, 2000.

[HofkA] **Hoffmann + Krippner GmbH. k.A.** Sensofoil®: Verschleißfreie und kostengünstige Folienpotentiometer als Weg- und Winkelsensoren. *http://www.sensofoil.de/.* [Online] Hoffmann + Krippner GmbH, k.A. k.A. [Zitat vom: 29. Juni 2011.] http://www.sensofoil.de/.

[Hon071] **Honda. 2007.** *ASIMO Technical Information.* [PDF] Japan / Germany : Honda Motor Co., Ltd., Public Relations Division, 2007.

[Hon07] **Honda Motor Co., Ltd. September 2007.** *ASIMO Technical Information.* [PDF] s.l. : Honda Motor Co., Ltd., September 2007. kA.

[Hus07] **Huston, Vince. 2007.** Design Patterns. [Online] 19. Januar 2007. http://www.vincehuston.org/dp.

[Int09] **Intel Corp. 2009.** *D510MO Technical Product Specification.* [PDF] Denver, CO 80217-9808, USA : Intel Corp., 2009. E74523-001US .

[Jan10] **Janssen, Herbert, Principal Scientist. 2010.** Personal Communication Honda ASIMO. [EMail/Phone]. Honda Research Institute Europe GmbH, Carl-Legien-Strasse 30, 63073 Offenbach/Main, Germany : s.n., 2010.

[Jew07] **Jewgenij Botaschanjan, Alexander Gruler, Alexander Harhurin, Leonid Kof,. 2007.** *Towards Modularized Verification of Distributed Time-Triggered Systems.* [PDF] D-85748, Garching bei München, Germany : Institut für Informatik, TU München, BMW Group Research and Technology, 2007.

[Juc10] **Juckett, Ryan, Director of Technology, High Impact Games, Burbank, CA, USA. 2009.** [Online] 11. August 2009. [Zitat vom: 15. Mai 2010.] www.raynjuckett.com.

[McG97] **McGowan, Christopher. 1997.** *The Raptor and the Lamb.* London : Henry Holt & Company, 1997. ISBN: 0805042989, ISBN-13: 9780805042986.

[MCSkA] **MCS Electronics . k.A..** *Easy TCP/IP.* [PDF] De Paal 112, Almere-Haven, HOLLAND, 1351JJ : MCS Electronics , k.A.

[MIA10] **MIAG Fahrzeugbau GmbH. 2010.** Das Antriebskonzept. *MIAG GmbH.* [Online] k.A.. k.A. 2010. [Zitat vom: 10. Juli 2011.] http://www.miag.de/Produkte/OCS/Mecanumrad/mecanumrad.html.

[Mil11] **Miller, Paul. 2011.** Automaton. *IEEE Specturm inside technology.* [Online] IEEE Spectrum, 28. März 2011. [Zitat vom: 21. Juli 2011.] http://spectrum.ieee.org/automaton/robotics/robotics-software/robots-are-the-next-revolution-so-why-isnt-anyone-acting-like-it.

[Nac10] **Nachtigall, Werner. 2010.** *Bionik als Wissenschaft.* Heidelberg : Springer-Verlag, 2010. ISBN 978-3-642-10319-3 .

[Nat11] **National Aeronautics and Space Administration (NASA). 2011.** *MSL Fact Sheet, Mars Science Laboratory.* [PDF] Pasadena, California : Jet Propulsion Laboratory, California Institute of Technology, 2011. JPL 400-1416 3/11.

[NVM09] **NV Melexis SA. 2009.** *MLX90316 DataSheet V5.* [pdf] Rozendaalstraat 12, 8900 Ieper, Belgium : NV Melexis SA, 2009. k.A..

[Ott10] **Ott GmbH & Co. KG. 2010.** *Motor GMPD/404980.* [PDF] Baarstr 3, D-78652 Deisslingen : Ott GmbH & Co. KG, 2010.

[Rob09] **Robotikhardware. 2009.** *RN-VNH2 Dualmotor .* [.pdf] Germany : http://www.robotikhardware.de, 2009. k.A..

[Sch96] **Schneider, Eberhard. 1996.** *Jagdlexikon (Wildbiologie).* München : BLV Verlagsgesellschaft mbH, 1996. ISBN 3-405-15131-7.

[Sch10] **Schröder, Dierk. 2010.** *Intelligente Verfahren, Identifikation und Regelung nichtlinearer Systeme.* München : Springer, 2010. ISBN 978-3-642-11397-0.

