Modulare und plattformunabhängige dynamische Bewegungserzeugung für autonome mobile humanoide Roboter

Modular and Platform-Independent Dynamic Motion Generation for Autonomous Mobile Humanoid Robots

Diplomarbeit von
Dorian Scholz

TU Darmstadt
Fachbereich Informatik
Informatik Diplom

Prof. Dr. Oskar von Stryk
Dipl.-Inform. Martin Friedmann

Darmstadt, Germany
Dezember 2008

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, Dezember 2008

Dorian Scholz

Erklärung zur Verwendung der Diplomarbeit

Hiermit erkläre ich mein Einverständnis mit den im Folgenden aufgeführten Verbreitungsformen dieser Diplomarbeit:

Verbreitungsform	**ja**	**nein**
Einstellung der Arbeit in die Bibliothek der TUD	X	
Veröffentlichung des Titels der Arbeit im Internet	X	
Veröffentlichung der Arbeit im Internet	X	

Ort, Datum Unterschrift Diplomand

Kurzzusammenfassung

Im Rahmen dieser Diplomarbeit wurde eine Software zur Echtzeitsteuerung und -regelung von Robotern entwickelt. Zum Einsatz kommt die Software auf einem Mikrocontroller zur Ansteuerung der Aktoren und Sensoren auf humanoiden Robotern. Implementiert wurde ein plattformunabhängiges Basissystem und Bibliotheken zur Plattformabstraktion für mehrere Systeme. Unterstützte Plattformen sind zwei verschiedene Mikrocontroller-Systeme, sowie PCs mit Windows oder Linux und ein Simulator. Das System ist modular aufgebaut und beinhaltet mehrere Bewegungsmodule. Einen Schwerpunkt bilden das Laufmodul und die Balanceregelung. Weiterhin wurde zu Debugging- und Visualisierungszwecken eine grafische Benutzeroberfläche implementiert, die mit dem System über eine serielle Verbindung kommuniziert.

Abstract

In the course of this thesis a software system for the real-time open and closed loop control of robots was developed. The system is used on a microcontroller interfacing with the actuators and sensors of humanoid robots. The implementation consists of a platform-independent base system and a platform abstraction library for multiple systems. Supported platforms are two microcontroller systems and PC based systems running Windows, Linux or a simulator. A modular architecture is in use to be able to host multiple motion modules. Biped locomotion and balancing were the main focus. Furthermore a graphical user interface was developed to aid in development, debugging and visualization of data on the microcontrollers.

Inhaltsverzeichnis

Abbildungsverzeichnis

Glossar

GUI grafische Benutzeroberfläche (graphical user interface)

Baudrate Übertragungsrate in Symbolen pro Sekunde (entspricht hier bei binärer Übertragung Bit pro Sekunde)

RoboCup jährliche Weltmeisterschaft im autonomen Roboterfußball: `http://www.robocup.org/`

EEPROM Electrically Erasable and Programmable Read Only Memory

Roll-Pitch-Yaw-Winkel Roll-Nick-Gier-Winkel zur Beschreibung der Orientierung im dreidimensionalen Raum

1 Einleitung

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist die Entwicklung einer Mikrocontrollerfirmware zur Steuerung und Regelung verschiedener Roboter.

Dies beinhaltet die Kommunikation sowohl mit den Aktoren und Sensoren des Roboters, als auch mit der übergeordneten Steuereinheit. Weiterhin müssen zur Steuerung die Gelenkwinkelkoordinaten mit Hilfe inverser Kinematik berechnet werden. Zur Regelung werden die Sensorinformationen gefiltert und ausgewertet.

Um das System Debuggen und Weiterentwickeln zu können soll es plattformunabhängig und modular aufgebaut werden. Dies ermöglicht es zu Test- und Analysezwecken die Software auf einem PC auszuführen und entweder einen simulierten oder auch einen realen Roboter steuern zu lassen.

Weiterhin soll eine grafische Debugoberfläche entwickelt werden, die auch Einblicke in das auf dem Mikrocontroller laufende System in Echtzeit erlaubt.

1.2 Motivation

Die Erfahrungen aus dem RoboCup haben gezeigt, dass eine einfach zu debuggende und gut zu wartende Mikrocontrollerfirmware eine wichtige Basis für den Betrieb von Robotern bildet. Hierzu gehört auch die Visualisierbarkeit der Programmdaten in Echtzeit, um z.B. das Einstellen der Regelungsparameter zu vereinfachen.

Es ist zur Erfüllung der verschiedenen Anforderungen des RoboCup

unerlässlich eine schnelle und robuste Kommunikation mit den Komponenten des Roboters zu gewährleisten. Dazu gehört der bidirektionale Datenaustausch mit den Servoeinheiten, die sowohl Aktoren als auch Sensoren enthalten. Hierbei müssen die gewünschten Positionen zu den Motoren geschickt werden und die aktuellen Positionen ausgelesen werden. Dies erfordert einen schnellen Datentransfer, um ein möglichst gering verzögertes Modell der Gelenkstellungen erstellen zu können.

Des weiteren ist eine Auswertung der Sensoren in Echtzeit notwendig, um eine Balanceregelung zu ermöglichen.

2 Grundlagen (Theorie)

2.1 Echtzeitsystem

Als Echtzeitsystem beschreibt man ein Computersystem, wenn der Erfolg einer Berechnung nicht nur von der Richtigkeit des Ergebnisses abhängt, sondern auch noch von der benötigten Zeit. Dabei unterscheidet man zwischen zwei Echtzeitanforderungen. [9] [19]

2.1.1 Harte Echtzeit

Dies wird gefordert, wenn das zu berechnende Ergebnis nach Ablauf der vorgegebenen Antwortzeit nutzlos wird und dieser Fall zu einer massiven Störung oder dem Ausfall des System führt.

2.1.2 Weiche Echtzeit

Hierbei führt eine nicht zeitgerechte Berechnung nur zu Einbußen in der Qualität des Dienstes, erlaubt dem System aber fortzufahren.

2.2 Stabilitätskriterien beim zweibeinigen Laufen

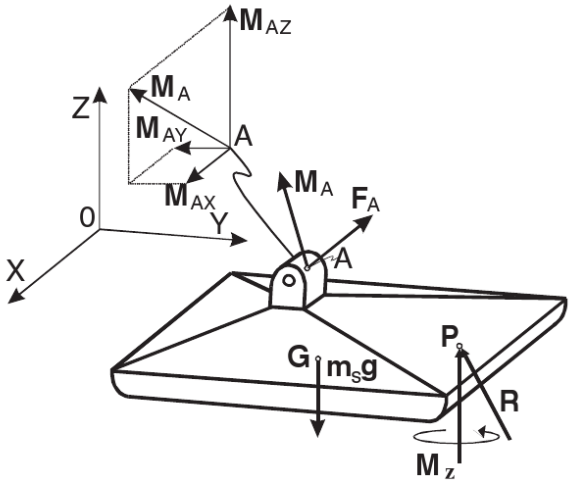
Grundsätzlich wird beim zweibeinigen Laufen zwischen statischer und dynamischer Stabilität unterscheiden. Das Stabilitätskriterium für statisch stabiles Laufen ist der in Richtung des Gravitationsvektors auf den Boden projizierte

Schwerpunkt des Roboters. Liegt dieser zu jeder Zeit innerhalb der konvexen Hülle der Bodenkontaktpunkte, ist die statische Stabilität gewährleistet. Dem kann aber bei langsamen Laufbewegungen genügt werden. Um ein schnelleres Laufen zu ermöglichen müssen die auch dynamischen Effekte betrachtet werden. Dies ermöglicht höhere Laufgeschwindigkeiten und eine bessere Energieeffizienz. Bei dynamisch stabilem Laufen wird als Stabilitätskriterium der sogenannte Zero-Moment-Point (ZMP) genutzt.

2.2.1 Zero-Moment-Point (ZMP)

Für diese Betrachtung kann der Teil des Roboters oberhalb des Sprunggelenkes des Standfußes vernachlässigt werden, indem man nur den Einfluss durch einwirkende Kraft und Moment am Sprunggelenk betrachtet.

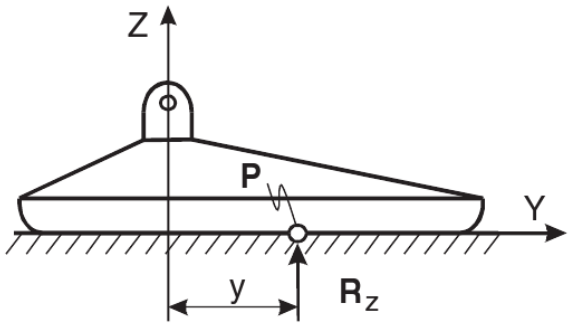
Abbildung 2.1: Kräfte und Momente am Fuß (Quelle [21])



Seien F_A und M_A die am Sprunggelenk auf den Fuß übertragenen Kräfte und Momente des Roboters, so wirken die Bodenreaktionskräfte R und

Bodenreaktionsmomente M diesen entgegen (Abbildung 2.1). Bei einem nicht rutschenden Fuß werden die horizontal wirkenden Kräfte und Momente R_x, R_y und M_z durch die Reibung aufgebracht und heben die korrespondierenden Kräfte und Momente im Sprunggelenk auf. Die vertikale Kraftkomponente wird durch die immer aufwärts gerichtete Reaktionskraft R_z des Bodens ausgeglichen. Daraus resultiert, dass die verbleibenden horizontalen Anteile der übertragenen Momente M_A nur ausgeglichen werden können, indem der Angriffspunkt P der Reaktionskraft R innerhalb der konvexen Hülle der Bodenkontaktpunkte verschoben wird (Abbildung 2.2).

Abbildung 2.2: Verschiebung des Angriffspunktes (Quelle [21])



Definition Es gibt einen Punkt P in der Bodenebene, an welchem für die horizontalen Komponenten der Bodenreaktionsmomente M gilt:

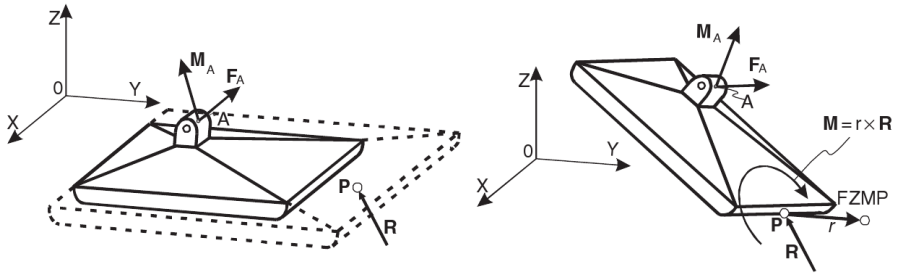
$$M_x = 0 \ \wedge \ M_y = 0$$

Falls ein solcher Punkt P innerhalb der konvexen Hülle der Bodenkontaktpunkte existiert, so heißt er Zero-Moment-Point (ZMP). □ [21]

Sollte die konvexe Hülle der Bodenkontaktpunkte nicht groß genug sein, um durch Verschieben des Angriffspunktes P die einwirkenden Momente

auszugleichen, so existiert kein ZMP. Die Reaktionskraft R wirkt dann an der Kante des Fußes und die nicht komplett ausgeglichenen einwirkenden Momente bringen den Roboter zum Kippen über die Fußkante. In diesem Fall kann ein fiktionaler Angriffspunkt der Reaktionskraft außerhalb der konvexen Hülle der Bodenkontaktpunkte gefunden werden, an welchem die einwirkenden Momente ausgeglichen werden könnten. Diesen bezeichnen wir als fiktionalen ZMP (FZMP).

