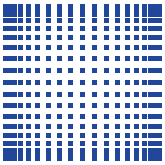


hochschule mannheim

Fakultät Informatik
Institut für Robotik

# Diplomarbeit

# Entwicklung eines statisch stabilen Laufalgorithmus für einen zweibeinigen Laufroboter

**eingereicht von:** Jörg Fellmann

**eingereicht am:** 24. April 2007

**Betreuer:** Herr Prof. Dr. Thomas Ihme

**Zweitkorrektor:** Frau Prof. Dr. Miriam Föller

**erstellt bei:** Hochschule Mannheim
Windeckstraße 116
68309 Mannheim

# Abstract

Mobile autonome Roboter können bei schweren oder gefährlichen Arbeiten den Menschen ersetzen bzw. unterstützen. Die zweibeinige Fortbewegung bietet im Gegensatz zur radgetriebenen höchstmögliche Mobilität in der heutigen von Menschen geschaffenen Umwelt. Eine große Herausforderung liegt in der Entwicklung eines leistungsfähigen Laufalgorithmus. Dieser muss eine ausreichende Stabilität bieten und gleichzeitig energieeffizient und flexibel sein. Um erste Erfahrungen mit einem an der Hochschule Mannheim konstruierten zweibeinigen Laufroboter zu sammeln, wird für diesen in der vorliegenden Arbeit ein statisch stabiler Laufalgorithmus konzipiert und implementiert. Hierzu wird ein parameterisiertes Laufmuster basierend auf einem zuvor erstellten Stabilitätskriterium geplant. Das im kartesischen Raum vorliegende Laufmuster wird mit Hilfe der aus der Robotik bekannten Methoden der Kinematik in entsprechende Gelenkstellungen umgewandelt. Während der Arbeit mit dem Roboter wurden problematische Eigenschaften in der Konstruktion entdeckt und eventuelle Lösungen vorgeschlagen. Durch die mathematische Analyse der aus dem Laufalgorithmus gewonnenen Daten werden geeignete Parameter für das Laufmuster gesucht und der Algorithmus bewertet. Die Ergebnisse dieser Analyse lassen ein für statisch stabile Laufalgorithmen positives Resultat im praktischen Einsatz vermuten. Jedoch können erst praktische Versuche, die zum Zeitpunkt dieser Arbeit noch nicht möglich waren, die tatsächliche Qualität des Laufalgorithmus unter Beweis stellen.

# Danksagung

Ich möchte besonders Herrn Prof. Dr. Thomas Ihme für seine Unterstützung und Betreuung im Verlauf dieser Arbeit danken.

Bedanken möchte ich mich auch bei meiner Familie. Insbesondere bei meiner Schwester Katja für das intensive Korrekturlesen während der Erstellung der Arbeit, aber auch bei meinem Bruder Claus und meinem Schwager Jürgen für ein letztes Korrekturlesen der finalen Version. Nicht zuletzt will ich hiermit auch meiner Mutter danken, ohne deren Unterstützung ich mein Studium vermutlich nicht begonnen hätte.

Auch meinen Kommilitonen Rafael Troilo und Kai Wetzelsberger möchte ich für die Zusammenarbeit und beratende Funktion während der Arbeit danken.

Mannheim im April 2007

Jörg Fellmann

# Inhaltsverzeichnis

# 1 Einleitung

Die vorliegende Arbeit beschreibt die Entwicklung eines statisch stabilen Laufalgorithmus für einen zweibeinigen Laufroboter.
Der erste Teil dieses Kapitels erläutert die Motivation, die hinter der Forschungsarbeit mit zweibeinigen Laufrobotern steht. Im zweiten Teil wird der aktuelle Stand der Forschung aufgezeigt und der letzte Teil beschreibt das Ziel und den Aufbau dieser Arbeit.

