technische universität
dortmund

Diplomarbeit

Analyse und Kompensation laufbezogener
Effekte zweibeiniger Roboter

Oliver Urbann
oliver.urbann@tu-dortmund.de

2. September 2010

Prüfer:

Prof. Dr. Uwe Schwiegelshohn

Abteilung Informationstechnik
Institut für Roboterforschung

Prof. Dr. Günter Rudolph

Lehrstuhl Algorithm Engineering
Fakultät für Informatik

INHALTSVERZEICHNIS

1

EINLEITUNG

1.1 Motivation

In der Robotik stellen humanoide Roboter einen besonderen Forschungszweig dar. Aufgrund ihrer Ähnlichkeit zum Menschen erfahren sie eine hohe Akzeptanz (KIIO04) und können sich in menschlichen Umgebungen besser bewegen, wobei ihre menschenähnliche Gestalt hilfreich ist. Gleichzeitig ist das aber ein schwieriges Forschungsfeld, da die Fortbewegung auf zwei Beinen, insbesondere für größere oder günstigere Roboter, aufgrund mechanischer Probleme eine größere Schwierigkeit darstellt. Dazu kommen Störungen des Laufs, die für menschliche Umgebungen normal sind und für den Menschen keine Schwierigkeit darstellen. Unebene Böden wie Pflastersteine oder Wellen im Teppich sind Beispiele dafür. Stöße von Menschen oder anderen Robotern müssen ebenfalls ausgeglichen werden können.
Der Robocup[1] stellt einen Benchmark für Robotik dar, insbesondere für das zweibeinige Laufen. Er verfolgt das Ziel im Jahr 2050 gegen den Fußball-Weltmeister mit einem Team aus Robotern anzutreten. Von einem Spiel unter realistischen Bedingungen ist der Wettbewerb aber noch weit entfernt. Neben idealisierten Bedingungen für die Bilderkennung wird in den humanoiden Ligen die Herausforderung auch für das zweibeinige Laufen abgemildert, indem ein möglichst ebener und relativ glatter Teppichboden als Spielfeld verwendet wird. Dennoch lassen sich gewisse Unebenheiten und Kollisionen mit anderen Robotern nicht vermeiden, so dass der Robocup nach wie vor eine Herausforderung bei der Laufforschung ist.

[1] http://www.robocup.org

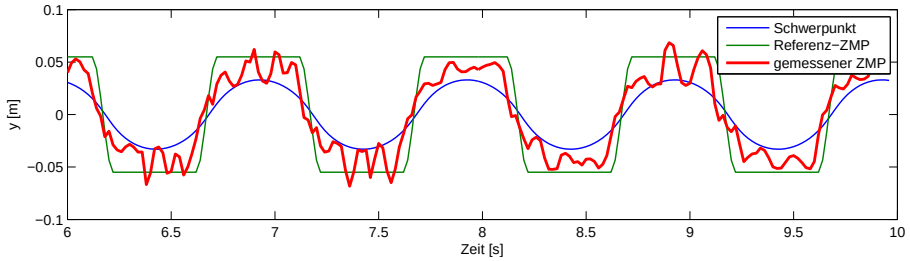


Abbildung 1.1: Abweichung des gemessenen ZMP vom Soll-ZMP bei einem Lauf mit einer Geschwindigkeit von $5\frac{cm}{s}$ in SimRobot.

Speziell in der „Standard Platform League“ ist die Entwicklung von guten Laufbewegungen von großer Wichtigkeit, da hier alle Teilnehmer die gleiche Plattform, den zweibeinigen Roboter Nao der französischen Firma Aldebaran Robotics, verwenden und nicht verändern dürfen. Existierende Probleme müssen daher softwareseitig gelöst werden. Das Team „Nao Devils Dortmund“ [2] nimmt ebenfalls in dieser Liga teil. Es setzt seit 2008 die „Dortmund Walking Engine“ (kurz DWE) zur Erzeugung der Laufmuster ein.

Beobachtungen

In ihrer derzeitigen Version sind einige Ungenauigkeiten beim Lauf zu beobachten, teilweise auch schon in der Simulation SimRobot [3] (LSR06).

1a. Bei einer Schrittdauer[4] im Bereich von 1 Sekunde lässt sich ein Aufschwingen des Roboters entlang der x-Achse beobachten (im Folgenden ***Vorwärtsschwingung*** genannt). Bei langsameren Laufbewegungen tritt das Aufschwingen weniger stark auf und bei 2.5 Sekunden ist es nicht mehr zu erkennen.

1b. Etwas Ähnliches zeigt sich entlang der y-Achse, die ***Seitwärtsschwingung***. Sie tritt allerdings erst bei einer Schrittdauer von 1.5 Sekunden bis 2.3 Sekunden auf. Gemein ist den beiden Vorgängen, dass sie in SimRobot nicht zu beobachten sind, sondern nur auf dem realen Roboter.

2. Nicht nur auf dem realen Roboter, sondern auch in SimRobot ist ein anderes Problem zu erkennen. So folgt der im SimRobot gemessene Zero Moment Point (kurz ZMP) nicht exakt dem Soll-ZMP (***ZMP-Abweichung***). Der ZMP ist der Punkt am Boden, an dem die Drehmomente um die x- und y-Achse 0 betragen. Somit ist der Roboter stabil, sofern der ZMP innerhalb der konvexen Hülle der Kontaktpunkte vom Roboter mit dem Boden liegt (VB04), Details folgen in Abschnitt 2.4.2. In

[2] http://www.nao-devils.de

[3] SimRobot ist eine Simulation speziell für den Roboterfußball, entwickelt vom DFKI in Bremen. Er basiert auf der Festkörpersimulation „Open Dynamics Engine“ (Smi02), welche in Abschnitt 2.2 vorgestellt wird.

[4] Die Schrittdauer bezeichnet die Dauer eines Doppelschritts, also jeweils zwei Phasen mit einem bzw. zwei Füßen am Boden, und wird in Abschnitt 2.4.4 näher erläutert.

Abbildung 1.1 sieht man den Verlauf des gemessenen ZMP im Vergleich zu dem Referenz-ZMP.

3. Die Problematik beschränkt sich nicht nur auf den inkorrekten ZMP-Verlauf, sondern resultiert in einem mehr oder minder starken ***Einknicken*** des Roboters um die x-Achse in Richtung des Schwungbeins[5], welches dadurch den Boden unbeabsichtigt berühren kann. Das führt beim realen Roboter bei hohen Geschwindigkeiten oft zum Sturz. Sollte eine solche Berührung nicht zum Sturz führen, verursacht sie aber mit hoher Wahrscheinlichkeit eine Änderung der Laufrichtung.

4. Neben der Rotationsgeschwindigkeit werden ebenfalls die Translationsgeschwindigkeiten nicht richtig umgesetzt. Während der Roboter in SimRobot eine geringere Geschwindigkeit als gewünscht erreicht, läuft der reale Roboter **schneller** als vorgegeben, vor allem bei hohen vorgegebenen Geschwindigkeiten. Diese Beobachtung wird als ***Geschwindigkeitssteigerung*** bezeichnet.

Diese fünf Punkte stellen die derzeit wichtigsten Beobachtungen dar, die die Stabilität des Roboters beim Laufen gefährden, die langsame Bewegungen nur bedingt zulassen und die vorgegebene Geschwindigkeit nur ungenau umsetzbar machen.

1.2 Aufgabenstellung

Diese Arbeit hat zum Ziel, die Ursachen der Beobachtungen zu **identifizieren**, um die Auswirkungen zu **kompensieren**.
Für diese Arbeit wurde eine formalisierte Einteilung der Abstraktionsgrade entwickelt, die dabei hilft die Ziele festzulegen und die zu deren Erreichung notwendigen Werkzeuge zu entwickeln und anzuwenden. Sie wird im Folgenden vorgestellt.

Abstraktionsgrade

Abbildung 1.2 zeigt den Zusammenhang zwischen Software zur Laufmustererzeugung (im Folgenden Walking Engine) und Simulatoren bzw. realen Robotern. Hier soll verdeutlicht werden, was die Ein- und Ausgabe der beiden Richtungen ist, und inwiefern diese von den verwendeten Modellen beeinflusst wird. Ein realer oder simulierter Roboter hat als Eingabe die gewünschten Gelenkwinkel, Winkelgeschwindigkeiten, Drehmomente oder ähnliches, durch deren Ausführung eine Kinematik und Dynamik folgt bzw. errechnet wird. Diese Berechnungsrichtung wird im Folgenden als **Sim-Richtung** bezeichnet. Die Walking Engine ist informell gesehen die umgekehrte Funktion davon und wird als **WE-Richtung** bezeichnet. Sie hat als Eingabe zum Beispiel den gewünschten

[5] Schwungbein kennzeichnet das Bein während einer Phase mit nur einem Fuß am Boden, das keinen Kontakt zum Boden hat. Details folgen in Abschnitt 2.4.

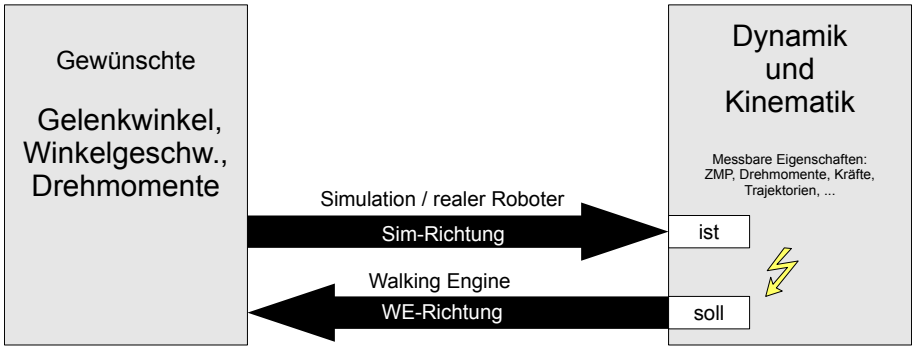


Abbildung 1.2: Berechnungsrichtung der Walking Engine und eines simulierten bzw. realen Roboters mit den jeweiligen Ein- und Ausgaben.

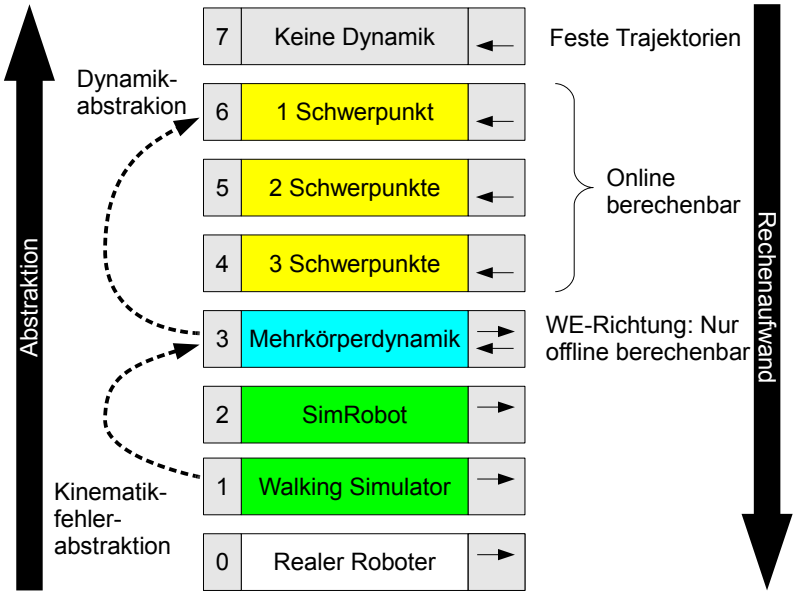


Abbildung 1.3: Sieben Stufen von Modellen des realen Roboters. Die Stufen sind durchnummeriert nach dem Grad der Abstraktion, und mit Richtungspfeilen markiert, die die gewöhnliche Verwendung zeigen (Pfeil nach links: WE-Richtung, Pfeil nach rechts: Sim-Richtung).

ZMP-Verlauf, dazu einige kinematische Bedingungen, wie vertikaler Oberkörper, und erzeugt daraus die Soll-Werte. Verwendet man für die Walking Engine und Simulation das gleiche Modell, folgt üblicherweise daraus, dass die der Walking Engine vorgegebenen Werte und Bedingungen auch vom Simulator so wiedergegeben werden. Bezogen auf das oben genannte Beispiel wäre das ein vertikaler Oberkörper und ein Ist-ZMP, der exakt dem gewünschten ZMP folgt. Sind die verwendeten Modelle unterschiedlich, sind Abweichungen sehr wahrscheinlich, was bedeutet, dass ein realer Roboter nicht den Vorgaben exakt folgen wird, da ein Modell immer eine Abstraktion darstellt. Einen Überblick über die für diese Arbeit wichtigen Abstraktionsgrade gibt Abbildung 1.3. Dem realen

Roboter kommt hierbei die Ebene 0 zu und ist ein Spezialfall, da er kein Modell ist, und damit nur für die Sim-Richtung verwendet werden kann.

Als nächstes folgt der Simulationsblock, in der Abbildung 1.3 grün dargestellt. Dazu gehört SimRobot (LSR06), ein Simulator für Roboterfußball, basierend auf der Open Dynamics Engine (ODE) (Smi02), welche die Bewegung von Festkörpern im Raum durch Lösen von Bewegungsgleichungen berechnet. Details folgen in Abschnitt 2.2. Simulationen sind immer eine Abstraktion von der Wirklichkeit, unterscheiden sich aber beim verwendeten Modell. Physiksimulationen für Roboter, wie die hier erwähnten, oder auch für Computerspiele, simulieren z.B. nicht die elektromagnetischen Felder des Motors oder den Luftwiderstand. Prinzipiell wäre das von einer Simulation genutzte Modell auch für eine Walking Engine anwendbar, aber zu komplex, um eine ausreichend effiziente Online-Berechnung zu ermöglichen.

Ein Modell unterhalb der Simulation ist die Mehrkörperdynamik (oder auch Mehrkörpermodell genannt, kurz MKM). Bei einem solchen Modell werden kein Kippen oder Rutschen und keine Motoren sowie deren Ansteuerung nachgebildet. Außerdem sind alle Teile starr und unterliegen keiner Reibung. In Bezug auf die Dynamik ist es allerdings so genau wie eine Simulation und betrachtet alle Schwerpunkte und Trägheitstensoren der Festkörper. Es eignet sich daher noch für die Sim-Richtung, wurde aber von Vukobratović auch zur Verwendung in einer Walking Engine vorgeschlagen (Vuk90). Hierbei liegen gewünschte Fußpositionen vor, woraus sich Vorgaben an den ZMP-Verlauf ergeben, der dann in Soll-Gelenkwinkel umgesetzt wird. Die dazu aufzustellenden Gleichungen können allerdings nicht in Echtzeit gelöst werden.

Verschiedene Autoren haben sich daher darauf konzentriert einfachere, linearisierbare Modelle zu entwickeln, die auch online berechnet werden können. Dabei reicht die Abstraktion von Modellen mit drei Schwerpunkten (BLB$^+$07), über solche mit zwei Schwerpunkten (SE07), bis hin zu einem Schwerpunkt (KKK$^+$03a). Letztere wird auch für die derzeit im Team Nao Devils Dortmund[6] verwendete Walking Engine (CKU09b; CKU09a) genutzt, die in Abschnitt 2.4 vorgestellt wird. Sie basiert auf der Arbeit von Kajita et al. (KKK$^+$06), und wurde um eine Sensorkontrolle zur Ausregulierung unvorhergesehener Störungen erweitert. Für gewöhnlich werden Modelle mit einer festen Anzahl von Schwerpunkten nur für die WE-Richtung verwendet.

Ein weiterer Spezialfall ist die Ebene 7 der festen Trajektorien. Dabei folgen die Arme und Füße vorher festgelegten Bahnen, die von Hand oder von evolutionären Algorithmen bestimmt werden. Zur Laufmustererzeugung finden also keine physikalischen Berechnung (bis auf eventuelle inverse Kinematiken) statt. Beispiele dafür sind die Walking Engine der Firma Aldebaran Robotics, die dem humanoiden Roboter Nao (siehe Abschnitt 2.1) mitgeliefert wird, oder die Walking Engine von Ralf Kosse (Kos06).

[6] http://www.nao-devils.de

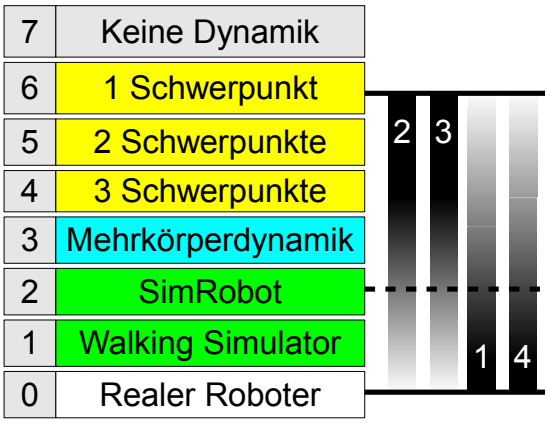


Abbildung 1.4: Zuordnung der Beobachtungen zu Bereichen der Abstraktionen, bei denen die Ursachen wahrscheinlich zu finden sind.

Identifikation von Ursachen

Die Identifizierung von Ursachen für die Beobachtungen soll durch Betrachtung der in Abbildung 1.3 dargestellten Abstraktionsebenen geschehen. SimRobot ermöglicht eine erste Einteilung, die in Abbildung 1.4 gezeigt wird.
Beobachtung *ZMP-Abweichung* und *Einknicken* zeigen sich auch in SimRobot und sind daher vermutlich eine Folge der Abstraktion von der Ebene SimRobot zur Ein-Schwerpunkt-Ebene. Dagegen sind die Beobachtungen *Vorwärtsschwingung*, *Seitwärtsschwingung* und *Geschwindigkeitssteigerung* in SimRobot nicht zu erkennen. Ihre Ursachen liegen daher vermutlich bei den Elementen, von denen SimRobot im Vergleich zum realen Roboter abstrahiert.
Aufgrund des hohen Aufwandes einer Umsetzung einer neuen Walking Engine mit komplexerem Modell soll in dieser Arbeit ausschließlich die bestehende Walking Engine zur Laufmustergenerierung genutzt werden.
Da alle Beobachtungen auch bei Geradeausläufen gemacht werden können, ist es möglich, die Komponenten auf einen Geradeauslauf zu beschränken.
Die folgenden Komponenten sollen entwickelt werden:

Erste neue Ebene: Im ersten Schritt soll statt SimRobot eine Berechnung der Mehrkörperdynamik entwickelt und implementiert werden. Hiermit soll festgestellt werden, welche Ungenauigkeiten bei der Abstraktion von einem Mehrkörpersystem zu einem einzelnen Schwerpunkt entstehen. In Abbildung 1.3 ist diese Abstraktion als **Dynamikabstraktion** bezeichnet. Der hier benötigte Algorithmus zur Berechnung der Dynamik mit Hilfe des Mehrkörpermodells wird in der Literatur als inverse Dynamik bezeichnet (Fea07).

Zweite neue Ebene: Simulationen haben den wesentlichen Vorteil, dass man Situationen nachstellen kann, Daten sich rauschfrei aufzeichnen lassen und der reale Roboter nicht verschleißt. Zudem führt in der Realität die Gesamtheit aller Einflüsse und Komponenten wie Reibung, Getriebe, Motoren usw. zum beobachtbaren physikalischen Verhalten. In einer Simulation hingegen lassen sich Einzelheiten getrennt von anderen betrachten. Es soll daher ein neuer Simulator, im Folgenden als „Walking Simulator“ (kurz WS) bezeichnet, entwickelt, implementiert und geeignet parametrisiert werden. Dieser wird dann anstatt des realen Nao zur Untersuchung eingesetzt. Er muß daher die Funktionalität von SimRobot besitzen und zusätzliche Erweiterungen aufweisen, um eine realistischere Simulation zu ermöglichen. Dazu gehört die Flexibilität der *Verbindungselemente* zwischen den Gelenken, insbesondere aber auch die Flexibilität und Toleranzen der *Gelenke* selbst. *PID-Regler*, die beim Nao und vielen anderen Robotern die Gelenke regeln, verursachen aufgrund ihrer Regelabweichung ebenfalls falsch angefahrene Positionen. Diese Elemente, sowie Rutschen und Kippen, verursachen Fehler in der Umsetzung des Laufs. Die Abstraktion vom WS zur MKM-Ebene wird in Abbildung 1.3 daher **Kinematikfehlerabstraktion** genannt.
Das Ziel bei der Neuentwicklung und Parametrisierung des Walking Simulators ist ein möglichst reales Laufverhalten. Dazu gehört, dass die Beobachtungen, die sich bisher nur auf dem realen Roboter zeigen, auch hier erkennbar werden. Ein Aufschwingen sollte in der Simulation ähnlich schnell zum Fall führen wie in der Realität. Außerdem sollte die tatsächliche Geschwindigkeit qualitativ dasselbe Verhalten zeigen.

Geplante Kompensation

Die gefundenen Systematiken in den Bewegungen und Drehmomenten, welche die Vorgänge, die zu den Beobachtungen führen, verursachen oder zumindest begünstigen, sollen nun geeignet kompensiert werden. Die Kompensation soll nicht durch eine Neuentwicklung einer Walking Engine erfolgen, sondern durch eine Verbesserung der Parametrisierung der existierenden Walking Engine. Durch Änderung in den folgenden drei Bereichen können dann die Bewegungs- und Drehmomentsverläufe beeinflusst werden:

- Änderung des Soll-ZMP-Verlaufs
- Armbewegung
- Änderungen in der Trajektorie des Schwungbeins

1.3 Struktur und Aufbau

Zunächst werden in dieser Arbeit einige Grundlagen erläutert, die für die späteren Kapitel wichtig sind (vgl. Kapitel 2). Darunter sind die gewählte Roboterplattform Nao,

Grundlagen zur Dortmund Walking Engine und zu Evolutionären Algorithmen, mit deren Hilfe der Walking Simulator parametrisiert wird.
Es folgt das Kapitel über die Entwicklung der Algorithmen zur Berechnung der Sim-Richtung auf der Mehrkörpermodellebene (vgl. Kapitel 3). Damit ist es möglich, aus gegebenen Gelenkwinkeln die Drehmomente und den ZMP in einem dort definierten Weltkoordinatensystem (kurz WKS) zu berechnen, womit Aussagen zur Dynamikabstraktion getroffen werden können.
Anschließend, in Kapitel 4, wird die Entwicklung und Parametrisierung des Walking Simulators beschrieben, der zur Analyse und Kompensation der Kinematikfehler dient. Die Kompensation der Dynamikabstraktion lässt sich mit seiner Hilfe ebenfalls bewerten.
In Kapitel 5 werden die beiden entwickelten Werkzeuge dann zur Analyse der Ursachen herangezogen und Kompensationen entwickelt und bewertet. Dabei werden zunächst die Beobachtungen analysiert, bei denen die Dynamikabstraktion die vermutliche Ursache ist. Danach werden die Beobachtungen entlang der y-Achse (*Seitwärtsschwingung* und *Einknicken*) thematisiert und das Kapitel mit der Analyse und Kompensation der Beobachtungen entlang der x-Achse (*Vorwärtsschwingung* und *Geschwindigkeitssteigerung*) abgeschlossen.
Abschließend wird in Kapitel 6 eine Zusammenfassung der Arbeit und ihrer Ergebnisse gegeben und weiterführende Forschungsfragen vorgestellt.

2

GRUNDLAGEN

In diesem Abschnitt werden einige grundlegende Verfahren und Komponenten erläutert, auf denen diese Arbeit aufbaut. Es beginnt mit der Vorstellung der Roboterplattform Nao von Aldebaran Robotics (vgl. Abschnitt 2.1). Darauf folgend werden die für das Simulationskapitel 4 wichtigen Teile der „Open Dynamics Engine“, einer Bibliothek zur Festkörpersimulation, erklärt (vgl. Abschnitt 2.2). Der nächste Abschnitt behandelt zunächst Grundlagen zu Evolutionären Algorithmen, gefolgt von Erläuterungen zu Verfahren, die in dieser Arbeit zur Parameterstudie verwendet werden (vgl. Abschnitt 2.3). Den Abschluss der Grundlagen bildet ein Abschnitt zur Dortmund Walking Engine (vgl. Abschnitt 2.4).

2.1 Nao

In diesem Abschnitt wird der Nao von Aldebaran Robotics[1] gemäß der Version 1.0 der Dokumentation[2] vorgestellt (Abbildung 2.1). Der Nao ist ein humanoider Roboter mit 21 Freiheitsgraden, Abbildung 2.2 zeigt den kinematischen Aufbau und Achsbezeichnungen nach Angaben des Herstellers. Die Achsen drehen sich um die z-Achse des eingezeichneten Koordinatensystems[3]. Die beiden Achsen der Hüfte und des Knies schneiden sich in einem Punkt. Eine Besonderheit stellen die Gelenke LHipYawPitch und RHipYawPitch dar. Sie sind mechanisch miteinander verbunden und haben einen gemeinsamen

[1] http://www.aldebaran-robotics.com

[2] http://robocup.aldebaran-robotics.com/documentations.php

[3] Die eingezeichneten Koordinatensysteme werden vom Hersteller definiert und verwendet. In dieser Arbeit werden später andere Koordinatensysteme definiert.

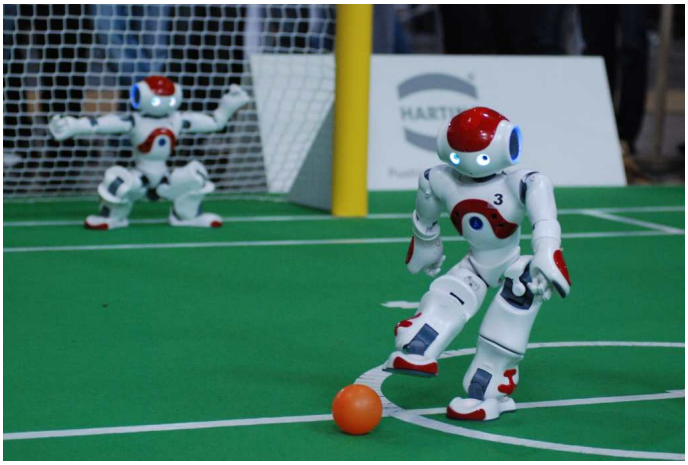


Abbildung 2.1: Naos des Teams „Nao Devils Dortmund“ während des Spiels.

Bezeichnung	Länge [mm]
NeckOffsetZ	126.5
ShoulderOffsetY	98
UpperArmLength	90
LowerArmLength	50.65
ShoulderOffsetZ	100
HandOffsetX	58
HandOffsetZ	15.9
HipOffsetZ	85
HipOffsetY	50
Thigh Lenght	100
Tibia Lenght	100
Foot Height	46

Tabelle 2.1: Tabelle der Dimensionen.

Motor. Der Winkel lässt sich somit nur auf den für beide gleichen Wert setzen. Die Beine haben demnach nicht, wie z.B. beim Bioloid (LGCM09) üblich, 12 Freiheitsgrade, sondern nur 11.

Das in Abbildung 2.2 eingezeichnete Roboterkoordinatensystem wird auch im Folgenden als Roboterkoordinatensystem O_{RKS} verwendet.

Abbildung 2.3 zeigt zusammen mit Tabelle 2.1 die Abstände zwischen den Achsen. Weitere Abmessungen für den Nao werden zur Zeit der Arbeit von Aldebaran nicht vorgegeben.

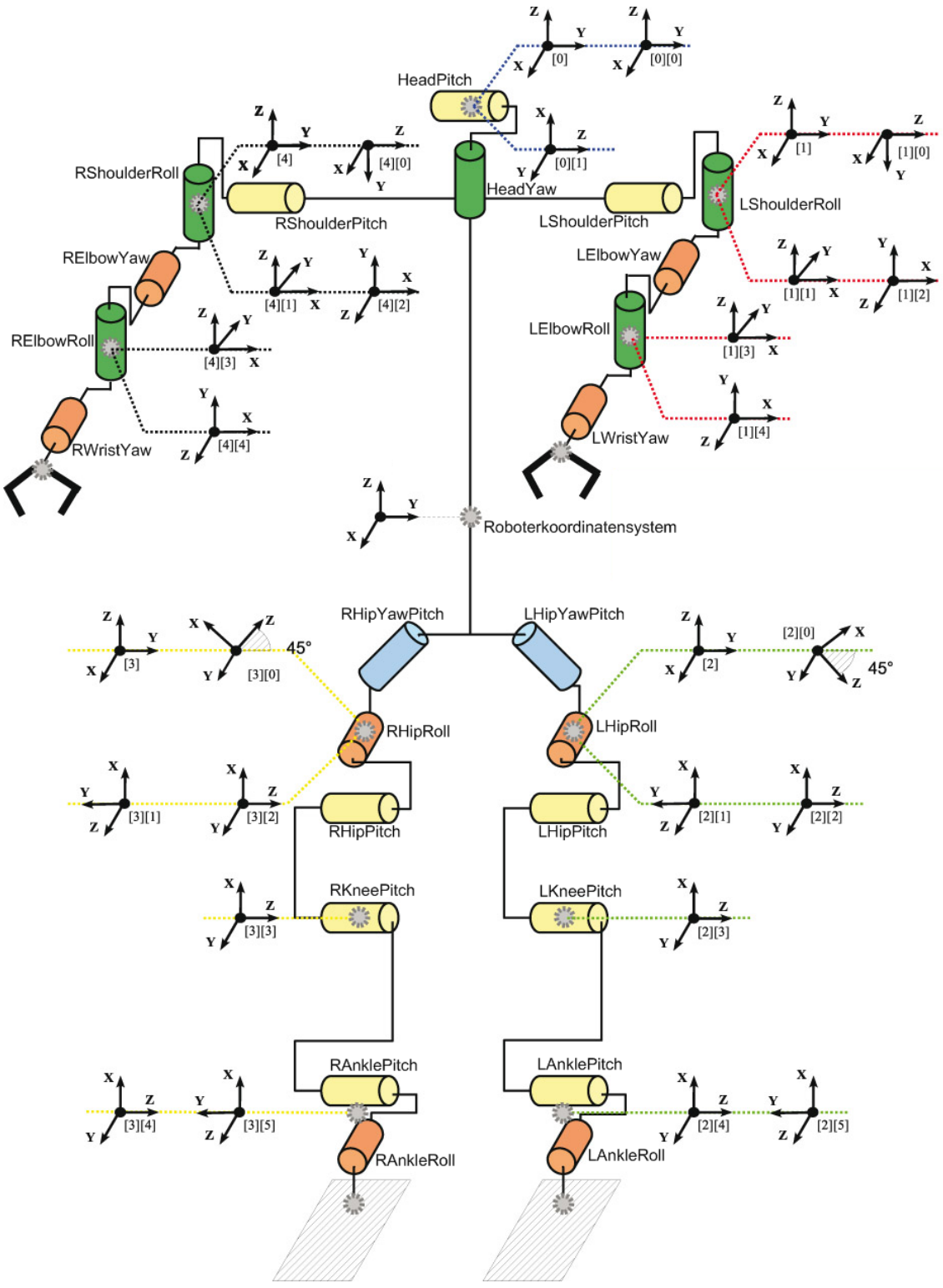


Abbildung 2.2: Bezeichnungen und Koordinatensysteme der Achsen.[1]

[1]Quelle: http://robocup.aldebaran-robotics.com/documentations.php

Getriebe und Motoren

In den Beinen des Nao kommen Getriebe mit zwei verschiedenen Übersetzungsverhältnissen zum Einsatz. Bei HipYawPitch, HipRoll und AnkleRoll beträgt es 201.3:1 und bei HipPitch, KneePitch und AnklePitch 130.85:1. Mit Angaben von Aldebaran zu den Motoren ist es möglich den verwendeten Motortypen beim Hersteller zu identifizieren, um die meisten der von Aldebaran nicht spezifizierten aber notwendigen Konstanten zu erhalten (siehe Tabelle 2.2).

Die Winkel können mit 50 Hz vorgegeben werden. Auf welche Weise sie umgesetzt werden, ist unklar. Daneben ist es möglich, das Drehmoment zu begrenzen, das die Motoren ausüben. Der Parameter heißt **„Stiffness“** und hat einen Wertebereich von 0 (Motor ab-

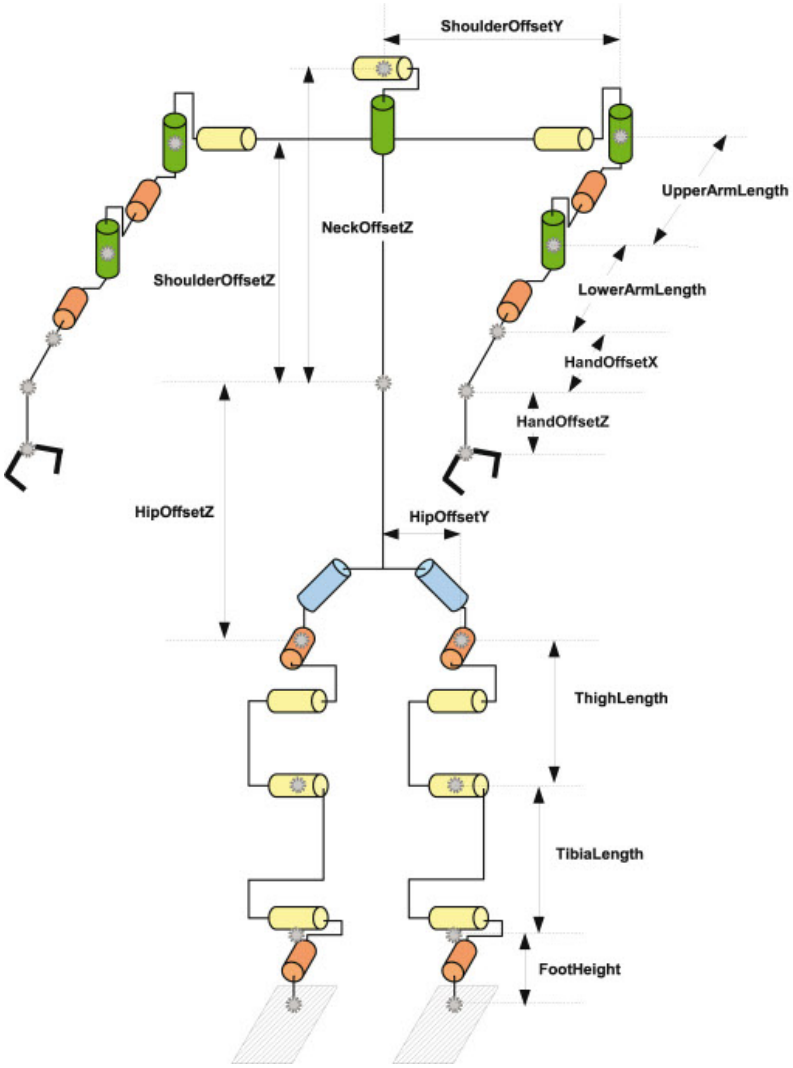


Abbildung 2.3: Bezeichnungen der Dimensionen.[1]

[1]Quelle: http://robocup.aldebaran-robotics.com/documentations.php

Variable / Konstante	Bedeutung	Herstellerangabe
L	Induktivität	$0.000309H$
R	Widerstand	6.44Ω
K_t	Drehmomentskonstante	$0.0195\frac{Nm}{A}$
K_s	Gegen-EMF-Konstante	keine Angabe

Tabelle 2.2: Technische Daten der Beinmotoren.

geschaltet) bis 1 (volles Drehmoment). Die genaue Funktionsweise ist aber nicht bekannt. Auf welche Weise der Parameter hier umgesetzt wird, ist in Abschnitt 4.1 beschrieben.

Messung der Winkel

Der Nao verwendet Hall-Sensoren zur Messung der Winkelposition der Achsen. Dabei wird nicht die Stellung der Motoren gemessen, wie es z.B. in Servos üblich ist, sondern die Hall-Sensoren sind an den Achsen angebracht. Die Fehler, die durch das Getriebe entstehen, sind somit ebenfalls in der Messung enthalten.

2.2 Open Dynamics Engine

In diesem Abschnitt werden die Grundlagen der „Open Dynamics Engine“ (kurz ODE) (Smi02) eingeführt. Es handelt sich dabei um eine Bibliothek zur Simulation der Festkörperdynamik. Zum Einsatz kommt sie in Computerspielen, zur Simulation von Fahrzeugen oder von Robotern, unter anderem im freien Simulator SimRobot (LSR06) und im kommerziellen Simulator Webots (Mic04).
Die ODE basiert auf den Euler-Lagrange Gleichungen (SV89; Hau09). Die Integration von Gelenken und Kontakten geschieht über die Formulierung von Nebenbedingungen für die Bewegungsgleichungen. Für mathematische Details sei auf die Literatur verwiesen (SV89).
Für den Walking Simulator (siehe Abschnitt 4) wird die ODE ausgewählt, da sie sich in diesem Umfeld, z.B. beim SimRobot, bewährt hat.