Abbildung 2.3: fiktionaler ZMP (Quelle [21])

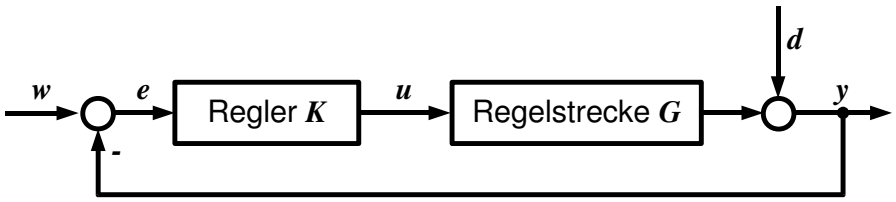


Für ein dynamisch stabiles Laufen ist es also entscheidend, den ZMP innerhalb der konvexen Hülle der Bodenkontaktpunkte und nach Möglichkeit auf Abstand zu den Rändern zu halten. Sollte jedoch durch eine Störung der ZMP aus der konvexen Hülle der Bodenkontaktpunkte verschoben werden und somit nur noch als FZMP existieren, so können aus dessen Position hilfreiche Informationen zur Wiederherstellung des Gleichgewichts gewonnen werden.

2.3 Regelung

Ein Regler vergleicht in einem Regelkreis (Abbildung 2.4) den Soll-Wert mit dem zurückgeführten Ist-Wert (Regelgröße). Aus der Differenz dieser beiden Werte (Regeldifferenz) e berechnet er die Stellgröße u. Diese beeinflusst die Regelstrecke so, dass die Regeldifferenz minimiert wird. [15] [16]

Abbildung 2.4: Regelkreis



Nicht verwechselt werden sollte dies mit der Steuerung, bei welcher keine Rückführung des Ist-Wertes erfolgt.

Klassische lineare Regler bestehen aus einer Kombination der folgenden Teile:

- Proportionalteil (P-Glied)
- Integralteil (I-Glied)
- Differentialteil (D-Glied)

Beispielhaft sei hier die Übertragungsfunktion eines PID-Reglers gezeigt:

$$u(t) = K_p(e(t) + \frac{1}{T_n}\int_0^t e(\tau)d\tau + T_v\frac{d}{dt}e(t))$$

Dabei werden die Einstellparameter K_p, T_n und T_v so gewählt, dass sie die vorliegende Regelaufgabe möglichst gut lösen. Dazu gilt es sicherzustellen, dass die Regelung im Anwendungsfall stabil ist und dass das Regelungsverhalten dem gewünschten Verhalten möglichst nahe kommt was Einstellverzögerung und Überschwingungsverhalten betrifft.

3 Grundlagen (Hardware)

Neben dem weitgehend plattformunabhängigen Basisteil dieser Software, ist auch die plattformabhängige Anbindung an verschiedene Hardware implementiert worden. Dazu ist eine Kenntnis der Funktionsweise der eingesetzten Hardwarekomponenten unerlässlich. Dieses Kapitel gibt einen Überblick über die verwendeten Roboter, sowie deren Komponenten.

3.1 Roboter

Zum Einsatz kommt die Mikrocontrollerfirmware auf zwei humanoiden Robotertypen. Sie bauen auf von Hajime Research Ltd. entwickelten Roboterplattformen auf und wurden vom Team der Darmstadt Dribblers erweitert, um am RoboCup-Soccer-Wettbewerb in der Humanoiden Kid-Size Liga teilnehmen zu können. Diese Erweiterungen beinhalten unter anderem eine Webcam, ein PC-104 Prozessorboard, sowie eine WLAN Schnittstelle und einen Massenspeicher. Hierdurch wird das autonome Fußballspielen in bekannter Umgebung ermöglicht. Da diese Komponenten aber für die vorliegende Arbeit nicht weiter relevant sind, werden sie im Folgenden nicht weiter beschrieben.

3.1.1 DD2007

Das Robotermodell Darmstadt Dribblers 2007 basiert auf der Roboterplattform Hajime Robot 18. Diese besitzt 21 Freiheitsgrade (Abbildung 3.2), welche alle über elektrische Servomotoren realisiert werden. Zum Einsatz kommen hier zwei

Typen von Servomotoren (DX-117 und RX-64) der Firma Robotis mit eingebauten Potentiometern zur Positionsbestimmung. Gesteuert werden sie von einem Mikrocontroller des Typs SuperH 7145F (SH7145F) [2] von Renesas. Diesem mit 50 MHz getaktetem Chip stehen ein Megabyte RAM und 128 Kilobyte EEPROM als externe Speicher zur Verfügung. Angeschlossen sind alle 21 Servomotoren an einen seriellen Bus des Mikrocontrollers. Die Kommunikation erfolgt mit der maximalen Baudrate des Mikrocontrollers von 500000 Bit pro Sekunde.

3.1.2 DD2008

Das Robotermodell Darmstadt Dribblers 2008 (Abbildung 3.1) basiert auf der Roboterplattform Hajime Robot 30. Auch diese kann in 21 Freiheitsgraden von elektrischen Servomotoren bewegt werden. Die kinematische Kette (Abbildung 3.2) gleicht der des DD2007. Jedoch wurden die Servomotoren des Typs DX-117 durch den neueren RX-28 abgelöst. Auch der Mikrocontroller ist eine neuere Generation aus der SuperH Serie von Renesas. Der mit 160 MHz getaktete SuperH 7211 (SH7211) [4] kann hier auf 16 Megabyte RAM und 128 Kilobyte EEPROM in externen Speicherbausteinen zugreifen. Anders als beim DD2007 sind die Servomotoren auf zwei serielle Busse des Mikrocontrollers aufgeteilt worden. Somit sind mit einem Bus zehn und mit dem anderen Bus elf Servomotoren verbunden, was einen höheren Datendurchsatz ermöglicht. Die Kommunikation erfolgt jeweils mit der maximalen Baudrate der Servomotoren von 1000000 Bit pro Sekunde.

3.2 Sensoren

In dieser Arbeit kamen drei Sensortypen zum Einsatz:

- Gyroskope: zur Balanceregelung
- Beschleunigungssensoren: zur Lagebestimmung
- Potentiometer: zur Bestimmung der Gelenkstellungen

Abbildung 3.1: DD2008

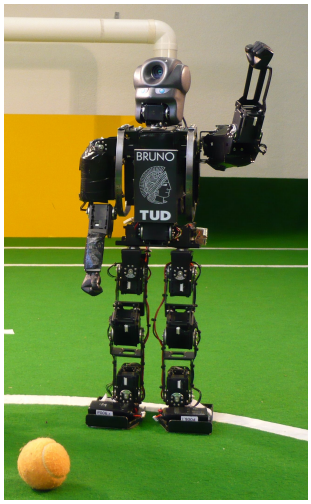
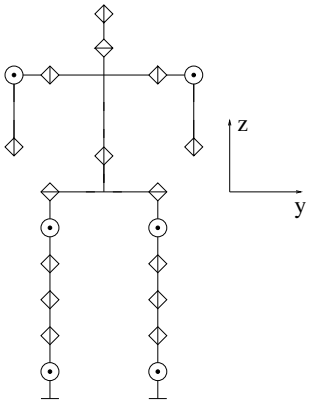


Abbildung 3.2: Kinematische Kette DD2007 / DD2008



3.2.1 Gyroskope

Ein Gyroskop ermöglicht die Messung der Winkelgeschwindigkeit um eine Achse. Aus den Messdaten von drei paarweise rechtwinklig stehenden Gyroskopen können die momentane Rotationsgeschwindigkeit und -achse berechnet werden. Dies ermöglicht eine Balanceregelung des Roboters.

3.2.2 Beschleunigungssensoren

Beschleunigungssensoren messen die lineare Beschleunigung in Richtung einer Achse. Gemessen wird hierbei die lokale Beschleunigung der das System ausgesetzt ist, überlagert von der auf der Erde allgegenwärtigen Erdbeschleunigung. Mit Hilfe von drei paarweise orthogonal angeordneten Beschleunigungssensoren lässt sich die Richtung der Erdbeschleunigung bestimmen. Daraus lässt sich direkt auf die Lage des Roboters im Raum schließen.

3.2.3 Potentiometer

Ein Potentiometer ist ein stetig einstellbarer Widerstand. Wird ein Drehpotentiometer so an einem Gelenk angebracht, dass seinen Stellung der des Gelenkes folgt, kann durch Messen seines Widerstands auf die Gelenkstellung geschlossen werden. Dieses Verfahren kommt hier innerhalb der verwendeten Servoeinheiten zum Einsatz.

3.3 Aktoren

Aktoren wandeln eine Eingangsgröße in eine andersartige Ausgangsgröße um. In dieser Arbeit werden ausschließlich Servomotoren als Aktoren verwendet.

3.3.1 Servomotoren

Servomotoren sind elektrische Antriebe die zusammen mit einem Regler in einem geschlossenen Regelkreis verwendet werden. Sie verfügen über eine Messeinrichtung zur Bestimmung der Position der Abtriebswelle und können somit eine vorgegebene Position anfahren.

Zur Ansteuerung von Servomotoren bieten sich grundsätzlich zwei Verfahren an. Entweder die Steuerung über ein pulsweitenmoduliertes (PWM) Signal zur Positionsvorgabe oder über ein digitales Bussystem.

Die Ansteuerung über ein PWM-Signal ist im Modellbau weit verbreitet, benötigt jedoch pro Servomotor eine eigene Steuerleitung und ermöglicht keine Rückmeldung. Somit lässt sich nicht überprüfen, ob der Servomotor die vorgegebene Position auch erreicht hat oder ob er eventuell überlastet ist.

Bei einem Bussystem lässt sich über eine Leitung für alle Servomotoren eine Kommunikation in beiden Richtungen ermöglichen. Dazu erhält jeder Servomotor eine Identifikationsnummer (ID) über die er explizit angesprochen werden kann. Dies ermöglicht es Status- und Fehlermeldungen zurück an die übergeordnete Steuereinheit zu schicken, was im autonomen Betrieb sehr wichtig ist.

Die hier verwendeten Servomotoren kommen bei den Robotern als Antriebe der Drehgelenke zum Einsatz. Sie werden über ein Bussystem nach dem RS-485 Standard angesteuert (Details siehe 5.1.5). Zur Kommunikation benötigt dabei jeder Servomotor einen Mikrocontroller. Dieser übernimmt auch gleichzeitig die Positionsregelung. Dazu wird ein Potentiometer wie oben beschrieben zur Positionsbestimmung eingesetzt.

3.4 Mikrocontroller

Ein Mikrocontroller ist ein Ein-Chip-Computersystem, auch System-On-Chip (SoC) genannt. Er vereinigt einen Prozessor mit Speicher- und Peripherieeinheiten auf einem Mikrochip. Der Mikrocontroller wird meist als Hauptkomponente einer speziell für eine Aufgabe entwickelten Schaltung verwendet und mit den anderen

Komponenten auf einer Platine aufgelötet. Weitere Komponenten sind dabei zum Beispiel externer Speicher oder Bausteine zur seriellen Kommunikation.

Bei der Programmierung von Mikrocontrollern ist ein gutes Verständnis der Komponenten des Mikrocontrollers und ihres Zusammenspiels von Nöten. Deshalb hier ein kurzer Überblick über die für diese Arbeit wichtigsten Bauteile und Konzepte.

3.4.1 Interrupt

Ein Interrupt ist eine Unterbrechung des normalen Programmablaufs in der für diesen Interrupt spezifischer Code ausgeführt wird. Ausgelöst wird ein Interrupt durch eine Unterbrechungsanforderung. Diese werden auf einem Mikrocontroller meist von internen oder externen Peripheriegeräten ausgelöst. Sie dienen der ereignisgesteurten Verarbeitung von Daten oder auch zur Fehlerbehandlung. Mögliche Peripheriegeräte sind Zeitgeber, serielle Schnittstellen oder Ähnliches. Ein Interrupt wird typischerweise exklusiv verarbeitet, das heißt während die Unterbrechungsroutine arbeitet, kann kein weiterer Interrupt ausgelöst werden. Nach der Abarbeitung der Unterbrechungsroutine wird der Prozessorzustand wieder hergestellt, sodass der normale Code transparent weiterarbeiten kann.