## 1.1 Wozu zweibeinige Laufroboter?

Heutzutage sind stationäre Roboter bei der Produktion in der Industrie kaum mehr wegzudenken. Hauptsächlich eingesetzt an Fließbändern sind diese als Schweiß-, Niet- oder Montageroboter ein schnellerer und genauerer Ersatz für menschliche Arbeiter. Viele Aufgaben, die von Robotern erledigt werden könnten, benötigen allerdings eine große Mobilität, daher sind auch mobile Roboter bereits seit langem Teil der Forschung auf dem Gebiet der Robotik. Erste praxistaugliche Anwendungen haben sich daraus auch schon entwickelt: In manchen Krankenhäusern ist es heutzutage üblich, mit mobilen Robotern auf abgesperrten Wegen Essen, Wäsche oder anderes automatisch auf vorgegebene Sammelpunkte zu verteilen oder von dort einzusammeln. Auch kleine Saugroboter die automatisch den Boden systematisch abfahren und reinigen sind bereits erhältlich.

All diese Anwendungen basieren auf Robotern, die sich mit Rädern fortbewegen. Eine andere Fortbewegungsart bieten Laufroboter: Roboter, die sich mit Hilfe einer beliebigen Anzahl an Beinen fortbewegen. Laufroboter sind bei der Konstruktion und Steuerung weitaus komplizierter als Roboter mit Radantrieb. Ebenso wurde mit Laufrobotern noch keine vergleichbar hohe Fortbewegungsgeschwindigkeit erreicht. Radgetriebene Roboter haben allerdings den Nachteil, dass sie nur wenig Flexibilität in Bezug auf den Boden haben auf dem sie sich fortbewegen [10]. Treppen oder auch nur ein zu steiniger Untergrund können hier oft schon eine unüberwindbare Hürde sein. Die Fortbewegung mit Beinen bietet hingegen eine höhere Mobilität: Bei Laufrobotern können die Auftrittspunkte fle-

xibel gesetzt werden, somit wird keine kontinuierliche Spur benötigt. Außerdem ist es möglich, Hindernisse bis etwa zur Beinhöhe zu überschreiten.

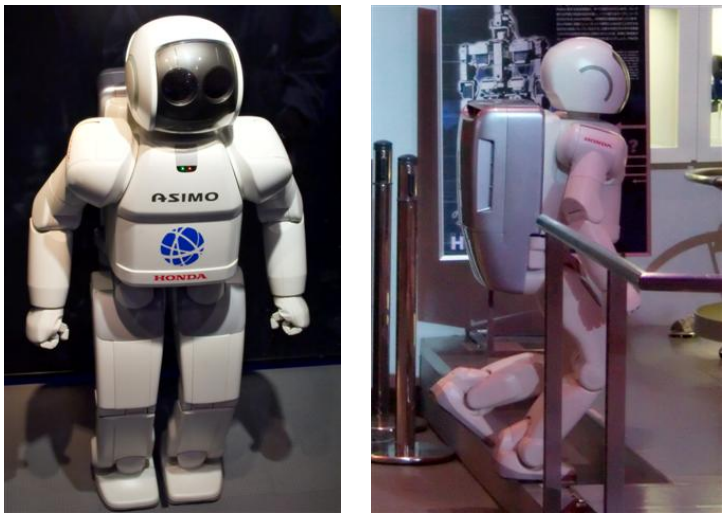
Obwohl Laufroboter theoretisch mit einer beliebigen Anzahl an Beinen gefertigt werden können, haben sich sechs-, vier- und zweibeinige Laufroboter als häufigste Form herausgestellt. Bei sechs-, und vierbeinigen Robotern ist die Entwicklung eines stabilen Gangs verhältnismäßig einfach. Allerdings bieten zweibeinige Laufroboter einige entscheidende Vorteile im Vergleich zu sechs- und vierbeinigen: In unserer heutigen Welt ist vieles auf zweibeinige Fortbewegung ausgelegt. Daher kann auch nur ein zweibeiniger Laufroboter die Flexibilität bieten, um in unserer menschengeschaffenen Umwelt vielseitig eingesetzt zu werden. Auch psychologische Aspekte sprechen für zweibeinige Laufroboter. Aufgrund ihrer menschenähnlichen Anatomie und Bewegung fällt es dem Menschen leichter, sich an eine zunehmende Anzahl Roboter in seiner direkten Umwelt zu gewöhnen [8]. Aufgrund der Menschenähnlichkeit zweibeiniger Laufroboter wird für diese auch häufig die Bezeichnung humanoide Roboter verwendet.