2.2.1 Festkörper und Gelenke

Größe, Form und Gewicht der Festkörper können frei vorgegeben werden. Neben Standardformen wie Quader oder Zylinder sind auch Dreiecksnetze zur Formung von Körpern möglich. Die Kollisionserkennung bei Körpern aus Dreiecksnetzen verläuft aber oft fehlerhaft. Massen und Größen sollten bei Körpern, die miteinander verbunden sind, ungefähr gleich skaliert sein. Empfohlen wird, dass sich Längen und Massen um nicht mehr als den Faktor 100 unterscheiden. Es ist somit zum Beispiel ungünstig, ein Gelenk

mit sich schneidenden Achsen zu modellieren, indem zwei einzelne Gelenke durch einen Körper mit Größe und Gewicht 0 verbunden werden. Hier sollte auf die vorhandenen Gelenkarten mit mehreren Achsen zurückgegriffen werden.
Ebenfalls nicht möglich sind Realisierungen von Schwerpunktpositionen, bei denen der eigentliche Körper ein sehr kleines Gewicht bekommt, und ein weiterer sehr kleiner Körper an der gewünschten Schwerpunktposition befestigt wird. Als Lösung ist in der ODE vorgesehen, die Schwerpunktposition für einen Körper direkt vorzugeben. Tests haben allerdings ergeben, dass das zu einem physikalisch unlogischem Verhalten führt. Hingegen keine Schwierigkeit bereitet die Vorgabe des Trägheitstensors.
Kollisionen werden ähnlich wie Gelenke behandelt. Wird eine Kollision erkannt, wird an der entsprechenden Stelle ein oder mehrere Kontakt-Gelenk(e) eingefügt, die sich wie normale Gelenke verhalten und verhindern sollen, dass ein Körper sich durch den anderen bewegen kann.

2.2.2 Simulationsparameter

Im Folgenden werden die hier wichtigen Parameter zur Konfiguration der ODE besprochen.

- Die ODE arbeitet mit Zeitschritten fester Länge (Δt_s) zur Lösung der Bewegungsgleichungen und zur Kollisionsdetektion. Die Länge der Zeitschritte kann frei gewählt werden und liegt üblicherweise zwischen $0.02s$ und $0.001s$, je nach Anforderung an die Genauigkeit.

- Motorisierte Achsen können auf zwei verschiedene Arten angesteuert werden. Einerseits ist es möglich eine gewünschte Geschwindigkeit anzugeben. Die ODE setzt diese dann exakt um. Für das dazu aufzubringende Drehmoment kann ein Maximalwert τ_{max} vorgegeben werden. Die andere Möglichkeit ist das Drehmoment selbst vorzugeben. Beide Ansteuerungen können auch gleichzeitig vorgenommen werden. Das lässt sich nutzen, um Reibung im Gelenk zu simulieren, indem das Soll-Drehmoment vorgegeben wird und als Soll-Geschwindigkeit 0 gewählt wird. Es wird dann mit dem Drehmoment τ_{max} die Achse abgebremst.

- Neben der Reibung im Getriebe gibt es auch Reibung bei Kontakt zwischen Körpern, welche mit dem Parameter μ_r eingestellt wird.

- Gelenke beschränken die Bewegung zweier Körper, so dass sie relativ zueinander nur bestimmte Positionen einnehmen können. Grundsätzlich sind die Gelenke als fest anzusehen, üblicherweise treten aber während der Simulation Fehler auf, so dass die Körper ihre Position zueinander verlieren. Mit Hilfe des „error reduction parameter“ (erp, Wertebereich 0 bis 1) lässt sich eine Kraft einstellen, die die Körper

in die notwendige relative Position zueinander zwingt. Der Parameter erp_{soft} ist für Oberflächen zuständig. Je niedriger der Wert, desto mehr können die Körper in den Boden „einsinken“.

- Der Parameter „constraint force mixing“ (cfm, Wertebereich 0 bis 1) hat zu erp eine antiproportionale Bedeutung. Ein höherer Wert erlaubt höhere Fehler in den Gelenken, die dadurch „weich“ werden. Zusammen mit dem erp kann der cfm verwendet werden, um Gelenke federartig oder schwammig zu machen. Wie erp_{soft} ist cfm_{soft} für Oberflächen zuständig.

2.2.3 Instabilitäten

Auch wenn die in Abschnitt 2.2.1 beschriebenen Regeln zur Erzeugung stabiler Simulationen eingehalten werden, kann es durch hohe Kräfte und Drehmomente dennoch zu instabilen Simulationen kommen. Das äußert sich in Fehlermeldungen der ODE oder in „explodierenden“ Robotern. Im letzteren Fall war die ODE nicht mehr in der Lage, die durch Gelenke verbundenen Körper in ihren Positionen zu halten, woraufhin die Gelenke ihre Bindung verlieren.
Es ist möglich solche Probleme zu mindern, indem eine günstige Wahl von cfm und erp getroffen wird. Auch eine Verringerung von Δt_s führt zu einer stabileren Simulation, bedeutet aber auch, dass die Simulation zeitaufwändiger wird.

2.3 Evolutionäre Algorithmen

Analysen von Bewegungsabläufen auf dem realen Roboter haben verschiedene Nachteile. Daten lassen sich nicht rauschfrei aufzeichnen und Einflüsse wie Reibung, Getriebe, Motoren, usw. nicht getrennt untersuchen. Zudem lassen sich Positionen und Geschwindigkeiten des Roboters und seiner einzelnen Bauteile kaum bestimmen.
Der Walking Simulator soll die Lücke zwischen dem realen Roboter und einer einfachen Simulation schließen, muß aber dazu realistischer simulieren als z.B. SimRobot oder andere ODE basierte Simulatoren. Es bedarf dazu einiger Erweiterungen der grundlegenden physikalischen Simulation der ODE und einer geeigneten Parametrisierung, die die Anforderung der realitätsnahen Simulation erfüllt. Einige Parameter sind fest vorgegeben, aber nicht alle, und müssen durch eine Optimierung gefunden werden. Die Auswahl an Optimierungsverfahren wird durch die Tatsache eingegrenzt, dass die zu optimierende Funktion nicht bekannt ist, man spricht auch von einer Black-Box-Optimierung. Das schließt Verfahren aus, die die erste oder zweite Ableitung der zu optimierenden Funktion benötigen, darunter das Newton-Verfahren oder das Gradientenverfahren. Unter den Black-Box-Verfahren haben sich die „Evolutionären Algorithmen“ (Kra09) als effektiv er-

wiesen (HNF06). Ihr Vorbild ist die Evolution der Natur und sie arbeiten nach dem Darwinschen Prinzip „survival of the fittest“. Für Optimierungen im Zusammenhang mit Simulationen und laufenden Robotern wurden die von H.-P. Schwefel und Rechenberg entwickelten „Evolutionsstrategien“ (kurz ES) (Sch95; Rec73) bereits erfolgreich eingesetzt (Kos06; Heb09; HNF06). Sie stellen neben den „Genetischen Algorithmen“ und der „Evolutionären Programmierung“ ein Teilgebiet der „Evolutionären Algorithmen“ dar.

In diesem Abschnitt werden zunächst kurz die notwendigen Begriffe eingeführt, gefolgt von der Vorstellung zwei der am weitest entwickelten Evolutionsstrategien, deren Effizienz im Bezug auf die hier durchzuführende Parametersuche mit einander verglichen werden sollen.

2.3.1 Grundbegriffe

Mit den Evolutionären Algorithmen haben sich einige Begriffe etabliert (Kra09), die ihren Ursprung bei der Evolution in der Natur haben. Sie werden nun eingeführt.

Ein Parametersatz, der zur Ausführung des Simulators benötigt wird, heißt **Individuum**. Ein Individuum stellt einen Punkt im Suchraum dar, in dem das absolute Extremum gefunden werden soll. Aus einer Multimenge von Individuen, der **Population** der Größe μ, werden bei der Suche nach dem Optimum durch **Mutation** und **Rekombination** λ neue Individuen erzeugt. Diese werden dann durch die **Fitnessfunktion** (in der mathematischen Optimierung auch **Zielfunktion** genannt) bewertet, und die „fittesten“ Individuen werden zur neuen **Generation**. Letzteres nennt sich **Selektion**.

Es wird dabei die **Komma-Selektion** (Kurzschreibweise (μ, λ)) und die **Plus-Selektion** (kurz $(\mu + \lambda)$) unterschieden. Bei der Komma-Selektion werden die Eltern für die nächste Generation nur aus den Nachkommen selektiert, während bei der Plus-Selektion die fittesten Individuen aus den Nachkommen und den Eltern ausgewählt werden. Die Komma-Selektion hat mehrere Vorteile. Es ist wahrscheinlicher lokale Extrema zu verlassen (Kra09). Es kann auch vorkommen, dass durch Messfehler zu gut bewertete Individuen entstehen. Sie sollten nicht erneut zur Nachkommenerzeugung verwendet werden. Eltern nicht zu übernehmen kann aber auch ein Nachteil sein, wenn in einer Generation aus guten Eltern durch Zufall nur schlechte Nachkommen erzeugt wurden, obwohl eventuell noch Verbesserungen möglich wären, wenn man die Eltern länger leben lassen würde. Als Kompromiss kann bei der Plus-Strategie die maximale Lebensdauer durch einen Parameter κ beschränkt werden.

Die folgenden beiden Abschnitte beschreiben zwei Ansätze von Plus-Evolutionsstrategien, die sich hauptsächlich in der Art der Nachkommenerzeugung unterscheiden.

2.3.2 $(\mu + \lambda)$-ES mit σ-Selbstadaption

Bisher wurde ausgeklammert, wie die Evolutionsstrategie die Nachkommen erzeugt. Das kann zum einen durch Rekombination geschehen. Zufällig Werte von Individuen zu vermischen soll dazu führen, dass gute Werte mit einander kombiniert gute Individuen ergeben. Es ist aber im Falle des in Abschnitt 4 behandelten Walking Simulator zu erwarten, dass einige Parameter korrelieren. Zum Beispiel existieren beim Walking Simulator mehrere Elemente, die sich federnd verhalten können, wie z.B. PID-Regler oder federnde Gelenke. Angenommen, es gäbe ein gutes Individuum, bei dem eine Federung durch die PID-Regler erreicht wird, aber nur wenig durch die Gelenke. Daneben gäbe es ein gutes Individuum, bei dem es sich umgekehrt verhält. Da die Reglerparameter und die Parameter für das Gelenk nicht unabhängig von einander zu einer hohen Fitness führen, ist davon auszugehen, dass eine Rekombination durch Austauschen der Parameter beider Individuen nicht von Vorteil ist. Es wäre zu untersuchen, inwiefern andere Rekombinationsvarianten erfolgversprechend sind, was aber mit zeitlich aufwändigen Tests einhergehen würde. Auf Rekombination wird daher verzichtet.

Die Mutation wird im allgemeinen durchgeführt, indem zu jedem Parameter X_i eines Individuums $\boldsymbol{X} = (X_1 \dots X_n)$ (n Anzahl der Parameter) ein Zufallswert addiert wird (Kra09). Üblicherweise handelt es sich dabei um eine $\mathcal{N}\left(0, \sigma^2\right)$-verteilte Zufallsvariable $z_i : X_i\left(t\right) = X_i\left(t-1\right) + z_i$. Für mehr Flexibilität kann eine eigene Standardabweichung für jedes z_i' verwendet werden ($z_i' \sim \mathcal{N}\left(0, \sigma_i^2\right)$).

Wahl der Standardabweichung

Der Standardabweichung σ kommt bei evolutionären Algorithmen die Rolle der Schrittweite zu und wird **Strategieparameter** genannt (Kra09). Es ist möglich sie fest zu wählen, aber nicht empfehlenswert. Zu Anfang der Evolution sind eher hohe Schrittweiten sinnvoll, während in der Nähe des Optimums die Schrittweiten kleiner werden sollten. Dazu wurde die „1/5-Erfolgsregel“ entwickelt. Sie arbeitet aber mit nur einer Schrittweite σ für alle Parameter, was hier nicht sinnvoll ist, da die X_i unterschiedlich skaliert sind. Einige liegen im Bereich von 10^{-6}, andere bei 10^6.

Eine andere Möglichkeit ist, die Schrittweiten ebenfalls der Evolution zu unterziehen. Die Idee dabei ist, dass gute Strategieparameter zu guten Individuen führen. Das Verfahren nennt sich **Selbstadaption**. Ein Individuum besteht demnach nun aus dem Parametervektor $\boldsymbol{X} = (X_1 \dots X_n)$ und den Schrittweiten $\boldsymbol{\sigma} = (\sigma_1 \dots \sigma_n)$. Die $\boldsymbol{\sigma}$ werden mit Hilfe der logarithmischen Normalverteilung mutiert, da so kein σ_i negativ werden kann und der Erwartungswert der logarithmischen Normalverteilung 1 ist:

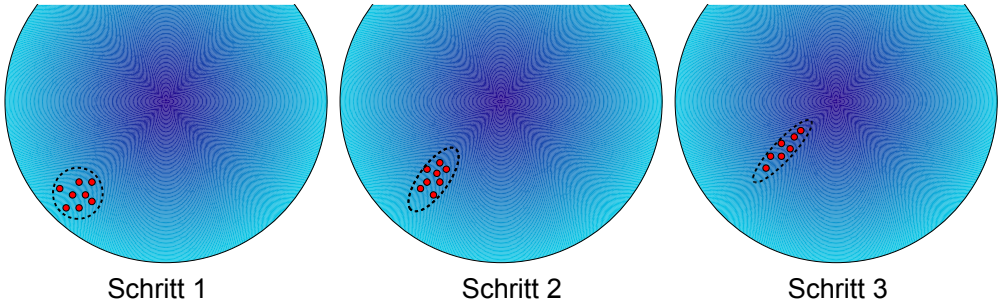


Abbildung 2.4: Auswirkung der Kovarianzmatrix-Anpassung beim CMA-ES gezeigt in drei Schritten. Die Kreise stellen den Suchraum dar, wobei die Schattierung die Fitness am jeweiligen Ort darstellt. Die Punkte repräsentieren einzelne Individuen, die als Generation im Verlauf der Evolution aufgrund der Anpassung der Kovarianzmatrix in Richtung Optimum ausgerichtet sind.

$$\sigma(t) = c_0 \cdot (c_1 \cdot \sigma_1(t-1) \dots c_n \cdot \sigma_n(t-1)) \tag{2.1}$$

$$\ln c_0 \sim \mathcal{N}(0, \tau_0^2) \tag{2.2}$$

$$\ln c_i \sim \mathcal{N}(0, \tau^2) \tag{2.3}$$

$$\tau_0 = \frac{1}{\sqrt{2n}} \tag{2.4}$$

$$\tau = \frac{1}{\sqrt{2\sqrt{n}}} \tag{2.5}$$

Um zu erreichen, dass mit guten Individuen auch gute Strategieparameter gewählt werden, muß die Mutation von $\boldsymbol{X}$ nach der Mutation von σ durchgeführt werden.

2.3.3 Covariance Matrix Adaption

Wie bereits erwähnt ist eine Abhängigkeit der Parameter untereinander zu erwarten. Daher liegt es nahe, die Zufallswerte zur Mutation der X_i nicht unabhängig von einander zu wählen, sondern einen multivariat normalverteilten Zufallsvektor der Dimension n zur Mutation zu verwenden. Das führt zum Verfahren der „Covariance Matrix Adaption“ (kurz CMA-ES) (Han06). Der hier verwendete Zufallsvektor ist $\mathcal{N}(0, C)$-verteilt bei einer Kovarianzmatrix C. Sie wird im Laufe des Verfahrens basierend auf dem Erfolg der Mutation angepasst, was die in Abbildung 2.4 gezeigte Auswirkung bei der Suche hat.
Während der Evolution werden die Fortschritte anhand von zwei sogenannten **Evolutionspfaden** beobachtet. Einer dient der Anpassung der Kovarianzmatrix. Der andere Evolutionspfad dient der Kontrolle der globalen Schrittweite, die zusätzlich zur Kovarianzmatrix bei der Normalverteilung angewendet wird. Für mathematische Details des Verfahrens sei auf die Literatur verwiesen (Han06).

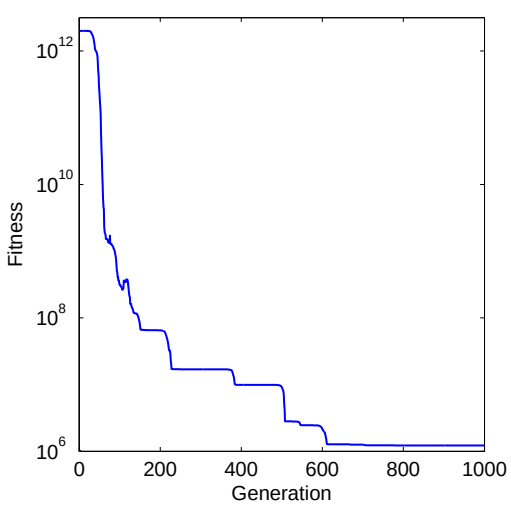
Um den Entwicklungsaufwand zu minimieren wird hier auf eine bereits vorhandene und frei verfügbare Implementierung des Algorithmus zurückgegriffen. Die Shark-Bibliothek (IGHM08) bietet eine dokumentierte und parallelisierbare $(1 + \lambda)$ Variante.

2.3.4 Validierung der Implementation

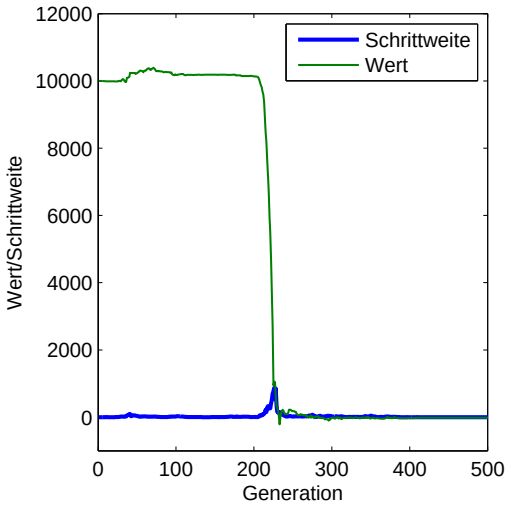
Im Folgenden werden die Implementationstests beschrieben und die Resultate gezeigt. Als Testfunktion wird hier SPHERE verwendet:

$$f_{SPH} = \sum_{i=1}^{n} X_i^2 \tag{2.6}$$

Diese Funktion ist zu minimieren, wobei eine Fitness von 0 zu erreichen ist.
Zur Erzeugung der Pseudozufallszahlen, die zur Generierung der normalverteilten Zufallsvariablen benötigt werden, kommt hier ein Mersenne Twister MT19937 (MN98) zum Einsatz, welcher für seine gut verteilten Werte und der langen Periode bekannt ist.
Bei dem Test wird $n = 19$ gewählt, was der Anzahl der später zu optimierenden Parameter des Simulators entspricht. Die Startwerte für die Parameter und Schrittweiten werden ungünstig gewählt, um einen guten Test für die Algorithmen zu erhalten. Die X_i liegen initial im Bereich von 10^{-4} bis 10^4, die Schrittweiten σ_i bei 1.



(a) Fitness.



(b) Wert X_1 und Schrittweite σ_1 von Parameter 1 des jeweils besten Individuums.

Abbildung 2.5: Ergebnisse des $(5 + 30)$-ES Tests

Die $(\mu + \lambda)$-ES mit σ-Selbstadaption wird in 3 Durchläufen als $(1 + 30)$-ES mit einem $\kappa = 3$ getestet. Abbildung 2.5(a) zeigt den Median der Fitness anhand einer logarithmischen Skala. Zu erkennen ist eine stärkere Minimierung zu Anfang, gefolgt von Phasen mit geringem Erfolg. In Abbildung 2.5(b) ist der Wert und die Schrittweite σ_1 des ersten

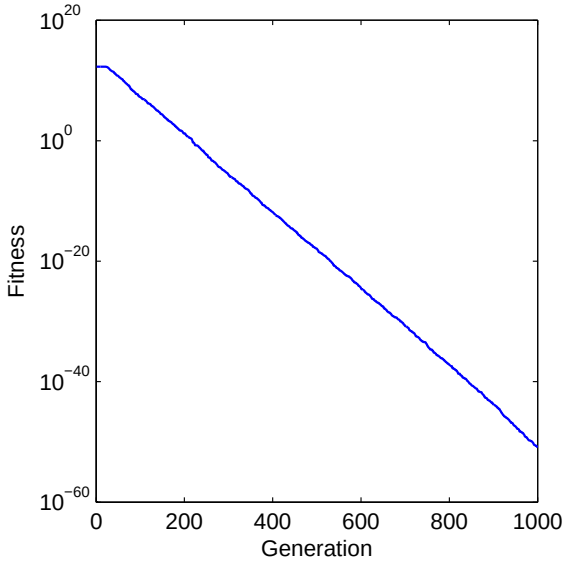


Abbildung 2.6: Resultat des $(1 + 30)$-CMA-Tests.

Parameters X_1 des besten Individuums eines Durchlaufs zu sehen. Der Startwert liegt bei 10000 und die Schrittweite bei 1, was zu klein ist, um den Wert erfolgreich zu minimieren. So ist anhand der Fitness zu erkennen, dass offensichtlich zunächst andere Werte kleiner werden. Erst nach ca. 200 Generationen steigt die Schrittweite für den Parameter auf fast 1000, woraufhin der Wert gegen 0 geht und sich aufgrund der erneut kleinen Schrittweite stabilisiert.
Der CMA-ES wird als $(1 + 30)$ mit den gleichen Startwerten, ebenfalls anhand von 3 Durchläufen getestet. In Abbildung 2.6 ist die bessere Performance des Algorithmus gegenüber der anderen Strategie zu erkennen. Der Fitnessverlauf ist auf der logarithmischen Skala nahezu eine Gerade und erreicht eine bessere Fitness.
Auch wenn die Überlegenheit des CMA-ES bei dem Beispielproblem SPHERE klar ist, gibt es Fälle, in denen der CMA-ES schlechte Ergebnisse liefert (HK04). Daher sind weitere Tests notwendig, die zeigen, welches Verfahren bei der Optimierung der Simulation überlegen ist.

2.4 Dortmund Walking Engine

Gegenstand dieser Arbeit ist die **Dortmund Walking Engine** (kurz DWE), die vom Team „Nao Devils Dortmund“ (CK09) und vom Vorgängerteam „BreDoBrothers“ seit dem Einsatz der Naos zur Erzeugung der Laufmuster verwendet wird. Alle Beobachtungen wurden daher auch unter dem Einsatz der DWE gemacht.
Zunächst werden einige Grundbegriffe eingeführt. Im Anschluss werden in der Literatur weit verbreitete Stabilitätskriterien zum zweibeinigen Laufen vorgestellt (vgl. Abschnitt

2.4.2). Es folgt ein Überblick über die Zusammenhänge der Module in der DWE (vgl. Abschnitt 2.4.3) und im Anschluss werden einige der Module im Detail vorgestellt.

2.4.1 Grundbegriffe

Beim humanoiden Laufen haben sich einige Begriffe etabliert, die nun vorgestellt werden. Ein Lauf besteht aus zwei unterschiedlichen Phasen. Übt der Boden nur über einen Fuß Kraft auf den Roboter aus, hat also nur ein Fuß Bodenkontakt, nennt man das die **Single-Support-Phase**. In dieser Phase bewegt sich der Fuß, der keinen Kontakt zum Boden hat, von einem Kontaktpunkt zum nächsten und wird **Schwungbein**[4] genannt, der Fuß ist dann der **Schwungfuß**. Im anderen Fall übt der Boden über beide Füße eine Kraft auf den Roboter aus, das ist die **Double-Support-Phase**. In beiden Phasen lässt sich um die Kontaktpunkte mit dem Boden ein konvexes Polygon bilden. Diese Fläche wird **Support-Polygon** genannt.

2.4.2 Stabilitätskriterien

Bei der Entwicklung einer Walking Engine stellt sich die grundlegende Frage, wie man den Lauf entwerfen muß, damit der Roboter stabil bleibt. Anschaulich klar ist, dass das der Fall ist, wenn sich der Schwerpunkt des Roboters bei langsamen Bewegungen über dem Support-Polygon befindet. In der Literatur wird der senkrecht auf den Boden projizierte Schwerpunkt als **„ground projected center of mass“** bezeichnet (BHK$^+$03). Liegt dieser innerhalb des Support-Polygon und beschleunigt bzw. bewegt sich der Roboter nur langsam, ist der Lauf stabil (SK08).
Die Annahme von nur langsamen Bewegungen schränkt allerdings stark ein. Für den allgemeinen Fall bezieht Vukobratović (VB04) die dynamischen Aspekte mit in die Betrachtung ein und erhält so den Zero Moment Point (kurz ZMP). An diesem Punkt am Boden herrschen keine Drehmomente um die x- und y-Achse. Liegt dieser Punkt innerhalb des Support-Polygons, übt der Boden die gesamte Kraft auf den Roboter an diesem Punkt aus, ohne dabei ein Drehmoment auf den Roboter zu übertragen. Der Roboter bleibt in dieser Situation stabil.
Für die Berechnung und Messung dieses Punktes existieren verschiedene Arten. Die abstrakteste Möglichkeit ist das 3D-LIP-Modell von Kajita et al. (KKK$^+$03b). Hier wird der Roboter auf einen Schwerpunkt abstrahiert, auf den die Theorie des invertierten Pendels angewendet werden kann. Abbildung 2.7 zeigt ein Modell zur Erläuterung des Prinzips. Ein Wagen ohne eigene Masse ist per Drehgelenk mit einer Stange veränderbarer Länge verbunden, welche ein Gewicht hält. Der Wagen kann über seine Räder beschleunigen, das Gelenk ist nicht motorisiert. Das Gewicht soll seine Höhe über dem Boden halten.

[4] In der englischen Literatur wird dieses Bein als **swinging leg** bezeichnet.

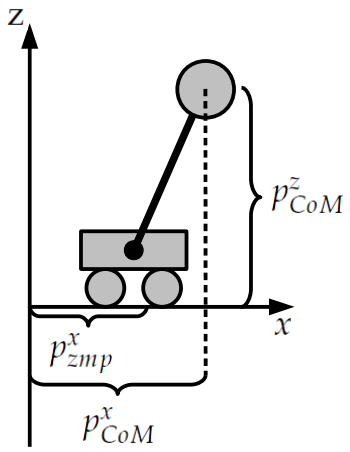


Abbildung 2.7: Wagen-Modell zur Theorie des invertierten Pendels.

Variable / Konstante	Bedeutung
g	Gravitation.
${}^x p^{WKS}_{CoM}, {}^y p^{WKS}_{CoM}, {}^z p^{WKS}_{CoM}$	Position des Schwerpunktes entlang der x-, y- bzw. z-Achse im WKS.
${}^x \ddot{p}^{WKS}_{CoM}, {}^y \ddot{p}^{WKS}_{CoM}$	Beschleunigung des Schwerpunktes entlang der x- bzw. y-Achse im WKS.
${}^x p^{WKS}_{zmp}, {}^y p^{WKS}_{zmp}$	Position des ZMP entlang der x- bzw. y-Achse im WKS.

Tabelle 2.3: Variablen und Konstanten der ZMP-Berechnung anhand des 3D-LIPM.

Für den Fall, dass der Wagen nicht beschleunigt, muß offensichtlich ${}^{x}p_{CoM}^{WKS} = {}^{x}p_{zmp}^{WKS}$ gelten (siehe Tabelle 2.3 für die hier verwendeten Variablen und Konstanten). Das ist in der Abbildung 2.7 nicht der Fall. Der Wagen muß hier in positiver Richtung beschleunigen, um zu erreichen, dass der ZMP die beispielhaft eingezeichnete Position hat.
Die folgenden beiden Gleichungen zeigen den Zusammenhang zwischen dem ZMP und dem Schwerpunkt (KKK+03b):

$$ {}^{x}p_{zmp}^{WKS} = {}^{x}p_{CoM}^{WKS} - \frac{{}^{z}p_{CoM}^{WKS}}{g} \cdot {}^{x}\ddot{p}_{CoM}^{WKS} \tag{2.7} $$

$$ {}^{y}p_{zmp}^{WKS} = {}^{y}p_{CoM}^{WKS} - \frac{{}^{z}p_{CoM}^{WKS}}{g} \cdot {}^{y}\ddot{p}_{CoM}^{WKS} \tag{2.8} $$

Der Vorteil dieser Methode ist, dass der ZMP sich leicht bestimmen lässt. Messbar ist er zum Beispiel, indem die Beschleunigung des Schwerpunktes mit Hilfe von Beschleunigungssensoren bestimmt wird.
Die Abstraktion auf einen Schwerpunkt deutet darauf hin, dass er auf diese Weise bei einem humanoiden Roboter nicht korrekt berechnet wird. Die korrekte Berechnung geschieht auf der Mehrkörpermodellebene, wie sie in Abschnitt 3.3 vorgestellt wird. Eine Messung auf der Mehrkörpermodellebene ist ebenfalls möglich, indem per Fußdrucksensoren die Kräfte auf den Fuß an verschiedenen Punkten gemessen werden (SK08).

2.4.3 Überblick über die Dortmund Walking Engine

In diesem Abschnitt wird ein Überblick über die Module der DWE und den dazwischen verlaufenden Informationen gegeben. Zu Anfang wird die gewünschte Laufgeschwindigkeit in Form eines Tripels (v_x, v_y, ω_z) aus Translationsgeschwindigkeit entlang der x- und y-Achse und der Rotationsgeschwindigkeit um die z-Achse festgelegt und der Walking Engine übergeben, siehe Abbildung 2.8. Der *PatternGenerator* legt zum Start des Laufs ein Weltkoordinatensystem O_{WKS} fest, dessen Ursprung an der Startposition des Roboters auf dem Boden liegt. In der Abbildung 2.8 sind alle Datenpfade rot eingezeichnet, die in diesem WKS arbeiten. Die blauen Datenpfade verwenden das Roboterkoordinatensystem O_{RKS}.
Der *PatternGenerator* errechnet in den folgenden Zeitschritten Sollpositionen $p_{f_r}^{WKS}$ und $p_{f_l}^{WKS}$ für den rechten Fuß f_r bzw. den linken Fuß f_l, sofern sie Kontakt zum Boden haben sollen. Diese kann man sich vorstellen wie Fußstapfen im Schnee. Details zum Verfahren befinden sich in Abschnitt 2.4.4. Das darauf folgende Modul *ZMP-Generierung* erzeugt aus den Fußpositionen Soll-ZMP-Positionen $p_{zmp,S}^{WKS}$, siehe Abschnitt 2.4.5. Diese werden dann dem *ZMP/IP-Controller* übergeben. Dieses Modul ist der zentrale Teil der Walking Engine, in dem auch das ausgewählte Abstraktionsmodell seine Verwendung findet. Es berechnet aus den vorgegebenen ZMP-Positionen einen dazugehörigen Schwerpunkt-

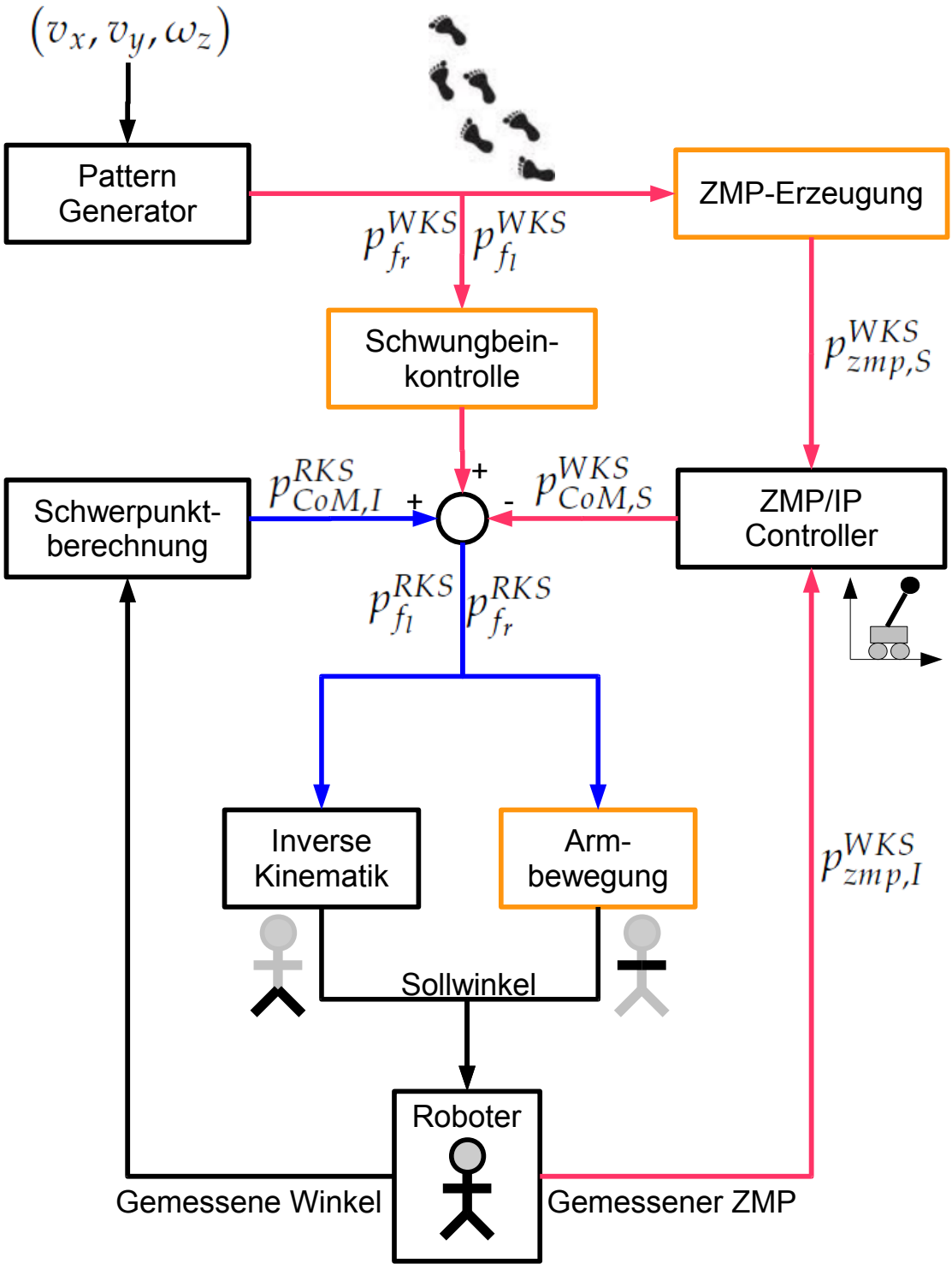


Abbildung 2.8: Aufbau der Dortmund Walking Engine. Module, bei denen Änderungen zur Kompensation der Beobachtungen möglich sind, sind orange gefärbt. Die roten Pfeile stellen Datenpfade im WKS dar, die blauen im RKS.

verlauf im WKS, dem der Schwerpunkt des Roboters folgen muß. Die Funktionsweise des *ZMP/IP-Controller* wird in Abschnitt 2.4.7 erläutert.
Die Schwerpunktposition wird nun vom WKS ins RKS konvertiert. Zunächst muß dazu die Trajektorie für den Schwungfuß festgelegt werden, mit der er sich von einer Position am Boden zur nächsten bewegt, ebenfalls im WKS (Modul *Schwungbeinkontrolle*, Abschnitt 2.4.6). Jetzt sind die Positionen beider Füße zu jedem Zeitpunkt festgelegt und man kann von ihnen die Soll-Schwerpunktposition subtrahieren. Daraus ergeben sich die Fußpositionen relativ zur Soll-Schwerpunktposition. Da der Gesamtschwerpunkt des Roboters normalerweise nicht mit dem Ursprung des RKS übereinstimmt, muß die tatsächliche Schwerpunktposition $p^{RKS}_{CoM,I}$ ebenfalls addiert werden. Zur Berechnung von $p^{RKS}_{CoM,I}$ werden die gemessenen Winkel von einer Vorwärtskinematik verwendet.
Obwohl die Dortmund Walking Engine grundsätzlich auf der Ein-Schwerpunkt-Ebene arbeitet, wird mit dieser Schwerpunktanpassung versucht, die daraus resultierenden Ungenauigkeiten zu kompensieren, indem die Mehrkörpermodellebene teilweise einbezogen wird. Die genaue Einordnung auf eine Abstraktionsebene wird dadurch schwierig, da unklar ist, wie sich eine simple Schwerpunktanpassung tatsächlich auswirkt.
Aus den nun vorliegenden Fußpositionen im RKS ($p^{RKS}_{f_r}$ und $p^{RKS}_{f_l}$) werden mit Hilfe der inversen Kinematik die Winkel für die Beine berechnet. Parallel dazu werden Winkel für die Armbewegung berechnet. Um eine Auslenkung der Arme entsprechend der Bewegung der Beine zu erreichen, werden die x-Koordinaten der Füße mit einem Faktor versehen als Winkel für die ShoulderPitch-Achsen (siehe Abbildung 2.2) verwendet. Alle anderen Armwinkel sind konstant.
Die Verwendung der Winkel aus dem letzten Zeitschritt zur Berechnung von $p^{RKS}_{CoM,I}$ ist nicht vollkommen korrekt, da sie zeitlich nicht zu denen passen, die zum aktuellen Zeitpunkt gesetzt werden müssen. Die Näherung hat aber den Vorteil, auf den gemessenen Winkeln zu basieren, und erlaubt eine schnellere Berechnung. Die exakte Berechnung wurde von Strom et al. verwendet (SSC09). Hier wird in einem iterativen Verfahren zunächst ein Satz Winkel berechnet, woraus sich eine vorläufige Ist-Schwerpunktposition ergibt. Da sie in die Berechnung der $p^{RKS}_{f_r}$ und $p^{RKS}_{f_l}$ eingeht, verändert sie die Soll-Winkel. Es wird daher ein weiterer Satz Winkel berechnet, wodurch sich $p^{RKS}_{CoM,I}$ erneut ändert. Das Verfahren kann bis zum Konvergieren der Winkel fortgesetzt werden. Ein Vergleich beider Ansätze wird in Abschnitt 5.1.1 vorgestellt.
Im Folgenden werden einige Module näher erläutert.