3.4.2 Zeitgeber

Ein Zeitgeber (Timer) wird benötigt, um bestimmte Abläufe zeitgesteuert auszuführen oder auch die vergangene Zeit zu messen. Realisiert werden Timer in einem Mikrocontroller als unabhängig vom normalen Programmfluss herunter- oder heraufzählende Register. Die Frequenz in der gezählt wird, ist dabei bestimmt von einem internen oder externen Taktgeber. Als Taktgeber kommt üblicherweise ein Quarzoszilator zum Einsatz. Da der Quarzoszilator ein sehr hochfrequentes Ausgangssignal hat, kann dieses über einen Vorteiler auf ein geeignete Frequenz reduziert werden.

3.4.3 Serielle Schnittstellen

Die seriellen Schnittstellen dienen zur Kommunikation mit anderen Bauteilen und Systemen. Sehr oft wird eine serielle Schnittstelle genutzt, um einen Mikrocontroller mit einem PC zu verbinden. Dieser ermöglicht dann die Interaktion mit der Firmware des Mikrocontrollers über ein serielles Terminal.

Um die serielle Schnittstelle eines Mikrocontrollers mit einem PC zu verbinden, wird meist nach dem RS-232 Standard gearbeitet. Dazu müssen die Signalpegel des Mikrocontrollers auf die des RS-232 Standards gewandelt werden. Dafür ist ein externer Pegelwandler notwendig.

3.4.4 Analog-Digital-Umsetzer

Da ein Mikrocontroller nur mit digitalen Daten arbeiten kann, benötigt er Analog-Digital-Umsetzer (ADU) um analoge Daten in Digitale umzusetzen. Die wichtigsten Parameter eines ADU sind seine Auflösung und seine Umsetzungsrate. Die Auflösung wird in Bit spezifiziert und beschreibt, in wie viele mögliche Ausgangswerte der ADU die Amplitude des analogen Signal quantifizieren kann. Somit stellt die Auflösung auch die Genauigkeitsgrenze des ADU dar. Die Umsetzungsrate beschreibt, wie häufig die Umsetzung des Signals stattfindet.

4 Anforderungen

4.1 Firmware

4.1.1 Plattformunabhängigkeit

Die zu entwickelnde Firmware soll primär auf zwei verschiedenen Mikrocontrollern betrieben werden. Jedoch ist eine möglichst breite Codebasis zu erstellen, die auch auf anderen Plattformen lauffähig ist. Dies ist zum einen wichtig für die Portierung auf neue Systeme, aber ermöglicht auch ein Debuggen großer Teile des Codes auf dem PC. Daraus ergeben sich erhebliche Vorteile gegenüber dem Debuggen auf dem Mikrocontroller selbst. Natürlich können aber die mikrocontrollerspezifischen Codeteile nur auf diesem auch in realer Umgebung getestet werden. Deshalb wird eine Anbindung an einen auf einem PC laufenden Debugger benötigt.

4.1.2 Struktur

Der Code soll einen modularen Aufbau haben. Dies ermöglicht es einzelne Teile zu überarbeiten, zu ersetzen oder auch hinzuzufügen, ohne das gesamte System zu verändern.

4.1.3 Laufzeit

Die Steuerung, Regelung und Kommunikation soll in Echtzeit erfolgen. Es reicht aus weiche Echtzeit zu fordern, bei der ein Durchlauf des Steuer- und Regelzyklus

nicht länger als ein vordefiniertes Intervall braucht. Sollte ein Zyklus dieses Intervall überschreiten, muss der darauf folgende Zyklus übersprungen werden, um die verlorene Zeit wieder aufzuholen. Dies bedeutet eine Beeinträchtigung der Qualität von Steuerung und Regelung, lässt das System aber weiter funktionieren.

4.1.4 Kommunikation

Die Kommunikation mit der übergeordneten Steuereinheit soll ein asynchrones Protokoll ermöglichen. Dazu müssen beide Seiten durchgehend empfangsbereit sein und mit geringer Latenz antworten können. Das System soll auch unaufgefordert regelmäßig oder ereignisabhängig Statusmeldungen verschicken können.

Der Datenaustausch mit den Servoeinheiten soll transparent im Hintergrund abgewickelt werden. Die Aufrufe der Kommunikationsfunktionen sollen also nicht-blockend erfolgen. Das heißt die zu sendenden Daten müssen in einem Puffer zwischengespeichert werden und die eigentliche Übertragung erfolgt im Hintergrund. Dabei sollen die Steuerbefehle an die Servoeinheiten Vorrang haben und die Statusabfragen die restliche Bandbreite des seriellen Busses nutzen.

4.1.5 Bewegungsmodule

Um die Roboter für den Einsatz beim RoboCup tauglich zu machen, sollen zwei grundlegende Bewegungsmodule implementiert werden.

Ein Bewegungsmodul soll die Möglichkeit schaffen, den Roboter so flexibel wie möglich zu bewegen. Dies wird benötigt, um Schuss- und Aufstehbewegungen zu erstellen und auszuführen.

Das zweite Bewegungsmodul soll eine parametrisierbare Laufbewegung erzeugen. Mit Hilfe dieses Moduls soll der Roboter in der Lage sein omnidirektional zu laufen. Dabei sollen Laufgeschwindigkeit und -richtung ständig anpassbar sein, um eine flüssiges Laufverhalten zu ermöglichen.

Des Weiteren wird ein Modul zur Steuerung des Kopfes benötigt, welches

unabhängig von der Bewegung des restlichen Roboters die Blickrichtung der Kopfkamera steuert.

4.1.6 Balanceregelung

Unabhängig vom aktuellen Bewegungsmodul soll eine Balanceregelung auf Basis der Sensordaten erfolgen. Es stehen dafür drei Gyroskope und drei Beschleunigungssensoren zur Verfügung.

4.1.7 Kopfreflex

Für den Fall, dass der Roboter umfällt, ist es wichtig die Kopfkamera zu schützen, da diese in manchen Positionen über den Körper hinaus steht und somit beschädigt werden könnte. Der 'Kopfreflex' soll den Kopf in eine sichere Position bringen, sobald ein Umfallen nicht mehr verhindert werden kann. Für einen erfolgreichen 'Kopfreflex' sind zwei Kriterien bei der Erkennung des Umfallens entscheidend. Es muss unter allen Umständen rechtzeitig erkannt werden, um den Kopf noch in die sichere Position bringen zu können, da ein Sturz auf die Kopfkamera direkt zur Spielunfähigkeit führen kann. Ein unnötiges Auslösen des 'Kopfreflexes' sollte vermieden werden, da es den Bewegungsraum der Kamera stark einschränkt.

4.1.8 Parametrisierung

Um das System einfach an verschiedene Gegebenheiten anpassen zu können, sollen alle relevanten Werte als Parameter zugänglich gemacht werden. Dazu gehört die Werte im laufenden Betrieb auslesen und verändern zu können.

4.1.9 Unit-Tests

Um die Komponenten der Firmware zu testen und ihre Funktionsfähigkeit zu gewährleisten, sollen Unit-Tests erstellt werden. Mit Hilfe von Unit-Tests lassen

sich einzelne Module, aber auch das Zusammenspiel mehrerer Komponenten verifizieren. [8]

4.2 Debugger

Um ein anschauliches Debuggen der Firmware im Betrieb zu ermöglichen soll ein Debugger mit grafischer Benutzeroberfläche implementiert werden. Diese soll die in der Firmware verwendeten Datenstrukturen darstellen können. Dabei soll es möglich sein die Werte der in den Datenstrukturen enthaltenen Elemente anzuzeigen und zu verändern. Weiterhin soll eine Visualisierung der Elemente in grafischer Form ermöglicht werden.

5 Realisierung

5.1 Firmware

Da die Firmware sowohl auf verschiedenen Mikrocontrollern als auch auf mehreren Betriebssystemen lauffähig sein sollte, wurde als Programmiersprache C [14] gewählt. Hierbei wurde nach dem C90 Standard gearbeitet, damit das Projekt auch mit Microsoft Visual Studio 2005 kompiliert werden kann.

5.1.1 Plattformabstraktion

Um eine möglichst hohe Plattformabstraktion zu gewährleisten, wurden feste Schnittstellen definiert, welche die jeweiligen plattformspezifischen Module implementieren. Diese umfassen zum Beispiel Schnittstellen zu Zeitnehmer- und Kommunikationsmodulen (siehe Übersicht Bild 5.1). Der größte Teil des Codes wurde dann aufbauend auf diesen Schnittstellen entwickelt und somit auf allen unterstützten Plattformen unverändert lauffähig. Die momentan unterstützten Plattformen sind:

- Renesas SuperH SH/7145F und SH2/7211 [1, 3]
- Win32 API [13]
- Posix kompatible [18]
- Multi Robot Simulation Framework (MuRoSimF) (Firmware eingebunden als dynamische Bibliothek [6, 7])

Renesas SuperH

Für die beiden verwendeten SuperH Mikrocontroller von Renesas musste im Vergleich zu den anderen Plattformen mehr Code entwickelt werden. Da hier kein Betriebssystem zum Einsatz kommt, mussten auch grundlegende Module zur Ansteuerung von zum Beispiel der Echtzeituhr oder der seriellen Schnittstellen komplett implementiert werden.

Serielle Schnittstellen Bei der Implementierung der Module zur Ansteuerung der seriellen Schnittstellen wurde großer Wert darauf gelegt, dass die Kommunikation durchweg mit nicht-blockenden Aufrufen ermöglicht wird. Dies ermöglicht einen transparenten Aufruf von Kommunikationsfunktionen an jeder Stelle der Firmware, ohne dass es zu Wartezeiten kommt. Realisiert wurde dies durch Ringpuffer für zu versendende und zu empfangene Daten. Diese werden dann von den Interruptroutinen im Hintergrund abgearbeitet und das Hauptprogramm kann weiterarbeiten.

Timer Das 'Multi-Function Timer Pulse Unit' (MTU) der Renesas Chips unterstützt mehrere Zeitgeber, welche nach Ablauf einen Interrupt auslösen können (Interrupt-Timer). Ein Interrupt-Timer wird verwendet, um die Taktung des Steuer- und Regelzyklus zu realisieren. Ein weiterer Interrupt-Timer kommt zum Einsatz um die Zeitüberschreitungen bei der seriellen Kommunikation abzufangen.

Direct-Memory-Access Controller Der Direct-Memory-Access Controller (DMAC) ermöglicht das Kopieren von Daten zwischen Hauptspeicher und Peripheriegeräten, ohne Prozessorlast zu erzeugen. Dies wurde benutzt, um die serielle Kommunikation effizienter zu machen. So werden die zu verschickenden Daten Byte für Byte vom DMAC aus dem Speicher in das serielle Ausgangsregister kopiert, ohne dass die CPU eingreifen muss. Auch beim Empfang serieller Daten kommt der DMAC zum Einsatz. Werden bei der seriellen Kommunikation Pakete

unbekannter Länge empfangen, muss zuerst der Paketkopf mit der Paketlänge empfangen werden, bevor die CPU die Kontrolle an den DMAC übergeben kann.