Neben den direkten Anwendungen, für die zweibeinige Laufroboter eingesetzt werden können, gibt es auch noch einen indirekten Nutzen aus der Forschung mit zweibeinigen Laufrobotern. Viele Teilbereiche zweibeiniger Laufroboter, aber auch von Laufrobotern allgemein, sind noch nicht ausreichend erforscht oder es wurden noch keine optimalen Lösungen gefunden. Durch die Ergebnisse der Forschung in diesen Teilbereichen können sich auch neue Technologien für andere Bereiche außerhalb der Robotik ergeben: Ein gutes Beispiel hierfür ist die Entwicklung von neuen Prothesen, die vermutlich von einigen Ergebnissen aus der Erforschung der Gelenke der Laufroboter direkt profitieren kann.

## 1.2 Forschung mit zweibeinigen Laufrobotern

Bereits seit langem wird auf dem Gebiet der zweibeinigen Laufroboter geforscht. Lange Zeit war dies allerdings hauptsächlich auf theoretische Arbeiten beschränkt. Erst in den letzten paar Jahren wurden, dank neu verfügbarer Technologien und Materialien, viele Erfolge bei praktischen Versuchen erreicht. Der wohl bekannteste in der Industrie entwickelte zweibeinige Laufroboter ist Hondas Asimo[1] (Abb. 1.1(a)). Er bietet beeindruckende Eigenschaften in Bezug auf die Fortbewegung. Asimo kann bei einer Größe von nur 1,30 m problemlos mit einer Geschwindigkeit von bis zu 2,7 km/h gehen und rennen mit bis zu 6 km/h. Auch das Treppenlaufen stellt kein Problem dar (Abb. 1.1(b)).

[1] http://world.honda.com/ASIMO/




(a) Asimo (b) Auf der Treppe

Abbildung 1.1: Hondas Asimo

Ein autonomer Betrieb ist allerdings nicht möglich, die Steuerung erfolgt noch von Menschenhand. Außerdem reicht die integrierte Stromversorgung, trotz der großen Batterien auf dem Rücken, nur für etwa 40 Minuten.

Asimo besteht aus insgesamt 26 Freiheitsgraden. Als Sensoren werden, abgesehen von den Kameras im Kopf, Kreisel und Beschleunigunsmesser im Oberkörper und je ein Kraft-Moment Sensor pro Fuß verwendet. Da Asimo von einem Unternehmen mit wirtschaftlichen Interessen und nicht von einer Forschungseinrichtung entwickelt wurde, stehen leider keine detaillierten Informationen über die Konstruktion zur Verfügung.

Um die Forschung auf dem Gebiet der autonomen Roboter voranzutreiben, wurde der RoboCup[2] ins Leben gerufen. Diese seit 1997 jährlich stattfindende Veranstaltung bietet Forschergruppen aus aller Welt die Gelegenheit, im Fußball ihre Entwicklungen gegeneinander antreten zu lassen. Ziel des RoboCups ist es, bis 2050 eine Mannschaft mit autonomen humanoiden Robotern zu entwickeln, die den aktuellen Fußball Weltmeister schlagen kann. Neben Ligen für radgetriebene Roboter, gibt es auch welche für zweibeinige Laufroboter.

Voraussetzung für die Teilnahme ist die völlige Autonomie der Roboter. Lediglich die Kommunikation zwischen den Robotern eines Teams ist erlaubt. Neben dem eigentlichen Fußballspiel müssen alle Roboter auch noch in verschiedenen Einzeltests ihre Leistung

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

unter Beweis stellen. Diese wechseln jedes Jahr und können beispielsweise Elfmeterschiessen oder das Laufen über einen unebenen Untergrund enthalten. Das Teilnehmerfeld der Humanoiden Liga vom Jahr 2005 ist in Abb. 1.2 zu sehen.