2.4.4 PatternGenerator

Zur Erzeugung der Fußpositionen arbeitet der *PatternGenerator* mit verschiedenen Laufphasen. Er fasst zwei Single-Support-Phasen, in denen entweder nur $p^{WKS}_{f_r}$ oder $p^{WKS}_{f_l}$ definiert ist, und zwei Double-Support-Phasen zu einer Schrittfolge zusammen, die eine

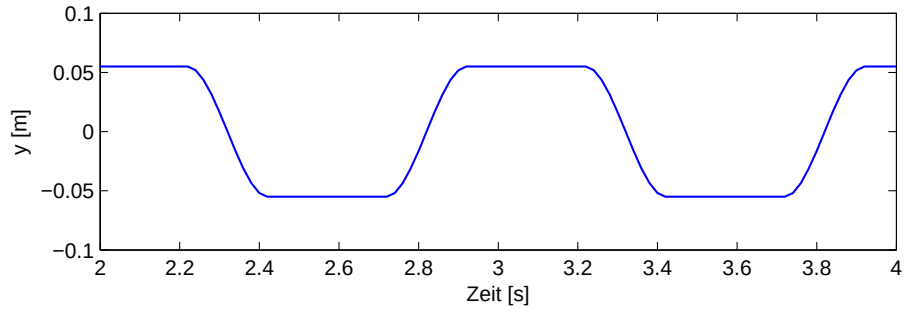


Abbildung 2.9: Verlauf des Soll-ZMPs entlang der y-Achse über die Zeit.

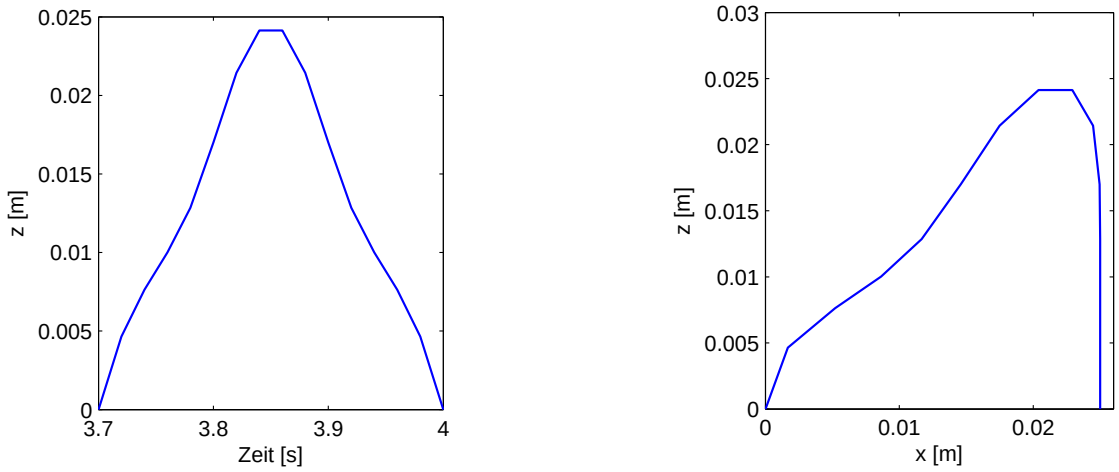
einstellbare aber konstante Dauer hat, die **Schrittdauer**. Aus der Geschwindigkeit und der Schrittdauer ergibt sich die Schrittweite in den jeweiligen Richtungen und damit auch die Positionen der Füße im WKS.

2.4.5 ZMP-Generierung

Aus der Fußposition muß für den *ZMP/IP-Controller* erst eine ZMP-Position berechnet werden. Der ZMP kann innerhalb des Support-Polygons frei gewählt werden, da dort jede Position theoretisch zu einem stabilen Lauf führt. Sinnvollerweise sollte sich in der Single-Support-Phase der ZMP in der Fußmitte befinden, damit Abweichungen vom Soll nicht so schnell zu Instabilitäten führen.
Plötzliche Änderungen im ZMP führen auch zu schnellen Beschleunigungsänderungen, die sich negativ auf die Roboterstabilität auswirken können. In der Double-Support-Phase wechselt der ZMP daher nicht schlagartig von einem Fuß zum anderen, sondern langsam mit Hilfe einer zweidimensionalen Bézierkurve, die getrennt für x und y berechnet wird. Die erste Dimension der Bézierkurve ist für den Ort vorgesehen (x- bzw. y-Koordinate), die zweite Dimension für die Zeit, was mehrere Punkte zu einem Zeitpunkt ermöglicht. Zur Festlegung der Kurve wird ein Kontrollpolygon mit insgesamt 4 Punkten verwendet. Der erste Punkt liegt zum Startzeitpunkt der Double-Support-Phase in der Mitte des Startfußes. Der vierte Kontrollpunkt ist die Mitte des Zielfußes zum Endzeitpunkt der Double-Support-Phase. Punkt 2 und 3 liegen örtlich auch in der Mitte des Start- bzw Endfußes und zeitlich in der Mitte der Double-Support-Phase. Das Ergebnis ist für die y-Achse über die Zeit in Abbildung 2.9 zu erkennen.

2.4.6 Schwungbeinkontrolle

Die vom *PatternGenerator* festgelegten Fußpositionen betreffen nur die Positionen der Füße am Boden und man kann sie sich daher auch als Fußstapfen im Schnee vorstellen. Es fehlen somit Informationen über die Fußpositionen des Schwungfußes während der Single-Support-Phase. Diese fügt die *Schwungbeinkontrolle* hinzu. Ihre Aufgabe ist es, ei-



(a) Fußhöhe über die Zeit.

(b) Fußhöhe über x-Achse.

Abbildung 2.10: Bewegung des Schwungfußes im WKS.

ne Trajektorie für den Schwungfuß von seinem letzten Berührungspunkt mit dem Boden zum nächsten festzulegen. Hier wird eine B-Spline-Kurve vom Grad 4 mit einem Kontrollpolygon von 9 Punkten mit jeweils 3 Dimensionen für die x-, y- und z-Koordinaten verwendet. Die x- und y-Koordinaten der Kontrollpunkte werden entlang der Strecke zwischen Start und Zielpunkt festgelegt. Die z-Koordinaten werden Stufenweise um $\frac{1}{6}$ der maximalen Höhe erhöht bzw. erniedrigt. Die maximale Höhe ist einstellbar und wird als **Schritthöhe** bezeichnet.
Abbildung 2.10(a) zeigt die Höhe des Fußes über die Zeit. Das Kontrollpolygon wird so gewählt, dass der Fuß mit ungefähr gleicher Geschwindigkeit steigt wie sinkt, um den Einfluss der Trägheit des Beines gleichmäßig zu verteilen.
Würde der Fuß sich gegen Ende der Bewegung jedoch noch entlang der x- oder y-Achse bewegen, wäre es wahrscheinlich, dass er die Position des Roboters verfälscht, da er häufig zu früh den Boden berührt. In Abbildung 2.10(b) ist zu erkennen, dass der Fuß sich daher gegen Ende der Single-Support-Phase nur noch absenkt, was die Auswirkungen einer zu frühen Berührung minimiert.

2.4.7 ZMP/IP-Controller

Bisher wurde die Funktionsweise des *ZMP/IP-Controller* ausgelassen. Seine Aufgabe ist es, aus dem Soll-ZMP einen Schwerpunkt-Verlauf zu berechnen.
Abbildung 2.11 zeigt die Problematik. Der Referenz-ZMP bleibt in der ersten Sekunde an Position $0m$ und springt bei Sekunde 1 auf $10m$. Ohne den ZMP-Verlauf vorher zu kennen, müsste der Schwerpunkt bei Sekunde 1 demnach in negativer y-Richtung zunehmend beschleunigt werden, was nach kurzer Zeit nicht mehr möglich ist.

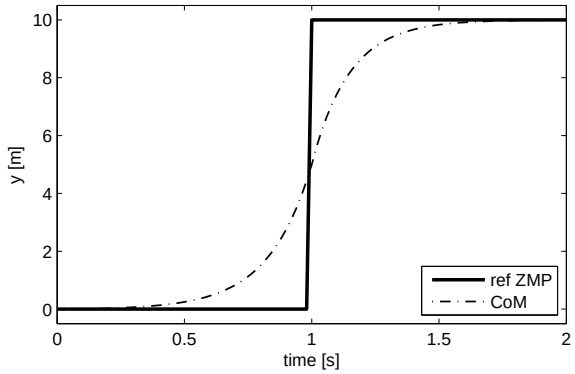


Abbildung 2.11: Um den Soll-ZMP (ref ZMP) zu erreichen, muß die Bewegung des Schwerpunktes beginnen bevor der ZMP sich bewegt (CKU09a).

Um online eine funktionierende Lösung berechnen zu können, schlägt Kajita et al. einen „Preview-Controller“ vor (KKK$^+$03a). Dieser benötigt eine Vorschau des Soll-ZMP (normalerweise ca. 1 Sekunde in die Zukunft) und ist in der Lage, daraus die Schwerpunktbewegung zu berechnen. In der Abbildung 2.11 führt diese Vorschau dazu, dass die Schwerpunktbewegung vor der Bewegung des ZMPs beginnt, so dass er bei Position $10m$ zum Stehen kommen kann.
Für die Dortmund Walking Engine wurde der Ansatz um eine Sensorkontrolle erweitert, die den vom Roboter gemessenen ZMP verwendet, um seinen aktuellen Zustand zu schätzen. Die Messung des ZMPs erfolgt ebenfalls nach dem Ein-Schwerpunkt-Modell, indem in die Gleichungen 2.7 und 2.8 die gemessene Beschleunigung eingesetzt wird. Zur Vereinfachung werden die vom Sensor gelieferten Daten direkt verwendet, also die Beschleunigung des Oberkörpers, statt sie auf den Schwerpunkt umzurechnen. Da dieser aber seine eigene Bewegung im RKS hat, entsteht ein kleiner Fehler. Zum Beispiel nimmt der gemessene ZMP entlang der y-Achse betraglich zu große Werte an, was in Abschnitt 5.1 von Wichtigkeit sein wird. Neben der Messung ist die Sensorkontrolle in dieser Arbeit nicht relevant, für weitere Details sei daher auf die Literatur verwiesen (CKU09a; CKU09b).

3

MEHRKÖRPERMODELLEBENE

Die Mehrkörpermodellebene betrachtet im Gegensatz zu den abstrakteren Ebenen den Schwerpunkt und die Trägheit jedes einzelnen Festkörpers. Der wichtige Unterschied zur Simulationsebene SimRobot ist, dass hier auf die Behandlung von Reibung und Kollisionsberechnung verzichtet wird. Daraus folgt, dass der Kontakt mit dem Boden fest ist und der Roboter weder rutschen noch kippen kann.

Die Mehrkörpermodellebene stellt damit eine Ebene dar, die eine separate Untersuchung der Dynamikabstraktion und der Kinematikfehler erlaubt (siehe Abschnitt 1.1). Im Folgenden werden die Anforderungen an die Algorithmen dieses Kapitels formuliert, um die Untersuchung zu ermöglichen.

Anforderung der Dynamikabstraktion

Für die in der Einleitung vorgestellten Beobachtungen wurden bereits Hypothesen zu ihren Ursachen vorgestellt. Die *ZMP-Abweichung* äußert sich in SimRobot in einem fehlerhaft verlaufenden ZMP. Ein fehlerhafter ZMP kann auch zu einem Kippen führen. Daher steht das *Einknicken* in einem möglichen Zusammenhang mit der *ZMP-Abweichung* und beide zeigen, aufgrund der Beobachtbarkeit in SimRobot, einen möglichen Zusammenhang mit der Dynamikabstraktion. Es ist daher nötig, den ZMP auf der Mehrkörpermodellebene zu berechnen, um ihn mit dem ZMP der Ein-Schwerpunkt-Ebene zu vergleichen. Er wird in einem vergleichbaren Weltkoordinatensystem benötigt.

Der ZMP ist ein Resultat der Drehmomente an den Gelenken. Für die Suche nach den genauen Ursachen sind also auch die Drehmomente von Interesse.

Anforderungen der Kinematikfehlerabstraktion

Die *Vorwärtsschwingung*, *Seitwärtsschwingung* und *Geschwindigkeitssteigerung* sind nicht in SimRobot zu erkennen, weswegen für sie Kinematikfehler in Frage kommen. Dazu zählen fehlerhaft angefahrene Gelenkwinkel, Rutschen und Kippen. Um die genauen Ursachen zu identifizieren, ist ein Vergleich der Ebene „Walking Simulator“ zur Mehrkörpermodellebene nötig, siehe Abbildung 1.3. Dieser Vergleich ist anhand der Gelenkwinkel, der Oberkörperorientierung und Oberkörperposition möglich, wie sie von den Berechnungen auf den beiden Ebenen ausgegeben werden. Die tatsächlichen Gelenkwinkel der Mehrkörpermodellebene stimmen mit den Soll-Gelenkwinkel überein, da hier Kinematikfehler im Modell nicht enthalten sind. Die tatsächliche Oberkörperorientierung der Mehrkörpermodellebene entspricht auch immer dem Soll, nämlich konstant senkrecht. Ein Vergleich mit der tatsächlichen Oberkörperorientierung und den tatsächlichen Winkeln anderer Abstraktionsebenen ist demnach ohne weitere Berechnungen möglich. Die Oberkörperposition bzw. der vom Oberkörper zurückgelegte Weg stimmt auch mit dem Soll überein, steht allerdings nicht direkt zur Verfügung. Zur Erinnerung: Die Eingabe der Walking Engine ist die Geschwindigkeit und die Ausgabe die Gelenkwinkel. Die Position des Oberkörpers muß daher aus den Gelenkwinkeln errechnet werden, was mit einer Vorwärtskinematik möglich ist.

Kapitelübersicht

In diesem Kapitel wird der Algorithmus *getTorqueZMP* hergeleitet, der aus den Gelenkwinkelverläufen die Drehmomente und den ZMP berechnet.

Zunächst werden Koordinatensysteme, Bezeichnungen und Nummerierungen der Roboterelemente in Abschnitt 3.1 definiert, wie sie von den Algorithmen benötigt werden. Es folgt die Vorstellung des Algorithmus zur Berechnung der Drehmomente an den Gelenkachsen zu einem Zeitpunkt t (vgl. Abschnitt 3.2). Im nächsten Abschnitt 3.3 wird gezeigt, wie man aus den Ergebnissen der Drehmomentsberechnung auch den ZMP zum Zeitpunkt t in einem lokalen Koordinatensystem berechnen kann. Diese Abschnitte beziehen sich zunächst auf die Single-Support-Phase während des Laufens. Auf die Schwierigkeiten in der Double-Support-Phase wird im Abschnitt 3.4 eingegangen. Die ZMP-Position liegt zu diesem Zeitpunkt noch nicht im Weltkoordinatensystem vor. Dafür wird in Abschnitt 3.5 ein Algorithmus gezeigt, der mittels Vorwärtskinematik aus den Gelenkwinkeln die Kontanktpunkte der Füße mit dem Boden berechnet. Daraus ergibt sich ein Lauf in Weltkoordinaten, und so kann auch aus den einzelnen ZMP-Punkten ein ZMP-Verlauf in Weltkoordinaten errechnet werden. Zuletzt werden Testverfahren gezeigt, mit denen sich die Algorithmen validieren lassen (vgl. Abschnitt 3.6).

3.1 Definitionen

In diesem Abschnitt werden den Festkörpern und Gelenken[1] Indizes zugeordnet, die den Algorithmen der folgenden Abschnitte eine Bearbeitung in Schleifenform ermöglichen. Die Indizierung beginnt beim Boden, der den Festkörper-Index 0 erhält. Da die Verbindung von Fuß zu Boden über ein Gelenk definiert sein muß, wird hier ein virtuelles Gelenk ohne Freiheitsgrad eingefügt. Dieses virtuelle Gelenk erhält den Index 1, der Fuß ist Festkörper 1. Man beachte hier, dass es bei einem zweibeinigen Roboter verschiedene Möglichkeiten gibt, welcher Fuß zu welchem Zeitpunkt Kontakt zum Boden hat. Beim Laufen erhält in der Single-Support-Phase der Fuß mit dem Kontakt zum Boden den Index 1. In der Double-Support-Phase wird dafür der linke Fuß festgelegt. Die Indizierung ändert sich also während des Laufens.

Auf den Fuß folgt die Achse 2, die den Fuß mit Körper 2 verbindet. Das wird zunächst mit der Regel, dass Achse i die Festkörper $i-1$ und i verbindet, über den Oberkörper bis zum anderen Fuß fortgesetzt, siehe auch Abbildung 3.1. Dieser ist aber nicht notwendigerweise der letzte Körper, in der Double-Support-Phase hat er ebenfalls Kontakt mit dem Boden. Dieser Kontakt soll auch hier über eine Achse erfolgen. Problematisch ist, dass sie nur zu bestimmten Zeitpunkten existieren würde. Die Autoren Nakamura und Yamane haben sich bereits mit dem Thema der strukturveränderlichen Systeme in Bezug auf humanoide Roboter beschäftigt (NY00) und schlagen vor, dass auf einen Körper, der einen Kontakt mit einem anderen Körper haben kann, ein virtueller Körper folgt. Für den Fall, dass der Kontakt besteht, sind der virtuelle Körper und der kontaktierte Körper als gleich anzusehen. Dieses Prinzip wird hier übernommen, so dass die virtuelle Achse zu jedem Zeitpunkt existiert. Sei j der Index des zweiten Fußes, dann wird eine virtuelle Achse $j+1$ eingefügt, die einen virtuellen Körper $j+1$ mit Körper j verbindet. Wenn der Fuß Kontakt zum Boden hat, dann ist der virtuelle Körper $j+1$ als Boden anzusehen.

Ohne Arme würde der Roboter nur aus den beiden Beinen bestehen, die durch den Oberkörper verbunden sind. Das ist eine Kettenstruktur, mit dem Boden als Wurzel, und dem virtuellen Körper $j+1$ als Blatt. Durch Einbindung der Arme wird aufgrund der Verzweigung am Oberkörper eine Baumstruktur. Nach dem Schema, mit dem das zweite Bein indiziert wurde, wird erst der linke Arm indiziert, beginnend mit Index $j+2$ für die Achse vom Oberkörper zum ersten Armkörper. Gleiches gilt danach für den rechten Arm.

Der Kopf wird aus zwei Gründen nicht modelliert. Zum einen wird der Roboter so angesteuert, dass der Oberkörper immer senkrecht bleibt, die Trägheit des Kopfes spielt demnach bezüglich Rotationen keine Rolle. Zum anderen wird der Kopf von der Dortmund

[1] Gelenke bestehen in diesem Kapitel aus einer Achse.

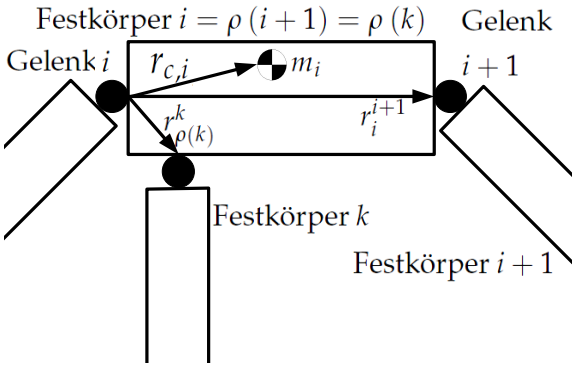


Abbildung 3.1: Darstellung der Konstanten und der Nummerierung.

Walking Engine nicht bewegt. Er wird daher mit dem Oberkörper zu einem Festkörper und einem Schwerpunkt zusammengefasst.

Koordinatensysteme

Einige Ergebnisse dieses Kapitels werden in Weltkoordinaten benötigt. Dazu wird ein Weltkoordinatensystem O_{WKS} festgelegt, dessen Ursprung zum Startzeitpunkt am Boden unter dem Ursprung des Roboterkoordinatensystems O_{RKS} liegt, zu diesem parallel und fest mit dem Boden verbunden ist. Das entspricht dem WKS in Simulationen.
Daneben gibt es für jede Achse i ein lokales Koordinatensystem O_i. Der Ursprung von O_i stimmt zu jedem Zeitpunkt mit der Position der Achse i überein. Zum Startzeitpunkt ist die Orientierung von O_i parallel zu O_{RKS} und dreht sich im Folgenden mit Festkörper i mit.
Mit diesen Definitionen können nun die Fußpositionen im RKS festgelegt werden: $p^{RKS}_{f_l} = p^{RKS}_{O_0}, p^{RKS}_{f_r} = p^{RKS}_{O_{j+1}}$.

3.2 Rekursiver Newton-Euler-Algorithmus

Der Algorithmus zur Berechnung der Drehmomente ist Thema dieses Abschnittes. In der Literatur wird ein solcher Algorithmus als inverse Dynamik bezeichnet (Fea07). Es gibt verschiedene physikalische Ansätze für inverse Dynamiken. Zu den bekanntesten zählt die Euler-Langrage-Gleichung, die auch Grundlage für die ODE ist (Hau09). Daneben gibt es das Newton-Euler-Verfahren. Der Vorteil zum Euler-Lagrange-Ansatz ist, dass der Roboter nicht als ein Ganzes angesehen wird, sondern jeder Festkörper und jedes Gelenk nacheinander in einer rekursiven Berechnung einfließt. Dieses Verfahren liefert daher während der Berechnung Zwischenergebnisse wie Geschwindigkeiten und Beschleunigungen der Körper, was die Ursachen der entstehenden Kräfte und Drehmo-

mente deutlicher macht. Eine weitere Stärke des Newton-Euler-Algorithmus ist die effiziente Berechenbarkeit.
Der hier vorgestellte rekursive Newton-Euler-Algorithmus (kurz RNEA) wurde von Luh, Walker und Paul entwickelt (LWP80) und wird daher auch in der Literatur oft als Luh-Walker-Paul-Algorithmus (kurz LWP-Algorithmus) bezeichnet. Verschiedene Autoren haben ihn weiterentwickelt und vereinfacht, so dass unterschiedliche Versionen des Verfahrens bekannt sind. Ursprünglich für Kettenstrukturen entwickelt (MH04; SV89), gibt es auch eine Version für Baumstrukturen (SK08). Für die mathematische Herleitung der einzelnen Gleichungen des Algorithmus sei auf die genannten Quellen verwiesen.
Die hier vorgestellte Version lehnt sich an der Version von Mamadyl und He an (MH04) und verwendet für alle Konstanten und Zwischenergebnisse das Koordinatensystem 0. Die Verwendung der Indizes ist der in Abschnitt 3.1 vorgenommenen Definitionen angepasst und auf die überflüssige Berechnung der linearen Geschwindigkeit wird verzichtet. Durch Verwendung der Funktion $\rho\,(i)$, die einem Körper i einen Vorgänger zuordnet (siehe auch Tabelle 3.1), ist es möglich den Algorithmus auch auf Baumstrukturen anzuwenden. Dieses Konzept wird in (SK08) ebenfalls verwendet.

Ein- und Ausgabe

Ausgabe des Algorithmus sind der ZMP im Koordinatensystem 0 (siehe Abschnitt 3.3) und die Drehmomente um die frei drehbaren Achsen, zunächst für die Single-Support-Phase.
Eingabe sind die Winkelpositionen, -geschwindigkeiten und -beschleunigungen der Gelenke zu einem Zeitpunkt t, sowie das Modell des Systems. Die Werte der Konstanten des Modells (siehe Tabelle 3.1) lassen sich nahezu vollständig aus den Spezifikationen von Aldebaran errechnen, wobei sich wegen der zwei möglichen Startpunkte der Indizierung zwei Modelle ergeben.
Die Trägheitstensoren I_i^j werden vom Hersteller Aldebaran nicht spezifiziert, und eine nachträgliche Messung ist nicht möglich. Daher wird, wie auch in Simulationen üblich, für jeden Festkörper von einem Quader mit homogen verteilter Masse ausgegangen.
Dann gilt für den Trägheitstensor (Hau09):

$$I_i^i = \frac{m}{12} \cdot \begin{pmatrix} {}^2l_i^2 + {}^3l_i^2 & 0 & 0 \\ 0 & {}^1l_i^2 + {}^3l_i^2 & 0 \\ 0 & 0 & {}^1l_i^2 + {}^2l_i^2 \end{pmatrix} \tag{3.1}$$

und ausgedrückt im Koordinatensystem 0:

$$I_i^0 = R_i^0 \cdot I_i^i \cdot \left(R_i^0\right)^\top \tag{3.2}$$

Die Dimensionen der Körper werden passend zu denen im Walking Simulator gewählt, siehe dazu Abschnitt 4.3.

Variable / Konstante	Bedeutung
n	Anzahl der Gelenke inkl. der virtuellen Gelenke zur Welt (mindestens 2).
$q_i, \dot{q}_i, \ddot{q}_i$	Position, Geschwindigkeit, Beschleunigung von (virtuellem) Gelenk i.
R_i^j	Rotationsmatrix von Koordinatensystem i nach j.
m_i	Masse von Festkörper i. Virtuelle Festkörper haben Masse 0.
$\rho(i)$	Vorgänger von Festkörper i. Es gilt $\rho(i) = i - 1$ außer bei den Festkörpern der Arme, die mit dem Oberkörper verbunden sind.
r_i^j	Vektor von Gelenk i nach Gelenk j.
$r_{c,i}$	Vektor von Gelenk i nach Schwerpunkt i.
I_i^j	Trägheitstensor von Festkörper i ausgedrückt in Koordinatensystem j.
g	Gravitation der Form $\begin{pmatrix} 0 \\ 0 \\ -9.81 \end{pmatrix}$.
M_i	Drehmoment an Achse i.
p_{zmp}^0	Position des Zero Moment Point in Koordinatensystem 0.
$\omega_i, \dot{\omega}_i$	Winkelgeschwindigkeit/Winkelbeschleunigung von Festkörper i.
a_i	Lineare Beschleunigung von Achse i.
$a_{c,i}$	Lineare Beschleunigung von Schwerpunkt i.
f_i	Kraft ausgeübt von Festkörper $\rho(i)$ auf Festkörper i. ${}^x f_i$ selektiert die x. Komponente von f_i.
N_i	Drehmoment ausgeübt von Festkörper $\rho(i)$ auf Festkörper i. ${}^x N_i$ selektiert die x. Komponente von N_i.
${}^x l_i$	Die x. Komponente der Länge von Festkörper i.
b_i	Einheitsvektor entlang der Achse von Gelenk i.

Tabelle 3.1: Variablen und Konstanten für RNEA.

Funktionsweise des RNEA

Der Algorithmus arbeitet nach dem Prinzip der Newton-Euler-Gleichungen (Hah02). Dabei werden für jeden Festkörper die Kraft und das Drehmoment auf den Schwerpunkt bezogen berechnet, was die dazu nötigen Gleichungen vereinfacht. Das erlaubt eine effizientere Berechnung und wird daher in der Literatur üblicherweise zur Berechnung der inversen Dynamik verwendet.
Algorithmus 3.1 zeigt die hier verwendete Variante. Er ist eingeteilt in 2 Phasen: eine Phase für die Berechnung der Geschwindigkeiten und Beschleunigungen (Schleifen der Zeilen 1 bis 8), in der die Körper und Achsen in aufsteigender Reihenfolge betrachtet werden. In der zweiten Phase werden daraus die Kräfte und Drehmomente in absteigender Reihenfolge berechnet (Zeilen 9 bis 23).

Phase 1: Geschwindigkeiten und Beschleunigungen

Zunächst wird in der ersten Phase die von Phase 2 benötigte Winkelgeschwindigkeit (Zeile 2) und die Winkelbeschleunigung (Zeile 3) der Festkörper berechnet. Dazu wird zu der Winkelgeschwindigkeit $\omega_{\rho(i)}$ des Vorgängerfestkörpers die Rotationsgeschwindigkeit $\dot{q}_i$ der verbindenden Achse addiert. Analog wird mit der Winkelbeschleunigung verfahren. Zudem wird die lineare Beschleunigung des Schwerpunktes i ($a_{c,i}$) und des Gelenks i (a_i) berechnet (Zeilen 6 und 7).

Phase 2: Kräfte und Drehmomente

Zur Berechnung der Drehmomente werden die auftretenden Kräfte benötigt, die daher zuerst berechnet werden. In absteigender Reihenfolge wird die Kraft berechnet, die der Festkörper $\rho(i)$ auf Festkörper i ausübt (Zeile 12). An Verzweigungen der Baumstruktur haben mehrere Festkörper den selben Vorgänger, es ist daher sicherzustellen, dass die Berechnung nur einmal stattfindet (Zeile 11). Die Kraft, die ein Festkörper ausübt, muß auch auf ihn von seinem Vorgänger ausgeübt werden, daher addieren sich die Kräfte über die Iterationsschritte (Zeile 10).
Ähnlich verhält es sich bei der Berechnung der Drehmomente. Sie entstehen aus der Kraft auf den Nachfolger (Zeile 16), der Kraft vom Vorgänger und dem Drehmoment, das aus dem Trägheitsmoment des Festkörpers entsteht (Zeile 18).

Initialisierung

Für die Winkelgeschwindigkeit und -beschleunigung des Festkörpers 1 gilt $\omega_1 = 0$ und $\dot{\omega}_1 = 0$, da das erste Gelenk, das Festkörper 1 mit dem Boden verbindet, keinen Freiheitsgrad hat. Außerdem unterliegt das Koordinatensystem 0 keiner linearen Beschleunigung ($a_1 = 0$). Sei B die Menge der virtuellen Festkörper, die die Blätter der Baumstruktur dar-

Algorithmus 3.1 Rekursiver Newton-Euler-Algorithmus für Baumstrukturen (*RNEA*).

Eingabe: $q, \dot{q}, \ddot{q}$, Systemmodell: $n, R, m, \rho, r_c, r, I, g, b$

Ausgabe: M, p^0_{zmp}

for $i = 2$ **to** n **do**

$\omega_i = \omega_{i-1} + \dot{q}_i \cdot \left(R^0_{\rho(i)} \cdot b_i \right)$

$\dot{\omega}_i = \dot{\omega}_{i-1} + \ddot{q}_i \cdot \left(R^0_{\rho(i)} \cdot b_i \right) + \omega_{\rho(i)} \times \left(\dot{q}_i \cdot R^0_{\rho(i)} \cdot b_i \right)$

end for

for $i = 2$ **to** n **do**

$a_i = a_{\rho(i)} + \dot{\omega}_{\rho(i)} \times \left(R^0_{\rho(i)} \cdot r^i_{\rho(i)} \right) + \omega_{\rho(i)} \times \left(\omega_{\rho(i)} \times \left(R^0_{\rho(i)} \cdot r^i_{\rho(i)} \right) \right)$

$a_{c,i} = a_i + \dot{\omega}_i \left(R^0_i \cdot r_{c,i} \right) + \omega_i \times \left(\omega_i \times \left(R^0_i \cdot r_{c,i} \right) \right)$

end for

for $i = n$ **downto** 2 **do**

$f_{\rho(i)} = f_{\rho(i)} + f_i$

if $\rho(i) = i - 1$ **then**

$f_{\rho(i)} = f_{\rho(i)} - m_{\rho(i)} \cdot g + m_{\rho(i)} \cdot a_{c,\rho(i)}$

end if

end for

for $i = n$ **downto** 2 **do**

$N_{\rho(i)} = N_{\rho(i)} + N_i - \left(R^0_{\rho(i)} \cdot \left(r_{c,\rho(i)} - r^i_{\rho(i)} \right) \right) \times f_i$

if $\rho(i) = i - 1$ **then**

$N_{\rho(i)} = N_{\rho(i)} + \left(R^0_{\rho(i)} \cdot r_{c,\rho(i)} \right) \times f_{\rho(i)} + I^0_{\rho(i)} \cdot \dot{\omega}_{\rho(i)} + \omega_{\rho(i)} \times \left(I^0_{\rho(i)} \cdot \omega_{\rho(i)} \right)$

end if

end for

for $i = n$ **to** 1 **do**

$M_i = R^0_{\rho(i)} \cdot b_i \cdot N_i$

end for

$p^0_{zmp} = \begin{pmatrix} \frac{-^2 N_1}{^3 f_1} \\ \frac{^1 N_1}{^3 f_1} \\ 0 \end{pmatrix}$

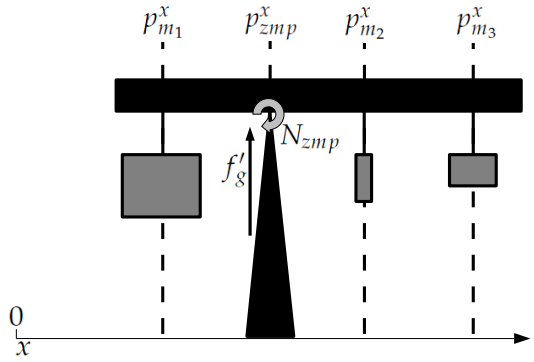


Abbildung 3.2: Eine Balkenwaage, die am ZMP gestützt wird.

stellen. Dann gilt für den Fall, dass der Roboter nur über Fuß 1 einen Kontakt zur Welt hat:

$$\forall i \in B : f_i = 0, N_i = 0 \tag{3.3}$$

Das gilt genau dann, wenn der Roboter sich in der Single-Support-Phase befindet. Auf den Fall der Double-Support-Phase wird in Abschnitt 3.4 eingegangen.

3.3 ZMP

Die Ergebnisse des RNEA aus dem letzten Abschnitt lassen sich auch zur Berechnung des ZMP nutzen. Per Definition ist N_1 und f_1 das externe Drehmoment bzw. die externe Kraft, die der Boden auf den Festkörper 1 ausübt. Nimmt man an, dass diese Kraft am ZMP p^0_{zmp} auf den Roboter ausgeübt wird, lässt sich folgende Formel für den ZMP aufstellen (SK08):

$$N_1 = p^0_{zmp} \times f_1 + N_{p^0_{zmp}} \tag{3.4}$$

Diese Gleichung entspricht, vereinfacht betrachtet, dem physikalischen Prinzip einer Balkenwaage. In Figur 3.2 erzeugen mehrere Gewichte eine Gewichtskraft von insgesamt $f_g = f_{m_1} + f_{m_2} + f_{m_3}$ und ein Gesamtdrehmoment von $N_g = N_{m_1} + N_{m_2} + N_{m_3}$, betrachtet im dargestellten Koordinatensystem x. Sei nun festgelegt, dass der Balken sich nicht bewegen soll. Dann muß der Boden eine Kraft von $f'_g = -f_g$ auf den Balken ausüben, gleiches gilt für das Drehmoment ($N'_g = -N_g$). Angenommen es übt eine Stütze auf den Balken die nötige Kraft f'_g und dazu ein Drehmoment N_{zmp} (z.B. durch einen Motor) an einem zunächst beliebigen Punkt p^x_{zmp} aus. Da der Balken stabil sein soll, gilt dann analog zu Gleichung 3.4, dass das Gesamtdrehmoment des Balkens (N_g) durch die Stütze ausgeglichen werden muß:

$$N'_g = p^x_{zmp} \times f'_g + N_{zmp} \tag{3.5}$$

Da man weder beim Roboter, noch bei einer Balkenwaage ein Drehmoment von außen wirken lassen kann (die Annahme, dass ein Motor bei p^x_{zmp} existiert ist normalerweise falsch), muß $N_{zmp} = 0$ gelten. Es bleibt in der Gleichung jetzt eine Unbekannte, die Position p^x_{zmp}, an der die Stütze greift. Sie muß so gewählt werden, dass $N'_g = p^x_{zmp} \times f'_g$ erfüllt ist. Dann entspricht der Punkt p^x_{zmp} dem ZMP.
Die Gleichung 3.4 muß noch nach p^0_{zmp} umgeformt werden. Mit $N_{p^0_{zmp}} = 0$ gilt:

$$
\begin{aligned}
& \begin{pmatrix} {}^1N_1 \\ {}^2N_1 \\ {}^3N_1 \end{pmatrix} = \begin{pmatrix} {}^2p^0_{zmp} \cdot {}^3f_1 - {}^3p^0_{zmp} \cdot {}^2f_1 \\ {}^3p^0_{zmp} \cdot {}^1f_1 - {}^1p^0_{zmp} \cdot {}^3f_1 \\ {}^1p^0_{zmp} \cdot {}^2f_1 - {}^2p^0_{zmp} \cdot {}^1f_1 \end{pmatrix} \\
\Rightarrow \quad & {}^1N_1 = {}^2p^0_{zmp} \cdot {}^3f_1 - {}^3p^0_{zmp} \cdot {}^2f_1 \\
& {}^2N_1 = {}^3p^0_{zmp} \cdot {}^1f_1 - {}^1p^0_{zmp} \cdot {}^3f_1 \\
\Leftrightarrow \quad & \left.\begin{aligned} {}^1p^0_{zmp} &= \frac{-{}^2N_1 + {}^3p^0_{zmp} \cdot {}^1f_1}{{}^3f_1} \\ {}^2p^0_{zmp} &= \frac{{}^1N_1 + {}^3p^0_{zmp} \cdot {}^2f_1}{{}^3f_1} \end{aligned}\right\} \text{ZMP liegt auf dem Boden} \Rightarrow {}^3p^0_{zmp} = 0 \\
\Rightarrow \quad & {}^1p^0_{zmp} = \frac{-{}^2N_1}{{}^3f_1}, {}^2p^0_{zmp} = \frac{{}^1N_1}{{}^3f_1}
\end{aligned}
\tag{3.6}
$$

Die Gleichungen 3.6 befinden sich in Algorithmus 3.1 in Zeile 24. Sie geben den ZMP im Koordinatensystem 0 an.