Analog-Digital-Umsetzer Die beiden Renesas Mikrocontroller verfügen jeweils über zwei eingebaute Analog-Digital-Umsetzer, die via Multiplexer jeweils auf vier Kanälen Messungen durchführen können. Nach erfolgter Messung werden die Daten in einer Interruptroutine verarbeitet. Die maximalen Abtastfrequenz liegt je Kanal bei 6250 Hz (SH7145F) bzw. 12500 Hz (SH7211) und die Auflösung bei 10 Bit.

Da die gemessenen analogen Signale verrauscht sind, werden sie durch einen Filter geglättet. Die Wahl fiel hierbei auf einen rekursiven Tiefpassfilter erster Ordnung, da dieser nur geringe Rechenzeit und wenig Speicherbedarf benötigt [17]. Der Ausgabewert out_i im Zeitschritt i berechnet sich aus dem Eingabewert in_i nach folgender Formel.

$$out_i = in_i \cdot (1-p) + out_{i-1} \cdot p \tag{5.1}$$

Dabei ist p ein Faktor zwischen 0 und 1, der den Einfluss des Eingabewerts auf den Ausgabewert bestimmt.

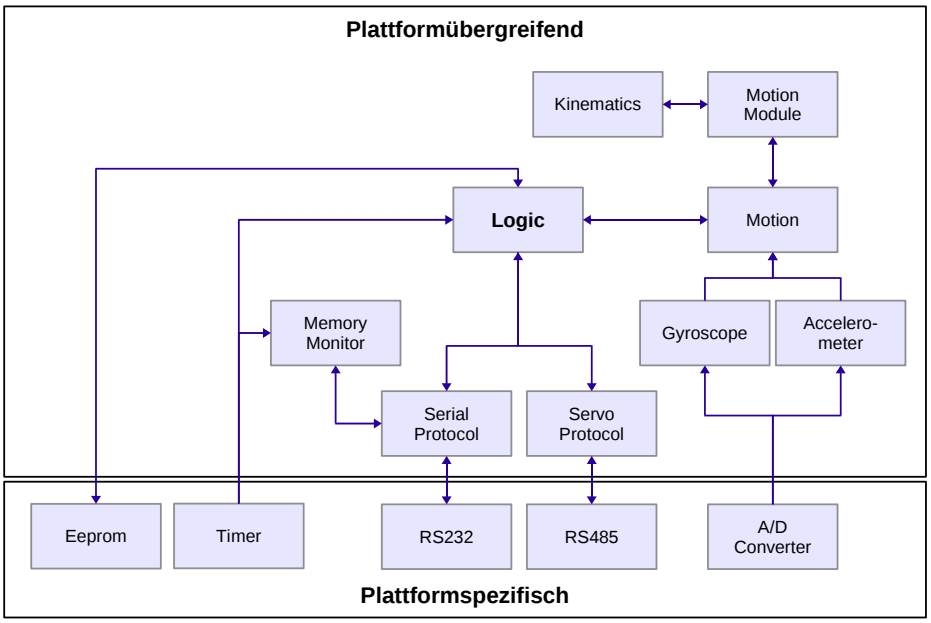
Bei Tests stellte sich heraus, dass die Mikrocontroller nicht ausreichend Rechenleistung besitzen um bei ihrer jeweiligen maximalen Abtastfrequenz auf Fließkommabasis zu filtern. Das erklärt sich damit, dass die beiden Mikrocontroller keine Fließkommaeinheit (FPU) besitzen und somit die Fließkommaberechnungen in Software durchgeführt werden. Um die Rechenzeit zu reduzieren, wurde der Tiefpassfilter auf Fixkommabasis mit 6 Bit Nachkommaanteil implementiert. Dazu wird die Formel 5.1 mit 2^6 erweitert und mit einer Division mit Rest der Ausgabewert out_i und ein Rest $remain_i$ berechnet.

$$temp = 2^6 \cdot (in_i \cdot (2^6 - p) + out_{i-1} \cdot p) + remain_{i-1}$$
$$remain_i = temp \bmod 2^6$$
$$out_i = \lfloor \frac{temp}{2^6} \rfloor$$

Der Filter auf Fixkommabasis ist ausreichend schnell, um auf allen acht Kanälen des ADU bei maximaler Abtastfrequenz eingesetzt zu werden. Das Thema Rechenzeit wird in Kapitel 6 ausführlicher diskutiert.

EEPROM Das externe EEPROM (Electrically Erasable and Programmable Read Only Memory) kommt als permanenter Speicher für roboterspezifische Daten zum Einsatz. Die Ansteuerung erfolgt über einen Daten- und einen Zeitgeberpin. Der Datenpin wird dabei je nach Operation als Ein- oder Ausgabepin umgeschaltet.

Abbildung 5.1: Übersicht über den Datenfluss zwischen den Modulen



5.1.2 Programmablauf

Hauptprogramm

Das Hauptprogramm besteht aus dem Aufruf der Initialisierungsfunktionen der einzelnen Module, sowie der Hauptschleife. Die Hauptschleife ist eine Endlosschleife, die zwei Funktionen aufruft.

Zum einen die Steuerungs- und Regelungslogik und zum anderen die Verarbeitung von eingehenden Paketen der übergeordneten Steuereinheit. Dabei wird die Steuerungs- und Regelungslogik in einem festen Intervall ausgeführt, welches über einen Timer realisiert wird. Dieser setzt bei jedem Aufruf einen Marker der in der Hauptschleife abfragt wird. Bei gesetztem Marker führt die Hauptschleife den Steuer- und Regelzyklus aus und setzt den Marker zurück.

Die Verarbeitung von eingehenden Paketen der übergeordneten Steuereinheit wird immer durchgeführt, wenn neue Pakete vorhanden sind.

Falls bei einem Durchlauf keiner dieser beiden Fälle eintritt, läuft die Hauptschleife ab, ohne etwas zu bearbeiten. Sie wartet also aktiv (busy waiting) auf das nächste Ereignis. Dieses auf modernen Computersystemen mit mehreren laufenden Prozessen unerwünschte Verhalten ist in der Mikrocontrollerprogrammierung durchaus legitim, da es nur einen einzigen Prozess gibt.

Hintergrundabläufe

Um keine unnötigen Wartezeiten innerhalb des Hauptprogramms einhalten zu müssen, wurden alle Funktionen die zeitgesteuert ablaufen, oder von externen Signalen abhängen, auf Interruptbasis realisiert. Somit können sie Ereignisabhängig das Hauptprogramm unterbrechen, um kleine Aufgaben abzuarbeiten. Auf Interruptbasis laufen folgende Funktionalitäten:

Datenübertragung mit dem PC Hier wird je ein Ringpuffer für Ein- und Ausgabedaten verwendet. Der Eingabepuffer wird dabei von dem Interrupthandler

der seriellen Schnittstelle gefüllt und die Daten werden im Hauptprogramm ausgewertet. Der Ausgabepuffer wird vom Hauptprogramm mit Daten gefüllt und vom Interrupthandler der seriellen Schnittstelle abgearbeitet. Bei der Ausgabe der Daten wird der Direct Memory Access Controller (DMAC) zu Hilfe genommen. Dieser ermöglicht es Daten aus dem Speicher in das Ausgaberegister der seriellen Schnittstelle zu kopieren, ohne dabei Rechenzeit zu beanspruchen. Beim Einlesen der Daten ist das nicht ohne weiteres möglich, da aufgrund des asynchronen Protokolls die zu lesende Datenmenge nicht vorher bekannt ist.

Datenübertragung mit den Servoeinheiten Auch hier werden Ringpuffer für die Ein- und Ausgabe eingesetzt. Jedoch sind diese in einzelne Datenpakete unterteilt und werden paketweise abgearbeitet. Dies ist erforderlich, da die Kommunikation synchron über einen halb-duplexen Bus erfolgt. Das heißt jedem ausgehenden Datenpaket liegt die Information bei, ob es eine Antwort darauf geben wird und welche Länge diese haben wird. Notwendig ist dies, da eine Antwort nur empfangen werden kann, wenn der halb-duplexe Bus in den Empfangsmodus umgeschaltet wird. Da die Kommunikation synchron und die Antwortlänge bekannt ist, konnte hier neben dem Datenversand auch der Datenempfang über den DMAC realisiert werden. Dies spart aufgrund der hohen Datenrate auf diesem Bus erheblich Rechenzeit ein.

Neben dem Ausgabepuffer der die zu verschickenden Pakete des Hauptprogramms aufnimmt gibt es einen weiteren Ausgabepuffer. Dieser beinhaltet die Statusabfragepakete, welche immer verschickt werden, wenn der Hauptausgabepuffer leer ist. Somit ist gewährleistet, dass den Steuerpaketen des Hauptprogramms Vorrang gewährt wird und außerdem die verfügbare Bandbreite des Busses ausgenutzt wird, um möglichst aktuelle Statusinformationen über die Servoeinheiten zu beziehen.

Sensorauswertung Die Auswertung der Sensoren erfolgt interruptgesteuert. Der Analog-Digital-Umsetzer wurde so eingestellt, dass er nach jeder Umsetzung eine Unterbrechungsanforderung auslöst. Somit kann eine Interruptserviceroutine

die neuen Daten auslesen und direkt dem Tiefpassfilter zur Rauschunterdrückung weiter reichen. Das Hauptprogramm kann somit zu jeder Zeit auf die schon gefilterten Daten zugreifen.

Echtzeitfähigkeit

Die geforderte weiche Echtzeitfähigkeit kann überprüft werden anhand eines Zählers für aufgetretene Zeitüberschreitungen. Dieser wird hochgezählt, falls bei einem Timeraufruf der Marker noch nicht zurückgesetzt war.

5.1.3 Modularisierung

Bewegungsmodule

Um eine einfache Erweiterbarkeit der Bewegungsmöglichkeiten des Roboters zu gewährleisten, wurde Wert darauf gelegt, dass sich Bewegungsmodule einfach austauschen und hinzufügen lassen. In der vorliegenden Implementierung können mehrere Bewegungsmodule integriert sein und über serielle Befehle einzeln aufgerufen werden. Hierbei werden auch für das jeweilige Modul spezifische Parameter übergeben.

Die Entscheidung darüber, wann ein neues Bewegungsmodul ausgeführt werden kann, obliegt hierbei dem momentan aktiven Bewegungsmodul. So kann dieses dafür sorgen, dass der Roboter in eine stabile Position zurückgeführt wird, bevor es die Kontrolle an ein anderes Modul abtritt.

Des Weiteren können mitunter mehrere Bewegungsmodule gleichzeitig laufen. Die macht Sinn, um zum Beispiel den Kopf unabhängig vom Rest des Roboters bewegen zu können. So wird das im Folgenden beschriebene HeadPath Modul vor dem aktiven Bewegungsmodul ausgeführt und setzt eine Kopfposition. Das aktive Bewegungsmodul kann nun entscheiden, ob es die Kopfposition überschreibt oder nicht, hat also eine höhere Priorität.

Es folgt eine Beschreibung der bereits implementierten Bewegungsmodule:

Walk Erzeugt die Laufbewegung (Details siehe 5.1.4).

SpecialAction Dieses Modul bewegt den Roboter in mehrere aufeinander folgende Zielpositionen. Die Zielpositionen wurden als Zustände einer Zustandsmaschine implementiert. Jeder Zustand besitzt die Informationen über Gelenkwinkelkoordinaten, Verweilzeit und zwei mögliche Nachfolgezustände. Beim Wechsel in einen neuen Zustand werden die Gelenkwinkelkoordinaten über lineare Interpolation auf die neuen Werte überführt. Das erfolgt in der durch die Verweilzeit gegebenen Zeit und diese hat somit direkten Einfluss auf die Bewegungsgeschwindigkeit. Nach Ablauf der Verweilzeit springt die Zustandsmaschine abhängig vom Empfang eines Abbruchsignals in den entsprechenden Nachfolgezustand. Dies ermöglicht es Bewegung bei Bedarf vorzeitig abzubrechen oder auch Schleifen zu definieren und diese dann wieder zu verlassen. Dabei sollte auch beim Erstellen des Abbruchpfades darauf geachtet werden, dass dieser stabil ausgeführt werden kann.