Abbildung 1.2: RoboCup Teilnehmer 2005

## 1.3 Ziele und Übersicht der Arbeit

Am Institut für Robotik der Hochschule Mannheim soll ein zweibeiniger Laufroboter entwickelt werden, der bei einer zukünftigen Teilnahme der Hochschule am RoboCup eingesetzt werden soll. In einer ersten Diplomarbeit [21] wurden bereits die mechanischen Grundlagen erforscht und eine entsprechende Konstruktion entworfen.

Das Ziel dieser Arbeit ist es, aufbauend auf der bereits erstellten Konstruktion, einen Laufalgorithmus zu planen und zu implementieren. Dieser soll als Grundlage für erste praktische Laufversuche mit dem Roboter dienen. Da zuvor noch kein Laufalgorithmus für diesen Roboter entwickelt wurde, müssen auch die kinematischen Eigenschaften der Konstruktion analysiert werden. Die Ergebnisse der Analyse werden verwendet, um ein entsprechendes mathematisches Modell des Roboters zu entwerfen. Für die eigentliche Entwicklung des Laufalgorithmus ist es notwendig, ein Stabilitätskriterium aufzustellen, auf dessen Grundlage das Laufmuster geplant werden kann.

Die Arbeit ist in insgesamt acht Kapitel aufgeteilt. In diesem Kapitel wurde bereits die Motivation, die hinter der Entwicklung von zweibeinigen Laufrobotern steht, erläutert

und ein kurzer Einblick in den aktuellen Stand der Forschung gegeben.
In Kapitel 2 werden grundlegende Konzepte von Laufalgorithmen dargelegt. Der erste Teil beschreibt in der Robotik übliche Stabilitätskriterien. Im zweiten Teil wird erläutert, wie ein Laufmuster geplant und mit Sensorinformationen sinnvoll manipuliert werden kann.
Die mathematischen Grundlagen werden in Kapitel 3 genauer dargestellt. Diese beinhalten zum einen die Interpolation, um aus dem Laufmuster geeignete Zwischenschritte für die Motorsteuerung berechnen zu können, und zum anderen die in der Robotik verbreiteten Methoden der Kinematik, mit deren Hilfe das mathematische Modell eines Roboters entwickelt werden kann.
Kapitel 4 gibt eine kurze Übersicht über die Eigenschaften des Roboters, der in dieser Arbeit verwendet wird.
Die Entwicklung der für den Laufalgorithmus nötigen Konzepte wird in Kapitel 5 beschrieben. Zunächst wird die Kinematik des Roboters modelliert, danach ein geeignetes Stabilitätskriterium ausgewählt und geplant und zuletzt ein parameterisiertes Laufmuster entwickelt.
Die Implementierung der vorgestellten Lösung wird in Kapitel 6 genau beschrieben.
In Kapitel 7 wird der entwickelte Laufalgorithmus bewertet. Hierfür werden die berechneten Daten des Laufmusters auf verschiedene Eigenschaften hin untersucht.
Schließlich enthält das Kapitel 8 eine abschließende Zusammenfassung. Zusätzlich wird hier noch ein kurzer Ausblick auf weiterführende Arbeiten gegeben.

# 2 Grundlegende Konzepte für die Entwicklung von Laufalgorithmen

Grundlage eines jeden Laufalgorithmus bilden vorgegebene Trajektorien für die Beine, auch Laufmuster genannt. Diese können durch zeitabhängige Funktionen oder einfach durch die Angabe mehrerer markanter Punkte beschrieben werden. Wichtig für die Planung ist es, ein Kriterium für die Stabilität zu haben. Der erste Teil dieses Kapitels beschreibt daher mathematische Konzepte für die Stabilität, die in der Entwicklung von Laufrobotern bereits Verwendung finden.
Im zweiten Teil wird gezeigt, wie man ein Laufmuster planen kann und auf welche Eigenschaften man, abgesehen von der Stabilität, noch Wert legen sollte. Die geplanten Laufmuster bieten hierbei zwar eine gute Grundlage, reichen aber meist nicht für die Entwicklung eines effektiven Laufalgorithmus, insbesondere bei zweibeinigen Laufrobotern, aus. Dies kommt zum einen durch Ungenauigkeiten bei der Berechnung der Stabilität. Zum anderen können äußere Einflüsse, wie beispielsweise eine Änderung des Untergrunds oder von außen auf den Roboter einwirkende Kräfte, die geplante Stabilität stören. Daher ist es empfehlenswert, auch während der Ausführung des Laufalgorithmus das Laufmuster ständig an aktuelle Gegebenheiten anzupassen. Dies kann durch verschiedene Sensorinformationen erreicht werden.
Der letzte Teil dieses Kapitels beschreibt mögliche Sensorinformationen. Wichtiger Punkt hierbei ist, wie die Sensorinformationen zur Modifikation des Laufmusters verwendet werden können.