3.4 Schwierigkeiten der Double-Support-Phase

Ein Problem, das bei der Bezeichnung des Roboters als Baumstruktur zunächst ausgeblendet wird, ist die korrekte Berechnung in der Double-Support-Phase. In dieser Phase handelt es sich beim Modell zwar noch um eine Baumstruktur, welches aber die Realität physikalisch nicht richtig abbildet, da die Füße zusammen mit dem Boden einen Zyklus bilden (SK08). Das stellt bei der Berechnung nicht nur ein graphentheoretisches Problem dar.
Die Freiheitsgrade des Fußes mit dem Index j, $j > 1$, werden nichtlinear beschränkt, da gilt: ${}^3p^0_{O_{j+1}} \geq 0$. Das ist zunächst unkritisch, diese Beschränkung gilt auch für jeden nicht-humanoiden Roboter. In der Double-Support-Phase gilt ${}^3p^0_{O_{j+1}} = 0$ und $f_{j+1} \neq 0$, es wird also eine Kraft auf beide Füße ausgeübt, und es können zwei Fälle unterschieden werden:
Im ersten Fall ist die Reibung zwischen Fuß j und dem Boden niedrig, und für die Reibungskraft ${}^{1,2}f_R$ gilt $\left|{}^1f_R\right| < \left|{}^1f_j\right|$ und $\left|{}^2f_R\right| < \left|{}^2f_j\right|$, was bedeutet, dass der Fuß rutscht. Dieser Fall kommt bei korrekter Umsetzung der Winkel nicht vor, da sie in der Double-Support-Phase so gewählt werden, dass die Füße ihre Position zueinander halten.

Im zweiten Fall gilt $-^1f_R = {}^1f_j$ und $-^2f_R = {}^2f_j$. Der Fuß rutscht also nicht. Nun sind nicht mehr ausschließlich die Motoren dafür verantwortlich die Position des Fußes zu halten, sondern auch die Reibung sorgt dafür. Dadurch verringert sich die Anzahl der Freiheitsgrade, woraus folgt, dass die Anzahl der generalisierten Koordinaten des Roboters größer als die Anzahl der Freiheitsgrade ist. Es gibt dann unendlich viele Lösungen zur Aufteilung des nötigen Gesamtdrehmoments und der nötigen Gesamtkraft auf die Gelenke. Das Problem wird auch in der Literatur beschrieben (Vuk90; NG89; NY00). Als Lösung wird eine Optimierung, z.B. mit dem Ziel nach möglichst niedrigem Energieverbrauch, vorgeschlagen. Hier kommen allerdings Regler zum Einsatz, die die Drehmomente bestimmen. Auf dieser Modellebene werden ihre Ungenauigkeiten ausgeklammert. Die Drehmomente, die aufzubringen sind, um die gewünschten Winkel zu erreichen, werden immer erreicht. Dennoch gibt es unendlich viele gültige Lösungen, und es ist nicht klar, welche von den Reglern angestrebt wird. Eine Optimierung müsste daher darauf abzielen diese Lösung zu finden, was im Rahmen dieser Arbeit nicht möglich ist. Auf die Berechnung der Drehmomente in der Double-Support-Phase wird daher auf der Abstraktionsebene des Mehrkörpermodells verzichtet.
Die genannten Schwierigkeiten haben allerdings keinen Einfluss auf die Berechnung des ZMPs. Wenn auch die nötigen Drehmomente an den Gelenken mehrdeutig sind, so gilt das nicht für die Gesamtdynamik des System. Die Bewegung der Festkörper ist auch in der Double-Support-Phase eindeutig, und damit die Kräfte und Drehmomente, die der Boden auf den Roboter ausübt, ebenfalls.
Das kann man sich anhand der in Abschnitt 3.3 erwähnten Balkenwaage klar machen. Zur Erinnerung: Gleichung 3.5 besagt, dass das Gesamtdrehmoment N'_g vom Boden auf den Balken ausgeübt werden muß, damit der Balken stabil ist. Eine Möglichkeit ist, die Gegenkraft f'_g am ZMP wirken zu lassen, was das nötige Drehmoment erzeugt. Eine andere wäre, den Balken an einem anderen Punkt zu stützen, wobei aber das Drehmoment N_{zmp} im System fehlen würde. Eine zweite Stütze s_2 kann das System stabilisieren. Dadurch verteilt sich die Gesamtkraft auf zwei Stützen, bleibt insgesamt aber gleich. Auch das von der ersten Stütze s_1 ausgeübte Drehmoment $p^x_{s_1} \times f'_1$ ändert sich und mit $f'_g = f'_1 + f'_2$ gilt:

$$N'_g = p^x_{zmp} \times f'_g + N_{zmp} \tag{3.7}$$
$$= p^x_{s_1} \times f'_1 + N_{s_1} + p^x_{s_2} \times f'_2 + N_{s_2} \tag{3.8}$$

Es wird deutlich, dass es nicht auf die Art oder die Anzahl der Stützen bzw. Füße ankommt, mit der das Gesamtdrehmoment N'_g und die Gesamtkraft f'_g auf den Balken/Roboter ausgeübt wird. Der Ort des ZMP hängt nur von den beiden Größen ab, welche für jede Lösung der Kraft-/Drehmomentsberechnung gleich sind (sofern das gleiche Koordinatensystem gewählt wird).

Variable / Konstante	Bedeutung
$q(t), \dot{q}(t), \ddot{q}(t)$	Alle Winkelpositionen, -geschwindigkeiten, -beschleunigungen zum Zeitpunkt t.
$p_{O_{RKS}}^{WKS}(t)$	Position des Roboters im WKS zum Zeitpunkt t.
$p_{f_r}^{WKS}(t), p_{f_l}^{WKS}(t)$	Position des rechten / linken Fußes zum Zeitpunkt t.
$p_{zmp}^{WKS}(t)$	Position des ZMPs im WKS zum Zeitpunkt t.
T	Zeitobergrenze.
Δt	Zeitlicher Abstand zweier Frames.
f_l, f_r	Fuß links / rechts.
$\text{kin}_{f_a}(q(t))$	Vorwärtskinematik für Fuß a im RKS bei Winkel q zum Zeitpunkt t.
$\text{reverse}(q(t), \dot{q}(t), \ddot{q}(t))$	Liefert $q(t), \dot{q}(t), \ddot{q}(t)$ zum Zeitpunkt t in jeweils umgekehrter Reihenfolge und mit negierten Vorzeichen.
$\text{diff}(q_i)$	Diskrete Differenzierung der Winkelverläufe q_i.

Tabelle 3.2: Variablen und Konstanten für *getTorqueZMP*.

3.5 Weltkoordinatensystem-Konvertierung und -Berechnung

Der ZMP wird von Algorithmus 3.1 im Koordinatensystem 0 zu einem Zeitpunkt t zurück gegeben. Für einen besseren Vergleich zu den Ergebnissen der Simulation wird aber ein Verlauf im WKS benötigt (p_{zmp}^{WKS}).
Diese Konvertierung sollen die hier entwickelten Algorithmen leisten. Da keine Untersuchungen von Läufen mit Rotation geplant sind, sind die Algorithmen auf Läufe ohne Drehung beschränkt.

Fußpositionen im WKS

Algorithmus 3.2 zeigt den Ablauf zur Berechnung der Fußpositionen im WKS bei gegebenen Gelenkwinkeln. Sie stimmen mit den $p_{f_l}^{WKS}$ und $p_{f_r}^{WKS}$ aus Abschnitt 2.4.6 überein. Die Idee ist, zunächst die Fußpositionen im RKS aus den Winkelpositionen mit Hilfe der Vorwärtskinematik zu errechnen (Zeilen 1 bis 4). Für den ersten Zeitschritt werden die Positionen entlang der x- und y-Achse mit denen im RKS initialisiert (Zeilen 5 und 6). Zur Wandlung ins WKS ist für die folgenden Zeitschritte nur zu beachten, dass sich nur der Schwungfuß bewegt, während der Standfuß am Boden fest ist. Die Bewegung des Schwungfußes entspricht dem Abstand der beiden Füße im RKS. Das wird in Zeilen 10-12 berechnet.

Algorithmus 3.2 Algorithmus zur Berechnung der Fußpositionen im WKS für Läufe entlang der x-Achse (*getFootPositions*).

Eingabe: $\forall t : q(t)$

Ausgabe: $\forall t : p_{f_l}^{WKS}(t), p_{f_r}^{WKS}(t)$

1: **for** $t = 0$ **to** T **step** $t = t + \Delta t$ **do**
2: $p_{f_l}^{RKS}(t) = \text{kin}_{f_l}(q(t))$
3: $p_{f_r}^{RKS}(t) = \text{kin}_{f_r}(q(t))$
4: **end for**
5: $p_{f_l}^{WKS}(0) = \begin{pmatrix} {}^1p_{f_l}^{RKS}(0) \\ {}^2p_{f_l}^{RKS}(0) \\ 0 \end{pmatrix}$
6: $p_{f_r}^{WKS}(0) = \begin{pmatrix} {}^1p_{f_r}^{RKS}(0) \\ {}^2p_{f_r}^{RKS}(0) \\ 0 \end{pmatrix}$
7: **for** $t = 0$ **to** T **step** $t = t + \Delta t$ **do**
8: $p_{f_l}^{WKS}(t) = p_{f_l}^{WKS}(t - \Delta t)$
9: $p_{f_r}^{WKS}(t) = p_{f_r}^{WKS}(t - \Delta t)$
10: **if** Single-Support-Phase oder folgende Double-Support-Phase mit Fuß f_a am Boden, Fuß f_b ist (bzw. war) in der Luft **then**
11: $p_{f_b}^{WKS}(t) = p_{f_a}^{WKS}(t) + p_{f_b}^{RKS}(t) - p_{f_a}^{RKS}(t)$
12: **end if**
13: **end for**

ZMP im WKS

Algorithmus 3.3 Algorithmus zur Berechnung des ZMPs im WKS für Läufe entlang der x-Achse (*getTorqueZMP*).

Eingabe: $\forall t : q(t)$, Links-Modell, Rechts-Modell
Ausgabe: $\forall t : p_{zmp}^{WKS}(t)$

1: **for** $\forall i$ **do**
2: $\quad \dot{q}_i = \text{diff}(q_i)$
3: $\quad \ddot{q}_i = \text{diff}(\dot{q}_i)$
4: **end for**
5: $p_{f_l}^{WKS}, p_{f_r}^{WKS} = \text{getFootPositions}(\boldsymbol{q})$
6: **for** $t = 0$ **to** T **step** $t = t + \Delta t$ **do**
7: $\quad$ **if** f_l am Boden **then**
8: $\quad\quad p_{zmp}^{0}(t) = \text{rnea}(q(t), \dot{q}(t), \ddot{q}(t), \text{Links-Modell})$
9: $\quad\quad p_{zmp}^{WKS}(t) = p_{zmp}^{0}(t) + p_{f_l}^{WKS}(t)$
10: $\quad$ **else**
11: $\quad\quad p_{zmp}^{0}(t) = \text{rnea}(\text{reverse}(q(t), \dot{q}(t), \ddot{q}(t)), \text{Rechts-Modell})$
12: $\quad\quad p_{zmp}^{WKS}(t) = p_{zmp}^{0}(t) + p_{f_r}^{WKS}(t)$
13: $\quad$ **end if**
14: **end for**

Algorithmus 3.3 ist als zentraler Algorithmus des Kapitels zu verstehen, da er die einzelnen Algorithmen verwendet, um aus den gegebenen Gelenkwinkeln den ZMP im WKS zu bestimmen. Die Eingabe sind die Winkel und die Modelle, die sich durch einen Indizierungsstart am linken Fuß (Links-Modell) und am rechten Fuß ergeben (Rechts-Modell).

Als erstes müssen die Winkelgeschwindigkeiten und -beschleunigung errechnet werden. Sie stehen hier nicht direkt zur Verfügung und müssen aus den Positionen durch diskrete Differenzierung errechnet werden (Zeilen 1 bis 4). In der Schleife, die die einzelnen Zeitschritte bearbeitet, wird nach den beiden erwähnten Fällen unterschieden, ob mit der Indizierung am linken Fuß oder am rechten Fuß gestartet wurde (Zeile 7). Der RNEA wird hier verwendet um den ZMP zu berechnen und benötigt dazu, neben den Winkeldaten, das entsprechende Robotermodell. Für das Rechts-Modell müssen die Winkeldaten in Zeile 11 umsortiert und die Vorzeichen, bei gleichbleibenden Rotationsachsen, negiert werden, was von *reverse* durchgeführt wird. Zuletzt wird in den Zeilen 9 und 12 der ZMP im WKS berechnet.

3.6 Validierung der Implementation

In diesem Abschnitt werden Möglichkeiten vorgestellt, die Implementation der obigen Algorithmen zu testen, und die Ergebnisse gezeigt.

Validierung gegen Ein-Schwerpunkt-Modell

In Abschnitt 1 wurde festgestellt, dass die Walking Engine Vorgaben nicht korrekt umsetzt, wenn sie mit einem anderen Modell arbeitet, als ein Algorithmus in Sim-Richtung. Nach (KKK+03c) arbeitet das zu Grunde liegende 3D-LIP-Modell für den Fall eines einzelnen Schwerpunktes physikalisch korrekt. Ein erster Test ist daher das Modell für den RNEA so abzuändern, dass nur noch der Oberkörper eine Masse besitzt. Dadurch rechnet *getTorqueZMP* mit dem gleichen Modell wie die Walking Engine, was dazu führen sollte, dass der ausgegebene ZMP mit der Vorgabe übereinstimmt. Das testet die Berechnung der inversen Kinematik und die statischen und dynamischen Bereiche des RNEA sowie die Algorithmen *getTorqueZMP* und *getFootPositions*. Das Ergebnis für die ZMPs entlang der y-Achse zeigt Abbildung 3.3(a) und bestätigt die korrekte Funktionsweise.

Statische Tests

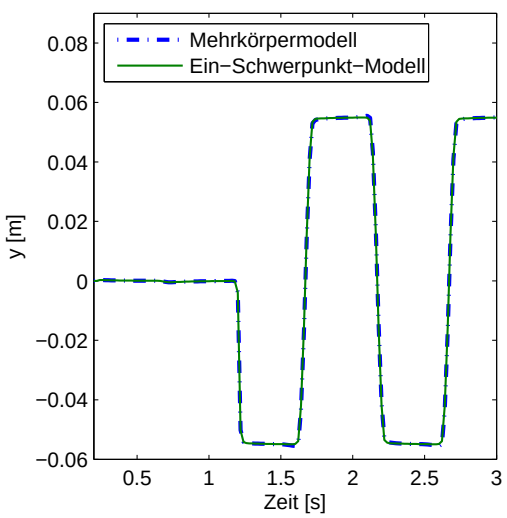
Ein derart abstraktes Modell lässt aber viel Raum für Fehler, die bei diesem einfachen Test nicht in Erscheinung treten. Daher besteht eine weitere Testmöglichkeit darin, in Algorithmus 3.3 in den Zeilen 8 und 11 die Vektoren $\dot{q}(t)$ und $\ddot{q}(t)$ durch einen Vektor aus Nullen zu ersetzen. Der so errechnete ZMP entspricht dann dem GCoM[2]. Er sollte mit dem von der Walking Engine vorgegebenem Schwerpunkt genau übereinstimmen. Auf die Art lassen sich die Berechnung des Gesamtschwerpunktes und der inversen Kinematik der Dortmund Walking Engine, die statischen Anteile des RNEA und die Algorithmen *getTorqueZMP* und *getFootPositions* überprüfen. Abbildung 3.3(c) zeigt das Ergebnis für die y-Achse, einen übereinstimmenden Verlauf.

Validierung des RNEA gegen existierende Implementationen

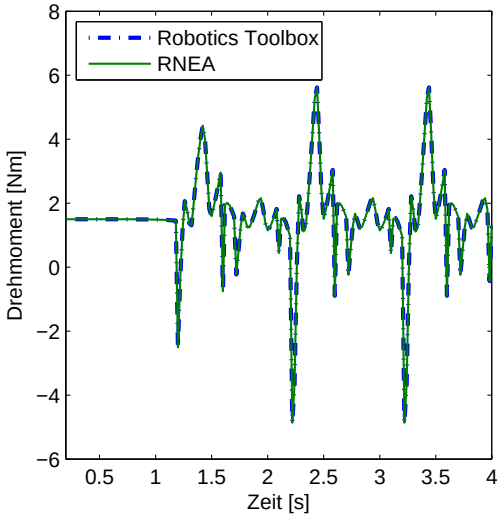
Die bisherigen Tests beinhalten zwar die Berechnung des ZMPs, aber damit nur die Berechnung des Gesamtdrehmoments und der Gesamtkraft. Um die Implementation des RNEA weiter zu testen, ist es daher notwendig, die Drehmomente mit Hilfe einer anderen Implementation zu überprüfen. Die Robotics Toolbox (Cor96) ist eine solche. Sie beschränkt sich allerdings auf kinematische Ketten, und berechnet nur die Drehmomente, nicht den ZMP. Dazu werden die Beine und der Oberkörper des Nao als serieller

[2]In der Literatur wird der auf den Boden projizierte Schwerpunkt als der „ground projected center of mass“ bezeichnet. Er ist eine einfache Methode, den ZMP näherungsweise zu bestimmen, indem die dynamischen Aspekte vernachlässigt werden.

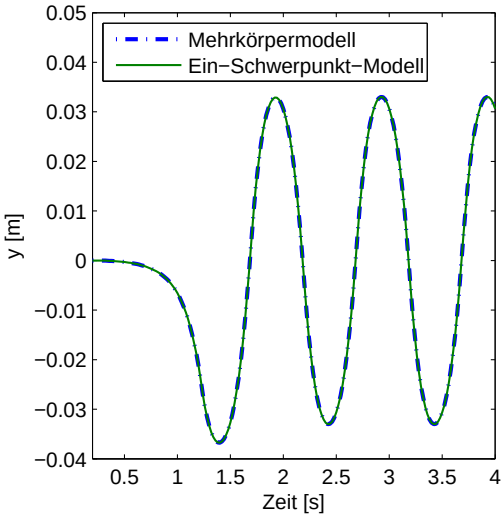
Roboter aufgefasst, das Modell in D-H-Parametern (SV89) ausgedrückt und der Robotics Toolbox übergeben. Abbildung 3.3(b) zeigt die übereinstimmenden Ergebnisse der beiden Algorithmen beispielhaft für das linke Knie.



(a) Test mit vereinfachtem Mehrkörpermodell.



(b) Drehmomentstest.



(c) Test per GCoM.

Abbildung 3.3: Ergebnisse der Tests.

4

WALKING SIMULATOR

In diesem Kapitel wird die Entwicklung des „Walking Simulator“ (WS) beschrieben. Mit ihm soll untersucht werden, welche Unterschiede sich in den Beobachtungen ergeben, wenn neben einer vollständigen Mehrkörperdynamik (Abschnitt 3) auch Kinematikfehler simuliert werden, siehe Abbildung 1.3. Zu den Kinematikfehlern gehört die Möglichkeit, dass die Füße nicht flach auf dem Boden aufliegen, mangelnde Haftung der Füße am Boden und insbesondere auch Einflüsse, die Fehler bei der Umsetzung der Winkel verursachen. Für letzteres ist die ODE nur wenig geeignet, die auch Grundlage dieses Simulators sein soll. Üblicherweise steuern Simulatoren die Gelenke an, indem sie die Winkelpositionen differenzieren und die Winkelgeschwindigkeit zur ODE senden, die diese umsetzt, ohne dabei Fehler zu machen. So arbeitet auch SimRobot, und lässt nur die Begrenzung des maximalen Drehmoments als Ungenauigkeit zu.

Das Ziel dieses Abschnittes ist eine realistischere Simulation. Sie soll die in der Einführung besprochenen Beobachtungen ebenfalls zeigen. Das bedeutet, der Roboter im Walking Simulator soll die *Vorwärtsschwingung* und *Seitwärtsschwingung* ebenfalls zeigen und ähnlich schnell zu Fall kommen. *Einknicken* sollte ebenfalls zu erkennen sein und die Geschwindigkeit sollte qualitativ das selbe Verhalten zeigen. Dazu werden Komponenten entwickelt, von denen theoretisch angenommen wird, dass sie diese Beobachtungen verursachen können. Der Erfolg wird in Abschnitt 5 untersucht.

Die Ansteuerung der Gelenke per Geschwindigkeitsvorgabe und deren ideale Umsetzung ist unrealistisch und wird durch einen PID-Regler und anschließendem Motormodell ersetzt, siehe Abschnitt 4.1. Der Motor überträgt sein Drehmoment auf das Getriebe, das aufgrund des Materials flexibel sein kann und herstellungsbedingt Toleranzen aufweist. Diese Winkelabweichungen können von den Sensoren des Naos gemessen wer-

Variable / Konstante	Bedeutung	Herstellerangabe
U	Vom Regler erzeugte Spannung.	-
L	Induktivität im Motormodell.	$0.000309H$
R	Widerstand im Motormodell.	6.44Ω
$\tau = \frac{L}{R}$	Zeitkonstante.	-
K_t	Drehmomentskonstante.	$0.0195\frac{Nm}{A}$
e_m	Gegen-EMF-Spannung.	-
i	Stromstärke im Motormodell.	-
ω_M	Winkelgeschwindigkeit des Motors.	-
K_s	Gegen-EMF-Konstante.	-
F_c	Konstante für geschwindigkeitsunabhängige Reibung.	-
B_v	Konstante für geschwindigkeitsabhängige Reibung.	-
T_f	Reibungsdrehmoment der geschwindigkeitsabhängigen Reibungskomponente.	-
U_{bat}	Spannung der Batterie des Naos.	$21.4V$
U_R	Spannung am Widerstand in der Ersatzschaltung.	-
S	*Stiffness* Parameter des Naos.	$[0, \ldots, 1]$
q_T	Ziel-Winkel.	-
q_A	Ist-Winkel.	-
T_M	Ausgabedrehmoment vom Motor.	-

Tabelle 4.1: Variablen und Konstanten für das Motormodell.

den. Die Umsetzung wird in Abschnitt 4.2.1 beschrieben. Eine Flexibilität, deren Auswirkung vom Nao nicht gemessen werden kann, und damit anders modelliert werden muß, ist die der Körper des Roboters, siehe Abschnitt 4.2.2. Die Umsetzung dieser beiden Elemente wird in Abschnitt 4.2.3 getestet. Der Nao wird danach in Abschnitt 4.3 aufgebaut. Da er ohne eine geeignete Parametrisierung der Simulation nicht wie gewünscht laufen wird, beschäftigt sich damit Abschnitt 4.4.

4.1 Motormodell

Einer der wichtigsten Unterschiede vom Nao zu Simulationen auf der Ebene von SimRobot ist, dass die ODE es erlaubt, die Geschwindigkeit von Gelenken vorzugeben. Die

ODE setzt diese Geschwindigkeit dann exakt um. Die Notwendigkeit einer weniger abstrakten Simulation ist auch von anderen Autoren bereits erkannt worden. Lima et al. (LGCM09) ersetzen diese Geschwindigkeitsansteuerung durch ein Motormodell mit Reibung und PID-Regler.

Algorithmus 4.1 Algorithmus zur Simulation von Motoren.

Eingabe: q_T, q_A, ω_M

Ausgabe: T_M

1: $U = PID\left(q_T, q_A\right)$
2: **if** $U > U_{bat}$ **then**
3: $\quad U = U_{bat}$
4: **end if**
5: $U_R\left(t + \Delta t\right) = U_R\left(t\right) + \left(U - U_R\left(t\right)\right) \cdot \left(1 - e^{-\frac{t}{\tau}}\right)$
6: $T_M = \underbrace{\frac{U_R}{R} \cdot S \cdot K_t}_{\text{Drehmoment durch den Strom}} \underbrace{- \left(B_v + K_s\right) \cdot \omega_M}_{\text{Drehmoment durch Reibung und Gegen-EMF-Spannung}}$

Einen Überblick der hier verwendeten Motorsimulation zeigt Algorithmus 4.1. Eingabe sind die Soll-Gelenkwinkel und die gemessenen Gelenkwinkel, Ausgabe ist das vom Motor erzeugte Drehmoment. Eine Übersicht der hier verwendeten Konstanten gibt Tabelle 4.1. Im Folgenden werden die einzelnen Komponenten beschrieben.

PID-Regler

Wenn auch klar ist, dass Regler im Nao zum Einsatz kommen, um die gewünschte Gelenkwinkelposition umzusetzen, sind Details leider unbekannt. Da aber nur Gelenkwinkelpositionen übergeben werden, liegt die Vermutung nahe, dass ein klassischer PID-Regler zum Einsatz kommt, wobei die Regelgröße die Winkel und die Stellgröße die Spannungen zu den Motoren sind. Ein solcher Regler wird daher hier eingesetzt, zu finden in Algorithmus 4.1, Zeile 1.

Reibung

In (LGCM09) wird auch die Reibung des Systems betrachtet. Es gibt eine winkelgeschwindigkeitsabhängige und -unabhängige Komponente. Die geschwindigkeitsunabhängige Komponenten F_c wird von der ODE, wie in Abschnitt 2.2 beschrieben, umgesetzt und ist damit den Achsen zuzuordnen. Die Komponente für geschwindigkeitsabhängige Reibung befindet sich in Algorithmus 4.1, Zeile 6.

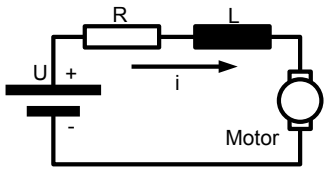


Abbildung 4.1: Elektrisches Modell des Motors.

Ersatzschaltung

Das Motormodell, wie in (LGCM09) beschrieben, besteht aus einer Ersatzschaltung mit einem Widerstand und einer Induktivität, einer Spannung, die vom Motor selbst verursacht wird und dem Motor selbst, wie in Abbildung 4.1 zu sehen. Nach (LGCM09) gilt für das Netz:

$$U = e_m + Ri + L \cdot \frac{\partial i}{\partial t} \tag{4.1}$$

Wir betrachten zunächst die Ersatzschaltung bestehend aus der Spannungsquelle, dem Widerstand und der Induktivität, die hier ebenfalls als Spannungsquelle betrachtet wird. Nach (HMS07) lässt sich das für eine konstante Spannung U_k, die zum Zeitpunkt 0 eingeschaltet wird, nach der Spannung am Widerstand auflösen zu:

$$U_R(t) = U_k \cdot \left(1 - e^{-\frac{t}{\tau}}\right) \tag{4.2}$$

Diese Gleichung ist ungeeignet für die Anwendung in einer Simulation. Günstiger wäre, die Spannung über Zeitschritte der Länge Δt iterativ zu berechnen.
Für die Zeitkonstante τ gilt bei den Motoren des Nao:

$$\tau = \frac{L}{R} = 0.00004781s \tag{4.3}$$

Typische Frequenzen für Physiksimulationen sind zum Beispiel $50Hz$ bei SimRobot bis zu mehreren hundert Hz. Sei zunächst keine Spannung angelegt und das System zum Zeitpunkt $t = 0$ eingeschaltet. Bei einer Simulationsfrequenz von $1kHz$ beträgt nach Gleichung 4.2 die Spannung U_R nach $\Delta t = 0.001s$ bereits 99.99999991% der Spannung $U(0)$. Der zu erwartende Einfluss der Simulation des elektrischen Feldes ist daher gering. Die iterative Simulation des Feldes befindet sich in Algorithmus 4.1, Zeile 5.
Bisher nicht betrachtet wurde die vom Motor induzierte Spannung e_m. Sie induziert eine von der Geschwindigkeit vom Motor abhängige Spannung $e_m(t) = K_s \cdot \omega_M(t)$. Vernachlässigt man den Einfluss des elektrischen Feldes, entspricht das einer geschwindigkeitsabhängigen Reibung, die bereits betrachtet wird. Da der Einfluss des elektrischen Feldes hier gering ist, und zudem die Konstante K_s vom Motorhersteller nicht angegeben wird, wird die Spannung e_m hier nicht weiter betrachtet und der Reibung zugeordnet.

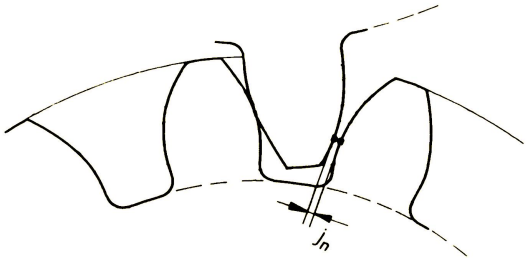


Abbildung 4.2: Toleranz bei einem Zahnradgetriebe (MMW$^+$01).

Nichtlinearitäten

Die maximale Spannung im System entspricht der Batteriespannung U_{bat} und beschränkt daher die Spannung U.
Die Begrenzung des Stroms und damit des ausgeübten Drehmoments wird von Aldebaran im Zusammenhang mit dem Stiffness Parameter S vorgenommen, siehe Abschnitt 2.1. Die Funktionsweise ist nicht näher dokumentiert, die Bedeutung lässt sich aber anhand von Beobachtungen bestimmen. Der Parameter wird hier umgesetzt, indem der Strom am Motor mit S multipliziert wird. Ob und inwiefern eine echte Obergrenze existiert, ist unklar.

4.2 Flexibilität und Toleranz

Dieser Abschnitt beschäftigt sich zunächst mit der Simulation von Flexibilität und Toleranz im Getriebe und anschließend mit flexiblen Körpern des Roboters. Während die Winkelfehler, die durch die Getriebe entstehen, vom Nao messbar sind (siehe Abschnitt 2.1), sind die Fehler der flexiblen Körper nicht messbar. Beide Flexibilitäten separat zu simulieren macht daher nicht nur wegen den unterschiedlichen Ausgangsorten Sinn.

4.2.1 Flexibilität und Toleranz im Getriebe

Toleranzen sind eine gewollte Eigenschaft von Zahnradgetrieben, da damit Fertigungsungenauigkeiten und Ausdehnung aufgrund von Erwärmung ausgeglichen werden können (MMW$^+$01). Das am Nao verwendete Material leidet außerdem unter starkem Verschleiß, der ebenfalls zu Toleranzen führt. Abbildung 4.2 zeigt die Toleranz bei einem Zahnradgetriebe.
Die Problematik der Toleranzen in Getrieben wurde bereits in einer anderen Arbeit versucht umzusetzen (Hei07), verursachte aber eine instabile Simulation. Der Grund dafür ist, dass versucht wurde, die Simulation der Toleranz auf ODE-Seite zu implementieren. Daher wird das Modell des Roboters hier nicht zur Simulation von Getrieben abgeändert, sondern Vorberechnungen eingesetzt.

Algorithmus 4.2 Algorithmus zur Simulation von Getrieben.

Eingabe: T_M

Ausgabe: T_O

1: $T_O = 0$
2: $T_M = T_M \cdot \sigma$
3: **for** $i = 0$ **to** $\frac{\Delta t_s}{\Delta t_g}$ **do**
4: $\quad p_h = 0$
5: $\quad$ **if** $p_m > l_h$ **then**
6: $\quad\quad p_h = p_m - l_h$
7: $\quad$ **end if**
8: $\quad$ **if** $p_m < 0$ **then**
9: $\quad\quad p_h = p_m$
10: $\quad$ **end if**
11: $\quad T_F = D_g \cdot p_h$
12: $\quad p_m = p_m + \dot{p}_m \cdot \Delta t_g$
13: $\quad \dot{p}_m = \dot{p}_m + \frac{\left(T_M - T_F - B_g \cdot \dot{p}_m\right)}{m_g} \cdot \Delta t_g$
14: $\quad T_O = T_O + T_F \cdot \frac{\Delta t_g}{\Delta t_s}$
15: **end for**

Ähnliche Auswirkungen wie die Toleranz hat die Flexibilität von Bauteilen. So ist zum Beispiel eine Verformung der einzelnen Zähne des Getriebes denkbar, die von der Stärke der Zähne abhängt. In diesem Abschnitt wird die Flexibilität daher zusammen mit der Toleranz von Getrieben simuliert.

Die Simulation des Getriebes ist nach der Motorsimulation geschaltet, hat demnach als Eingabe das Drehmoment, das von Algorithmus 4.1 ausgegeben wird, und als Ausgabe das Drehmoment, dass der ODE zur Anwendung auf die Festkörper übergeben wird. Tabelle 4.2 zeigt die verwendeten Variablen und Konstanten und Algorithmus 4.2 das Verfahren, auf das im Folgenden näher eingegangen wird.

Flexibilitäten und Toleranzen treten normalerweise über das gesamte Getriebe in Erscheinung. Um das exakt nachzuempfinden, müsste das Drehmoment für jedes einzelne Zahnrad berechnet werden, um die Kräfte zwischen den einzelnen Zähnen berechnen zu können. Das ist hier allerdings nicht möglich, daher wird zunächst die Übersetzung auf das vom Motor ausgeübte Drehmoment angewendet (Zeile 2).

Abbildung 4.3 zeigt das Modell, dem die Simulation der Flexibilität und Toleranz zu Grunde liegt. Es arbeitet nicht mit Drehmomenten und Winkelgeschwindigkeiten, sondern mit linearen Größen. Das liegt nah an der Realität, da die einzelnen Zähne Kräfte aufeinander ausüben, und die Toleranz j_n in Abbildung 4.2 ebenfalls eine Strecke ist. Zur

p_h	Position der oberen Kante der Hülle.
p_m	Position der Masse (Masse selbst ist punktförmig).
$\dot{p}_m$	Geschwindigkeit der Masse.
D_g	Federkonstante der Getriebefederung.
B_g	Reibungskoeffizient der geschwindigkeitsabhängigen Reibung der Masse.
Δt_s	Länge eines Zeitschrittes der gesamten Physiksimulation.
Δt_g	Länge eines Zeitschrittes der Getriebesimulation. Es gilt: $\Delta t_s \geq \Delta t_g$.
l_h	Länge der Hülle.
T_O	An die ODE übergebene Drehmoment.
σ	Übersetzungsverhältnis.
T_F	Von der Feder auf den Festkörper ausgeübte Kraft.
m_g	Gewicht der Masse.

Tabelle 4.2: Variablen und Konstanten für die Getriebesimulation.

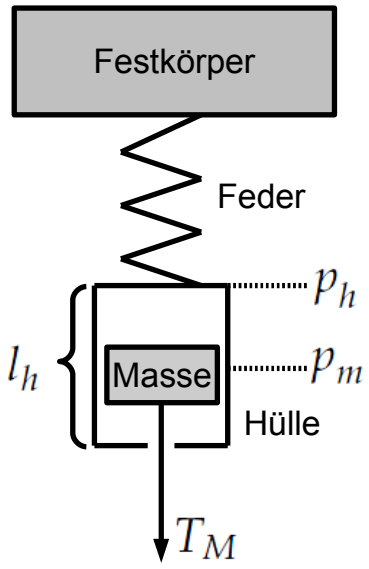


Abbildung 4.3: Modell zur Simulation der Flexibilität und Toleranz.

Umrechnung von Kraft zu Drehmoment und umgekehrt wird der Radius der Zahnräder benötigt. Dieser ist nicht bekannt und würde nur die anderen, ebenfalls unbekannten, Größen skalieren, und hat so keinen wesentlichen Einfluss. Aus Effizienzgründen wird daher hier zur Umrechnung von einem Zahnrad mit dem Radius 1 ausgegangen.

Die Kraft des Motors greift zunächst an einer neuen, kleinen Masse, die der Masse der beweglichen Teile des Getriebes entspricht. Ihre Dimension in der Grafik dient aber nur zur Verdeutlichung, sie ist als punktförmig zu betrachten. Ohne eine Reibung gerät die Masse und die Feder in eine Schwingung, die nicht endet. Die Masse unterliegt daher einer virtuellen Reibung zur Umgebung, ist aber nicht mit einem der anderen Elemente verbunden. Tests haben ergeben, dass eine geschwindigkeitsunabhängige Reibung zu einer Simulation führt, bei der es kaum möglich ist, Simulationsparameter zu finden, bei denen der Roboter in der Lage ist, zu laufen. Daher handelt es sich um eine geschwindigkeitsabhängige Reibung (siehe Zeile 12).

Die Masse wird von einer Hülle umgeben, die die Toleranz simuliert. Für den Fall, dass die Feder nicht gespannt ist und keine Kraft ausübt, wird die Position p_h der Hülle, die mit der Feder verbunden ist, als 0 definiert. In diesem Fall bewegt sich die Masse frei innerhalb der Hülle. Da die Feder und die Hülle keine Masse und damit keinen Impuls hat, hat die Masse genau dann keinen Kontakt zur Hülle (Bedingung in Zeile 4 und 7 ist nicht wahr), wenn $p_h = 0$ gilt. Gilt $p_h \neq 0$, so wird die Feder gespannt und die Kraft der Feder (Zeile 10) wird direkt auf die Masse angerechnet (siehe Zeile 12).