HeadPath Dieses Modul implementiert die Kopfbewegung analog zum SpecialAction Modul. Jedoch erlaubt es den direkten Abbruch einer Bewegung zu jeder Zeit. Das ist hier problemlos möglich, da es im Gegensatz zu Bewegungen des gesamten Roboters, bei der Kopfbewegung keine instabilen Positionen gibt.

5.1.4 Laufbewegungserzeugung

Die Erzeugung der Laufbewegung setzt sich aus zwei Teilen zusammen.

Als Erstes müssen die jeweils nächsten anzufahrenden Punkte auf den beiden Fußtrajektorien berechnet werden. Diese Punkte werden im dreidimensionalen Raum zuzüglich einer Dimension für die Fußdrehung um die Z-Achse (Yaw) berechnet.

Als Zweites muss der resultierende vierdimensionale Vektor dann in Gelenkwinkelkoordinaten umgerechnet werden. Die Gelenkwinkelkoordinaten stellen jedes Gelenk eines Beins dar, sind also in diesem Fall ein

sechsdimensionaler Vektor. Die Umrechnung in Gelenkwinkelkoordinaten erfolgt mit Hilfe der inversen Kinematik (IK). [20]

$$(x, y, z, yaw) \xrightarrow{IK} (hip_{roll}, hip_{pitch}, hip_{yaw}, knee_{pitch}, foot_{roll}, foot_{pitch})$$

Fußtrajektorien

Um bei der Berechnung der Fußtrajektorien für eine Laufbewegung Rechenzeit zu sparen, werden vorberechnete Tabellen verwendet (Bild 5.2). Die Werte aus diesen Tabellen geben für beide Füße die jeweilige Auslenkung in jeder Achse als Faktor an. Hierbei ist die Tabelle für die Y-Achsen-Auslenkung nach dem Zero-Moment-Point Kriterium berechnet. Es wurden die folgenden Formeln aus [5] zur Berechnung der Y-Achsen-Trajektorie der Füße y_{traj} verwendet:

$$y_{traj}(t_i) = y(t_i) - \frac{h}{g} \cdot \ddot{y}(t_i) \tag{5.2}$$

$$y(t_i) = C_1 \cdot e^{\sqrt{\frac{g}{h}} \cdot t_i} + C_2 \cdot e^{-\sqrt{\frac{g}{h}} \cdot t_i} + y_{ZMP} \tag{5.3}$$

$$C_1 = C_2 = -y_{ZMP} \cdot (e^{\sqrt{\frac{g}{h}} \cdot \frac{T}{4}} + e^{-\sqrt{\frac{g}{h}} \cdot \frac{T}{4}})^{-1} \tag{5.4}$$

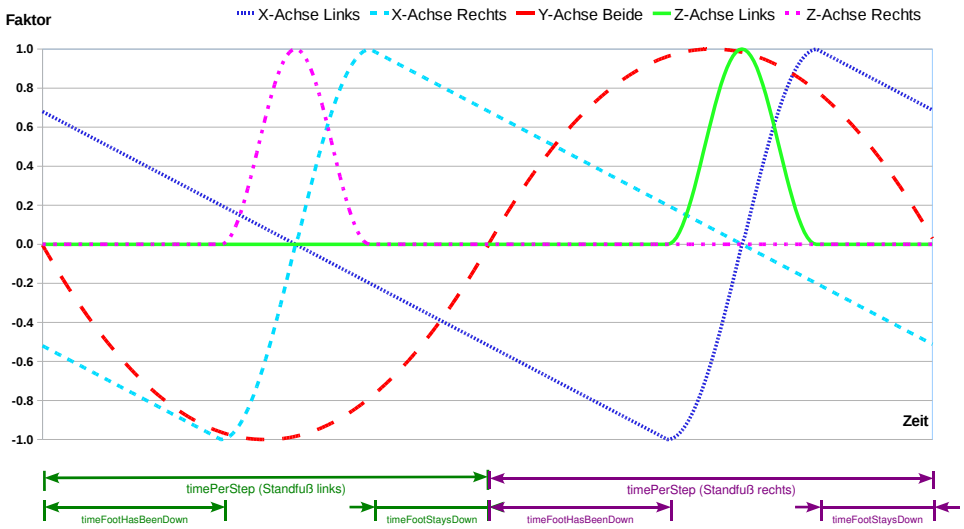
Hierbei sind y_{ZMP} die gewünschte Y-Position des ZMP, g die Gravitation, T die Dauer eines Schrittes, t der aktuelle Zeitschritt und h die Höhe des Schwerpunktes.

Zuzätzlich zu diesen Tabellen gibt es Laufparametersätze, welche die maximalen Auslenkungen der Füße in den drei Achsen und in der Rotation um die Z-Achse definieren.

Es folgt eine kurze Beschreibung der Laufparameter. Die Zeiteinheit ist Millisekunden, die Streckeneinheit ist Millimeter und die Winkeleinheit 0,29 Grad (Auflösung der Positionsencoder der Servoeinheiten).

timePerStep Zeitdauer eines einzelnen Schrittes (siehe Abb. 5.2)

Abbildung 5.2: Vorberechnete Tabellen zur Fußtrajektorienberechnung



timeFootHasBeenDown Zeitdauer die ein Fuß am Boden sein muss, bevor er der neue Standfuß wird (siehe Abb. 5.2)

timeFootStaysDown Zeitdauer die der ehemalige Standfuß noch am Boden bleibt, bevor er zum neuen Schritt abhebt (siehe Abb. 5.2)

stepLength Schrittlänge (maximale Distanz zwischen den Füßen in Richtung der X-Achse)

stepWidth Schrittweite (maximale Distanz zwischen den Füßen in Richtung der Y-Achse)

stepHeight Schritthöhe (maximale Distanz zwischen Fuß und Boden)

stepYaw Schrittdrehung (maximaler Winkel zwischen den Füßen um die Z-Achse)

hipSideAmplitude Hüftschwung (maximale Auslenkung der Hüfte in Richtung der Y-Achse)

footSideAmplitude Fußseitschwung (maximale seitwärtige Auslenkung des schwingenden Fußes)

footSideOffset Fußseitversatz (seitwärtiger Versatz des Standfußes)

forwardBackwardDiscrepancy Hüftversatz in Richtung der X-Achse als Anteil der Schrittlänge

Dabei gibt es zwei Laufparametersätze - jeweils einen für das Laufen in Richtung der X-Achse und der Y-Achse. Beim omnidirektionalen Laufen wird dabei je nach Laufrichtung zwischen diesen beiden Parametersätzen interpoliert. Weiterhin sollte ein flüssiger Übergang zwischen verschiedenen Laufrichtungen und -geschwindigkeiten ermöglicht werden. Dazu wird der aktuelle in einen neuen Laufparametersatz durch Interpolation über mehrere Schritte hinweg überführt. Dies dient auch dazu ein langsames Anlaufen zu ermöglichen.

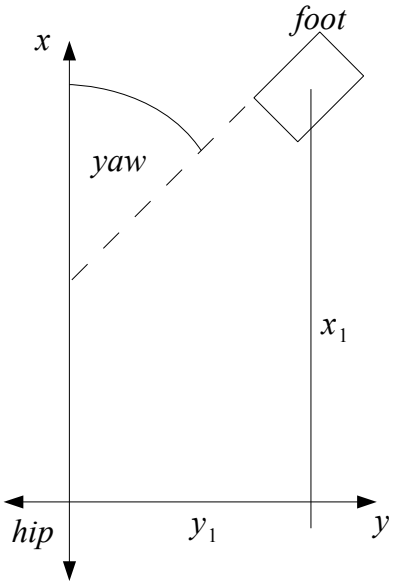
Inverse Kinematik

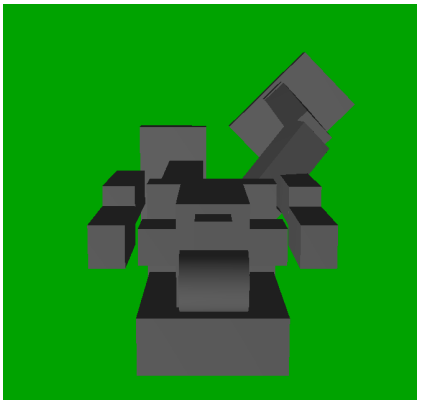
Die inverse Kinematik für die hier eingesetzten Roboter DD2007 und DD2008 wurde geometrisch ermittelt. Die Berechnung der Gelenkwinkelkoordinaten kann in drei logische Schritte unterteilt werden.

1. Schritt

   - Weise hip_{yaw} den Wert von yaw zu, da dies das einzige Gelenk ist, welches um diese Achse drehbar ist
   - Rechne die X- und Y-Koordinaten entsprechend der Yaw-Rotation des Fußes um

$$foot = (x_{1,}\, y_{1,}\, z_{1,}\, yaw)$$
$$hip_{yaw} = yaw$$
$$x_2 = x_1 \cos(yaw) + y_1 \sin(yaw)$$
$$y_2 = y_1 \cos(yaw) - x_1 \sin(yaw)$$
$$z_2 = z_1$$





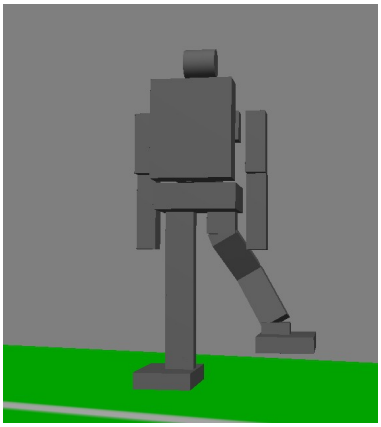
(Bild erstellt mit der am Fachgebiet eingesetzten humonoid Simulation [6])

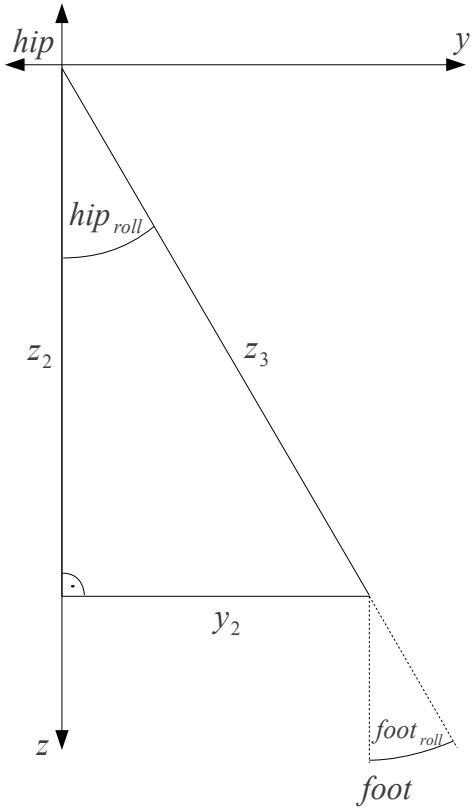
2. Schritt

   - Berechne hip_{roll} aus y_2 und z_2
   - Setze $foot_{roll} = -hip_{roll}$, um den Fuß waagerecht zu halten

$$x_3 = x_2$$
$$y_3 = y_2$$
$$z_3 = \sqrt{y_2^2 + z_2^2}$$

$$hip_{roll} = \arctan(y_2, z_2)$$
$$foot_{roll} = -hip_{roll}$$





(Bild erstellt mit der am Fachgebiet eingesetzten humonoid Simulation [6])