In der Literatur werden Laufalgorithmen oft in zwei Kategorien eingeteilt [10]: statisch stabile und dynamisch stabile Laufalgorithmen. Statisch stabile Laufalgorithmen, auch statische Laufalgorithmen genannt, beziehen in die Stabilitätsanalyse nur statische Kräfte mit ein. Der Roboter wird folglich immer so betrachtet, als wäre er in Ruhe. Um statische Stabilität zu erreichen, genügt es, den Massenschwerpunkt immer über der Standfläche des Roboters zu halten. Der Vorteil liegt hierbei darin, dass der Roboter zu jeder Zeit die Bewegung stoppen kann, ohne umzufallen. Allerdings wird der Einfluss der

dynamischen Kräfte mit steigender Geschwindigkeit schnell so groß, dass es nicht mehr ausreicht, nur statische Kräfte in die Analyse mit einzubeziehen. Dies gilt im Besonderen für zweibeinige Laufroboter, da diese einen sehr hoch gelegenen Schwerpunkt auf einer sehr geringen Standfläche haben.

Bei dynamisch stabilen Laufalgorithmen, auch dynamische Laufalgorithmen genannt, kann der Roboter die statische Stabilität verlassen; der Massenschwerpunkt muss also nicht mehr über der Standfläche liegen. Durch die auftretenden dynamischen Kräfte wird der Roboter dabei am Umfallen gehindert. In diesem Fall führt ein plötzliches Stoppen der Bewegung jedoch zum Umfallen des Roboters, da die dynamischen Kräfte dann wegfallen. Durch das Einbeziehen der dynamischen Kräfte in die Analyse, sind jedoch deutlich höhere Geschwindigkeiten als bei der Benutzung eines statisch stabilen Laufalgorithmus möglich.
Ein dynamisches Laufmuster vor der Laufzeit zu berechnen, würde allerdings nur zu unwesentlich besseren Ergebnissen führen als mit einem statischen Stabilitätskriterium [22], da viele Faktoren - beispielsweise das Spiel in den Motorgetrieben und die auftretenden Schwingungen in den Gelenken - bei der Berechnung nicht beachtet werden können. Sinnvoll ist ein dynamisch stabiler Laufalgorithmus daher nur bei der Verwendung von Sensoren zur Modifikation des Laufmusters.

## 2.1 Konzepte für die Stabilitätsanalyse

Nachfolgend werden die in der Robotik verbreiteten Konzepte für eine Stabilitätsanalyse beschrieben [35][16]. Je nach gewünschter Qualität des Laufalgorithmus und Beinanzahl des Roboters kann es genügen, nur statische Kräfte in der Analyse zu verwenden. Hierzu reicht es, den Massenschwerpunkt und das Stützpolygon des Roboters zu kennen. Bessere Ergebnisse erhält man, wenn auch dynamische Kräfte in die Analyse miteinbezogen werden. Hierfür können die Konzepte des *Zero Moment Point* und *Foot Rotation Indicator* verwendet werden.