Die Summe der auf die Masse ausgeübten Kräfte führt zu einer Beschleunigung, die nach der Euler-Methode (Hau09) integriert wird zur Geschwindigkeit (Zeile 12) und zur Position (Zeile 11).

Bisher außer Acht gelassen wurde die iterative Berechnung. Wie auch bei anderen Physiksimulationen stellen die ungleich skalierten Massen der Festkörper des Roboters und der Masse des Getriebemodells eine Schwierigkeit dar. Das große Drehmoment der Motoren beschleunigt die Masse so stark, dass die Kollision mit der Hülle in manchen Fällen zu spät erkannt wird und die Feder deutlich stärker gedehnt wird, als eigentlich richtig wäre. Das führt zu einer instabilen Simulation, so dass die Zeitschritte, die zwischen zwei Kollisionsprüfungen liegen, verkleinert werden müssen. Das vergrößert auch die Genauigkeit des Integrators. Da kleinere Zeitschritte bei der gesamten Simulation allerdings auch zu einem deutlichen Anstieg der Simulationsdauer führen würden, werden pro Zeitschritt der ODE mehrere Simulationsschritte der Getriebesimulation durchgeführt. Das ausgegebene Drehmoment ist der Durchschnitt aller Drehmomente der $\frac{\Delta t_s}{\Delta t_g}$ Zeitschritte.

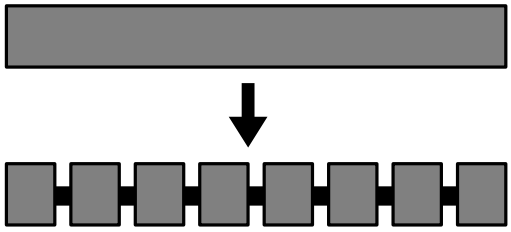


Abbildung 4.4: Beispiel der Unterteilung eines Festkörpers in mehrere kleinerer Größe zur Bildung eines flexiblen Körpers.

4.2.2 Flexible Körper

Neben den Flexibilitäten der Getriebe, ist eine weitere Flexibilität denkbar, die der Körper des Roboters. Zwar ist die ODE eine Festkörpersimulation, es ist aber möglich, flexible Körper zu einem gewissen Grad nachzubilden.
Ein flexibler Körper wird nachgebildet, indem der eigentliche Festkörper entlang einer wählbaren Achse χ durch eine feste Anzahl kleinerer Körper ersetzt wird. Das Gewicht des flexiblen Körpers wird gleichmäßig auf die kleineren Festkörper verteilt. Abbildung 4.4 zeigt ein Beispiel. Die kleineren Körper werden durch Gelenke mit jeweils zwei Freiheitsgraden (Achsen senkrecht zu χ) verbunden. Die Achsen üben ein Drehmoment in Abhängigkeit vom ausgelesenen Winkel α (und der Federkonstanten D_B) auf die verbunden Körper nach dem Hookeschen Gesetz aus:

$$T_B = -D_B \cdot \alpha \tag{4.4}$$

Zur Dämpfung der Schwingung wird eine Reibung benötigt. Hier wird eine geschwindigkeitsunabhängige Reibung $T_{R,B}$ verwendet, da sie von der ODE selbst umgesetzt werden kann.

4.2.3 Test der Flexibilität und Toleranz

Bevor die entwickelten Getriebe und flexiblen Körper im Nao-Modell eingesetzt werden können, sind Tests nötig, die die zu erwartende Funktion sicherstellen.

Test des Getriebes

Zum Test des Getriebes wird ein Festkörper fest mit dem Boden verbunden. An ihm wird ein weiterer Festkörper über einen Motor mit Getriebe verbunden. Der Winkel ist frei einstellbar und ist neben dem ausgelesenen Winkel die Eingabe für das Gelenk. Abbildung 4.5 zeigt den Testaufbau.
Abbildung 4.6 zeigt die Ein- und Ausgabe des Getriebes. Ausgangssituation ist ein Soll- und Ist-Winkel 0. Zum Zeitpunkt von ca. $0.1s$ wird ein positiver Winkel gesetzt und das

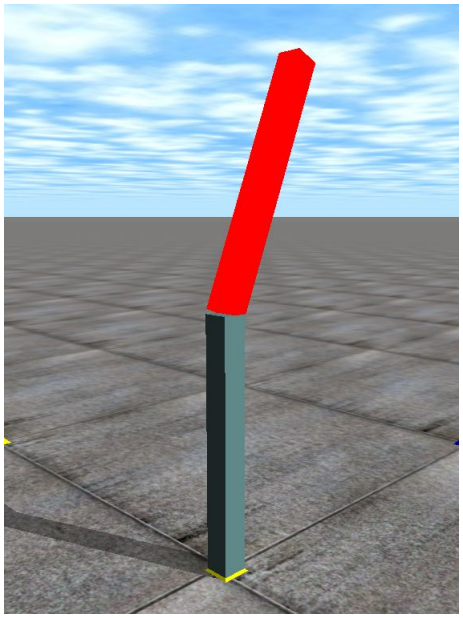


Abbildung 4.5: Aufbau zum Test des Getriebes.

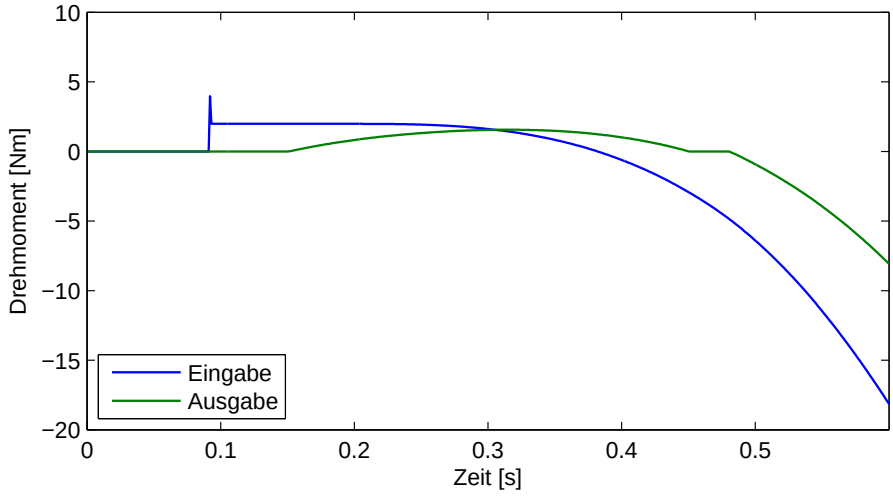


Abbildung 4.6: Darstellung der Übertragung des Drehmoments vom Getriebe-Eingang zum Getriebe-Ausgang.

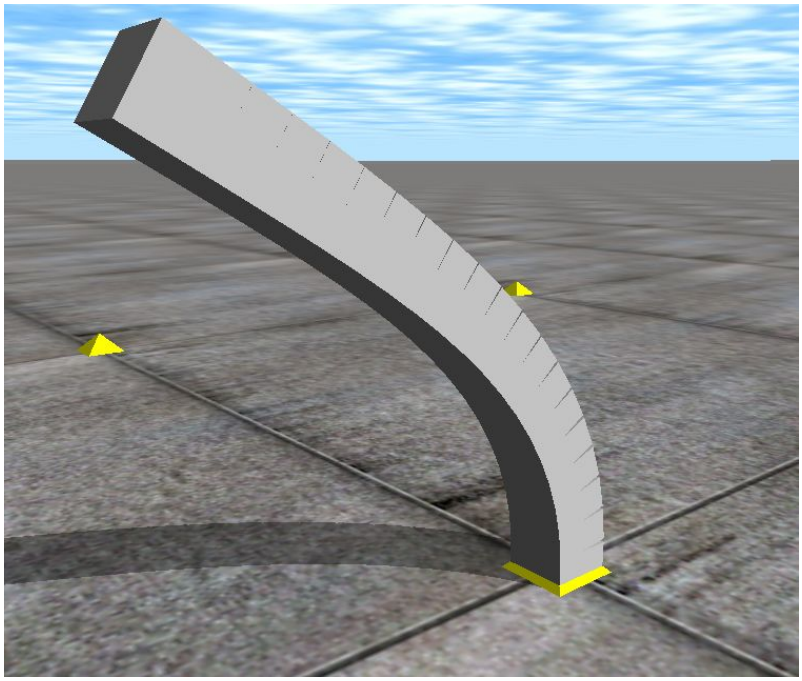


Abbildung 4.7: Aufbau zum Test des flexiblen Körpers.

Getriebe erhält nun ein positives Drehmoment. Zunächst wird die Masse beschleunigt, und es vergehen einige hundert Millisekunden bevor die Masse das Hüllenende erreicht und die Feder spannt, wodurch ein ansteigendes Drehmoment ausgegeben wird. Obwohl bei ca. $0.3s$ der Winkel sich dem Ziel nähert und das Eingabedrehmoment kleiner wird, steigt das Ausgabedrehmoment zunächst weiter an, was mit der Trägheit der Masse zu erklären ist, die zunächst die Feder weiter spannt. Kurz darauf wird die Soll-Position erreicht und überschritten, es wechselt also das Vorzeichen des Eingabedrehmoments um das Gelenk abzubremsen. Beim Vorzeichenwechsel des Ausgabedrehmoments zeigt sich erneut die Toleranz des Getriebes: Für eine kurze Zeit wird kein Drehmoment ausgegeben, genau dann, wenn die Masse keinen Kontakt zur Hülle hat.

Test des flexiblen Körpers

Für den Test des flexiblen Körpers wird ein Aufbau wie in Abbildung 4.7 gewählt, die einen sich in Schwingung befindenden Körper zeigt. Ein einzelner flexibler Körper besteht hier aus 20 kleineren Festkörpern. Der unterste ist fest mit dem Boden verbunden, und auf den oberen können Kräfte angewendet werden, die den Körper verformen und schwingen lassen. Zu erkennen ist die steigende Abweichung von der Nullposition bei den unteren Gelenken. Das ist ein Resultat des steigenden Gewichts, das die unteren Gelenke zu stützen haben. Ansonsten ist in der Simulation das zu erwartende gedämpfte Schwingverhalten zu sehen.

Test mit unterschiedlichen Parameterkonfigurationen wie mehr Unterkörpern oder anderer Reibung oder Federkonstanten zeigen, dass diese Parameter einen Einfluß auf die Stabilität der Simulation haben. Mehr Unterkörper, kleinere Reibung und höhere Federkonstanten führen zu instabilen Simulationen. Zur Simulation von Körpern mit einer kleinen Flexibilität ist demnach eine kleine Anzahl von Unterkörpern zu wählen, damit höhere Federkonstanten verwendet werden können.

4.3 Aufbau des Naos

Die dargestellten Elemente, die zu einer realistischeren Simulation führen sollen, werden nun zu einem Modell vom Nao zusammengesetzt. Einige der Elemente können in diesem Abschnitt nicht vollständig parametrisiert werden, da Angaben der Hersteller fehlen, oder die gezeigten Simulationselemente bisher nicht für eine Simulation des Naos verwendet wurden. Es bedarf daher des im nächsten Abschnitt vorgestellten Verfahrens zur Findung von Parametern, die zu einem sich realistisch verhaltenden Nao führen.

Festkörper und flexible Körper

Das Modell vom Nao, dass der ODE übergeben wird, besteht aus Quadern in Form von Festkörpern und flexiblen Körpern. Vor allem längliche Körper wie Beine können in der Realität flexibel sein, weniger der Oberkörper oder die Füße. Deswegen werden für eine stabilere Simulation nur die Ober- und Unterschenkel aus flexiblen Körpern mit jeweils 2 Unterkörpern bestehen. Kopf, Oberkörper und Füße sind Festkörper. Die Gelenke zwischen Ober- und Unterarmen werden von der Dortmund Walking Engine konstant auf die Position 0 gesetzt. Die Arme bestehen daher nur aus jeweils einem Festkörper.

Massen und Dimensionen

Gewichte und Schwerpunktpositionen der Körper entsprechen denen, die in Abschnitt 3.2 festgelegt werden, und damit den Vorgaben von Aldebaran. Laut API-Dokumentation ist es möglich, der ODE Schwerpunktpositionen für jeden Festkörper vorzugeben, die nicht der Mitte des Festkörpers entsprechen. Tests haben aber ergeben, dass die ODE physikalisch falsch arbeitet, sobald diese Funktion verwendet wird. Daher werden die Zentren der Quader auf die vorgegebenen Schwerpunktpositionen gelegt. Als Dimension der Quader sollten auch Vorgaben von Aldebaran verwendet werden. Allerdings gibt Aldebaran nur bestimmte Dimensionen implizit durch Angabe der Achsabstände vor. Die Angaben werden daher durch Abmessen des realen Roboters vervollständigt. Die Position der Quader beeinflusst aber deren Dimensionen. Da eine Überlappung der Quader vermieden werden soll, muss die Dimension auf die maximale Größe begrenzt

werden, die durch die Festlegung der Mittelpunkte auf die Schwerpunktposition noch möglich ist (Abstand vom Schwerpunkt zur Kante des nächsten Quaders).
Eine Ausnahme bilden die Füße. Deren Dimensionen und Positionen bestimmen auch den Zeitpunkt der Berührung mit dem Boden, wenn sie nicht zu ihm parallel sind. Da die Laufergebnisse von diesem Zeitpunkt abhängen, haben die Festkörper der Füße eine weitgehend reale Größe und Position, unabhängig von der Schwerpunktposition.
Die hier vorgestellten Abmessungen werden auch für die Modelle des Algorithmus 3.1 aus Abschnitt 3.2 verwendet.

Gelenke und Achsen

Die Achsen der Gelenke werden nach den Angaben von Aldebaran gewählt. Eine Ausnahme bildet *LHipYawPitch*. Die ODE bietet keine Möglichkeit, ein dreiachsiges Gelenk zu bilden, und eine zusätzliche Achse durch einen Körper der Dimension 0 mit dem zweiachsigen Hüftgelenk zu verbinden würde die Simulation destabilisieren. Ein Verzicht auf *LHipYawPitch* hat zwar den Nachteil, dass auch dessen Einfluss auf den Lauf fehlt und der Roboter sich nicht drehen kann (was nach Aufgabenstellung aber auch nicht notwendig ist). Eine stabile Simulation wird aber in dieser Arbeit priorisiert.
In den Beinen des Naos befinden sich Getriebe mit zwei unterschiedlichen Übersetzungsverhältnissen. Davon ausgehend, dass diese beiden Varianten sich nicht nur durch ihr Übersetzungsverhältnis unterscheiden, werden sie auch unterschiedlich parametrisiert, je einen Parametersatz für die beiden Übersetzungsverhältnisse. In Tabelle 4.3 sind die Parameter, die für jedes Übersetzungsverhältnis separat vorhanden sind, mit einer zwei markiert. Das sind die Getriebeparameter und Reibungsparameter des Motors, da letztere auch einen Einfluss auf die Reibung im Getriebe haben.
Die Armgelenke würden einen weiteren Parametersatz benötigen, der auch die Reglerparameter beinhalten müsste. Da Ungenauigkeiten in den Armen aber einen geringeren Einfluss haben als in die Beinen, werden sie, zur Reduktion der zu optimierenden Parameter und für eine stabilere Simulation, ausschließlich durch die Vorgabe der Winkelgeschwindigkeit angesteuert.

4.4 Parametrisierung

Die Entwicklung des Simulators verfolgt das Ziel einer physikalisch realistischen Simulation. In den vorigen Abschnitten wurden dazu einige Elemente vorgestellt, die die ODE dazu erweitern. Ohne eine geeignete Parametrisierung ist ein realistisches Verhalten allerdings nicht zu erreichen, eine solche ist daher Ziel dieses Abschnittes. Diese ist allerdings nicht von Hand zu erreichen, sondern wird mit einem Evolutionären Algorithmus gesucht. Das Augenmerk liegt dabei auf den in der Einführung besprochenen Beobach-

Variable / Konstante	Bedeutung	Abschnitt
K_P	P-Anteil des Reglers.	4.1
K_I	I-Anteil des Reglers.	4.1
K_D	D-Anteil des Reglers.	4.1
$B_v + K_s$ (2)	Summe für geschwindigkeitsabhängige Reibung plus Gegen-EMF-Konstante.	4.1
F_c (2)	Konstante für geschwindigkeitsunabhängige Reibung.	4.1
B_g (2)	Reibungskoeffizient der geschwindigkeitsabhängigen Reibung der Masse.	4.2.1
l_h (2)	Länge der Hülle.	4.2.1
m_g (2)	Gewicht der Masse.	4.2.1
D_g (2)	Federkonstante der Getriebefederung.	4.2.1
D_B	Federkonstante für flexible Körper.	4.2.2
$T_{R,B}$	Geschwindigkeitsunabhängige Reibung der Verbindungen bei den flexiblen Körpern.	4.2.2
cfm	Constraint Force Mix.	2.2
$soft_{cfm}$	Softness Constraint Force Mix.	2.2

Tabelle 4.3: Übersicht über die zu optimierenden Parameter.

tungen, die sich auch in der Simulation zeigen sollen. Dies ist aber eine subjektive Betrachtungsweise, die man einem Evolutionären Algorithmus nicht als Zielfunktion bzw. Fitnessfunktion vorgeben kann. Eine objektive Beschreibung des gewünschten physikalischen Verhaltens könnte über die Position und Orientierung der Körper geschehen. Diese Daten lassen sich aber nicht ohne weiteres beim realen Roboter messen. Was zur Verfügung steht, sind die generalisierten Koordinaten in Form von Gelenkwinkel. Sie lassen sich direkt vom Nao messen. Das Ziel der Parameterstudie ist somit eine Minimierung der quadratischen Differenz der gemessenen Gelenkwinkel q_r vom *realen* Roboter gegenüber den gemessenen Gelenkwinkeln q_s vom *simulierten* Roboter. Der Hintergrund ist, dass ein weitgehend gleicher Verlauf der Ist-Winkel auch zu vergleichbaren Orientierungen und Positionen des Roboters führen sollte.
Die *zu minimierende* Fitnessfunktion für alle (tatsächlich simulierten) Zeitschritte $t = 1, \ldots, T$ und alle Gelenke $i = 1, \ldots, n$ ist damit:

$$F\left(q_r, q_s\right) = \frac{\sum_{t=1}^{T} \sum_{i=1}^{n} \left(q_r\left(t\right) - q_s\left(t\right)\right)^2}{T^m} \tag{4.5}$$

Bei m handelt es sich um eine Gewichtung der Simulationsdauer. Ist der Wert größer, bevorzugt die Fitnessfunktion lange Simulationen, statt nur die Summe im Zähler zu minimieren. Die konkrete Wahl von m wird weiter unten beschrieben.

Simulationsaufbau

Um die gewünschten Beobachtungen auch in der Simulation zu erhalten, läge es nahe, die q_r in den entsprechenden Situationen aufzuzeichnen, in denen die Beobachtungen auftreten. Das kann aber dazu führen, dass die daraus erlernten Parameter zwar geeignet sind die Beobachtungen zu zeigen, sie aber die Realität im Allgemeinen nicht korrekt widerspiegeln. Um die Übertragbarkeit der Ergebnisse der Simulation auf die Realität zu stärken, sollen daher nicht die Beobachtungen direkt angelernt werden, sondern real funktionierende Läufe, bei denen die Sensorkontrolle deaktiviert ist, um unerwartete Einflüsse auszuschließen. Das betrifft nicht die Messung der Gelenkwinkel, auf der die Berechnung des Gesamtschwerpunktes basiert, sie kann nicht deaktiviert werden.
Insgesamt werden die q_r und q_s von vier realen Läufen auf dem selben Nao mit einer Dauer von jeweils ca. 10 Sekunden aufgezeichnet. Dazu kommen einige Sekunden zu Anfang, in denen der Roboter steht. Es werden zwei unterschiedliche Walking Engine-Parametersätze verwendet, jeweils einer für zwei Läufe, um allgemeinere Simulationsparameter zu erhalten. Da es sich beim Laufen um sich wiederholende und gleichförmige Bewegungen handelt, wird zusätzlich noch eine Schussbewegung aufgezeichnet. Die schnelle Beinbewegung und der Stand auf einem Bein sorgt für allgemeingültigere und stabilere Simulationsparameter.

Insgesamt bewertet ein Simulationslauf demnach anhand von gleichzeitig 5 Robotern ein Individuum. Die Anzahl der Gelenke n bei der Fitnessfunktion (Gleichung 4.5) ist die Summe aller Gelenke der 5 Roboter und entsprechend summiert sie über die Gelenke aller Roboter. Die Simulation endet, wenn sie 12 Sekunden dauerte, oder einer der Roboter umgefallen ist.

Parallele Fitnessauswertung

Für die Länge eines Zeitschrittes der Physiksimulation wird $0.001s$ gewählt, was einer Frequenz von $1kHz$ entspricht. Damit lässt sich eine hohe Genauigkeit und Simulationsstabilität erzielen (vgl. Abschnitt 2.2), führt aber zusammen mit der hohen Anzahl der einzelnen Körper von 5 Robotern zu einem deutlich höheren Zeitbedarf zur Auswertung eines Individuums. Außerdem ist mit einer hohen Anzahl von Auswertungen zur Optimierung der großen Anzahl von Parametern zu rechnen. Die Bewertung der Individuen wird daher parallelisiert auf dem Rechencluster des Instituts für Roboterforschung[1] durchgeführt, wo bis zu 180 CPU-Kerne zur Verfügung stehen. Die zu bewertenden Kinder einer Generation werden gleichmäßig auf die Knoten verteilt und parallel ausgewertet. Um den Netzwerk-Overhead niedrig zu halten, sollten pro Generation mehrere Individuen pro Knoten ausgewertet werden.

4.4.1 Vorbereitende Tests zur Parameterstudie

Die Wahl von m in Gleichung 4.5 ist noch ungeklärt. Um einen guten Wert zu finden, werden Tests mit einem vereinfachtem Aufbau durchgeführt. Die Simulation besteht aus nur einem Roboter, bei dem die flexiblen Körper und Getriebe deaktiviert sind. Als Zielwinkel für die Fitnessfunktion werden die Soll-Werte verwendet. Das Ziel des Tests ist zunächst Parameter zu erhalten, bei denen der Roboter über die gesamte Simulationsdauer nicht fällt.
Es zeigt sich, dass ein $m = 0$ zu Parametern führt, bei denen der Roboter sofort fällt, so dass die Simulation innerhalb weniger Zeitschritte beendet ist. Dadurch kann die Summe im Zähler der Fitnessfunktion nicht weiter wachsen, wodurch kürzere Simulationen zu kleineren Fitnesswerten führen. Ein $m = 1$ bedeutet, dass der Durchschnitt über die Zeit gebildet wird. Das führt zu einer längeren Simulation, die aber dennoch nicht die maximale Dauer von 12 Sekunden erreicht. Die Fitnessfunktion bewertet hier Parameter als gut, bei denen die Simulation solange dauert, wie der Roboter steht, da in dieser Phase der Durchschnitt klein bleibt. Würde der Roboter los laufen, stiege der Durchschnitt, daher ergeben sich Parameter, die einen Roboter sofort zu Fall bringen, sobald die Standphase beendet ist. Zur Erinnerung: Fallen führt nur zu einem kleineren T, nicht automatisch zu einer schlechteren Fitness. Zu erwarten wäre, dass mit besserer Fitness auch die

[1] http://www.irf.tu-dortmund.de

Simulationsdauer steigt, was aber nicht der Fall ist. Die lokalen Minima können nicht mehr verlassen werden. Es ist daher sinnvoll, zunächst Parameter anzustreben, die eine Simulationsdauer von 12 Sekunden ermöglichen, ohne zu fallen. Die Wahl von $m = 5$ führte zu den gewünschten Ergebnissen.

Simulationsparameter

Bisher nicht näher betrachtet wurde die Auswahl der zu optimierenden Parameter. Das sind zunächst alle, die nicht, z.B. durch Herstellerangaben, klar sind. Tabelle 4.3 fasst die 19 zu optimierenden Parameter zusammen. Die Werte können zwar nicht auf bestimmte Bereiche eingegrenzt werden, es ist aber klar, dass sie nur positiv Sinn machen. Für eine Beschleunigung der Studie werden daher Individuen mit negativen Werten vermieden.
Drei Parameter, die eigentlich ebenfalls in der Tabelle stehen sollten, müssen aber von Hand festgelegt werden, da die hier verwendeten Evolutionsstrategien nicht in der Lage sind, für sie sinnvolle Werte zu finden, wie sich in Tests mit dem vereinfachten Aufbau zeigt.
Dazu gehören die ODE-Parameter erp und erp_{soft}. Wie in Abschnitt 2.2.2 beschrieben, lassen sich mit diesen Parametern die Kräfte einstellen, mit denen gekoppelte Festkörper in ihrer relativen Position zueinander gehalten werden, wenn sie simulationsbedingt auseinander driften. Es zeigt sich, dass mit diesen Parametern schnell hohe Fitnesswerte erreichbar sind, indem sie besonders niedrig gewählt werden. Das führt unter anderem z.B. dazu, dass die Füße unrealistisch weit in den Boden einsinken. Dieses lokale Minimum kann aber nicht mehr verlassen werden, daher wird dieser Wert auf die Voreinstellungen der ODE festgelegt.
Ähnliches gilt für den Reibungskoeffizienten μ_r. Auch hier führen niedrige Werte schnell zu guten Individuen in einem lokalen Minimum, dass nicht mehr verlassen werden kann. Der Wert wird daher auf 1 festgelegt. Das entspricht ungefähr der Reibung von Aluminium zu Aluminium oder Gummi auf Asphalt.
Die Gründe dafür, warum niedrige Werte hier schnell zu guten Individuen führen, sind nicht klar. Man kann spekulieren, dass bei niedriger Reibung und geringem Widerstand durch den Boden die Gelenke besonders gut kontrollierbar sind, da sie kaum Drehmomentsspitzen zu bewältigen haben. Das allein führt leicht zu stabil laufenden Robotern.

Vergleich von ES-Optionen

In Abschnitt 2.3.4 werden zwei implementierte Strategien getestet. Dort zeigt sich, dass die CMA-ES der anderen Strategie überlegen ist, was aber nur für die zum Test verwendete Zielfunktion gilt. Es gibt Beispiele, für die die CMA-ES ungeeignet ist (HK04).
Um Aussagen bezüglich des hier vorliegenden Problems treffen zu können, sind weitere Tests der Strategien notwendig, jetzt mit dem vollständigen Aufbau und den zur Studie

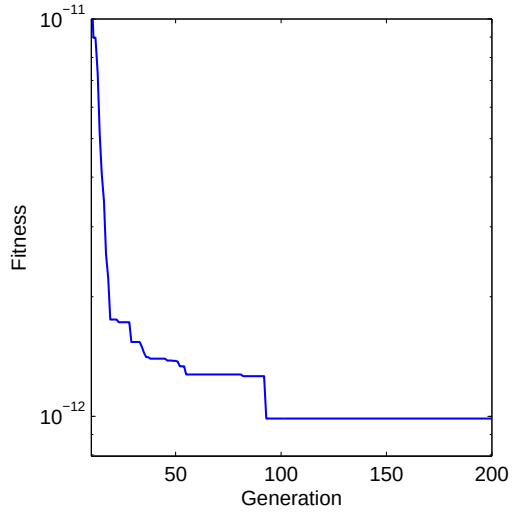
ausgewählten Parametern. Konkret muß untersucht werden, welche μ, λ und κ geeignet sind, und welche der beiden Evolutionsstrategien die geeignetere ist. Bei der Wahl der zu untersuchenden Optionen muß auch der Zeitrahmen der letztendlichen Parameterstudie beachtet werden, daher werden nur wenige Optionen betrachtet.

- Bei der CMA-ES ist $\mu = 1$, siehe Abschnitt 2.3.3. Bei der selbstadaptiven Plus-Strategie ist dieser Wert nicht klar und es ist zu untersuchen, ob eine kleine Population ($\mu = 10$) oder eine große Population ($\mu = 200$) sinnvoll ist.
- Die Anzahl der Nachkommen wird von den zur Verfügung stehenden CPU-Kernen bestimmt. Bei der selbstadaptiven Plus-Strategie wird ein λ von 800 gewählt, was ca. 4-5 Auswertungen pro Knoten bei jeder Generation entspricht. Bei der CMA-ES wird dieser Wert auf 180 festgesetzt, da eine hohe Anzahl von Kindern bei $\mu = 1$ keinen Vorteil mit sich bringt.
- Vor allem bei der selbstadaptiven Plus-Strategie kann eine begrenzte Lebensdauer einen Vorteil haben. Hier ist eine unbegrenzte Lebensdauer mit einem $\kappa = 3$ zu vergleichen.
- Es ist zwar nicht möglich die Parameter auf Wertebereiche einzuschränken, dennoch kann die Suche nach optimalen Parametern auf positive Werte beschränkt werden. Daher liegt es nahe, wie auch zur Mutation der Schrittweiten, die logarithmische Normalverteilung mit $\ln \widetilde{z}_i \sim \mathcal{N}\left(0, \sigma_i^2\right)$ zur Mutation zu verwenden: $X_i\left(t\right) = X_i\left(t-1\right) \cdot \widetilde{z}_i$. Es ist zu untersuchen, ob sie hier einen Vorteil gegenüber der üblicherweise verwendeten Normalverteilung, wie in Abschnitt 2.3.2 beschrieben, mit sich bringt.
- Außerdem ist zu klären, ob die CMA-ES der anderen Strategie grundsätzlich überlegen ist.

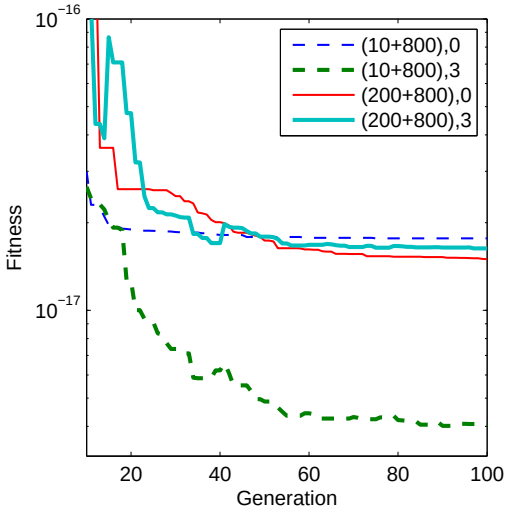
Da es sich um einen stochastischen Prozess handelt, werden für jede getestete Parameterkombination 3 Durchläufe durchgeführt und der Median zum Vergleich herangezogen.

4.4.2 Durchführung

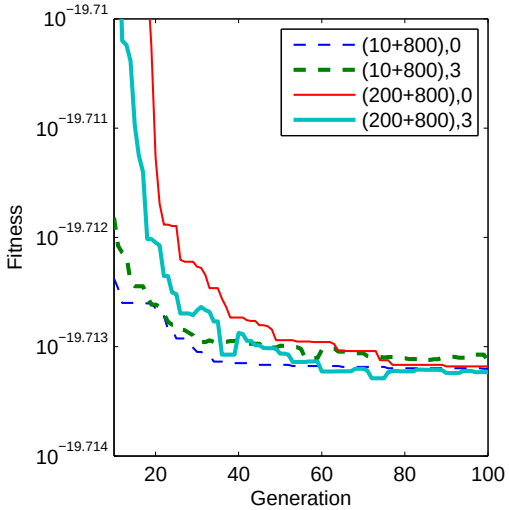
Abbildung 4.8 zeigt für die getesteten 9 verschiedenen Parameterkombinationen den Median der 3 Durchläufe. Je Durchlauf beinhalten die Tests jeweils 100 Generationen bei den ES mit (logarithmisch) normalverteilter Mutation und 444 beim CMA, um die gleiche Anzahl von Auswertungen zu erreichen.
In Abbildung 4.8(a) wird der Fitnessverlauf des Median beim Test des CMA-ES für Generation 10 bis 100 auf einer logarithmischen Skala gezeigt. Nach Generation 100 sind nahezu keine Fortschritte mehr zu erkennen. Der beste erreichte Wert liegt hier bei $4.9419 \cdot$



(a) CMA.



(b) Mutation per Normalverteilung.



(c) Mutation per logarithmischer Normalverteilung.

Abbildung 4.8: Vergleich verschiedener ES und verschiedener Strategieparameter.

10^{-13}, wobei nur rund 2 Sekunden simuliert werden, bevor die Simulation instabil wird. Auch eine veränderte initiale Schrittweite liefert keine besseren Ergebnisse und eine größere Anzahl von Generationen bringt keinen Vorteil, da sich die Fitness bei keinem Durchlauf nach Generationen 183 mehr verändert hat. Die CMA-ES ist demnach ungeeignet, da sie zu früh konvergiert.

Die in Abbildung 4.8(b) gezeigten Ergebnisse der ES mit normalverteilter Mutation sind deutlich besser, vor allem die Tests mit $(10 + 800)$-ES und $\kappa = 3$. Der beste Durchlauf erreicht eine Fitness von $3.2560 \cdot 10^{-20}$ bei einer Simulation ohne Sturz. Bei allen anderen Durchläufen ist die Fitness nicht so hoch, so dass die Simulation vorzeitig beendet wird

weil ein Roboter stürzt. Eine Evolution mit 500 Generation bei der $(10+800)$-ES mit $\kappa = 3$ zeigt, dass keine besseren Werte nach Generation 100 erreicht werden, was auch für andere Startschrittweiten gilt.
Da die Ergebnisse der meisten Durchläufe nicht verwendbar sind, wird statt der Normalverteilung die logarithmische Normalverteilung eingesetzt und getestet, siehe Abbildung 4.8(c). Mit ihr sind Fitnesswerte von bis zu $1.9341 \cdot 10^{-20}$ erreichbar. Hier werden auch die vollen 12 Sekunden simuliert, ohne dass ein Roboter fällt. Ansonsten sind zwischen den einzelnen Strategieparametern keine nennenswerten oder signifikanten Unterschiede zu erkennen, wobei hier die $(200+800)$-ES mit $\kappa = 3$ am besten abgeschnitten hat. Ein Durchlauf mit maximal 500 Generationen führt bei diesen Einstellungen nicht zu weiteren Verbesserungen, so dass die Parameterstudie damit auch beendet ist. Inwiefern die gefundenen Parameter die gewünschten Eigenschaften zeigen, wird im nächsten Kapitel geklärt.

5

Analyse und Kompensation

In diesem Kapitel werden die Dynamikabstrakion und die Kinematikfehlerabstraktion auf ihre Auswirkungen hin untersucht, und gegebenenfalls Kompensationen an den geplanten Stellen (siehe Abschnitt 2.4) vorgenommen.
In SimRobot ist die *ZMP-Abweichung* beobachtbar, und damit die Beobachtung auf der abstraktesten Ebene. Dieses Kapitel beginnt daher mit ihrer Untersuchung (vgl. Abschnitt 5.1). Die Beobachtungen, die erst auf komplexeren Ebenen gemacht werden können, werden von der hier gefundenen Kompensation ebenfalls profitieren, da sich die Dynamikabstraktion auch auf komplexere Ebenen auswirkt. Es wird sich zeigen, dass Beobachtung *Einknicken* bei der Beobachtung *Seitwärtsschwingung* eine wesentliche Rolle spielt. Abschnitt 5.2 wird sie daher zusammen behandeln. Aus ähnlichen Gründen wird der letzte Abschnitt 5.3 die Beobachtungen *Vorwärtsschwingung* und *Geschwindigkeitssteigerung* analysieren und kompensieren.

Abkürzungen

In diesem Kapitel werden zur Übersichtlichkeit für die Simulationselemente Abkürzungen benutzt. Mit MKM wird eine Rechnung auf der Mehrkörpermodellebene bezeichnet. BS steht für eine Basis-Simulation, die der Komplexität der SimRobot-Ebene entspricht, somit auch die Gelenke per Vorgabe der Winkelgeschwindigkeit ansteuert. MT steht für die Motorsimulation, und beinhaltet unter anderem die Simulation der Induktivität des Motors, die mit IN abgekürzt wird. Zur Motorsimulation gehören auch die PID-Regler, die den nötigen Strom regeln. Die MT ersetzt damit die Ansteuerung per Vorgabe der Winkelgeschwindigkeit. Ist die MT aktiviert, ist die Simulation der flexiblen Getriebe

möglich, die mit GT abgekürzt wird. FK steht für die flexiblen Körper der Beine. Sind sie deaktiviert, werden die flexiblen Körper durch Festkörper ersetzt.
Ist eine vollständige Simulation (Simulation auf der Walking Simulator-Ebene) gemeint, wird das als BS+MT+GT+FK oder VS bezeichnet, wird z.B. die Induktivität deaktiviert als VS-IN.