[Sic08] **Siciliano, Bruno und Khatib, Oussama. 2008.** *Handbook of Robotics.* Heidelberg 2008 : Springer-Verlag, 2008. ISBN: 978-3-540-23957-4 e-ISBN: 978-3-540-30301-5.

[Spe07] **Spenneberg, Dirk . 2007.** http://robotik.dfki-bremen.de. *ARAMIES.* [Online] 1. April 2007. [Zitat vom: 27. Juni 2011.] http://robotik.dfki-bremen.de/de/forschung/projekte/weltraumrobotik/aramies.html.

[Sup10] **SuperDroid Robots Inc. 2010.** *LT-F Compact Treaded All Terrain Surveillance & Inspection Robot.* [PDF] Fuquay Varina, NC 27526, USA : SuperDroid Robots Inc., 2010. LT-F Data Sheet, Revised February 6, 2010.

[Wet96] **Wettergreen, David. 1996.** *Gait Generation for Orthogonal Legged Robots.* [PDF] Riverside, CA 92505-3410, USA : National Aeronautics and Space Administration (NASA), 1996.

[Wiz06] **Wiznet Inc. 2006.** API for AVR. *www.wiznet.co.kr.* [Online] 14. Dec 2006. [Zitat vom: 12. April 2011.] k.A.. k.A..

[WIZ08] **WIZnet Inc. 2008.** *iinChip W3100A Datasheet v1.34 .* [pdf] Korea : WIZnet Inc., 2008. k.A..

9 Anhang

9.1 Vision in Bildern

Da in der Entwicklung mit einem Simulations-Tool gearbeitet wurde, sind als Nebenprodukt einige Bilder entstanden. Diese Bilder erfüllen keinen wissenschaftlichen Zweck, aber verdeutlichen die Vision des Projektes AMEE.

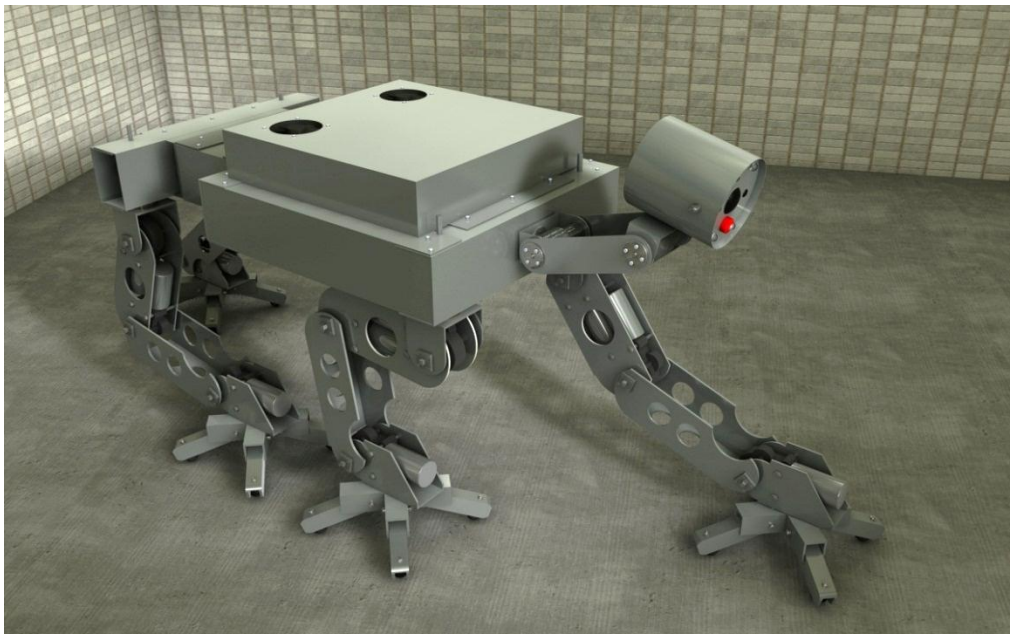

Abbildung 9-1 Gerendertes Bild des AMEE Roboters

Abbildung 9-2 Gerenderter Straßenzug mit Avatar und AMEE

Abbildung 9-3 Gerendertes Bild von AMEE, reales Hintergrundbild

9.2 Evolution der Realisierung

Nachfolgend einige Bilder, in chronologischer Reihenfolge, die die Entstehung des realisierten Beins dokumentieren. Abbildung 9-4 von links nach rechts. Der erste Entwurf. Zweiter Entwurf in der Simulation funktionsfähig. Ein Papiermodell um die Simulation zu prüfen. Das Simulationsmodell mit realen Größen und Antrieben.