### 2.1.1 Stützpolygon und Center of Pressure

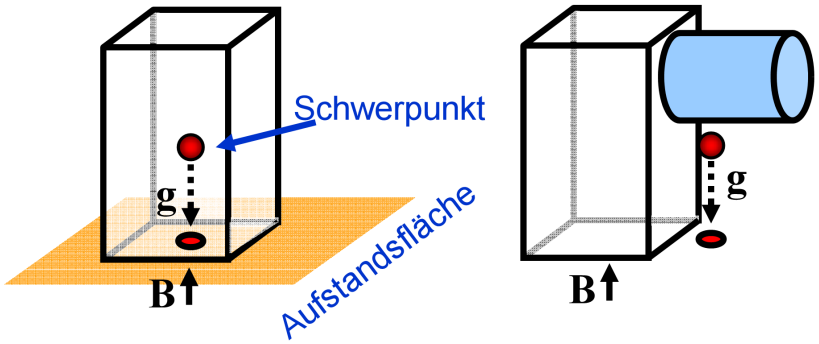
Das Stützpolygon bezeichnet die konvexe Hülle des Polygons, welches von den Kontaktpunkten der Füße aufgespannt wird. Für alle nachfolgend besprochenen Stabilitätsanalysen ist es grundlegend, die genaue Position des Stützpolygons zu kennen. Je größer die

Anzahl der Beine mit Bodenkontakt und je größer die Fläche des Stützpolygons, umso leichter ist es, den Roboter zu stabilisieren.

Innerhalb des Stützpolygons liegt der Center of Pressure (COP). Der COP ist aus der Strömungslehre bekannt. Er bezeichnet den Punkt an einer flachen Oberfläche, die in einer Flüssigkeit treibt, in dem die aufwärtsgerichtete Kraft wirkt. Aus der Position des COP können allerdings nur begrenzt Informationen über die Stabilität des Roboters gewonnen werden. Der COP liegt in jedem Fall innerhalb des Stützpolygons, unabhängig davon ob der Roboter noch stabil ist oder nicht. Ist der Roboter nicht mehr stabil, so liegt der COP auf der Kante des Stützpolygons entlang derer der Roboter umfällt.

### 2.1.2 Massenschwerpunkt und Center of Gravity

Unter dem Massenschwerpunkt, auch Gravitationszentrum genannt, versteht man in der Physik den Punkt, an dem die Schwerkraft wirkt. Wenn in einem Körper die ganze Masse in diesem Punkt vereint wäre, hätte die Schwerkraft die selbe Wirkung auf diesen Körper. Liegt bei einem Körper in Ruhe die vertikale Projektion des Massenschwerpunktes innerhalb der Standfläche, dann ist der Körper statisch stabil (vgl. Abb. 2.1(a)). Im Falle eines Roboters entspricht die Standfläche dem Stützpolygon. Sollte der Massenschwerpunkt allerdings außerhalb liegen, wie in Abb. 2.1(b) zu sehen, wird der Körper instabil und beginnt zu kippen. Der Körper fällt in diesem Fall in Richtung des Massenschwerpunktes.




(a) Stabil (b) Instabil

Abbildung 2.1: Massenschwerpunkt im ruhenden Körper

Der Center of Gravity (COG) bezeichnet die zuvor angesprochene vertikale Projektion des Massenschwerpunktes auf die Bodenfläche. Der COG kann als der Punkt verstanden

werden, an dem der COP sein müsste, um ein Umfallen des Körpers zu verhindern. Mit dieser Definition wird auch offensichtlich, warum der Körper instabil wird, wenn der COG außerhalb des Standfläche liegt.

Um die Stabilität eines Roboters zu analysieren, benötigt man den Massenschwerpunkt des gesamten Roboters. Ein Roboter besteht allerdings meist aus mehreren einzelnen Gelenken. Jedes dieser Gelenke wiederum besitzt eine eigene Masse und somit auch einen eigenen Massenschwerpunkt. Die jeweilige Stellung der einzelnen Gelenke beeinflusst somit den Massenschwerpunkt des gesamten Roboters. Um nun den gesamten Schwerpunkt des Roboters zu bestimmen, müssen die einzelnen Massenschwerpunkte zusammengefasst werden. Die Positionen der einzelnen Massenschwerpunkte werden hierfür nach der Masse gewichtet aufaddiert. Wenn der Roboter aus $i$ einzelnen Gelenken besteht, kann Formel (2.1) verwendet werden [36], um die Position des Massenschwerpunktes $\vec{s}$ zu berechnen. Der Vektor $\vec{s_i}$ enthält die Position des Schwerpunktes und $m_i$ die Masse des Elements $i$.

$$\vec{s} = \frac{\sum_{i=1}^{n} m_i \cdot \vec{s_i}}{\sum_{i=1}^{n} m_i} \tag{2.1}$$

Um den COG zu erhalten, muss weiterhin die vertikale Projektion des Massenschwerpunktes bestimmt werden. Idealerweise sollte der Massenschwerpunkt in einem Koordinatensystem vorliegen, welches eine Achse parallel zur Schwerkraft besitzt. Die Komponente entlang dieser Achse kann nun einfach verworfen werden um den COG zu erhalten.