3. Schritt

   - Berechne hip_{pitch} und $knee_{pitch}$ aus x_3, z_3, $l_{upperleg}$ und $l_{lowerleg}$ (eine effiziente Berechnung wird ermöglicht, da Ober- und Unterschenkel ein gleichschenkliges Dreieck aufspannen)
   - Setze $foot_{pitch} = -hip_{pitch} - knee_{pitch}$, um den Fuß waagerecht zu halten

$$d_{foot} = \sqrt{z_3^2 + x_3^2}$$

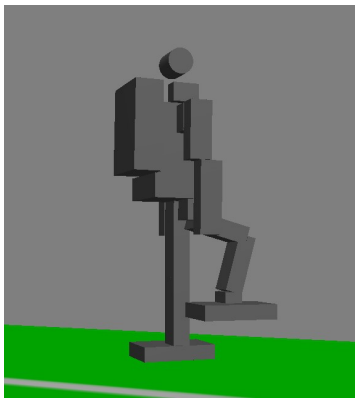
$$\alpha = \arctan(x_3, z_3)$$

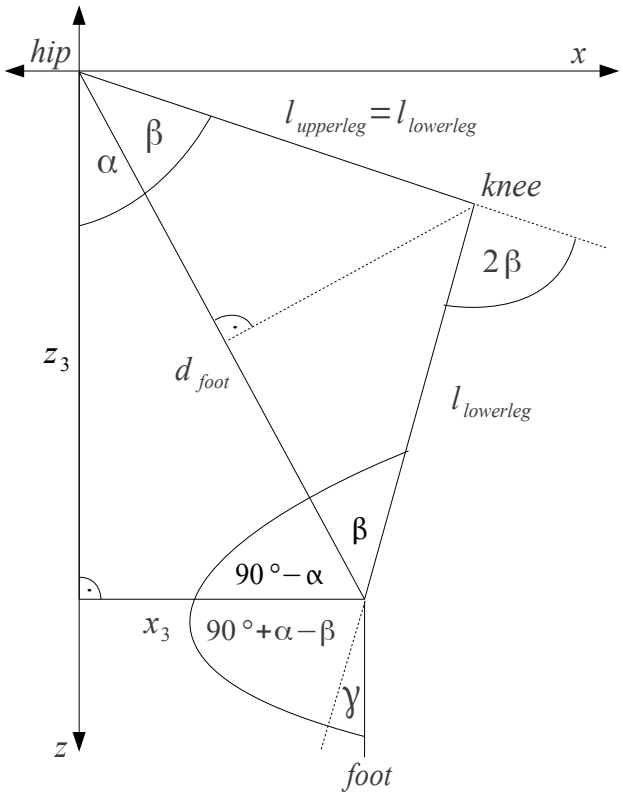
$$\beta = \arccos\left(\frac{d_{foot}}{2\, l_{upperleg}}\right)$$

$$hip_{pitch} = \alpha + \beta$$

$$knee_{pitch} = -2\beta$$

$$foot_{pitch} = \gamma = -\alpha + \beta$$





(Bild erstellt mit der am Fachgebiet eingesetzten humonoid Simulation [6])

5.1.5 Serielle Kommunikation

Serielle Kommunikation mit dem PC

Die Kommunikation mit der übergeordneten Steuereinheit erfolgt über einen seriellen Bus nach dem RS-232 Standard.

Serieller Bus (RS-232) Der serielle Bus nach dem RS-232 Standard ermöglicht voll-duplexe Kommunikation zwischen zwei Teilnehmern und hat sich zur Kommunikation mit eingebetteten System etabliert.

Serielles Protokoll Das serielle Protokoll zur Kommunikation mit der übergeordneten Steuereinheit wurde sehr einfach gehalten, um sowohl Rechenzeit und Speicherbedarf als auch den Kommunikationsoverhead gering zu halten. Implementiert wurde ein paketbasiertes Protokoll beginnend mit einem festen Framingbyte, gefolgt von einem minimalem Paketkopf (packet header) der nur die Paketlänge enthält. Der darauf folgende Paketkörper enthält eine Befehlsnummer, die Datenlänge und die Daten selbst. Um bei einem Übertragungsfehler eine erneute Synchronisation zu ermöglichen, darf der Paketkörper nie das Framingbyte enthalten.

Der Paketkörper wurde in einer ersten Implementierung hexadezimal Codiert. Das hat den Vorteil, dass die Daten mit einem einfachen Terminalprogramm les- und schreibbar sind und dass das Framingbyte nicht auftreten kann. In der frühen Entwicklungsphase konnten damit einfache Tests durchgeführt werden. Hierbei war wichtig, dass ein Fehler auf der PC-Seite ausgeschlossen werden konnte, um die möglichen Fehlerquellen zu reduzieren. Jedoch verdoppelt die hexadezimale Codierung die zu transportierenden Datenmengen.

Deshalb wurde nach ausreichenden Tests der Kommunikation, auf eine 8-Bit Codierung umgestellt. Diese hat hingegen den Nachteil, dass im Paketkörper jetzt das Framingbyte vorkommen kann. Folglich müssen die im Paketkörper auftretenden Framingbytes durch eine 16-Bit Escape-Sequenz ersetzt werden (byte stuffing). Diese Escape-Sequenz beginnt mit einem Escapezeichen, welches

auch sonst nie im Paketkörper vorkommen darf und selbst somit auch durch eine Escape-Sequenz ersetzt werden muss. Auch hier kann im schlimmsten Fall, bei einem Paket welches nur aus Framingbytes und Escapezeichen besteht, das Paket auf die doppelte Größe anwachsen. Statistisch gesehen ist jedoch die Wahrscheinlichkeit für das Auftreten eines der beiden zu escapenden Zeichen mit $1/128$ gering.

Serielle Kommunikation mit den Servoeinheiten

Die eingesetzten Servoeinheiten der Firma Robotis verwenden einen RS-485 Bus zur Kommunikation.

Serieller Bus (RS-485) Der serielle Bus nach dem RS-485 Standard in der hier verwendeten Ausführung, besitzt nur ein Adernpaar zur Datenübertragung und ist somit ein Halb-Duplex-Bus. Der Bus wird als Multipunktnetz benutzt, um eine Kommunikation zwischen mehreren Teilnehmern über eine gemeinsame Leitung zu ermöglichen. Hierbei darf zu einem Zeitpunkt immer nur ein Teilnehmer senden und alle anderen müssen empfangen. Deshalb fungiert ein Teilnehmer als Hauptsender (Master) und alle anderen als Nebensender (Slave). Der Hauptsender ist in diesem Fall die auf dem Renesas Mikrocontroller laufende Firmware und die Servoeinheiten sind die Nebensender.

Um die gegebene Bandbreite des Busses optimal auszunutzen, wurde bei der Implementierung auf ein schnelles Umschalten zwischen Sende- und Empfangsmodus Wert gelegt. Die gesamte Kommunikation erfolgt dementsprechend auf Interruptbasis, um die Kommunikation mit niedrigen Latenzen unabhängig von den Steuer- und Regelaufgaben des Systems betreiben zu können.

Um bei Kommunikationsfehlern nicht zu lange auf eine ausbleibende Antwort zu warten, wurde ein Zeitlimit für die Antworten der Servoeinheiten empirisch ermittelt. Das Zeitlimit wird realisiert über einen Zeitgeber der RTC, der bei einer Überschreitung des Zeitlimits eine Funktion des Kommunikationsmoduls aufruft.

So kann die Kommunikation fortgesetzt werden, auch wenn eine Servoeinheit nicht antwortet.

Serielles Protokoll Die Implementierung des seriellen Protokolls zur Kommunikation mit den Servoeinheiten richtet sich nach den von der Firma Robotis veröffentlichten Spezifikationen. [10, 11, 12]

Es handelt sich dabei um eine einfache Implementierung eines paketbasierten Protokolls. Der Paketstart wird durch zwei vorangestellte Framing-Bytes markiert. Darauf folgt ein Paketkopf (packet header) mit der Empfängeradresse (Bus ID) und der Paketgröße. Der nun anschließende Paketkörper (packet body) besteht aus den Nutzdaten und einem abschließendem Prüfsummenbyte.

5.1.6 Balanceregelung

Bei der Balanceregelung werden in mehreren Gelenken der Pitch- und Roll-Winkel angepasst, um eine Stabilisierung des Roboters zu erreichen. Die Berechnung der notwendigen Korrekturwerte erfolgt dynamisch auf den gefilterten Gyroskopdaten. Zum Einsatz kommt hier für jedes zu regelnde Gelenk ein PID-Regler. Die Einstellparameter der PID-Regler wurden empirisch ermittelt. Der Integralanteil wird jedoch derzeit noch nicht berücksichtigt, da die eingesetzten Gyroskope eine zu starke Drift aufweisen. Eine mögliche Lösung für dieses Problem wird in 8.2 diskutiert.

Der Stellwert u_t zum Zeitschritt t berechnet sich bei allen Gelenken nach der Formel:

$$u_t = K_p e_t + K_i E_t + K_d (e_t - e_{t-1})$$

Da der Sollwert w aus Abbildung 2.4 hier 0 ist, ergibt sich $e = -y$, also entspricht die Regeldifferenz e der negierten Regelgröße (Ist-Wert) y. Die Regelgröße y ist dabei die vom Gyroskop in der Achse dieses Gelenks gemessenen Winkelgeschwindigkeit.

Die Korrektur kann in den Sprung-, Hüft- und Schultergelenken jeweils im Pitch- und Roll-Winkel erfolgen. Die Auswahl der zu korrigierenden Gelenke erfolgt anhand der vom aktuellen Bewegungsmodul gelieferten Information über den momentanen Standfuß. Beim Stehen auf beiden Beinen wird in allen genannten Gelenken korrigiert. Während der Roboter nur auf einem Fuß steht wird nur in diesem Beim und in beiden Schultern korrigiert. Sollte es dazu kommen, dass der Roboter keinen Standfuß hat, also umgefallen ist, wird keine Balanceregelung vorgenommen.

5.1.7 Kopfreflex

Der implementierte 'Kopfreflex' dient dazu den Kopf in eine sichere Position zu bringen, bevor der Roboter beim Umfallen auf dem Boden auftrifft. Um eine sichere und rechtzeitige Erkennung des Umfallens zu ermöglichen wurden mit Hilfe des Debugger die Daten der Gyroskope und Beschleunigungssensoren beim Umfallen analysiert. Es zeigte sich, dass eine Betrachtung der Daten der Beschleunigungssensoren ausreichend ist, um den gestellten Anforderungen aus 4.1.7 zu genügen. Folglich wurde ein Schwellwert für die aus den Daten der Beschleunigungssensoren berechnete Neigung des Roboters ermittelt, ab welcher ein Umfallen unvermeidbar wird. In Kombination mit einer Hysterese wird dieser Schwellwert als Indikator für das Auslösen des Kopfreflexes benutzt. Implementiert wurde der 'Kopfreflex' im Programmfluss direkt vor der Ausgabe der Steuerbefehle an die Servoeinheiten, um ihm eine höhere Priorität als den Bewegungsmodulen zu verleihen.

5.1.8 Parametrisierung

Um die Parameter der einzelnen Module zugänglich zu machen, wurde eine zentrale Parameterverwaltung eingerichtet. Bei dieser kann sich jedes Modul mit seinen Parametern registrieren. Dabei wird jedem Parameter eine Identifikationsnummer zugeteilt, über welche der Parameter angesprochen wird.

So können alle Parameter von der übergeordneten Steuereinheit gelesen und geschrieben werden.

5.2 Unit-Tests

Zur Qualitätssicherung und Vermeidung von Fehlern wurden Unit-Tests benutzt. Diese ermöglichen das Testen der Funktionalität von einzelnen Modulen und auch deren Zusammenspiel.