5.1 *ZMP-Abweichung*

In diesem Abschnitt wird die Beobachtung *ZMP-Abweichung* untersucht und kompensiert. Bei dieser Beobachtung handelt es sich um einen gemessenen ZMP-Verlauf wie in Abbildung 1.1 dargestellt, bei dem der ZMP in den Single-Support-Phasen betraglich dauerhaft kleiner ist als der Soll-ZMP. Zur Erinnerung: Er wird auf der gleichen Abstraktionsebene gemessen und berechnet, wie auch die Dortmund Walking Engine arbeitet (siehe Abschnitt 2.4), nämlich anhand eines Ein-Schwerpunkt-Modells. Es wäre zu erwarten, dass entlang der y-Achse betraglich größere Werte gemessen werden als gewünscht, da zur Messung die Beschleunigung des Oberkörpers statt des Schwerpunktes verwendet wird. Zwei Hypothesen sind zu untersuchen:
Das Schwungbein wird entlang der z-Achse mehrfach während der Single-Support-Phase beschleunigt und wieder abgebremst. Das könnte den ZMP soweit beeinflussen, dass der Oberkörper in Richtung des Schwungbeins kippt, wie es auch ähnlich bei Beobachtung *Einknicken* beschrieben wird. Durch das Kippen würde der gemessene ZMP dem aus Abbildung 1.1 entsprechen, und die Beobachtung wäre eine Folge der Dynamikabstraktion. Außerdem läge dann ein Zusammenhang mit der Beobachtung *Einknicken* nahe.
Ebenso können Kinematikfehler, die von SimRobot simuliert werden, dafür verantwortlich sein. Die Drehmomente sind bei dem hier gezeigten Lauf eher gering, da der Roboter mit nur 5 $\frac{cm}{s}$ läuft. In der Realität sind vielfache Geschwindigkeiten möglich, und auch die Stiffness kann in der Realität deutlich geringer gewählt werden. Daher kann die Begrenzung der Drehmomente als Ursache nicht in Frage kommen. Möglich ist, dass mangelnde Bodenhaftung zur Beobachtung führt.
Zunächst wird daher in Abschnitt 5.1.1 anhand des Mehrkörpermodells der ZMP berechnet. Damit lässt sich klären, ob die Dynamikabstraktion für die Beobachtung verantwortlich ist.
In Abschnitt 5.1.2 wird der ZMP-Verlauf im Walking Simulator aufgezeichnet, indem die Oberkörperposition zweimal differenziert in die Gleichungen 2.7 und 2.8 eingesetzt wird. Der Simulator ist in der Lage, die Oberkörperposition mit einer Frequenz von $1kHz$ aufzuzeichnen, woraus sich ein ZMP-Verlauf höherer Auflösung ergibt. Der Reibungskoeffizient ist mit $\mu_r = 1$ in beiden Simulatoren gleich, daher sollte der ZMP im Walking Simulator auch die gleiche Tendenz zeigen, wenn mangelnde Bodenhaftung die Ursache ist.

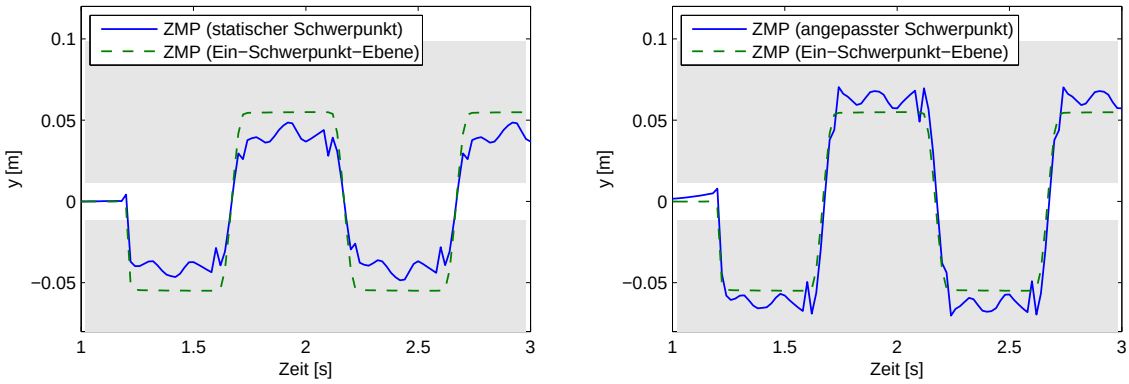


Abbildung 5.1: Verlauf des ZMP mit (Abb. rechts) und ohne (Abb. links) Gesamtschwerpunktanpassung, jeweils berechnet auf der Ein-Schwerpunkt-Ebene (gestrichelte Linie) bzw. der Mehrkörpermodellebene (durchgezogene Linie). Die grauen Flächen zeigen den Bereich am Boden, über dem sich die Füße befinden. Die graue Fläche stellt demnach das Support-Polygon dar, wenn der Fuß auf der entsprechenden Seite Kontakt mit dem Boden hat. In der Double-Support-Phase gehört die Fläche zwischen den Füßen ebenfalls zum Support-Polygon.

In Abschnitt 5.1.3 werden die gefundenen Ungenauigkeiten kompensiert und das Kapitel mit einem Fazit (vgl. Abschnitt 5.1.4) abgeschlossen.

5.1.1 Mehrkörpermodell-Untersuchung

In diesem Abschnitt wird die Hypothese überprüft, ob die Dynamikabstraktion zur Beobachtung *ZMP-Abweichung* führt. Dazu wird ein Lauf mit der Dortmund Walking Engine bei $5\frac{cm}{s}$ erstellt, und mit Hilfe des RNEA (Algorithmus 3.1) der Ist-ZMP auf der MKM-Ebene berechnet.

Zur Erinnerung: Die Dortmund Walking Engine arbeitet auf der Ein-Schwerpunkt-Ebene, passt aber den Gesamtschwerpunkt der aktuellen kinematischen Konfiguration an, was einer einfachen Kompensation entspricht. Diese Anpassung wird zunächst deaktiviert. Abbildung 5.1 zeigt links den ZMP-Verlauf entlang der y-Achse, berechnet mit dem Ein-Schwerpunkt-Modell bzw. mit dem Mehrkörpermodell. Die konstanten Abschnitte des Ein-Schwerpunkt-ZMPs bei $y = \pm 0.055m$ sind die Single-Support-Phasen. Der Mehrkörpermodell-ZMP ist in diesen Phasen betraglich zu klein (umgangssprachlich liegt er zu weit innen). Außerdem sind in den Single-Support-Phasen Schwingungen des ZMPs zu erkennen, für die das Schwungbein die wahrscheinlichste Ursache ist. Das entspricht aber nicht dem Lauf, der zur Beobachtung *ZMP-Abweichung* führt. Abbildung 5.1 zeigt rechts den gleichen Lauf mit Gesamtschwerpunktanpassung. Auch hier sind Schwingungen im ZMP-Verlauf zu erkennen, die wohl vom Schwungbein verursacht werden. Die Anpassung des Gesamtschwerpunktes ist aber offensichtlich keine

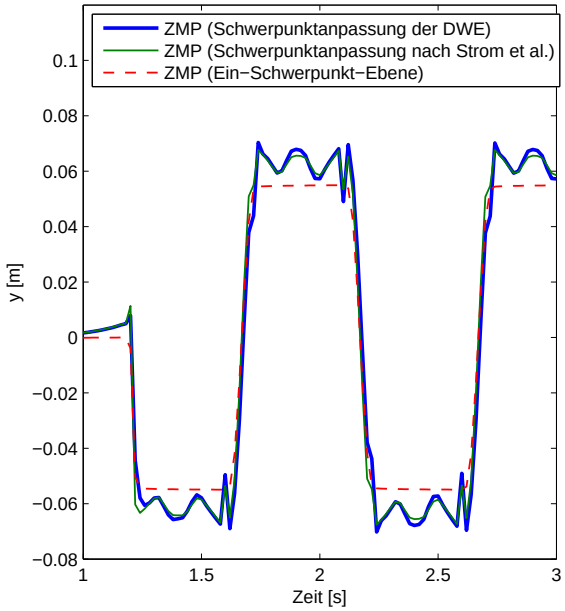


Abbildung 5.2: Verlauf des ZMP mit Gesamtschwerpunktanpassung der Dortmund Walking Engine bzw. nach Strom et al.

gute Kompensation, denn der Mehrkörpermodell-ZMP ist nun betraglich zu groß in den Single-Support-Phasen. Das bedeutet, dass nicht das Schwungbein zum Kippen des Roboters in seine Richtung führt, es verursacht nur eine relativ kleine Schwingung im ZMP-Verlauf.
Die Beobachtung *ZMP-Abweichung* kann daher nicht mit der Dynamikabstraktion erklärt werden.
Ein Vergleich mit der Gesamtschwerpunktanpassung von Strom et al. ist nun sinnvoll. Dazu wird die DWE entsprechend auf die iterative Berechnung, wie in Abschnitt 2.4.3 beschrieben, umgestellt. Der daraus resultierende Mehrkörpermodell-ZMP ist in Abbildung 5.2 zu sehen. Er unterscheidet sich kaum von dem ZMP, der sich durch das Anpassungsverfahren der Dortmund Walking Engine ergibt.

5.1.2 Einfache Simulation

In diesem Abschnitt wird die zweite Hypothese untersucht, ob eine zu geringe Reibung zur Beobachtung *ZMP-Abweichung* führt. Bei zu geringer Reibung würden die Füße des Roboters rutschen und könnten nicht die notwendige Beschleunigung erzeugen, die laut dem 3D-LIP-Modell für die Soll-ZMP-Position notwendig ist. Würde der ZMP das Support-Polygon dadurch verlassen und der Roboter kippen, wäre ein ZMP wie in Abbildung 1.1 messbar.
Um die Hypothese zu überprüfen, wird die **BS** ohne Erweiterungen benötigt. Es werden die Winkel des letzten Abschnittes als Eingabe verwendet, und es wird der ZMP nach

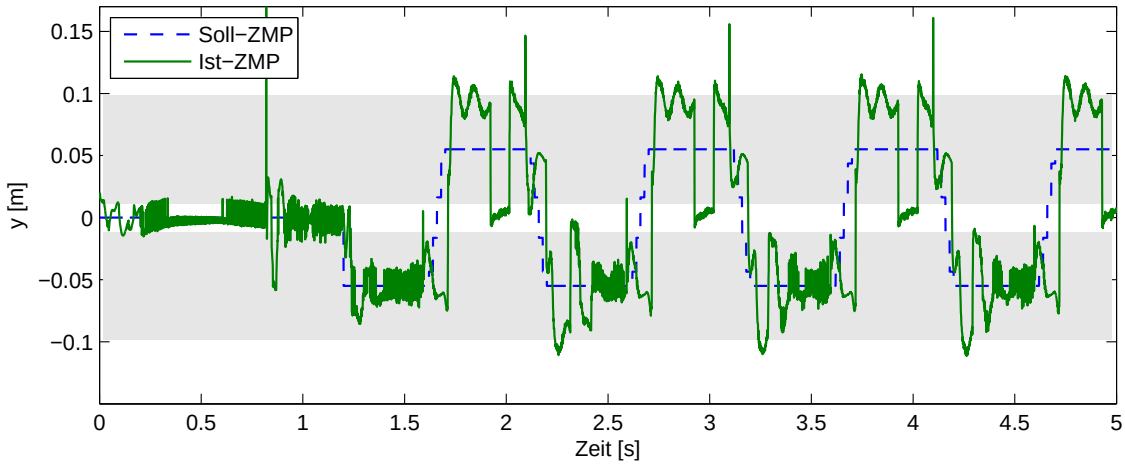


Abbildung 5.3: ZMP gemessen per Ein-Schwerpunkt-Modell aus der Beschleunigung des Oberkörpers. Die grauen Flächen zeigen den Bereich am Boden, über dem sich die Füße befinden.

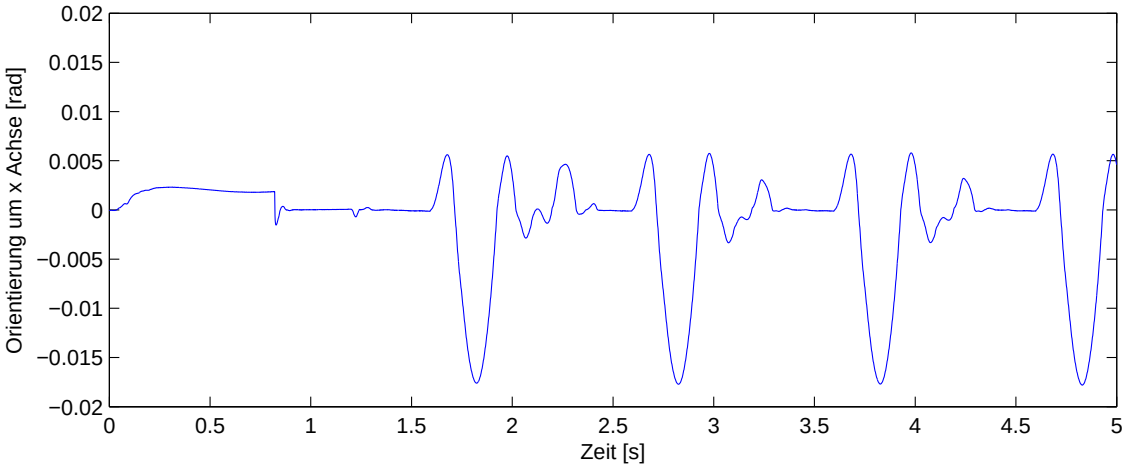


Abbildung 5.4: Orientierung des Oberkörpers während des Laufs.

Variable / Konstante	Bedeutung
$\boldsymbol{q}$	Gelenkwinkel zu allen Zeitpunkten.
p_{zmp}^{WKS}	ZMP-Verlauf nach Mehrkörpermodell.
$p_{zmp,S}^{WKS}$	Soll-ZMP-Verlauf.
$p_{zmp,S}^{WKS\ '}$	Kompensierter Soll-ZMP-Verlauf.
n	Anzahl der Kompensationsiterationen.

Tabelle 5.1: Variablen und Konstanten für *komp*.

dem Ein-Schwerpunkt-Modell mit einer Frequenz von $1kHz$ gemessen. Sollte mangelnde Reibung für die Beobachtung *ZMP-Abweichung* verantwortlich sein, muß der gemessene ZMP, wie in SimRobot, betraglich zu klein sein.

Zunächst eine Erläuterung zur Messung der Oberkörperorientierung. Als Orientierung des Oberkörpers $\varphi_x(t)$ um die x-Achse zum Zeitpunkt t wird der Winkel zwischen dem Einheitsvektor $(0,1,0)$ im Roboterkoordinatensystem und der Ebene mit dem Normalenvektor $(0,0,1)$ bezeichnet. Entsprechend für die y-Achse mit dem Einheitsvektor $(1,0,0)$ ($\varphi_y(t)$).

Abbildung 5.4 zeigt die Orientierung des Oberkörpers um die x-Achse während des Laufs und Abbildung 5.3 den ZMP entlang der y-Achse, jeweils über die Zeit. Zu sehen ist, dass der Oberkörper vor allem nach dem ersten Schritt bei ca. $1.7s$ schwankt, obwohl der Mehrkörpermodell-ZMP nach Abbildung 5.1 innerhalb des Support-Polygon liegt. Der gemessene ZMP spiegelt hauptsächlich diese Schwankung wieder, was bei dem zur Messung genutztem Modell auch zu erwarten ist. Die Ursache der Oberkörperschwankung ist, dass Festkörper in andere, je nach Einstellungen mehr oder weniger stark, eindringen können, da die Kontakt-Gelenke nicht vollständig fest sind (siehe Abschnitt 2.2). Auch wenn das logisch erscheinen mag, da auch zum Beispiel Teppiche nachgeben, ist die ODE eine Festkörpersimulation, so dass diese Schwankungen nicht auftreten dürften.

Außerdem zeigt der ZMP hochfrequente Störungen, vor allem in den ersten $1.5s$, die nachlassen, sobald der Roboter zu schwanken beginnt. Beim Schwanken liegt der Standfuß nicht flach auf dem Boden und daher werden auch weniger Kontakt-Gelenke erzeugt. Daraus lässt sich schließen, dass Kontakt-Gelenke Störungen in den Festkörperpositionen verursachen, die sich besonders nach zweimaligem Differenzieren zeigen.

Zusammengefasst ist ein betraglich dauerhaft zu kleiner ZMP wie in SimRobot hier nicht zu erkennen, obwohl die Reibung gleich groß ist. Daher kann die Reibung nicht dafür verantwortlich sein.

5.1.3 MKM/ZMP-Kompensation

Algorithmus 5.1 MKM/ZMP-Kompensationsalgorithmus (*komp*).

Eingabe: $p_{zmp,S}^{WKS}, n$
Ausgabe: $p_{zmp,S}^{WKS\,\prime}$

1: $p_{zmp,S}^{WKS\,\prime} = p_{zmp,S}^{WKS}, p_{zmp,offset}^{WKS} = \mathbf{0}$
2: **for** $i = 1$ **to** n **do**
3: $\quad \boldsymbol{q} = \text{walk}\left(p_{zmp,S}^{WKS\,\prime}\right)$
4: $\quad p_{zmp}^{WKS} = getTorqueZMP\,(\boldsymbol{q}, \ldots)$
5: $\quad p_{zmp,offset}^{WKS} = p_{zmp,offset}^{WKS} + p_{zmp,S}^{WKS} - p_{zmp}^{WKS}$
6: $\quad p_{zmp,S}^{WKS\,\prime} = p_{zmp,S}^{WKS} + p_{zmp,offset}^{WKS}$
7: **end for**

Wenn auch die Dynamikabstraktion nicht zu der Beobachtung führt, ist sie dennoch für einen fehlerhaften ZMP-Verlauf verantwortlich (siehe Abbildung 5.1), was auch die Schwankung in der Simulation verstärken kann.

Eine Kompensation ist daher sinnvoll, und wird in diesem Abschnitt vorgestellt, siehe Algorithmus 5.1 und Tabelle 5.1 für die verwendeten Variablen. Eingabe des Algorithmus ist der Soll-ZMP-Verlauf $p_{zmp,S}^{WKS}$. Das Verfahren arbeitet iterativ über n Iterationen. In jeder Iteration wird zuerst die DWE ausgeführt. Aus den erzeugten Winkelverläufen $\boldsymbol{q}$ wird mit Algorithmus 3.3 der Ist-ZMP-Verlauf p_{zmp}^{WKS} auf der MKM-Ebene berechnet. Die Differenz $p_{zmp,S}^{WKS} - p_{zmp}^{WKS}$ stellt den Fehler dar, der durch die Dynamikabstraktion entsteht. Sie wird auf einen Offset $p_{zmp,offset}^{WKS}$ addiert, der seinerseits auf den Soll-ZMP $p_{zmp,S}^{WKS\,\prime}$ addiert wird. Mit diesem veränderten Soll-ZMP-Verlauf wird die nächste Iteration ausgeführt, die eine kleinere Differenz aufweist. Das Verfahren konvergiert gegen einen Soll-ZMP, bei dem die DWE einen Lauf erzeugt, dessen Ist-ZMP auf der MKM-Ebene gleich $p_{zmp,S}^{WKS}$ ist. Abbildung 5.5 zeigt den Soll-ZMP und den Mehrkörpermodell-ZMP nach Iterationsschritt 1 bis 4. Nach dem 4. Schritt ist nahezu kein Unterschied zwischen Soll und Ist mehr zu erkennen, so dass im Folgenden für alle MKM/ZMP-Kompensationen $n = 4$ gewählt wird.

Ein ähnliches Verfahren wurde bereits von Kajita et al. zur Kompensation entwickelt (KKK$^+$03b). Dort wird allerdings nicht vor dem Lauf kompensiert, sondern während des Laufs die Differenz berechnet und im nächsten Schritt angewendet. Dieses Verfahren konvergiert ebenfalls und ist online berechenbar, aber nur bei einer Folge von gleichen Schritten.

Die Schwankung des Roboters durch das Eindringen der Füße in den Boden (siehe Abbildung 5.4) sollte in Festkörpersimulationen zwar nicht vorkommen, kann aber als Lauf auf einem weichen Teppich interpretiert werden. Daher ist es interessant zu testen, ob die MKM/ZMP-Kompensation zu einem stabileren Lauf führt.

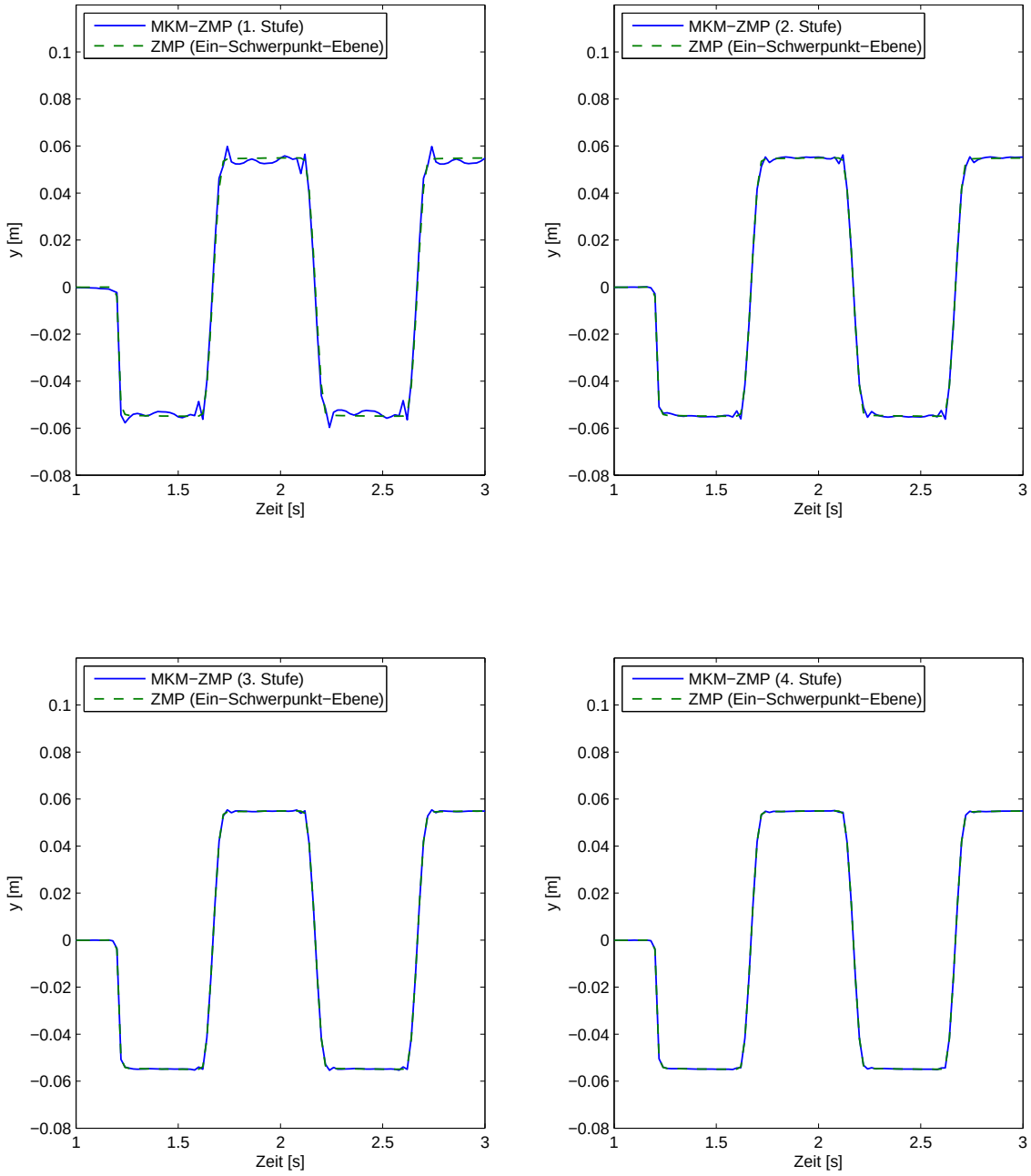


Abbildung 5.5: Fortschritt der MKM/ZMP-Kompensation über 4 Iterationsschritte des Algorithmus 5.1.

Um den Erfolg bewerten zu können, muß eine Metrik für Stabilität festgelegt werden. Üblicherweise wird dafür der ZMP verwendet. Diesen zu bestimmen ist aber nicht immer zuverlässig möglich. Eine andere Möglichkeit ist die Orientierung des Oberkörpers um die x- und y-Achse. Nach menschlicher Ansicht wirkt ein Roboter stabil, wenn er nicht schwankt, die Oberkörperorientierung also, wie vorgegeben, konstant ist. Als Größe zur Messung der Stabilität werden daher die folgenden Summen verwendet und im Weiteren als **Orientierungsfehler** bezeichnet:

$$f_x = \sum_{t=1}^{T} \varphi_x(t)^2 \tag{5.1}$$

$$f_y = \sum_{t=1}^{T} \varphi_y(t)^2 \tag{5.2}$$

Diese Art der Bewertung eines Laufs hat im Vergleich zum ZMP den Vorteil der Anschaulichkeit und besseren Messbarkeit im Walking Simulator, aber auch Einschränkungen. Während zum Beispiel sich mit dem ZMP noch auf der MKM-Ebene Aussagen zur Güte des Laufs machen lassen, ist das mit dieser Metrik nicht möglich.

Die Orientierungsfehler für den Lauf aus Abschnitt 5.1.2 betragen $f_x = 0.8477$ und $f_y = 0.000172$. Die MKM/ZMP-Kompensation reduziert die Fehler auf $f_x = 0.2399$ und $f_y = 0.0000725$.

5.1.4 Fazit

Zusammenfassend ist festzustellen, dass Beobachtung *ZMP-Abweichung* nicht eine Folge der Dynamikabstraktion ist, da der MKM-ZMP hier sogar betraglich zu groß ist. Sie ist auch keine Folge mangelnder Reibung, da die Reibung bei SimRobot und Walking Simulator gleich groß gewählt ist, sich aber dauerhaft betraglich zu kleine ZMP-Verläufe im Walking Simulator nicht zeigen. Ein Zusammenhang mit Beobachtung *Einknicken* lässt sich also nicht herstellen.

Auch wenn die Dynamikabstraktion die Beobachtung aus SimRobot nicht erklärt, ist sie dennoch dafür verantwortlich, dass die DWE einen Lauf erzeugt, der nach Mehrkörpermodell einen vom Soll abweichenden ZMP-Verlauf aufweist. Eine Kompensation ist daher mit dem gezeigten iterativen Algorithmus sinnvoll und verringert deutlich die Schwankung des Laufs im Walking Simulator. Da auch alle Beobachtungen auf komplexeren Ebenen als die Mehrkörpermodellebene von den Fehlern der Dynamikabstraktion beeinflusst werden können, ist daher auch in den folgenden Kapiteln die MKM/ZMP-Kompensation von Wichtigkeit.

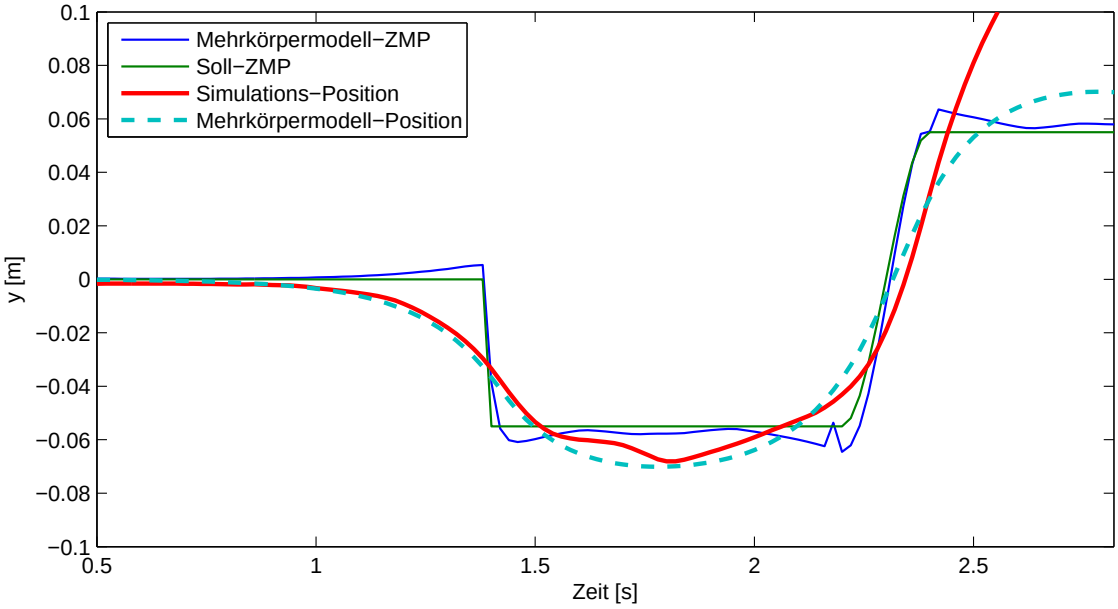


Abbildung 5.6: ZMP und Position des Roboters entlang der y-Achse über die Zeit.

5.2 *Seitwärtsschwingung* und *Einknicken*

Wird eine Schrittdauer zwischen 1.5*s* und 2.3*s* gewählt, ist beim realen Roboter ein sich verstärkendes Schaukeln entlang der y-Achse zu beobachten. Das führt, je nach Schrittdauer, nach einem bis mehreren Schritten zum Sturz. Diese Beobachtung soll in diesem Abschnitt zunächst auf die Ursachen hin untersucht werden (vgl. Abschnitt 5.2.1). Die zu prüfende Hypothese ist, dass der Roboter in der Single-Support-Phase nicht genug Drehmoment aufbringt, um die seitliche Bewegung des Schwerpunktes rechtzeitig abzubremsen, woraus eine stärker werdende Schwingung entsteht.
Das *Einknicken* ist ebenfalls eine Beobachtung, die sich entlang der y-Achse zeigt, und ist daher auch Gegenstand dieses Abschnittes. Nachdem im letzten Abschnitt die Hypothese widerlegt werden konnte, dass sie eine Folge der Dynamikabstraktion ist, wird hier nun eine weitere Hypothese untersucht: Den größten Anteil am Gesamtgewicht des Roboters hat der Oberkörper. Mögliche Gründe für das Einknicken sind demnach die Motoren, die zu geringe Drehmomente ausüben oder die Flexibilitäten und Toleranzen.
Anschließend werden die Ursachen kompensiert, so dass ein stabiler Lauf möglich ist (vgl. Abschnitt 5.2.2) und der Abschnitt mit einem Fazit (vgl. Abschnitt 5.2.3) abgeschlossen.

5.2.1 Analyse im Walking Simulator

In diesem Abschnitt werden zur Erläuterung der Vorgänge beim Laufen die Positionen des Roboters verwendet. Abbildung 5.6 zeigt die Position des Oberkörpers im Walking

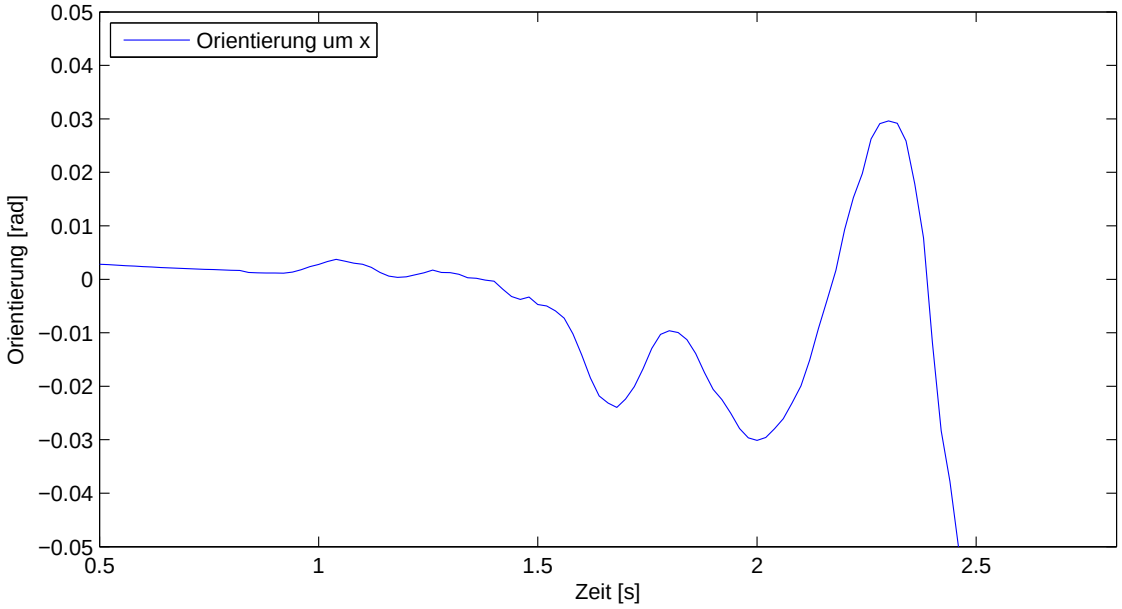


Abbildung 5.7: Orientierung des Oberkörpers um die x-Achse.

Simulator im Vergleich zur Soll-Position (die Ist-Position auf der Mehrkörpermodellebene stimmt genau mit der Soll-Position überein). Die Abbildung endet bei ca. 2.8*s*, da der Roboter in der Simulation bei einer Geschwindigkeit von $5\frac{cm}{s}$ und einer Schrittdauer von 2*s* bereits nach einem Schritt fällt.

Es ist zu erkennen, dass die Position in der Simulation von der Soll-Position abweicht, sobald die Bewegung beginnt. Kurz vor Sekunde 1.5 beginnt die erste Single-Support-Phase, bei der der Roboter vom rechten Bein gestützt wird. Hier steigt die Abweichung der simulierten Position im Vergleich zum Soll. Abbildung 5.7 zeigt die dazugehörige Orientierung des Oberkörpers um die x-Achse. Zu sehen ist, dass der Fehler in der Position des Oberkörpers aus seiner Orientierung resultiert, was auf Beobachtung *Einknicken* hindeutet. Kurz bevor die Double-Support-Phase beginnt schneidet die simulierte Position das Soll und weicht bis ca. 2.4*s* zur anderen Seite ab. Kurz darauf fällt der Roboter in der zweiten Single-Support-Phase um.

In der Abbildung 5.6 ist außerdem der Soll-ZMP sowie der ZMP der Mehrkörpermodellebene zu sehen. Sie zeigen eine zum letzten Abschnitt 5.1 ähnliche Differenz. Dort wurde die MKM/ZMP-Kompensation zur Minimierung der Auswirkungen der Dynamikabstraktion vorgestellt. Sinnvollerweise sollte diese auch hier zunächst angewendet werden, bevor nach weiteren Ursachen gesucht wird. Abbildung 5.8 zeigt den Nutzen. Vor allem um Sekunde 2 folgt die Position des Oberkörpers besser dem Soll, zeigt aber danach ähnliche Abweichungen und der Roboter stürzt auch zu einem ähnlichen Zeitpunkt. Die weitere Ursachensuche findet mit dem kompensierten Lauf statt, um Einflüsse der Dynamikabstraktion ausschließen zu können.

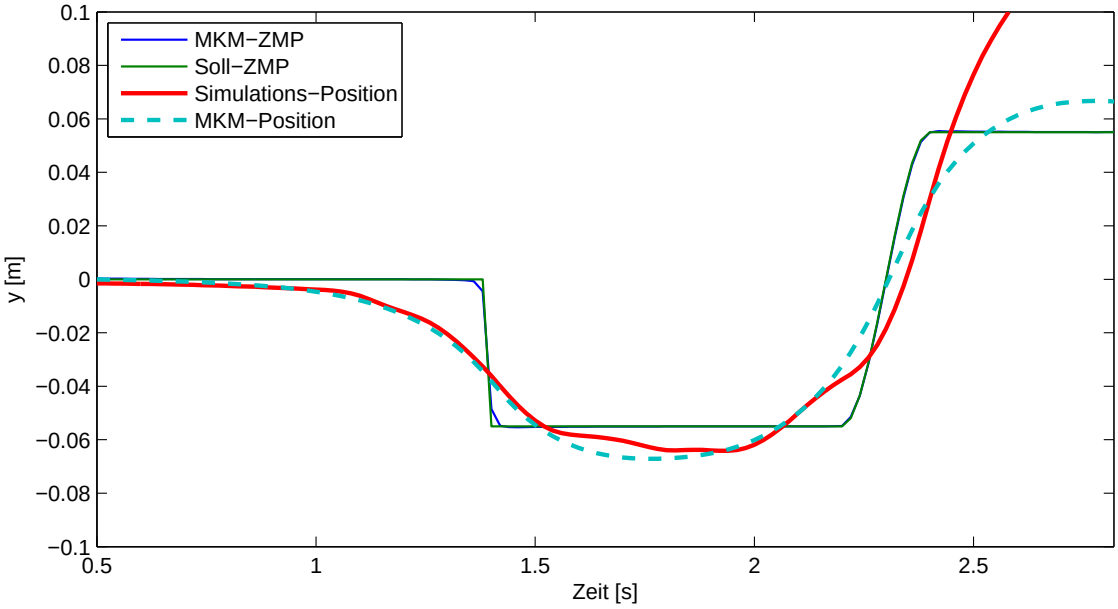


Abbildung 5.8: ZMP und Position des Roboters entlang der y-Achse über die Zeit beim MKM/ZMP-kompensierten Lauf.

Abbildung 5.9 zeigt Positionsverläufe des Oberkörpers bei denen einzelne Ungenauigkeiten deaktiviert wurden. Der abstrakteste Positionsverlauf stammt vom MKM und entspricht dem Soll. Zum Vergleich ist der simulierte Oberkörperverlauf aus Abbildung 5.8 ebenfalls eingetragen. Bei BS+MT+FK[1] erhält man einen ähnlichen Verlauf, der nur zu der Zeit des Einknickens einen sichtbaren Unterschied zeigt, allerdings keine Verringerung des Positionsfehlers. Verwendet man nur BS+MT knickt der Oberkörper im Vergleich zur vollständigen Simulation weniger ein, zeigt aber danach ebenfalls keinen Unterschied. Erst bei der BS entspricht die Oberkörperposition nahezu der Vorgabe. Die konstante Differenz entsteht in der Startphase während der Roboter in der Simulation die Startposition einnimmt.