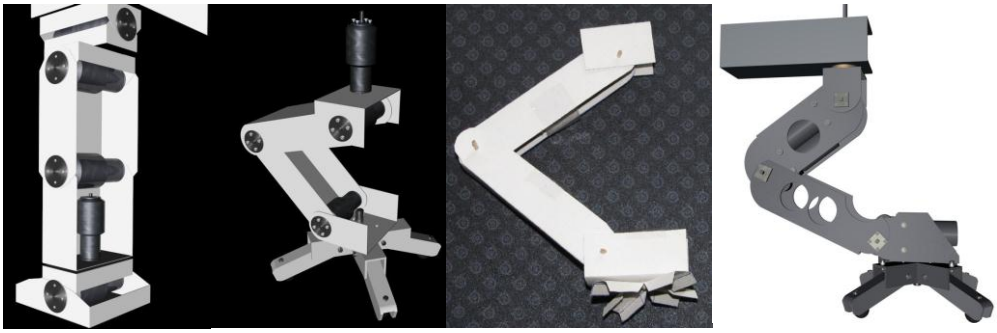

Abbildung 9-4 Konzeptentwicklung von links nach rechts

Der mechanische Aufbau in mehreren Baustufen bis zum 6.07.2011 (rechts)

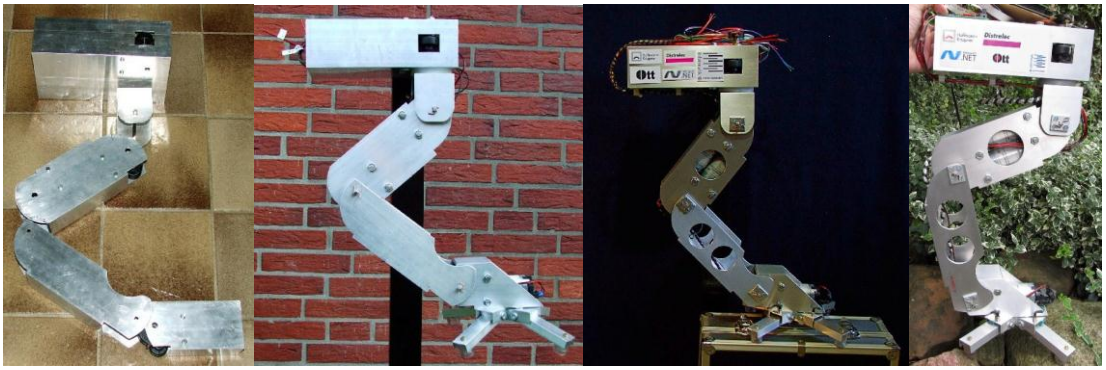

Abbildung 9-5 Realisierung Baustufen

Das Bein zerlegt, Abbildung 9-6. Es werden für eine Low-Cost-Lösung noch zu viele Bauteile benötigt. Für eine komplette Montage werden ca. 2 Stunden benötigt. Mit einem fertigen Kabelbaum erhöht sich die Montagedauer um ca. 40 Minuten, für die Verkabelung.