Zur Umsetzung der Unit-Tests wird das CUnit Framework[1] benutzt. Dies ermöglicht eine einfache Implementierung von Testfällen und die Überprüfung der Ergebnisse. Es wurde ein Projekt 'Testing' angelegt, welches alle Testfälle beinhaltet. So können alle Tests durch Bauen und Ausführen dieses Projekts durchgeführt werden. Für alle neuen Module sollten auch Tests zum Projekt hinzugefügt werden, um deren Funktionalität überprüfbar zu machen.

5.2.1 Testlauf

Ein Testlauf erfolgt durch einfaches Compilieren und Ausführen des 'Testing' Projekts. Beispielhaft folgt hier die Ausgabe eines Tests des Moduls zur Berechnung der inversen Kinematik. Hierbei werden aus verschiedenen vorgegebenen Fußpositionen die korrespondierenden Gelenkwinkelkoordinaten von dem Modul berechnet und mit den manuell berechneten verglichen.

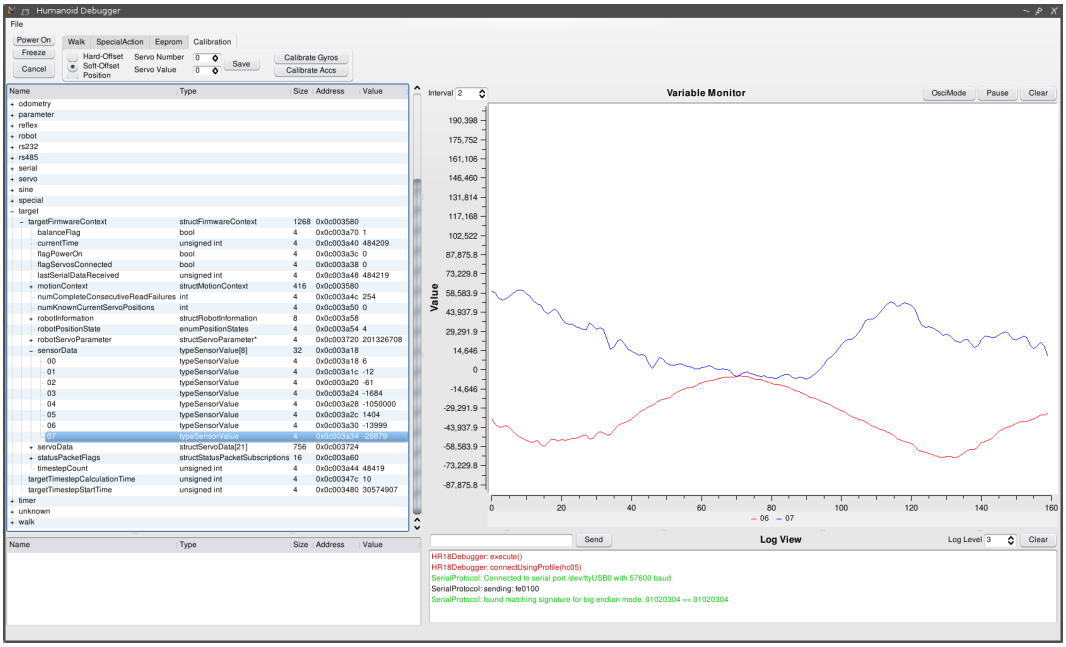
```
     CUnit - A Unit testing framework for C - Version 2.1-0
     http://cunit.sourceforge.net/

Suite: suiteKinematics
  Test: test of inverse kinematics (kinematicsFootPositionToLegAngles) ... passed

--Run Summary: Type      Total     Ran  Passed  Failed
               suites        1       1     n/a       0
               tests         1       1       1       0
               asserts      30      30      30       0
```

[1] http://cunit.sourceforge.net/

5.3 Debugger

Die Debuggerapplikation wurde in der Skriptsprache Python umgesetzt. Python bietet als weit entwickelte und gut unterstützte Hochsprache viele Vorteile. Dazu gehören unter anderem ein sehr kurzer Entwicklungszyklus, Plattformunabhängigkeit und eine Vielzahl von nutzbaren Modulen. Zur grafischen Ausgabe wurden hier die über die Pythonanbindung PyQt[2] nutzbaren plattformübergreifenden Qt-Bibliotheken[3] eingesetzt.

Abbildung 5.3: Die Humanoid Debugger Applikation



[2] http://www.riverbankcomputing.co.uk/
[3] http://www.qtsoftware.com/

5.3.1 Introspection

Um die Daten der Firmware übersichtlich darstellen zu können, ist es für den Debugger notwendig die Speicherbelegung der Firmware zu kennen. Weiterhin ist der Aufbau der Datenstrukturen von Interesse. Diese Informationen können automatisch aus den Debuginformationen des Compilers und des Linkers gewonnen werden. Hierzu wird die vom Linker generierte Speicherkarte (Map-File) eingelesen und die vom Compiler generierten Debuginformationen aus der Binärdatei extrahiert. Dies ermöglicht es alle globalen Variablen und Strukturen inklusive ihrer Elemente in einer Baumstruktur darzustellen.

Variablenbaum

Im Variablenbaum (Bild 5.3 Mitte links) können die Werte einzelner Variablen oder auch ganzer Strukturen durch ein einfaches Anklicken ausgelesen werden. Außerdem kann der Wert einer Variablen während des Betriebs geändert werden. Die Variablen können per Drag'n'Drop der Variablenüberwachung und dem Variablenplotter hinzugefügt werden.

Variablenüberwachung

Die Variablenüberwachung (Bild 5.3 unten links) zeigt den aktuellen Wert von ausgewählten Variablen in Textform an und wird bei Werteänderungen automatisch aktualisiert. Auch von hier aus kann der Wert einer Variable verändert werden.

Variablenplotter

Da bei schnellen Wertänderungen die Variablenüberwachung in Textform nicht gut lesbar ist, wurde der Variablenplotter (Bild 5.3 Mitte rechts) hinzugefügt. Dieser stellt den Wert von mehreren Variablen als Verlauf über die Zeit dar. Dies ermöglicht eine sehr übersichtliche Darstellung von z.B. den Sensorwerten.

5.3.2 Befehlsleiste

In der Befehlsleiste (Bild 5.3 oben links) können über verschiedene Knöpfe einige Befehle an die Firmware geschickt werden. Dies ermöglicht ein simples Testen der Grundfunktionalität und hilft z.B. bei der Kalibrierung der Sensoren.

5.3.3 Protokoll

Das Protokoll (Bild 5.3 unten rechts) ermöglicht das Mitlesen der über die serielle Verbindung ausgetauschten Nachrichten und Debugausgaben. Hierbei lässt sich die Ausgabe auf ein Prioritätslevel festlegen, um nur die wichtigsten Meldungen anzuzeigen.

6 Ergebnisse

6.1 Echtzeitfähigkeit

Der in 5.1.2 beschriebene Zähler für die Zahl der Zeitüberschreitungen wurde nach verschiedenen Testläufen des Systems ausgewertet. Hierbei wurde festgestellt, dass bei Verwendung des unter 5.1.1 beschriebenen Tiefpassfilters auf Fließkommabasis die Echtzeitfähigkeit des Systems nicht gewährleistet werden konnte. Nach der Umstellung des Tiefpassfilters auf Festkommabasis kam es zu keinen Zeitüberschreitungen mehr. Somit wurde empirisch gezeigt, dass das System mindestens eine weiche Echtzeitfähigkeit besitzt, solange ein geeignetes Ausführungsintervall für den Steuer- und Regelzyklus gewählt wird. Um eine geeignete Intervall-Länge zu ermitteln, wurde die Rechenzeit näher untersucht.

6.2 Rechenzeit

Um die Rechenzeit genauer zu überprüfen wurde eine Variable einfügt, welche auf die Zeitdifferenz zwischen Anfang und Ende des Steuer- und Regelzyklus gesetzt wird. Mit Hilfe des Variablenplotters des Debuggers wurde die Ausführungsdauer für einen Hauptschleifendurchlauf gemessen. Bei Ausführung des rechenzeitintensivsten Bewegungsmoduls zur Laufbewegungserzeugung mit inverser Kinematik liegt die Ausführungsdauer bei ca. 4 Millisekunden auf dem SH7211 und bei ca. 6 Millisekunden auf dem SH7145F.

Das Intervall für den Steuer- und Regelzyklus wurde auf 10 Millisekunden gesetzt. Somit ist es möglich noch zusätzliche Module auszuführen. Es wäre

zwar auch möglich das Intervall zu verkürzen, um häufiger Steuer- und Regelbefehle an die Servoeinheiten zu verschicken, begrenzender Faktor wäre dann aber die Bandbreite des RS-485 Busses. Da bei einem kürzeren Intervall mehr Steuerpakete verschickt würden, stünde weniger Bandbreite für die Statusabfragen der Servoeinheiten zu Verfügung.

6.3 Kommunikation mit den Servoeinheiten

Bei der Kommunikation mit den Servoeinheiten wurde vor allem untersucht, wie viele Statusabfragen pro Steuer- und Regelintervall neben dem Versenden der Steuerbefehle noch möglich sind. Dies ist interessant, da aktuelle Statusinformationen, speziell die momentanen Servopositionen, wichtige Daten für die übergeordnete Steuereinheit darstellen. Im hier betrachteten Anwendungsfall werden die momentanen Servopositionen benötigt, um zu den Kamerabildern der autonomen Roboter eine Kameramatrix berechnen zu können. Dabei kann durch häufigere Aktualisierung der Servopositionen eine genauere Berechnung ermöglicht werden.

Im Fall des DD2007 konnten neben dem Verschicken der Steuerbefehle noch 9 Statusabfragen in einem Intervall von 10 Millisekunden gemacht werden.

Beim DD2008 konnten deutlich höhere Werte erzielt werden. Theoretisch wäre ein bis zu viermal so hoher Durchsatz zu erreichen, da sowohl die Zahl der Busse, als auch die Baudrate verdoppelt wurde. Jedoch müssen auch hier feste Wartezeiten abgezogen werden, die von der höheren Baudrate unbeeinflusst bleiben. Dazu gehört das Umschalten zwischen Sende- und Empfangsmodus und die Verarbeitungszeit der Steuerbefehle durch die Servoeinheiten. Praktisch wurde eine Erhöhung der Zahl der Statusabfragen auf 22 pro Intervall gemessen.

6.4 Balanceregelung

Die Ergebnisse der Balanceregelung objektiv zu messen ist schwierig. Bei einem fußballspielenden humanoiden Roboter ist aber die Balance von erheblicher Bedeutung für seine Einsatzfähigkeit. Es ist erforderlich insbesondere während schneller Bewegungen wie dem Laufen oder Schießen auch unter störendem Einfluss der Gegenspieler das Gleichgewicht zu behalt. Daher kann der erfolgreiche Einsatz der in dieser Arbeit entwickelten Software auf den Robotern der Darmstadt Dribblers[1] beim RoboCup 2008 als positives Ergebnis angesehen werden.

Auch ein einfaches Anstoßen des stehenden Roboters mit und ohne Balanceregelung macht deren Wirkung deutlich sichtbar. Bei einem leichten Stoß ohne Balanceregelung pendelt der Roboter mehrere Male über seine Ausgangsposition hin- und zurück, bevor er zur Ruhe kommt. Mit Balanceregelung findet er hier seine Ausgangsposition deutlich schneller wieder. Durch einen stärkeren Stoß ist er ohne Balanceregelung erheblich einfacher umzuwerfen, als mit.