### 2.1.3 Zero Moment Point

Der Zero Moment Point (ZMP) wurde 1968 von Miomir Vukobratović eingeführt [33]. Der ZMP entspricht dem Punkt im Stützpolygon, an dem die Resultierende aus Schwer- und Trägheitskraft keine Momente entlang horizontaler Achsen hat. Es ist folglich der Punkt, an dem die nach oben gerichtete Kraft im Stützpolygon wirkt. Dies entspricht bei einem dynamisch stabilen Roboter auch der Definition des COP, daher befinden sich die beiden Punkte in diesem Fall auch an der selben Stelle im Stützpolygon.

Einige Veröffentlichungen (beispielsweise [20] und [27]) gehen davon aus, dass der ZMP immer innerhalb des Stützpolygons gehalten werden muss, um dynamische Stabilität zu erreichen. Dies ist allerdings eine ungenaue Aussage, da nach der Definition von Vukobratović der ZMP ohnehin nur innerhalb des Stützpolygons existiert. Ist ein Roboter

nicht mehr dynamisch stabil, wandert der COP, wie bereits oben erwähnt, an den Rand des Stützpolygons. Der ZMP verschwindet komplett, da es nun keinen Kraftpfeil im Stützpolygon mehr gibt, an dem kein Moment entsteht. Stattdessen beschreiben die entsprechende Veröffentlichungen den Foot Rotation Indicator, welcher im Folgenden Kapitel erläutert wird.

Zur Planung und Steuerung eines Laufmusters kann der ZMP gut verwendet werden. Oftmals wird dieser entlang einer vorgegebenen Trajektorie bewegt, wie beispielsweise in [17]. Hier wurde versucht den ZMP immer möglichst in der Mitte des Stützpolygons zu halten, damit eine ausreichende Stabilitätsreserve gegeben ist. Sobald der Roboter jedoch instabil wird, kann aus dem ZMP kein Nutzen mehr gezogen werden.

### 2.1.4 Foot Rotation Indicator

Der Foot Rotation Indicator (FRI) wurde 1999 von Ambarish Goswami definiert[14]. Er stellt einen um dynamische Kräfte erweiterten COG dar. Wie bereits dargestellt bezeichnet der COG den Punkt, an dem der COP sein muss, um lediglich der Schwerkraft entgegenzuwirken. Im Gegensatz hierzu bezeichnet der FRI den Punkt, an dem der COP sein muss, um der Resultierenden aus Schwerkraft und Trägheit entgegenzuwirken.

Der FRI wird oftmals, wie bereits weiter oben angesprochen, fälschlicherweise als ZMP bezeichnet. Tatsächlich sind ZMP und FRI bei einem dynamisch stabilen Roboter gleich. Im Gegensatz zum ZMP jedoch, der bei Verlust der dynamischen Stabilität verschwindet, existiert der FRI auch noch außerhalb des Stützpolygons. Ist der Roboter statisch stabil, liegen sogar COG, ZMP und FRI auf dem selben Punkt.

Solange der FRI innerhalb des Stützpolygons liegt, ist er gleich dem COP. In diesem Fall ist der Roboter dynamisch stabil, da der COP genau dem FRI entgegenwirken kann. Sollte der FRI das Stützpolygon verlassen, entsteht ein Moment im COP, welches den Roboter zum Kippen bringt. In diesem Fall kann der Abstand zwischen COP und FRI als Kriterium für die Instabilität des Roboters verwendet werden. Somit kann der FRI im Gegensatz zum ZMP auch bei einer bereits vorhandenen Instabilität noch nützliche Informationen liefern.