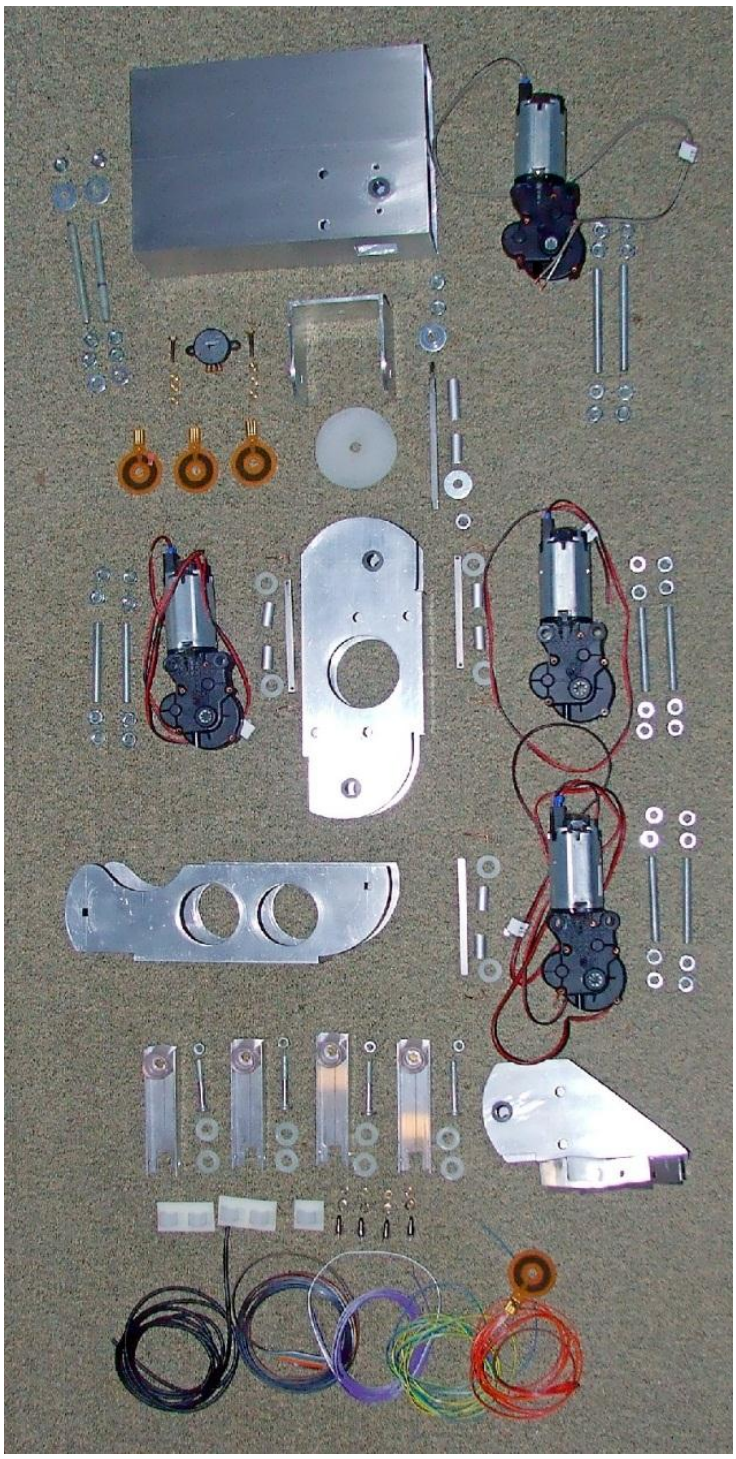

Abbildung 9-6 Bein zerlegt

9.3 Tests des „Touch Ctrl.“ Kontrollers

Nachfolgend einige Bilder die bei den Tests der „*Touch Ctrl.*“ (5.3.4.4.11) entstanden sind. Die Bilder werden mit der jeweiligen Situation erläutert.

Abbildung 9-7 Touch Ctrl Tests 1

In der Abbildung 9-7 (links) beendet die „Touch Ctrl.“ ohne Korrektur mit drei Auflagepunkten. (mittlere Bild) Der Fuß wird nach vorn gekippt bis der vordere (von der Kamera zeigende) Zeh den dritten Kontakt auslöst. (Bild rechts) Es wird nur ein Kontakt ausgelöst, da der Fuß auf dem rötlichen Stein in der Mitte aufliegt und nur ein Kontakt ausgelöst werden kann. Die „*Touch Ctrl.*“ bricht den Versuch ab.

Abbildung 9-8 Touch Ctrl Tests 2

Im Bild (links, Abbildung 9-8) wird der Fuß nach hinten gekippt bis alle vier Kontakte ausgelöst wurden. Im mittleren Bild können nur zwei Auflagepunkte hergestellt werden, da der Fuß nur auf der rechten Seite belastet wird. Im rechten Bild wird ein Gefälle von ca. 25% getestet, der Fuß wird nach vorn gedreht und löst alle vier Kontakte aus.

9.4 Robots Are the Next Revolution, So Why Isn't Anyone Acting Like It?

Das Magazin IEEE Specturm veröffentlichte im März 2011 einen Artikel, der die momentanen Probleme des Robotermarktes beschreibt.

„Robots Are the Next Revolution, So Why Isn't Anyone Acting Like It?

Back in 2006, when Bill Gates was making his tear-filled transition from the PC industry into a tear-filled career as a philanthropist, he penned an editorial on robotics that became a rallying cry for… no one. Titled "A Robot in Every Home," Bill Gates highlighted the obvious parallels between the pre-Microsoft PC industry and the pre-anybody personal robotics industry. Industrial use, research work, and a fringe garage hobby. That was the state of the computer industry before Bill Gates and Steve Jobs, and that's more or less the state of the robotics industry now, five years after Bill's editorial.