Auch wenn der Einfluss der FK gering zu sein scheint, zeigt ein Test bei BS+FK, dass flexible Körper alleine auch zu einem instabilen Lauf führen. Zwar fällt der Roboter nicht wie bei einer vollständigen Simulation nach einem Schritt. Der Einfluss von flexiblen Körpern ist aber groß genug, um den Roboter bei BS+FK nach mehreren Schritten stürzen zu lassen.

Weiteren Aufschluss zu den Ursachen der Instabilität gibt Abbildung 5.10. Sie zeigt die Soll- und Ist-Position sowie Soll- und Ist-Geschwindigkeit des Oberkörpers in der Simulation. Außerdem wurden die Gelenkwinkel aus der Simulation ausgelesen und anhand einer Vorwärtskinematik ebenfalls zu einer Position und Geschwindigkeit umgerechnet.

[1] Zur Erinnerung: BS ist die Basissimulation, MT steht für Motorsimulation und FK für flexible Körper. Siehe auch Seite 65.

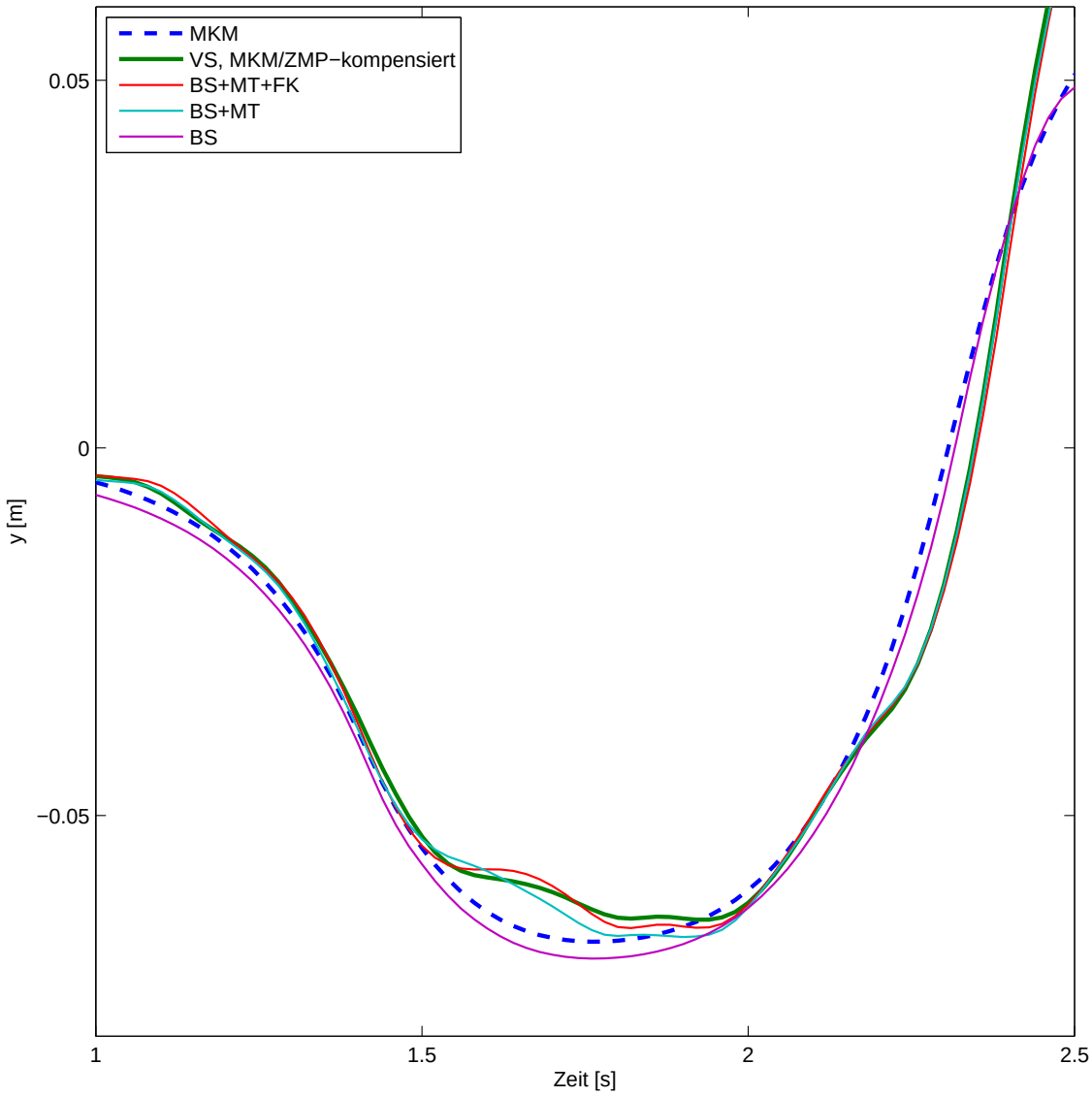


Abbildung 5.9: Positionen des Oberkörpers verschiedener Einflüsse im Vergleich.

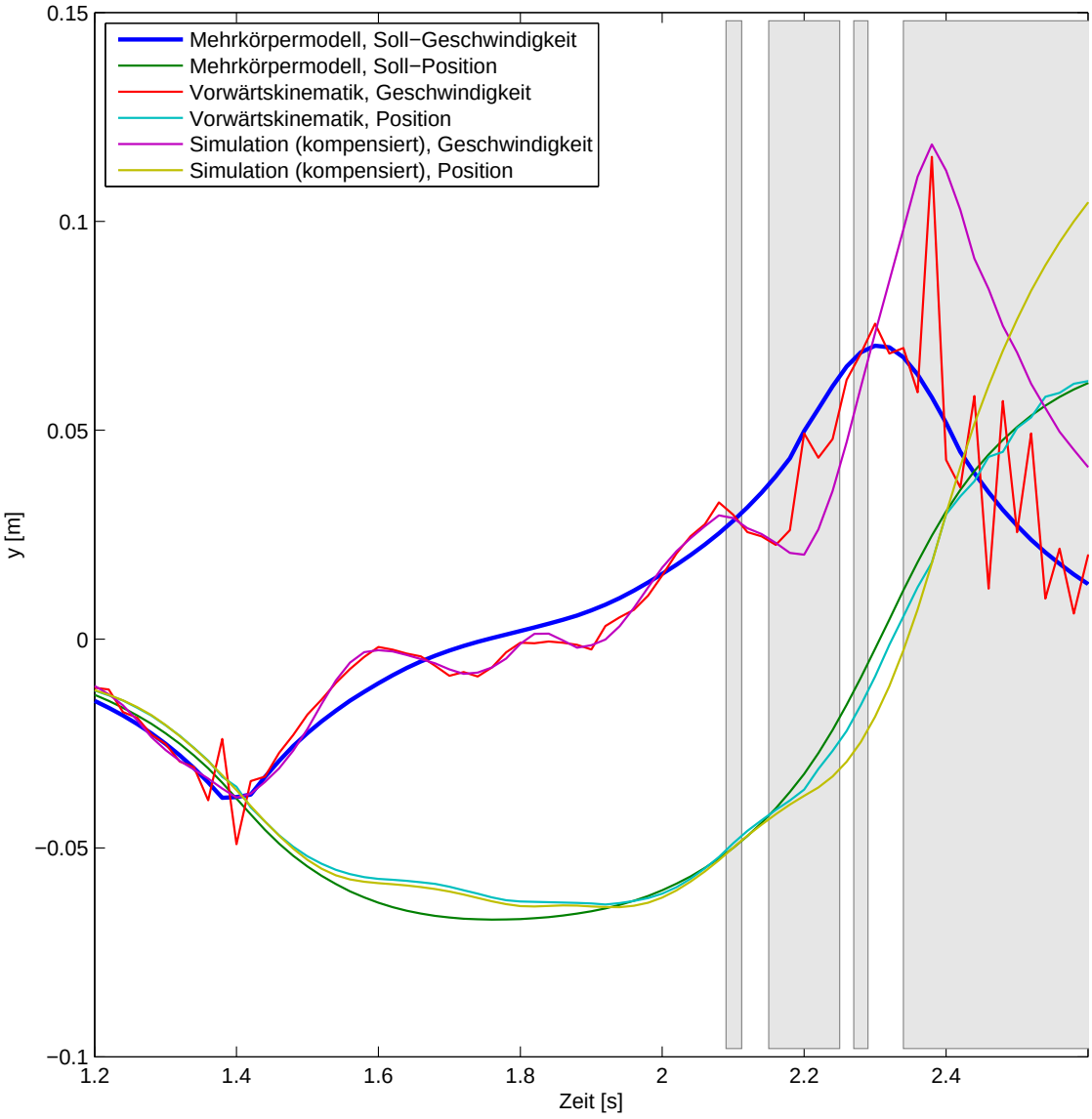


Abbildung 5.10: Position und Geschwindigkeit des Oberkörpers. Zeiten, zu denen der linke Fuß Kontakt zum Boden hat, sind grau hinterlegt.

Dazu zeigt Abbildung 5.11 die Orientierung des Oberkörpers. Die Zeitabschnitte, zu denen der linke Fuß Kontakt zum Boden hat, sind in beiden Grafiken grau eingezeichnet. Aus der Tatsache, dass bei 1.4s bis 2.1s kein Unterschied zwischen den Positionen der Vorwärtskinematik und den ausgelesenen Positionen zu erkennen ist, lässt sich schließen, dass Rutschen und Kippen ausgeschlossen werden können. Beobachtung *Einknicken* ist damit hauptsächlich eine Folge der Simulation der Motoren und ihrer Ansteuerung. In Abbildung 5.11 ist außerdem zu erkennen, dass kurz vor Beginn der Double-Support-Phase bei Sekunde 2.2 der Oberkörper erneut stärker einknickt. Dadurch hat der linke Fuß zu früh Kontakt mit dem Boden, den er kurzzeitig wieder verliert. Letzteres ist eine Folge des leicht federnden Bodens, so dass der Fuß nach einer zunächst nur leichten

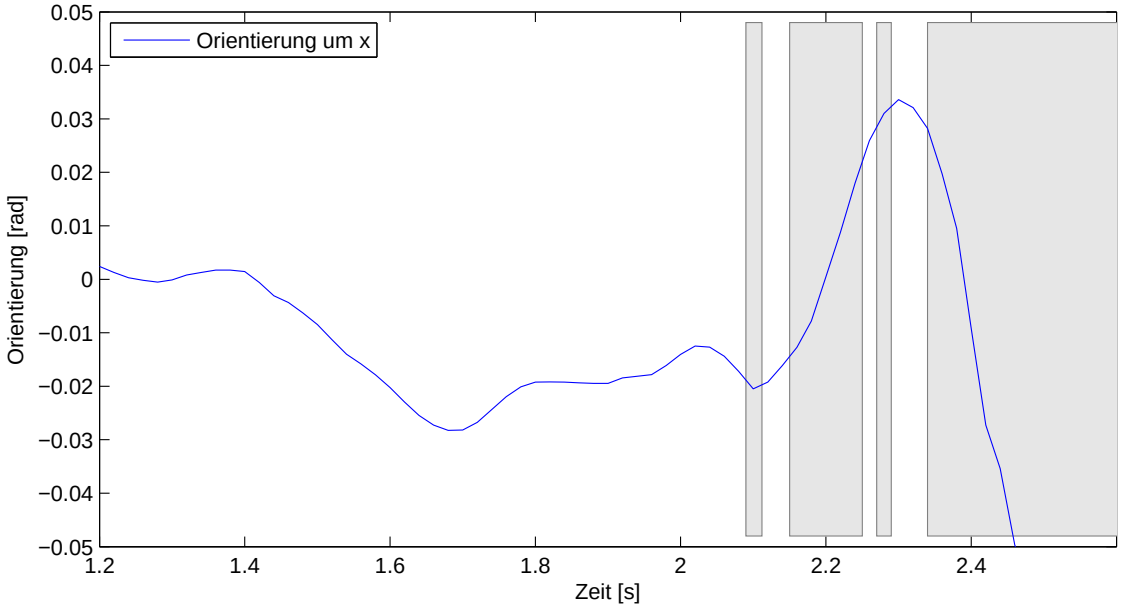


Abbildung 5.11: Orientierung des Oberkörpers. Zeiten, zu denen der linke Fuß Kontakt zum Boden hat, sind grau hinterlegt.

Berührung wieder abgestoßen wird. Zu dieser Zeit ist eine deutliche Verlangsamung des Oberkörpers zu sehen, die nicht statt finden dürfte und als Folge der Berührung anzusehen ist. Aus der Verlangsamung folgt eine Abweichung der Position vom Soll in die selbe Richtung. Ursachen für diese Abweichung sind einerseits Fehler aus der Regler- und Motorsimulation. Andererseits zeigt die Position aus der Vorwärtskinematik zeitweise einen kleineren Fehler als die ausgelesene Position. Der Roboter kippt demnach zusätzlich über seine äußere Fußkante, worauf auch Abbildung 5.11 hindeutet. Im weiteren Verlauf wird eine zu hohe Geschwindigkeit erzeugt, indem der Roboter zurück kippt und die Gelenke die korrekten Winkel erreichen. Der Roboter kann in der folgenden Single-Support-Phase auf dem linken Fuß nicht mehr gestoppt werden und stürzt.

5.2.2 Kompensation

Die gefundenen Ursachen für Beobachtung *Einknicken* machen eine Kompensation mit den vorgeschlagenen Mitteln schwierig. Mit ihnen wäre nur eine Vergrößerung der seitlichen Beschleunigungen möglich, so dass weniger Drehmoment nötig ist, um den Oberkörper aufrecht zu halten. Die erzeugbare zusätzliche Beschleunigung ist aber durch den Spielraum, den der ZMP innerhalb des Support-Polygons hat, gering, weshalb die Lösung nicht praktikabel ist.
Hier wird daher nur die Auswirkung des Einknickens, nämlich die *Seitwärtsschwingung* kompensiert. Die fehlerhafte Berührung des Bodens ist demnach nicht vermeidbar, aber ihre Auswirkungen. Dazu wird ein neuer Soll-ZMP-Verlauf entwickelt. Entlang der x-

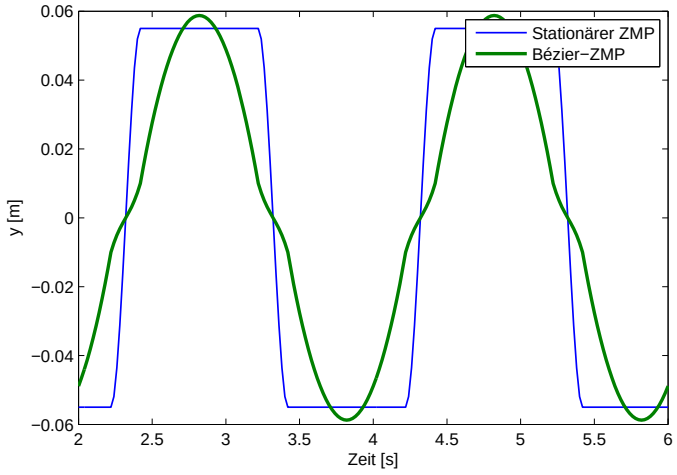


Abbildung 5.12: Bisheriger Soll-ZMP-Verlauf (stationärer ZMP) und neuer Soll-ZMP-Verlauf (Bézier-ZMP) entlang der y-Achse im Vergleich.

Achse verläuft der neue Soll-ZMP mit konstanter Geschwindigkeit. Dadurch folgt der Gesamtschwerpunkt mit konstanter Geschwindigkeit dem ZMP.

Die y-Koordinaten der ZMP-Kurve werden anhand einer eindimensionalen Bézier-Kurve bestimmt. Für jeden Fuß werden 4 Kontrollpunkte im Koordinatensystem des entsprechenden Fußes festgelegt. In der Single-Support-Phase wird die Bézier-Kurve von den Punkten des Standfußes definiert. In der Double-Support-Phase werden 4 Punkte zusammengestellt, bestehend aus dem letzten Punkt des vorhergehenden Standfußes, dem ersten des nächsten, und zwei neuen, die sich in der Mitte der beiden befinden.

Der resultierende Soll-ZMP-Verlauf, im Folgenden Bézier-ZMP genannt, ist in Abbildung 5.12 zu sehen. Er hat in der Single-Support-Phase den Vorteil sich so konfigurieren zu lassen, dass er die Form des Schwerpunkt-Verlaufs hat. Dadurch werden die Beschleunigungen am Anfang und Ende der Single-Support-Phase geringer und insgesamt gleichförmiger. In der Double-Support-Phase sorgen vor allem die beiden mittleren Kontrollpunkte dafür, dass der ZMP kurz langsamer wird, was auch den Schwerpunkt abbremst. Durch die beschriebene Wahl des ersten bzw. vierten Punktes stellt man einen weichen Übergang von einem Fuß zum nächsten sicher, egal an welcher Position im WKS sie sich befinden.

Den Effekt auf Beobachtung *Seitwärtsschwingung* zeigt Abbildung 5.13. Hier sind die Oberkörperpositionen beim herkömmlichen ZMP (MKM/ZMP-kompensiert) und beim Bézier-ZMP (ebenfalls MKM/ZMP-kompensiert) zu sehen. Zu erkennen ist, dass die Position ebenfalls zu Anfang und Ende der ersten Single-Support-Phase wie beim herkömmlichen Soll-ZMP-Verlauf abweicht. Durch die geringere Geschwindigkeit in der Double-Support-Phase ist aber eine zu hohe Geschwindigkeit in der folgenden Single-Support-Phase vermeidbar, und der Roboter läuft ohne zu fallen.

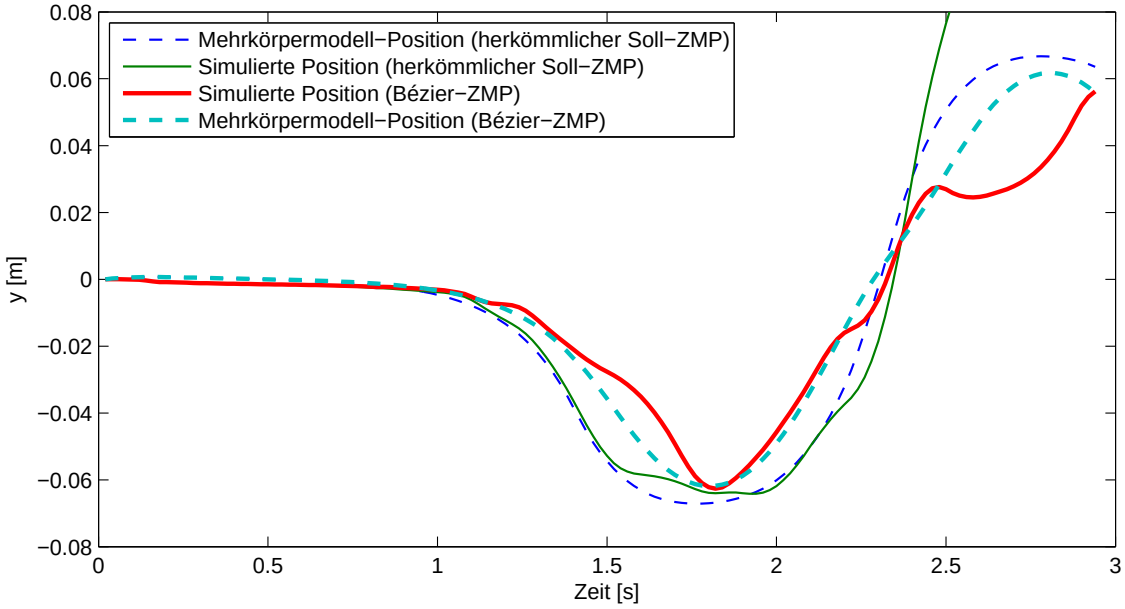


Abbildung 5.13: Vorteil des Bézier-ZMP im Vergleich zum herkömmlichen Soll-ZMP.

5.2.3 Fazit

In diesem Abschnitt wird zunächst gezeigt, dass der Walking Simulator in der Lage ist, realitätsnah die Beobachtungen *Seitwärtsschwingung* und *Einknicken* widerzuspiegeln. Die Übertragbarkeit der Ergebnisse zeigt Abbildung 5.14. Hier wird die Orientierung des simulierten Oberkörpers mit dem bei einem realen Lauf verglichen. Wenn auch die Winkel quantitativ wenig Ähnlichkeit haben, zeigen sie dennoch zu ähnlichen Zeitpunkten ein vergleichbaren qualitativen Verlauf, wobei die Art der Winkelmessung per Gyroskop und Beschleunigungssensoren für die Unterschiede verantwortlich sein kann. Zum Vergleich ist die Orientierung bei der **BS** eingetragen, woran man den Vorteil des Walking Simulators erkennen kann.

Bezüglich der Beobachtungen konnte gezeigt werden, dass die Hypothese für Beobachtung *Einknicken* weitgehend zutrifft. Hauptsächlich sind die Motoren und ihre Regelung dafür verantwortlich, dass der Oberkörper einknickt. Daraus folgt, dass der Oberkörper kurz vor der Double-Support-Phase einknickt, der linke Fuß zu früh den Boden berührt und den Oberkörper in die falsche Richtung stößt. Dadurch entsteht eine zu große Geschwindigkeit, die den Roboter zu Fall bringt, was in der Hypothese für Beobachtung *Seitwärtsschwingung* bereits vermutet wurde.

Eine Kompensation der Beobachtung *Einknicken* ist mit den gegebenen Mitteln nur schwer möglich. Ein veränderter ZMP-Verlauf, der den Schwerpunkt in der Double-Support-Phase langsamer werden lässt, ist aber in der Lage, die Auswirkung, also die *Seitwärtsschwingung*, so zu kompensieren, dass ein Lauf, bei dem der Roboter nach dem ersten Schritt fällt, stabilisiert wird.

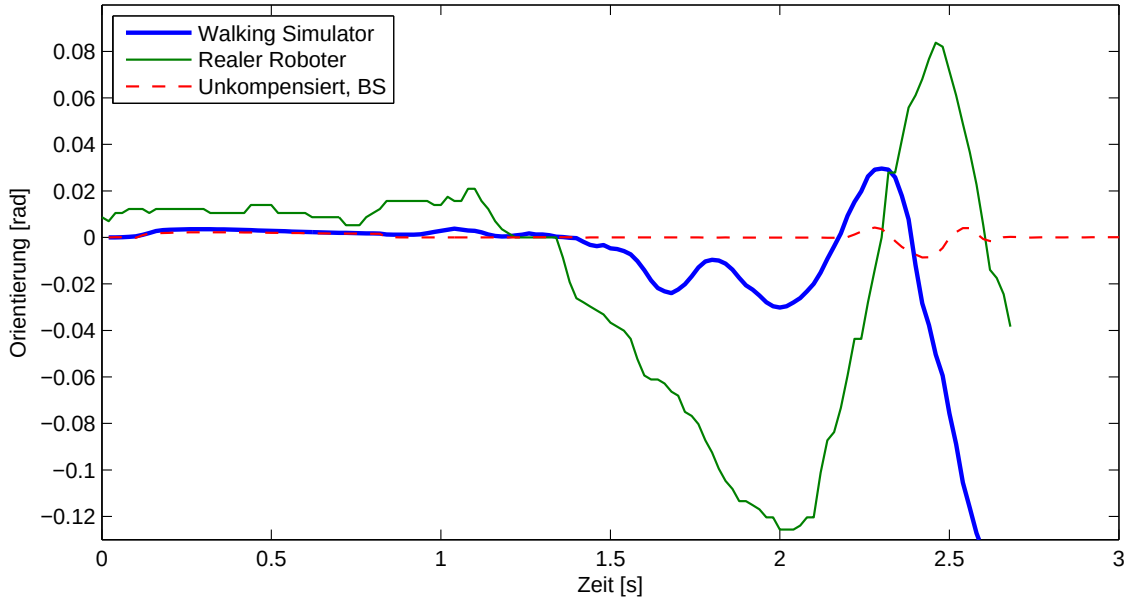


Abbildung 5.14: Orientierung des Oberkörpers vom simulierten zum realen Roboter im Vergleich.

5.3 *Geschwindigkeitssteigerung* und *Vorwärtsschwingung*

Im letzten Analyse- und Kompensationsabschnitt werden die Beobachtungen, die sich entlang der x-Achse äußern, thematisiert. Das ist zum einen die *Vorwärtsschwingung*. Dabei entsteht eine Schwingung des Oberkörpers entlang der x-Achse, die den realen Roboter nach wenigen Schritten stürzen lässt. Bei Beobachtung *Geschwindigkeitssteigerung* läuft der reale Roboter mit einer höheren Vorwärtsgeschwindigkeit als vorgegeben, was vor allem bei hohen Geschwindigkeiten vorkommt.

Zunächst werden beide Beobachtungen analysiert und ihre Zusammenhänge deutlich gemacht (vgl. Abschnitt 5.3.1). Dabei werden auch gefundene Kompensationen der vorigen Kapitel eingesetzt. Abschnitt 5.3.2 dient danach zur Entwicklung weiterer Kompensationsmöglichkeiten von Effekten, die zuvor nicht gänzlich beseitigt werden konnten. Den Abschluss bildet ein Fazit zu diesem Abschnitt (vgl. Abschnitt 5.3.3).

5.3.1 Analyse im Walking Simulator

Der Bézier-ZMP aus Abschnitt 5.2.2 erlaubt eine höhere Laufgeschwindigkeit als der stationäre ZMP. Mit ihm ist es möglich, in der Realität (mit Sensorkontrolle) und auch im Walking Simulator mit einer vorgegebenen Geschwindigkeit von $v_x = 20\frac{cm}{s}$ zu laufen.

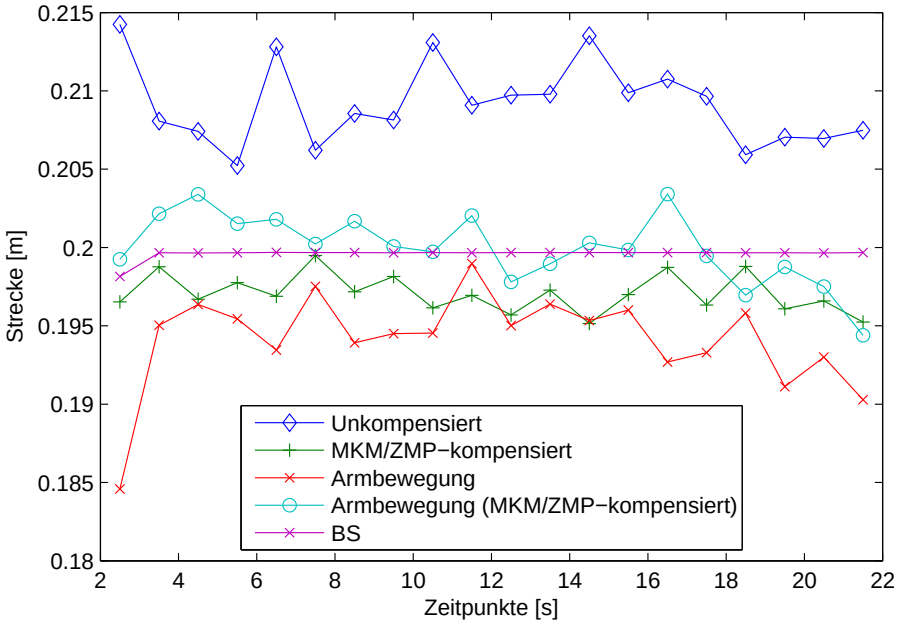


Abbildung 5.15: Geschwindigkeiten des simulierten Roboters.

Dabei ist die *Geschwindigkeitssteigerung* gut beobachtbar, in der Realität auf bis zu $25\frac{cm}{s}$. In diesem Abschnitt wird daher nur der Bézier-ZMP als Soll-ZMP verwendet.

Zur Messung der tatsächlichen Geschwindigkeit im Simulator werden für jede gelaufene Sekunde die vom Oberkörper zurückgelegten Strecken betrachtet. Bei einer Reibung der Füße mit dem Boden von $\mu_r = 1$ sind große Schwankungen in der Geschwindigkeit messbar, die keinen Schluss zulassen. Damit die Steigerung der Geschwindigkeit im Walking Simulator sichtbar wird, muß die Reibung gesenkt werden. Für $\mu_r = 0.25$ zeigt Abbildung 5.15 die zurückgelegten Strecken für 20 einzelne Sekunden ab $2.5s$, wo der Lauf sich der vollen Geschwindigkeit nähert. Von Sekunde 2.5 bis 22.5 beträgt die Geschwindigkeit im Durchschnitt $20.918\frac{cm}{s}$ bei einem zunächst nicht MKM/ZMP-kompensierten Lauf. Der Walking Simulator ist also in der Lage, die Geschwindigkeitssteigerung ebenfalls zu zeigen. Zum Vergleich sind auch die zurückgelegten Strecken bei **BS** eingetragen, welche nahezu genau $20\frac{cm}{s}$ in jeder Sekunde betragen (bis auf die Beschleunigungsphase zu Anfang).

Abbildung 5.16 zeigt die Orientierung um die y-Achse des simulierten Roboters, die eine Schwingung zeigt. Auf dem realen Roboter führt sie bei einer Geschwindigkeit von $v_x = 5\frac{cm}{s}$ nach wenigen Schritten zum Fall des Roboters, in der Simulation auch bei hohen Geschwindigkeiten nicht. Beobachtung *Vorwärtsschwingung* lässt sich also nur teilweise in der Simulation nachvollziehen.

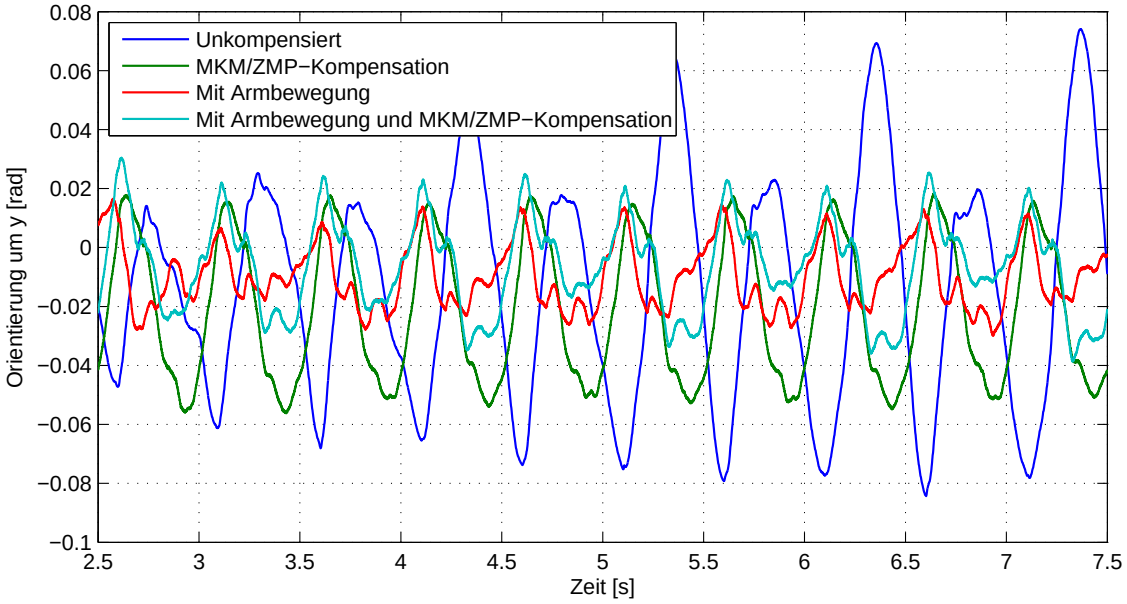


Abbildung 5.16: Orientierung des simulierten Oberkörpers um die y-Achse bei verschiedenen Kompensationen.

	f_x	f_y
Unkompensiert	7.0061	13.8960
MKM/ZMP-Kompensation	2.9498	8.7023
Mit Armbewegung	3.0982	2.5526
Mit Armbewegung und MKM/ZMP-Kompensation	2.9326	2.5187

Tabelle 5.2: Orientierungsfehler des Oberkörpers bei verschiedenen Kompensationsstufen (siehe Gleichungen 5.1 und 5.2).

	f_x	f_y
VS	7.0061	13.8960
VS-IN	8.5824	16.7813
BS+MT+FK	8.3269	23.6672
BS+MT	7.7779	15.7627
BS+FK	2.9498	8.7023
BS	0.0033	0.0013

Tabelle 5.3: Orientierungsfehler des Oberkörpers bei deaktivierten Simulationselementen (siehe Gleichungen 5.1 und 5.2).

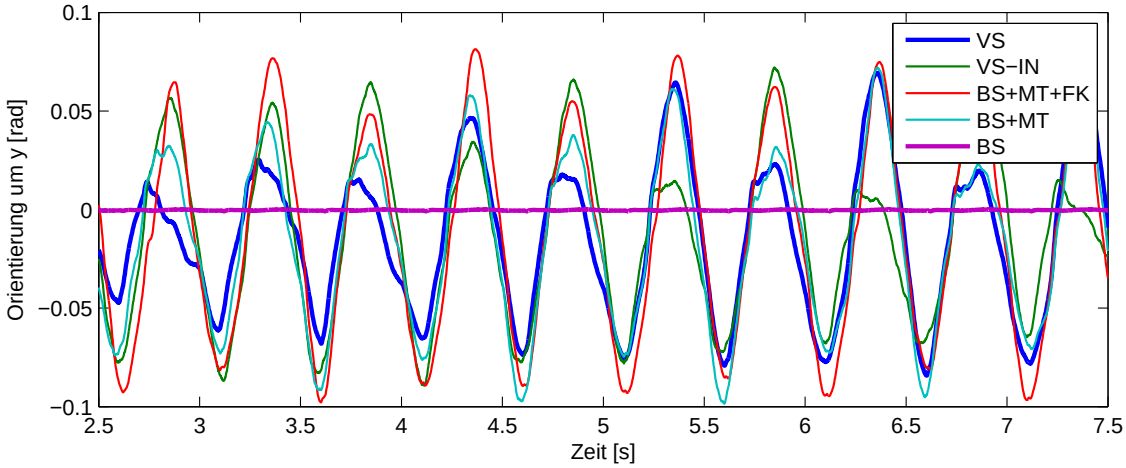


Abbildung 5.17: Einfluß verschiedener Simulationselemente auf die Orientierung.

Ursachenanalyse

Zunächst wird erneut die Dynamikabstraktion als mögliche Ursache in Betracht gezogen. Abbildung 5.15 zeigt die Geschwindigkeit für den gleichen Lauf mit MKM/ZMP-Kompensation. Offensichtlich hat die Dynamikabstraktion einen Einfluss auf die *Geschwindigkeitssteigerung*. Auch die Schwingung verringert sich sichtbar, wie in Tabelle 5.2 zu sehen ist.

Abbildung 5.17 und Tabelle 5.3 zeigen einen Vergleich der Einflüsse einzelner simulierter Kinematikfehler-Elemente[2] parallel zur Dynamikabstraktion. Die Simulation der Getriebe scheint sogar ein Vorteil in Bezug auf stabileres Laufen zu sein. Offenbar dämpfen sie die Schwingung, die durch die Simulation der Motoren und deren Ansteuerung entsteht. Flexible Körper sind eher von Nachteil.

Um die Stärke der Einflüsse besser einschätzbar zu machen, ist zum Vergleich ein Lauf eingezeichnet, bei dem nur die Simulation des elektrischen Feldes des Motors deaktiviert ist (VS-IN). Wie in Abschnitt 4.1 zu lesen ist, sollte ihr Einfluss gering sein, die Kurve zeigt aber dennoch ähnlich große Auswirkungen wie die Getriebe oder flexiblen Körper. Es kann daher nur sicher festgestellt werden, dass flexible Körper, wie auch die Motoren und ihre Ansteuerung, alleine den Roboter in Schwingung versetzen können.

Es ist nun zu klären, wie es zur *Geschwindigkeitssteigerung* kommt. In Abbildung 5.18 sind Punkte für die Fußposition eingetragen, wenn der Fuß Kontakt zum Boden hat. In der Zeit von 1.105*s* bis 1.125*s* rutscht er aus unklaren Gründen auf beiden Füßen in negative Richtung. Rutschen ist demnach kein Grund, es sorgt mehr dafür, dass der Roboter langsamer läuft als gewünscht. Die Fußdistanz an sich scheint sich kaum zu ändern. Abbildung 5.19 zeigt die Fußdistanz für den Fall, dass beide Füße Kontakt zum Boden

[2] Zur Erinnerung: BS ist die Basissimulation, MT steht für Motorsimulation und FK für flexible Körper. IN ist die Induktivität des Motors und VS stellt die Simulation mit allen Elementen dar. Siehe auch Seite 65.