[1] http://dribblers.de/

7 Zusammenfassung

Die in dieser Diplomarbeit erstellte Software eignet sich als Mikrocontrollerfirmware für die Steuerung mobiler autonomer Roboter. Sie wurde vor dem Hintergrund der Teilnahme am Fußballturnier in der humanoiden Kid-Size Liga des RoboCup entwickelt, ist jedoch darauf nicht beschränkt. Durch die Verwendung eines modularen Aufbaus ist sie flexibel und erweiterbar insbesondere was die Bewegungserzeugung betrifft. Daher eignet sie sich auch für den Einsatz auf Robotertypen mit anderer Kinematik oder Robotern mit völlig anderen Zielsetzungen.

Ein weiterer Vorteil des modularen Aufbaus liegt in der Möglichkeit Module zu entwickeln, ohne die tiefgehenden plattformspezifischen Teile der Software verstehen zu müssen. So arbeiten bereits mehrere Praktikumsgruppen an verschiedenen Modulen zur Erweiterung der Funktionalität.

Um eine einfache Portierung auf neue Systeme zu ermöglichen wurde eine strikte Trennung zwischen plattformspezifischen und plattformunabhängigen Modulen eingehalten. Eine erste Portierung wurde schon während der Entwicklung durchgeführt, als ein zweites Robotermodell mit neuerem Mikrocontroller angeschafft wurde.

Dabei erwies sich der grafischer Debugger als sehr hilfreich. Sobald die serielle Kommunikation auf der neuen Plattform eingerichtet war, konnte er verwendet werden, um die weiteren plattformspezifischen Module zu debuggen. Er eignet sich aber auch um die plattformunabhängigen Module im realen Betrieb auf dem Mikrocontroller zu debuggen, oder Daten der Sensoren in Echtzeit zu visualisieren.

Sowohl bei der Entwicklung des Debuggers als auch bei der zum Compilieren genutzten Toolchain wurde darauf geachtet, Betriebsystemunabhängig zu bleiben.

So können Entwickler unter Windows, MacOS oder Linux arbeiten. Es wurde ausschließlich für mehrere Betriebssysteme verfügbare Software benutzt: Python, Qt, CMake und GCC. Einzig zum Flashen der Renesas Mikrocontroller konnte keine Software gefunden werden, welche nicht Windows voraussetzt. Das Renesas Flash Development Toolkit kann jedoch auch auf einer Windows-Installation in einer mit VirtualBox aufgesetzten virtuellen Maschine verwendet werden.

8 Ausblick

8.1 Bewegungsmodule

Um eine einfache Erweiterung der Bewegungserzeugung zu ermöglichen wurde viel Wert auf die Modularisierung in diesem Bereich gelegt. Hier gibt es eine Vielzahl von interessanten Erweiterungsmöglichkeiten:

- Armbewegungen über inverse Kinematik (bereits in Arbeit)
- Schussbewegungen über inverse Kinematik
- direktere Übergänge zwischen Laufmodul und Schussbewegung

8.2 Balanceregelung

Bei der Balanceregelung kommt im Moment nur ein PD-Regler zum Einsatz, da der Integralanteil aufgrund der Drift der Gyroskope unbrauchbar ist. Über die kombinierte Filterung von Beschleunigungssensor- und Gyroskopdaten mit Hilfe eines Kalmanfilters ist es möglich die Drift der Gyroskope automatisch zu korrigieren. Dies würde es zum Beispiel erleichtern eine Regelung der Drehung des Roboters um die Z-Achse zu implementieren und so einen Ausgleich für ungewollte Rotationen bei der Laufbewegung ermöglichen.

Weiterhin besteht die Möglichkeit die Einstellparameter der PID-Regler durch ein Optimierungsverfahren zu verbessern. Dazu müssen aber noch geeignete Gütekriterien und Bewertungsverfahren gefunden werden, um einen Parametersatz automatisiert bewerten zu können.

8.3 Fußkontaktkraftsensoren

Durch das Anbringen von Fußkontaktkraftsensoren könnten gleich mehrere Vorgänge vereinfacht oder verbessert werden:

- Detektion des momentanen Standfußes
- Erkennung ob der Roboter umkippt
- Finden einer stabilen Standposition

A Praktischer Einsatz

A.1 Compilieren

Ein nicht zu unterschätzender Aufwand wurde bei dieser Arbeit bei der Suche nach geeigneten Entwicklungswerkzeugen betrieben. Da die benutzten Mikrocontroller vor allem im asiatischen Raum Verwendung finden, fällt die Unterstützung in deutscher und englischer Sprache gering aus.

A.1.1 Toolchains

Zum Compilieren der Firmware für die SuperH Mikrocontroller wurde eine auf dem GNU C Compiler (GCC) aufsetzende Toolchain (GNU-SH-ELF) in Version 8.03 benutzt. Diese wird frei zugänglich und quelloffen für Linux und Windows von der Firma KPIT Cummins Infosystems Limited bereitgestellt. Nähere Informationen hierzu findet man unter `http://www.kpitgnutools.com/`.

Um die Unit Tests und die Firmware für Linux zu kompilieren wurde der GNU C Compiler in Version 4.3 eingesetzt.

Die Unit Tests und die Firmware für Windows sowie die dynamische Bibliothek für den Betrieb mit dem Simulator können sowohl mit Microsoft Visual C als auch mit der Minimalist GNU for Windows (MinGW [1]) Toolchain kompiliert werden.

[1] `http://www.mingw.org/`

A.1.2 Projektverwaltung

Die Projektverwaltung wurde durch das plattformübergreifende quelloffene Build-System CMake[2] realisiert. Dies ermöglicht es das Projekt unter verschiedenen Plattformen zu bauen und auch Projektdateien für verschiedene Entwicklungsumgebungen zu erstellen. Unter anderem können Projektdateien für Eclipse CDT und Microsoft Visual Studio 2005/2008 automatisch generiert werden. Zusätzlich wurden auch Projektdateien für die proprietäre frei erhältliche Entwicklungsumgebung Highperformance Embedded Workshop (HEW) der Firma Renesas Technology Corp. erstellt.

A.2 Flashen

Die compilierte Firmware kann mit Hilfe des Flash Development Toolkits (FDT) von Renesas in den Flashspeicher der beiden Mikrocontroller kopiert werden. Da allerdings die Flashspeicher nur auf eine geringe Zahl von Schreibvorgängen ausgelegt sind und außerdem der Schreibvorgang recht lange dauert wurde auf die Möglichkeit zurückgegriffen, die Firmware per Bootloader in den externen RAM des Mikrocontrollerboards zu kopieren und von dort auszuführen. Dazu wird in den Flashspeicher ein Bootloaderprogramm kopiert, welches nun nach jedem Reset ausgeführt wird. Dieses kann dann per X-Modem-Übertragung eine speziell für die Ausführung im RAM compilierte Firmware in den RAM schreiben und starten. Hierbei dauert die Übertragung nur ein Drittel so lang, wie der entsprechende Schreibvorgang in den Flashspeicher. Jedoch ist der externe RAM beim Lesen nur halb so schnell wie der interne Flashspeicher und somit halbiert sich die Ausführungsgeschwindigkeit der Firmware.

[2] `http://www.cmake.org/`

A.3 Verzeichnisstruktur

Hier eine kurze Übersicht über die Verzeichnisstruktur der in dieser Arbeit entwickelten Software.

/**documents** Datenblätter und Anleitungen zu den verwendeten Hardwarekomponenten

/**tools** Hilfsskripte

/**tools**/**HR18Debugger** Python Code der Debuggerapplikation

/**tools**/**RobotisTool** Python Code einer Applikation zum Konfigurieren der Servomotoren der Firma Robotis

/**HR18**/**humanoid_flash** Flashprojekt für das Renesas Flash Development Toolkit

/**HR18**/**HR18Firmware**/**src** C Code für die Firmware mit plattformspezifischem Code in den Unterverzeichnissen 'hc03', 'hc05', 'linux', 'simWin32' und 'win32', sowie dem Unittest Code in 'testing'

/**HR18**/**HR18Firmware**/**build** Projektdateien der verschiedenen Buildsysteme, wobei 'cmake' inzwischen alle anderen ersetzt

Literaturverzeichnis

[1] Renesas Technology Corp. *Renesas SH-1, SH-2, SH-DSP Software Manual*. Renesas Technology Corp., 5.00 edition, 2004.

[2] Renesas Technology Corp. *Renesas SH7144 Group, SH7145 Group Harware Manual*. Renesas Technology Corp., 3.00 edition, 2004.

[3] Renesas Technology Corp. *Renesas SH-2A, SH2A-FPU Software Manual*. Renesas Technology Corp., 3.00 edition, 2005.

[4] Renesas Technology Corp. *Renesas SH7211 Group Harware Manual*. Renesas Technology Corp., 1.00 edition, 2006.

[5] M. Friedmann, J. Kiener, S. Petters, H. Sakamoto, D. Thomas, and O. von Stryk. Versatile, high-quality motions and behavior control of humanoid soccer robots. In *Proc. Workshop on Humanoid Soccer Robots of the 2006 IEEE-RAS Int. Conf. on Humanoid Robots*, pages 9–16, Genoa, Italy, Dec. 4-6 2006.

[6] M. Friedmann, K. Petersen, and O. v. Stryk. Adequate motion simulation and collision detection for soccer playing humanoid robots. In *Proc. 2nd Workshop on Humanoid Soccer Robots at the 2007 IEEE-RAS Int. Conf. on Humanoid Robots*, Pittsburgh, PA, USA, Nov. 29 - Dec. 1 2007.

[7] M. Friedmann, K. Petersen, and O. von Stryk. Simulation of multi-robot teams with flexible level of detail. In S. Carpin et al., editor, *Simulation, Modeling and Programming for Autonomous Robots (SIMPAR 2008)*, number 5325 in

Lecture Notes in Artificial Intelligence, pages 29–40, Venice, Italy, November 2008. Springer.

[8] Paul Hamill. *Unit test frameworks*. O'Reilly, 2004.

[9] Uwe Brinkschulte Heinz Wörn. *Echtzeitsysteme: Grundlagen, Funktionsweisen, Anwendungen*. Springer, 2005.

[10] Robotis Inc. *Dynamixel DX-113, DX-116, DX-117 User's Manual*. Robotis, Inc., 2005.

[11] Robotis Inc. *Dynamixel RX-64 User's Manual*. Robotis, Inc., 2006.

[12] Robotis Inc. *Dynamixel RX-28 User's Manual*. Robotis, Inc., 2007.

[13] David Iseminger. *Microsoft Win32 Developer's Reference Library*. Microsoft Press, 2000.

[14] Brian W. Kernighan and Dennis M. Ritchie. *The C programming language / Brian W. Kernighan, Dennis M. Ritchie*. Prentice-Hall, Englewood Cliffs, N.J., 1978.

[15] Jan Lunze. *Regelungstechnik 1 - Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen*. Springer Verlag, 2008.

[16] Jan Lunze. *Regelungstechnik 2 - Mehrgrößensysteme, Digitale Regelung*. Springer Verlag, 2008.

[17] Martin Meyer. *Signalverarbeitung - Analoge und digitale Signale, Systeme und Filter*. Verlag Vieweg, 1998.

[18] *System Application Program Interface (API) [C Language]*. Information technology—Portable Operating System Interface (POSIX). 1990.

[19] M. Seebode and W. Gerth. Echtzeitsystem für einen zweibeinigen Roboter mit adaptiver Bahnplanung. In P. Holleczek and B. Vogel-Heuser, editors,

Mobilität und Echtzeit. PEARL 2007, pages 88–97, Berlin Heidelberg, 2007. Springer.

[20] Prof. Dr. Oskar von Stryk. *Robotik 1*. TU Darmstadt - Fachgebiet Simulation und Systemoptimierung, 2004.

[21] Miomir Vukobratovic and Branislav Borovac. Zero-moment-point - thirty five years of its life. pages 157–173. 2004.