Of course, Bill hasn't been around to make the dream come true, he's been busy saving Africa and our public school system and the souls of fellow billionaires. He did leave behind a multi-billion dollar sotware company, however, that is perfectly poised to make "A Robot in Every Home" a piece of fact instead of fiction. Since then, Microsoft's one major (intentional) contribution to the industry has been the sporadically updated Microsoft Robotics Developer Studio. It's a good tool for prototyping and simulating simple robotics, but it isn't moving anything forward. In fact, it treats the robotics industry exactly like computer industry stalwarts treated the burgeoning PC industry: as a hobby. What Microsoft hasn't been doing over these past years is building a robot operating system, or making an even greater gamble on actual robots themselves.

Oddly enough, Microsoft's largest contribution to robotics as of yet was largely inadvertent. The Kinect sensor for the Xbox 360 was launched in November of 2010, and was a surprising success with consumers. While normal people snapped up the mysterious sensor by the millions, brought it into their living rooms, and realized how very-out-of-shape they were, pale hobbyists ("hackers," as they're known these days) quickly sequestered themselves in their garages (circa 2010/2011: poorly heated loft apartments), and taught the Kinect sensor new tricks. The piece of hardware that was originally intended to be a locked down add-on for the 360 became a multipurpose 3D sensor extraordinaire. Microsoft actually issued a mild out-of-touch (and never repeated) threat to the hackers, but the "damage" was done, and hundreds of burgeoning roboticists had a supremely powerful tool in their hands -- and incidentally generated millions of dollars worth of free PR for Kinect with YouTube videos of their exploits.

In 1974, when Intel released the 8080 microprocessor, it wasn't trying to invent the PC, it was just trying to improve upon its existing, limited 8-bit 8008 chip. It was up to the likes of MITS (the Altair 8800) and Microsoft (Altair Basic) to make good use of it. Clones and successors quickly followed, and Intel has obviously kept up over the years. Perhaps Microsoft would be happy to accidentally spark a robotics revolution with the Kinect sensor, but wouldn't it prefer to be at the center of it? Besides, Microsoft doesn't actually build the 3D sensor heart of the Kinect, those honors go to a company called PrimeSense, which is offering the same tech to anyone for a similarly low price.

Someone is going to figure this out. Willow Garage, fueled by some mysterious and apparently inexhaustible venture capital, is taking the open source angle with its ROS (Robot Operating System). The project already has a good amount of traction among bearded hackers and ambitious university robotics programs, since it allows altruistic types to build upon the innovation of others instead of continually "reinventing the wheel" (as Willow Garage puts it) and building their own robot operating system and hardware support from the ground up. Still, while ROS has made great strides and is

home to some very exciting innovation -- along with its fair share of Kinect hacks, of course -- it's nothing a consumer would find useful or even approachable. What the personal computing revolution did was take tools that were already commonplace in the enterprise and hand them to regular cro-mags who wanted to "balance a checkbook" with a spreadsheet application or "word process" without a typewriter ribbon. Microsoft put those tools in the hands of hobbyists, then Apple put them in the hands of regular people, and then Microsoft put them in the hands of everybody.
What we need a Microsoft or a Google or an Apple to do -- or if they won't do it, Enterprising Upstart X -- is build an operating system that runs on standardized hardware or commodity hardware, with built-in capabilities for doing things that are actually useful for a home user. Buzzing you in when you get locked out, signing for a package, taking that frozen chicken out of the freezer while you're at work, feeding your pet, and of course the veritable classic of robo-problems: getting you a beer. As simple as these things sound, they're actually incredibly complex in terms of where general robotics innovation is at currently. That's why EUX is a longshot, but there's still room for some barefaced ingenuity. The dawn of the PC was marked by incredible efficiency of code and hardware, techniques that made Bill Gates and Steve Wozniak famous. Currently, the retail robot closest to being able to manage all these tasks is Willow Garage's PR2, which costs $400,000, harbors two dual-processor Xeon servers (16 cores total) and is still slow as molasses.
Imagine a robot that you could buy at Best Buy for somewhere between $2k and $4k, unbox and configure in half an hour, and then just take for granted as an extremely reliable, whine-free household member for the next few years (or, if you bought it from Apple, exactly 12 months before the upgrade lust sets in). It would change everything. Of course, it sounds preposterous given the current state of this barely-there industry, but it's going to be a reality within the next decade. Who will get us there first?" [Mil11]

10 Versicherung über Selbstständigkeit

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

Geesthacht,02.08.2011 _____

Ort, Datum Unterschrift