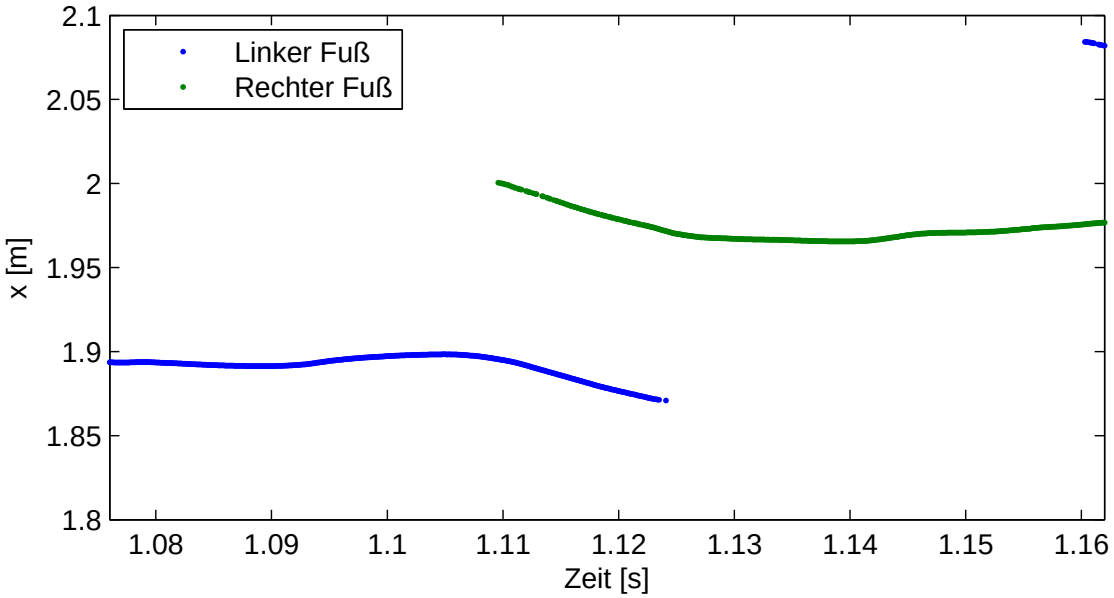


Abbildung 5.18: Position der Füße am Boden im Falle des Kontakts.

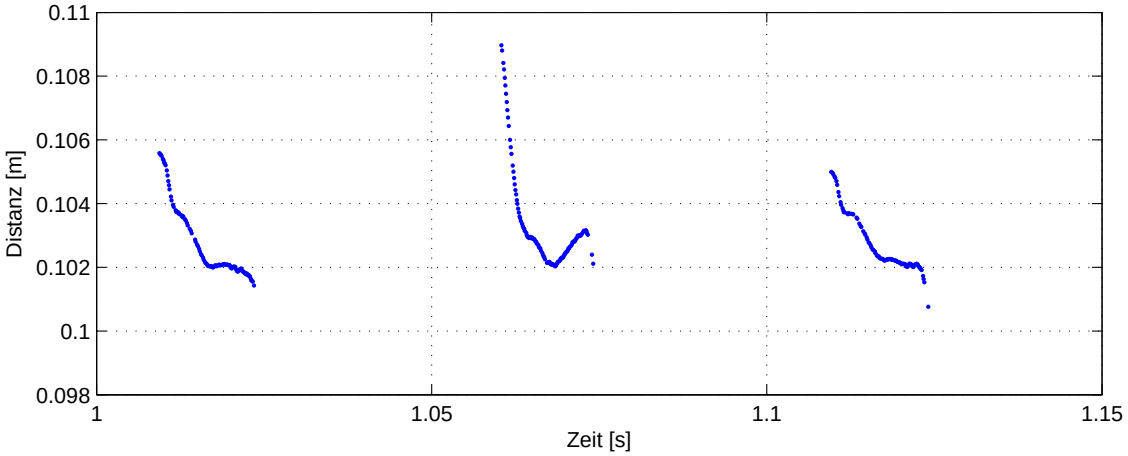


Abbildung 5.19: Distanz der Füße bei Kontakt mit dem Boden im unkompensierten Fall.

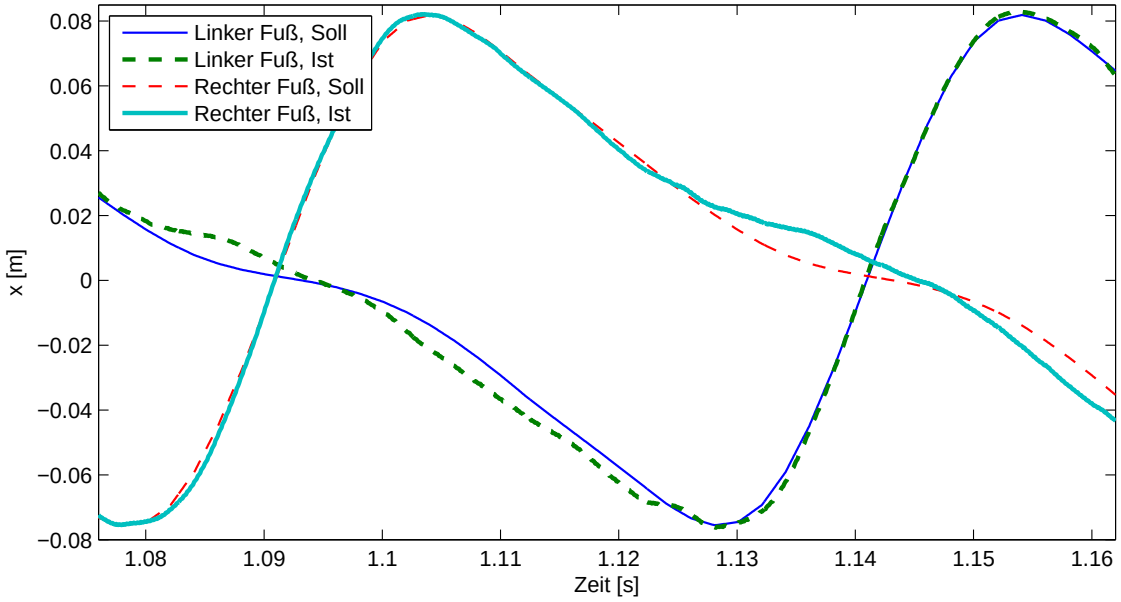


Abbildung 5.20: Soll- und Ist-Position (anhand der Vorwärtskinematik) der Füße im Roboterkoordinatensystem.

haben. Hier zeigt sich, dass eine zu große Schrittweite zur Geschwindigkeitssteigerung führt, richtig wäre bei einer Schrittdauer von einer Sekunde eine Schrittweite von nur $0.1m$. Abbildung 5.20 zeigt die Soll- und Ist-Position der Füße im Roboterkoordinatensystem, berechnet anhand der Vorwärtskinematik. Aus Abbildung 5.18 lassen sich dazu die Zeitpunkte entnehmen, zu dem eine Single-Support-Phase beginnt, was zum Beispiel bei $1.125s$ der Fall ist, wo der Roboter vom rechten Bein gestützt wird. Zu diesem Zeitpunkt endet auch die Positionsabweichung des linken Fußes und es entsteht ein Fehler im Standbein.
Zusammengefasst sind hauptsächlich die Motoren und ihre Ansteuerung nicht in der Lage, die nötigen Kräfte, die bei einem nicht kompensierten Bézier-ZMP auftreten, auszuüben. Daraus entsteht eine Schwingung, die dafür sorgt, dass die Füße mit einem zu großen Abstand den Boden berühren. Das führt zu einer höheren Geschwindigkeit.

5.3.2 Kompensation

Der neue Bézier-ZMP kommt mit MKM/ZMP-Kompensation bereits in Abschnitt 5.3.1 zum Einsatz und zeigt eine Verbesserung der Geschwindigkeitsabweichung und der Schwingung. Abbildung 5.15 und 5.16 zeigen, dass die Geschwindigkeit mit durchschnittlich $197.07\frac{mm}{s}$ noch nicht korrekt und die Schwingung so groß ist, dass weitere Kompensationen versucht werden sollten.
Abbildung 5.21 zeigt für das Standbein für die Gelenke, die sich um die y-Achse drehen, die Soll- und Ist-Winkel, sowie das Drehmoment, das laut Mehrkörpermodell notwendig

ist. Sie zeigt, dass zwischen dem notwendigen Drehmoment und der Winkelabweichung ein Zusammenhang besteht. Bei RKneePitch führt ein betraglich größeres Drehmoment auch zu einer betraglich größeren Abweichung. Bei den anderen Gelenken findet bei einem Vorzeichenwechsel des Drehmoments auch kurz darauf ein Wechsel der Abweichungsrichtung statt. Es wäre demnach ein Lauf erstrebenswert, der möglichst konstante und kleine Drehmomente an den Gelenken benötigt. Bei dem Knöchelgelenk ist das der Fall, wenn der ZMP sich konstant unterhalb des Gelenks befindet, da dann nur noch Kräfte vom Fuß auf den Unterschenkel übertragen werden, aber kein Drehmoment. Allerdings führt das bei den oberen Gelenken zu stärkeren Drehmomenten, was insgesamt zu einer stärkeren Schwankung führt. Daher ist, wie in der Einleitung dieses Abschnittes erwähnt, auch eine Geschwindigkeit von $v_x = 20\frac{cm}{s}$ mit dem stationären ZMP nicht möglich.
Neben RHipPitch ist außerdem auch das LHipPitch in Abbildung 5.21 eingetragen, also das entsprechende Gelenk des anderen Beines. Zwischen den beiden Gelenken liegt nur HipYawPitch, das hier nicht benötigt wird, und der Oberkörper. Das Drehmoment des Schwungbeines wird offensichtlich vom Oberkörper in seiner Form unverändert übertragen und verstärkt. Das Drehmoment vom Schwungbein lässt sich durch Änderung ihrer Trajektorie nur wenig verringern, da nur eine bestimmte Zeit zur Verfügung steht, um es die notwendigen $0.2m$ nach vorne zu bewegen. Für einen flacheren Drehmomentsverlauf wäre es daher sinnvoll, die Arme einzusetzen (speziell die Gelenke ShoulderPitch), um ein entgegengesetztes Drehmoment am Oberkörper zu erzeugen. Eine Möglichkeit ist eine Armbewegung anhand von Sinuskurven mit der Frequenz der Schwingung festzulegen. Getestet wurden gleiche wie auch phasenverschobene Winkel für RShoulderPitch und LShoulderPitch, dazu verschiedene Amplituden und Offsets. Der Erfolg bei Armbewegungen in dieser Form bleibt aber aus. Erfolgreich ist dagegen die Kopplung der Armwinkel mit der Orientierung des Oberkörpers. Die Orientierung wird zunächst gefiltert, um hochfrequente Störungen zu entfernen, und mit -10 multipliziert, wobei es sich um einen von Hand optimierten Faktor handelt, der gute Ergebnisse zeigt. Das Resultat wird auf beide ShoulderPitch angewendet. Den Effekt zeigen Abbildung 5.15 und 5.16. Die Schwankung kann nach Tabelle 5.2 um die y-Achse deutlich verringert werden, aber zur Verbesserung der Geschwindigkeit (nun durchschnittlich $194.16\frac{mm}{s}$) und der Schwankung um die x-Achse ist eine MKM/ZMP-Kompensation des Laufs mit Armbewegung notwendig. Dann wird eine durchschnittliche Geschwindigkeit von $199.96\frac{mm}{s}$ erreicht.

5.3.3 Fazit

In diesem Abschnitt wurde gezeigt, dass Beobachtung *Vorwärtsschwingung* vom Walking Simulator nur teilweise widerspiegelt wird. Eine Schwingung zeigt sich zwar auch hier,

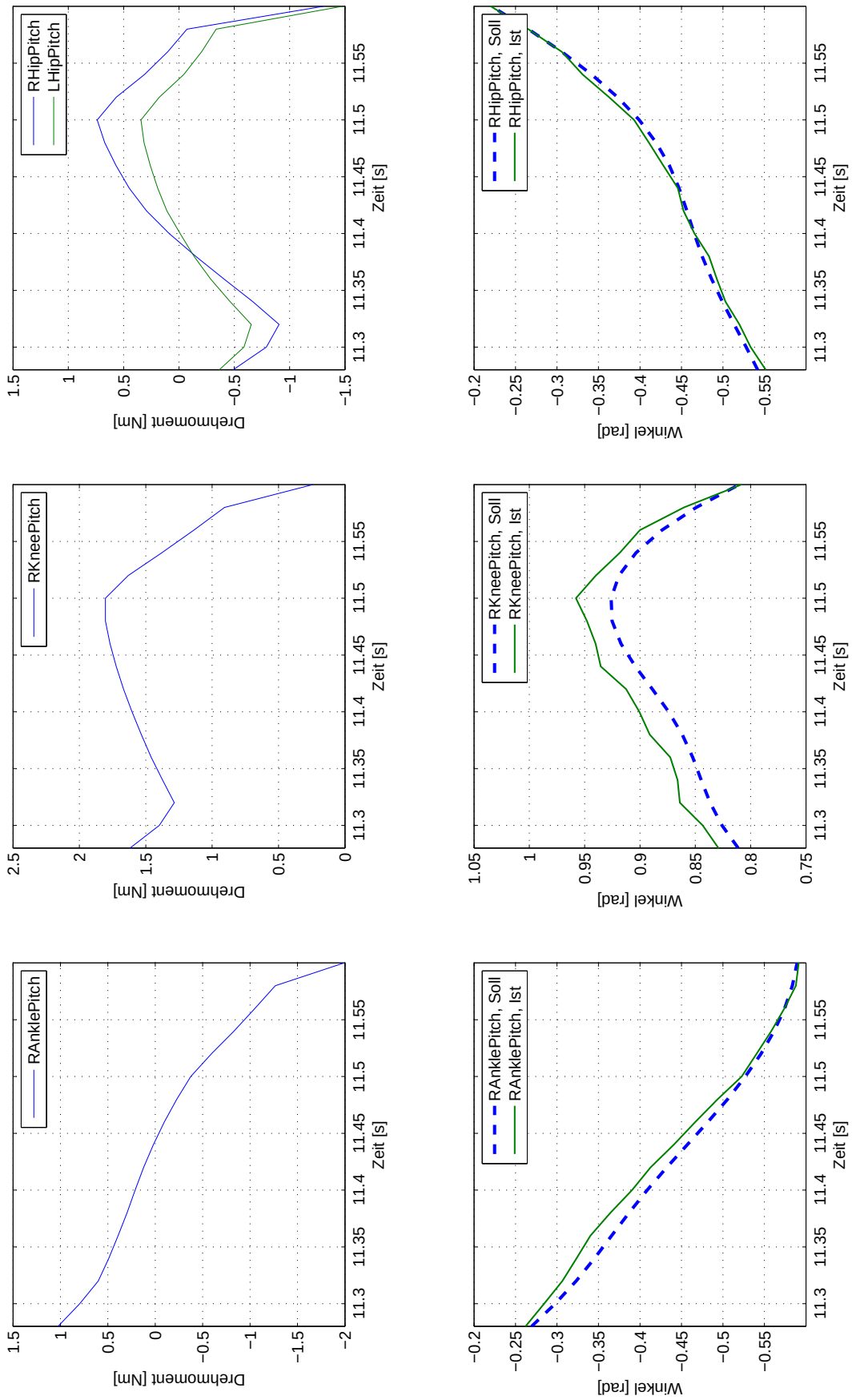


Abbildung 5.21: Drehmomente der Mehrkörperdynamik und daraus resultierende Winkelfehler der entsprechenden Gelenke in der Simulation.

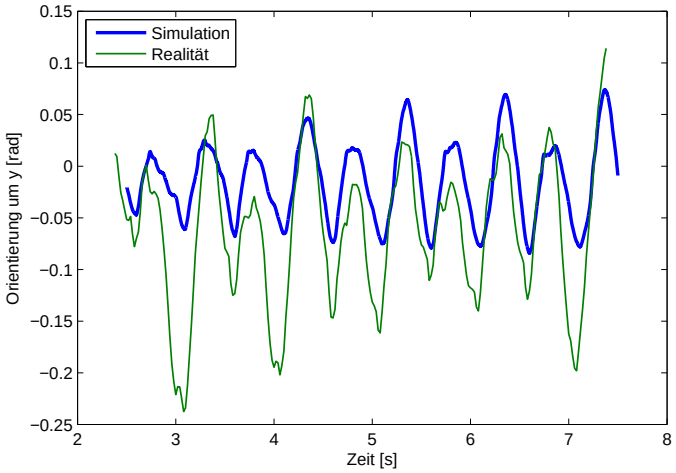


Abbildung 5.22: Oberkörperschwingung um die y-Achse vom realen Roboter im Vergleich zum simulierten Roboter.

bringt den Roboter aber nicht zu Fall. Die *Geschwindigkeitssteigerung* zeigt sich hingegen deutlich wie beschrieben. Ursachen für die Schwingung sind hauptsächlich die Dynamikabstraktion und die Simulation der Motoren und ihrer Ansteuerung. Das Standbein kann den Roboter nicht senkrecht halten, wodurch er zu schwingen beginnt. In dem Moment, wo der zweite Fuß den Boden berührt, ist dadurch der Abstand der beiden Füße zu groß. Aufgrund dieser zu großen Schrittweite entsteht die *Geschwindigkeitssteigerung*. Kompensiert werden kann die Schwingung und die Abweichung von der Sollgeschwindigkeit durch die MKM/ZMP-Kompensation bei gleichzeitigem Einsatz der Arme zum Ausgleich der Drehmomente, die der Oberkörper auf das Standbein ausübt.
Abbildung 5.22 zeigt die Realitätsnähe der Simulation und der Ergebnisse. Wie auch in Abschnitt 5.2 stimmen die Orientierungen quantitativ nicht überein. Die Schwingung hat in der Simulation aber die gleiche Frequenz und die gleiche Phase wie in der Realität. Es ist zudem in der Simulation, in Übereinstimmung mit der Realität, zu erkennen, dass jede zweite Schwingung nach vorne stärker bzw. schwächer ist. Daher ist die Schwingung wahrscheinlich auch in der Realität für die *Geschwindigkeitssteigerung* verantwortlich.

6

Zusammenfassung und Ausblick

Zweibeiniges Laufen humanoider Roboter ist ein wichtiges und herausforderndes Forschungsfeld. Das lässt sich auf dem jährlich stattfindenden RoboCup beobachten, wo Laufen einer der zentralen Punkte humanoider Ligen darstellt. Auch die „Dortmund Walking Engine“ des Teams „Nao Devils Dortmund“ zeigt einige Ungenauigkeiten, die wichtigsten davon wurden hier vorgestellt. Um ihre Ursachen finden und kompensieren zu können, wurde eine Hierarchie von Abstraktionsebenen vorgestellt. Die Walking Engine, auf einer abstrakten Ebene, berechnet einen Lauf, der auf ihrer Ebene zu den gewünschten Ergebnissen führt. Auf der komplexesten Ebene, dem realen Roboter, ist das nicht der Fall, und es ist damit klar, dass Elemente, von denen die verschiedenen Ebenen abstrahieren, für die Ungenauigkeiten verantwortlich sind. Eine erste Einteilung war mit SimRobot möglich, da sich einige der Beobachtungen hier bereits zeigen, andere dagegen nicht.

Die Mehrkörpermodellebene wurde als eine Ebene zur weiteren Untersuchung ausgesucht, da sie nah an der Ebene von SimRobot liegt, und mit ihr die Dynamikabstraktion als Ursache untersucht werden kann.

Daneben ist eine Untersuchung mit dem realen Roboter schwierig, da Daten sich nicht rausch- und verzögerungsfrei aufzeichnen lassen und Einflüsse wie Reibung, Getriebe, Motoren usw. nicht getrennt untersucht werden können. Daher wurde als weitere Untersuchungsebene der Walking Simulator festgelegt, der realitätsnäher simulieren soll wie SimRobot.

Entwicklungsphase

Die wichtigen Ausgaben der Rechnung in Sim-Richtung auf der MKM-Ebene sind die Oberkörper-Positionen und -Orientierungen, sowie die Drehmomente und der daraus resultierende ZMP. Auf dieser Ebene werden die Vorgaben, nämlich die Gelenkwinkel, exakt umgesetzt und der Roboter kann nicht kippen und nicht rutschen. Die Orientierung des Oberkörpers ist damit ohne weitere Rechnung klar, nämlich, wie gewünscht, senkrecht. Auch die Drehmomente werden korrekt umgesetzt, sind aber unbekannt, und müssen (beschränkt auf die Single-Support-Phase) von einer inversen Dynamik berechnet werden. Eine solche wurde für Roboter in Form von Baumstrukturen implementiert und getestet. Aus den Ergebnissen lässt sich auch der ZMP auf der MKM-Ebene berechnen. Da er danach in einem lokalen Koordinatensystem vorliegt, muß er in das WKS konvertiert werden. Dazu wurde ein Algorithmus entwickelt und getestet, der per Vorwärtskinematik die Positionen des Roboters bestimmt und die Konvertierung vornehmen kann.
Weiterhin wurde ein neuer Simulator entwickelt, der Walking Simulator, dessen Grundlage die Festkörpersimulation ODE ist. Damit ist die Simulation mindestens so komplex wie SimRobot und muß um einige Elemente erweitert werden, um eine realitätsnähere Simulation zu ermöglichen. Implementiert wurde die Simulation eines Motors, dessen Eingabe der Zielwinkel und Ausgabe das Drehmoment ist. Danach geschaltet ist die Simulation eines Getriebes, das flexibel sein kann und eine Toleranz aufweist. Daneben werden für die Beine flexible Körper verwendet. Die Elemente wurden auf ihre erwartungsgemäße Funktionsweise getestet und zu einem Modell vom Nao zusammengesetzt. Die Parametrisierung der Simulation ist von einem Evolutionären Algorithmus vorgenommen worden.

Ursachen der Beobachtungen

Die *ZMP-Abweichung* konnte im Walking Simulator nicht nachvollzogen werden, Ursachen dieser Beobachtung konnten daher nicht gefunden wurde. Eine Hypothese, dass die Dynamikabstraktion in Verbindung mit der Gesamtschwerpunktanpassung der Dortmund Walking Engine dafür verantwortlich ist, kann jedoch ausgeschlossen werden, da das zu einem betraglich zu großen ZMP entlang der y-Achse führt. Es konnte dagegen gezeigt werden, dass die Dynamikabstraktion großen Einfluß auf die *Vorwärtsschwingung* hat, woraus eine größere Schrittweite folgt, die zur *Geschwindigkeitssteigerung* führt.
Unter den Kinematikfehlern hat die Simulation der Motoren den größten Einfluss. Hauptsächlich dieses Element führt zum *Einknicken*, weswegen in der Single-Support-Phase der Schwungfuß zu früh den Boden berührt und den Roboter in die falsche Richtung stößt. Daraus folgt die *Seitwärtsschwingung*. Ohne die Simulation der Motoren würde die Dynamikabstraktion auch nicht zur *Vorwärtsschwingung* führen. Es ist vermutlich vornehm-

lich die Regelung durch PID-Regler, die die Motorsimulation für die Ungenauigkeiten verantwortlich macht.
Der Einfluss der Getriebe und der flexiblen Körper ist dagegen gering, aber nicht vernachlässigbar. So sind die flexiblen Körper alleine in der Lage, die *Seitwärtsschwingung* zu verursachen, wenn sie auch nicht so schnell zum Fall führen wie die Motorsimulation.

Kompensationen

Zur Kompensation der Auswirkungen der Dynamikabstraktion wurde die MKM/ZMP-Kompensation vorgestellt. Bei diesem iterativen Verfahren wird durch Vorberechnung die Differenz zwischen dem Ist-ZMP auf der MKM-Ebene und dem Soll-ZMP mit jedem Durchlauf verringert. Das führt zu einer deutlich geringeren *Vorwärtsschwingung*, die durch Einsatz der Arme weiter vermindert werden kann. Auch die *Geschwindigkeitssteigerung* wird dadurch nahezu ausgeglichen.
Für die *Seitwärtsschwingung* wurde ein veränderter ZMP-Verlauf entwickelt, der Bézier-ZMP. Dadurch, dass dieser Soll-ZMP die Form der Schwerpunktbewegung hat, sind die seitlichen Beschleunigungen geringer und die Geschwindigkeit im kritischen Moment kleiner. Der Roboter kann dadurch auch bei einer Schrittdauer von $2s$ stabil laufen.
Nicht kompensiert wurde das *Einknicken*. Mit den vorgeschlagenen Mitteln ist eine Kompensation nicht sinnvoll.

RoboCup 2010

Für den RoboCup wurden daher als Zwischenlösung einstellbare Winkeloffsets für jedes Gelenk implementiert, die in der Single-Support-Phase angewendet werden. Zusammen mit dem Bézier-ZMP war es möglich, eine Vorwärtsgeschwindigkeit von $444.7\frac{mm}{s}$ zu erreichen. Dazu wurde eine kurze Schrittdauer von $0.35s$ und ein niedriger Soll-Schwerpunkt verwendet, um die *Vorwärtsschwingung* zu verringern. Zum Vergleich: Das schnellste andere Team war in der Lage $330\frac{mm}{s}$ zu erreichen.

Ausblick

Die Winkeloffsets können nur eine Zwischenlösung sein. Es konnte gezeigt werden, dass der Winkelfehler im Zusammenhang mit dem Soll-Drehmoment steht. Eine sinnvollere Lösung wäre daher ein Offset in Abhängigkeit vom Soll-Drehmoment. Dazu muß zunächst die Drehmomentsberechnung auch in der Double-Support-Phase möglich sein.

Um die MKM/ZMP-Kompensation auf dem realen Roboter nutzbar zu machen, muß sie in eine Form gebracht werden, in der eine Berechnung online während des Laufs möglich ist. Die Schwierigkeit darin besteht in der begrenzten Rechenleistung. Alternativ wäre

eine Vorberechnung der $p^{WKS}_{zmp,offset}$ (siehe Algorithmus 5.1) für verschiedene Geschwindigkeiten möglich, die dann online zum Soll-ZMP addiert werden. Diese Lösung wäre aber nur für vorbestimmte Geschwindigkeiten optimal.

Auch die zur Kompensation verwendete Armbewegung ist nicht ohne weiteres auf dem realen Roboter anwendbar. Die rausch- und verzögerungsfreie Messung der Orientierung des Oberkörpers im Simulator ist auf dem realen Roboter nicht möglich. Es muß daher nach geeigneten Wegen gesucht werden, eine Armbewegung trotz dieser Schwierigkeiten umzusetzen.

Die entwickelten Werkzeuge, insbesondere der Walking Simulator, lassen sich auch in anderen Zusammenhängen einsetzen, z.B. zur Weiterentwicklung der Sensorkontrolle. Mit dem Walking Simulator können Dinge getestet werden, die andere Sensorik oder kurze Berechnungszeitschritte erfordern. So ist die Verzögerung der Messung derzeit ein großes Problem für die Sensorkontrolle. Wird der Walking Simulator um Sensoren erweitert, die eine einstellbare Verzögerung aufweisen, lässt er sich zum Testen von Algorithmen nutzen, die dieses Problem angehen. Eine weitere interessante Verbesserungsmöglichkeit der Dortmund Walking Engine wäre die Erhöhung der Berechnungsfrequenz von derzeit $50Hz$ (bzw. $100Hz$ bei der neusten Nao-Generation) auf zum Beispiel $1kHz$, die dann auch die Regelung der Gelenke beinhalten könnte, die in dieser Arbeit als eine der größten Schwierigkeiten ausgemacht wurde.

ABBILDUNGSVERZEICHNIS

LITERATURVERZEICHNIS

[BHK+03] BUSS, M. ; HARDT, M. ; KIENER, J. ; SOBOTKA, M. ; STELZER, M. ; STRYK, O. von ; WOLLHERR, D.: Towards an autonomous, humanoid, and dynamically walking robot: modeling, optimal trajectory planning, hardware architecture, and experiments. In: *Third IEEE International Conference on Humanoid Robots*. Karlsruhe, München : Springer Verlag, Sept. 30 - Oct. 3 2003, S. to appear

[BLB+07] BUSCHMANN, T. ; LOHMEIER, S. ; BACHMAYER, M. ; ULBRICH, H. ; PFEIFFER, F.: A collocation method for real-time walking pattern generation. In: *IEEE/RAS International Conference on Humanoid Robotics*, 2007

[CK09] CZARNETZKI, Stefan ; KERNER, Sören: Nao Devils Dortmund Team Description for RoboCup 2009. In: BALTES, Jacky (Hrsg.) ; LAGOUDAKIS, Michail G. (Hrsg.) ; NARUSE, Tadashi (Hrsg.) ; SHIRY, Saeed (Hrsg.): *RoboCup 2009: Robot Soccer World Cup XII Preproceedings*, RoboCup Federation, 2009

[CKU09a] CZARNETZKI, Stefan ; KERNER, Sören ; URBANN, Oliver: Applying Dynamic Walking Control for Biped Robots. In: *Proceedings RoboCup 2009 International Symposium*. Graz, Austria, July 2009

[CKU09b] CZARNETZKI, Stefan ; KERNER, Sören ; URBANN, Oliver: Observer-based dynamic walking control for biped robots. In: *Robotics and Autonomous Systems* 57 (2009), Nr. 8, S. 839–845. – ISSN 0921–8890

[Cor96] CORKE, P.I.: A Robotics Toolbox for MATLAB. In: *IEEE Robotics and Automation Magazine* 3 (1996), März, Nr. 1, S. 24–32

[Fea07] FEATHERSTONE, Roy: *Rigid Body Dynamics Algorithms*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2007. – ISBN 0387743146

[Hah02] HAHN, Hubert: *Rigid Body Dynamics of Mechanisms*. Springer, 2002

[Han06] HANSEN, N.: The CMA evolution strategy: a comparing review. In: LOZANO, J.A. (Hrsg.) ; LARRANAGA, P. (Hrsg.) ; INZA, I. (Hrsg.) ; BENGOETXEA,

E. (Hrsg.): *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*. Springer, 2006, S. 75–102

[Hau09] HAUSCHILDT, Daniel: *Entwicklung eines Systems zur Bewegungsund Kräfteschätzung eines humanoiden Roboters durch Festkörper-Simulation und Sensorfusion*, TU Dortmund, Institut für Roboterforschung, Diplomarbeit, 2009

[Heb09] HEBBEL, Matthias: *Evolutionäre Algorithmen zur Optimierung von Modellen für laufende Roboter*, TU-Dortmund University, Diss., 2009

[Hei07] HEIN, Daniel: *Simloid – Evolution of Biped Walking Using Physical Simulation*, Humboldt-Universität zu Berlin, Institut für Informatik, Diplomarbeit, 2007

[HK04] HANSEN, N. ; KERN, S.: Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In: YAO, X. (Hrsg.) u. a.: *Parallel Problem Solving from Nature PPSN VIII* Bd. 3242, Springer, 2004 (LNCS), S. 282–291

[HMS07] HERING, Ekbert ; MARTIN, Rolf ; STOHRER, Martin: *Physik für Ingenieure*. Springer-Verlag Berlin Heidelberg, 2007

[HNF06] HEBBEL, Matthias ; NISTICÒ, Walter ; FISSELER, Denis: Learning in a High Dimensional Space: Fast Omnidirectional Quadrupedal Locomotion. In: *RoboCup*, 2006, S. 314–321

[IGHM08] IGEL, Christian ; GLASMACHERS, Tobias ; HEIDRICH-MEISNER, Verena: Shark. In: *Journal of Machine Learning Research* 9 (2008), S. 993–996

[KIIO04] KANDA, T. ; ISHIGURO, H. ; IMAI, M. ; ONO, T.: Development and evaluation of interactive humanoid robots. In: *Proceedings of the IEEE* 92 (Nov. 2004), Nr. 11, S. 1839–1850. – ISSN 0018–9219

[KKK+03a] KAJITA, Shuuji ; KANEHIRO, Fumio ; KANEKO, Kenji ; FUJIWARA, Kiyoshi ; HARADA, Kensuke ; YOKOI, Kazuhito ; HIRUKAWA, Hirohisa: Biped walking pattern generation by using preview control of zero-moment point. In: *ICRA*, IEEE, 2003, S. 1620–1626

[KKK+03b] KAJITA, Shuuji ; KANEHIRO, Fumio ; KANEKO, Kenji ; FUJIWARA, Kiyoshi ; YOKOI, Kazuhito ; HIRUKAWA, Hirohisa: Biped walking pattern generation by a simple three-dimensional inverted pendulum model. In: *Advanced Robotics* 17 (2003), Nr. 2, S. 131–147

[KKK+03c] KAJITA, Shuuji ; KANEHIRO, Fumio ; KANEKO, Kenji ; FUJIWARA, Kiyoshi ; YOKOI, Kazuhito ; HIRUKAWA, Hirohisa: Biped walking pattern generation by a simple three-dimensional inverted pendulum model. In: *Advanced Robotics* 17 (2003), Nr. 2, S. 131–147

[KKK+06] KAJITA, Shuuji ; KANEHIRO, Fumio ; KANEKO, Kenji ; FUJIWARA, Kiyoshi ; YOKOI, Kazuhito ; HIRUKAWA, Hirohisa: Biped Walking Pattern Generator allowing Auxiliary ZMP Control. In: *IROS*, IEEE, 2006, S. 2993–2999

[Kos06] KOSSE, Ralf: *Planung und Implementierung eines evolutionären Ansatzes zur Steuerung eines zweibeinigen Roboters*, Uni Dortmund, Diplomarbeit, 2006

[Kra09] KRAMER, O.: *Computational Intelligence: Eine Einführung (Informatik Im Fokus)*. Springer-Verlag Berlin, 2009

[LGCM09] LIMA, José L. ; GONÇALVES, José A. ; COSTA, Paulo G. ; MOREIRA, Antonio P.: Humanoid Realistic Simulator - The Servomotor Joint Modeling. In: *ICINCO-RA*, 2009, S. 396–400

[LSR06] LAUE, T. ; SPIESS, K. ; RÖFER, T.: SimRobot - A General Physical Robot Simulator and Its Application in RoboCup. In: BREDENFELD, A. (Hrsg.) ; JACOFF, A. (Hrsg.) ; NODA, I. (Hrsg.) ; TAKAHASHI, Y. (Hrsg.): *RoboCup 2005: Robot Soccer World Cup IX*, Springer; http://www.springer.de/, 2006 (Lecture Notes in Artificial Intelligence 4020), S. 173–183

[LWP80] LUH, J. Y. S. ; WALKER, M. W. ; PAUL, R. P. C.: On-Line Computational Scheme for Mechanical Manipulators. In: *J. DYN. SYS. MEAS. & CONTR.* 102 (1980), Nr. 2, S. 69–76

[MH04] MAMADYL, Guanzheng T. ; HE, D. Huan He S.: On-line computational scheme for dynamic walking of anthropomorphic biped robots - II. In: *Intelligent Control and Automation, 2004. WCICA 2004. Fifth World Congress on* Bd. 6, 2004, S. 4922 – 4926

[Mic04] MICHEL, O.: Webots: Professional Mobile Robot Simulation. In: *Journal of Advanced Robotics Systems* 1 (2004), Nr. 1, S. 39–42

[MMW+01] MATEK, Wilhelm ; MUHS, Dieter ; WITTEL, Herbert ; BECKER, Manfred ; JANNASCH, Dieter: *Maschinenelemente*. Vieweg, 2001

[MN98] MATSUMOTO, Makoto ; NISHIMURA, Takuji: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. In: *ACM Trans. Model. Comput. Simul.* 8 (1998), Nr. 1, S. 3–30. – ISSN 1049–3301

[NG89] NAKAMURA, Y. ; GHODOUSSI, M.: Dynamics computation of closed-link robot mechanisms with nonredundant and redundant actuators. In: *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION* 5 (1989), Nr. 3, S. 294–302

[NY00] NAKAMURA, Y. ; YAMANE, K.: Dynamics computation of structure-varying kinematic chains and its application to human figures. In: *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION* 16 (2000), Nr. 2, S. 124–134

[Rec73] RECHENBERG, I.: *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart-Bad Cannstatt : Frommann-Holzboog, 1973 (Problemata 15)

[Sch95] SCHWEFEL, Hans-Paul: *Evolution and Optimum Seeking*. New York : Wiley Interscience, 1995 (Sixth-Generation Computer Technology). – ISBN 0–471–57148–2

[SE07] SEVEN, Utku ; ERBATUR, Kemalettin: An inverted pendulum based approach to biped trajectory generation with swing leg dynamics. In: *2007 7th IEEE-RAS International Conference on Humanoid Robotics*, 2007, S. 44

[SK08] SICILIANO, Bruno (Hrsg.) ; KHATIB, Oussama (Hrsg.): *Handbook of Robotics*. Berlin, Heidelberg : Springer, 2008. – ISBN 978–3–540–23957–4

[Smi02] SMITH, Russel: *Open Dynamics Engine*. www.ode.org, 2002

[SSC09] STROM, Johannes ; SLAVOV, George ; CHOWN, Eric: Omnidirectional Walking Using ZMP and Preview Control for the NAO Humanoid Robot. In: *RoboCup*, 2009, S. 378–389

[SV89] SPONG, M. W. ; VIDYASAGAR, M.: *Robot Dynamics and Control*. New York : John Wiley and Sons, 1989

[VB04] VUKOBRATOVIC, M ; BOROVAC, B: Zero-moment Point – Thirty Five Years of its life. In: *International Journal of Humanoid Robotics* 1 (2004), Nr. 1, S. 157–173

[Vuk90] VUKOBRATOVIC, M.: *Biped Locomotion*. Berlin, Germany : Springer-Verlag, 1990. – ISBN 3540174567

Hiermit bestätige ich, die vorliegende Diplomarbeit selbständig und nur unter Zuhilfenahme der angegebenen Literatur verfasst zu haben.

Ich bin damit einverstanden, dass Exemplare dieser Arbeit in den Bibliotheken der Universität Dortmund ausgestellt werden.

Dortmund, den 2. September 2010

Oliver Urbann