## 2.2 Planung von Laufmustern

Einen Laufalgorithmus für zweibeinige Laufroboter kann man in zwei Phasen aufteilen:

1. Double Support Phase: Beide Beine haben Bodenkontakt. Das Stützpolygon ist entsprechend groß. Diese Phase wird üblicherweise genutzt um das Gewicht von einem auf das andere Bein zu verlagern.

2. Single Support Phase: Nur ein Bein hat Bodenkontakt, das andere befindet sich in einer Vorwärtsbewegung. Die meiste Zeit eines Laufalgorithmus wird von dieser Phase beansprucht.

Eine Ausnahme zu dieser Unterteilung in zwei Phasen bilden Laufalgorithmen zum Rennen. Bei solchen Algorithmen hat zeitweise sogar kein Bein mehr Bodenkontakt. Dies erhöht natürlich die Anforderungen an das Stabilitätskriterium und an den Laufalgorithmus allgemein. Bis jetzt sind nur wenige erfolgreiche Umsetzungen eines frei rennenden Laufroboters bekannt. Ein Beispiel ist Hondas Asimo, welcher bereits in der Einleitung vorgestellt wurde. Dieser erreicht eine Geschwindigkeit beim Rennen von bis zu 6 km/h und kann sogar problemlos im Kreis rennen. Die meisten Arbeiten mit rennenden zweibeinigen Laufrobotern benutzen jedoch noch unterstützende Maßnahmen. In [11] wurde beispielsweise ein rennender Roboter entwickelt, der mit einer Stange seitlich gestützt wird. Durch diese Vereinfachung musste nur ein Fallen nach vorne und hinten beachtete werden, nicht jedoch seitlich. Laufalgorithmen zum Rennen werden aufgrund der Komplexität, die den Rahmen dieser Arbeit sprengen würde, hier nicht weiter betrachtet.

Neben dem Stabilitätskriterium gibt es noch weitere Eigenschaften eines Laufalgorithmus, die beim Planen beachtet werden können bzw. müssen. Die Energieeffizienz ist ein wichtiger Punkt. Viele zweibeinige Laufroboter können zwar bereits stabil Laufen, allerdings nur unter Verwendung von verhältnismäßig viel Energie. Ein gutes Beispiel hierfür ist wieder Asimo. Dieser hat zwar in Bezug auf Stabilität bereits hervorragende Eigenschaften, muss aber trotz der großen Batterien am Rücken schon nach 40 Minuten erneut aufgeladen werden. Ein Ansatz zur Steigerung der Effizienz sind die passiv dynamischen Laufroboter [1]. Diese nutzen den in den dynamischen Kräften gespeicherte Schwung aus, um Energie zu sparen, ähnlich wie dies auch beim Menschen der Fall ist.

Ebenso von Vorteil ist es, bei der Planung des Laufmusters auf möglichst geringe Drehmomente in den Gelenken zu achten. Sollte dies nicht in die Planung miteinbezogen werden, können die Drehmomente eventuell zu groß für die verwendeten Motoren werden. In diesem Fall ist es nicht mehr möglich, das geplante Laufmuster auszuführen.

Wenn hingegen auf möglichst geringe Drehmomente bei der Planung wert gelegt wird, können entsprechend kleinere Motoren gewählt werden. Dies senkt nicht nur die Kosten, sondern verringert auch das Gewicht und steigert somit die Stabilität und Energieeffizienz. Geringere Drehmomente sind auch schonender für Mechanik und Motoren und können so die Lebensdauer der verwendeten Komponenten erhöhen.

Auch auf eine flüssige Bewegung der Beine sollte Wert gelegt werden. Die Punkte für das Laufmuster alleine ergeben oft noch keine flüssige Bewegung. Hier müssen durch Interpolation ausreichend viele Zwischenschritte gebildet werden. Durch ruckhafte Bewegungen können ansonsten Vibrationen entstehen, welche die Sensoren negativ beeinflussen. Insbesondere Sensoren zur Beschleunigungs- und Neigungsmessung sind sehr empfindlich gegen solche Vibrationen. Im schlimmsten Fall könnten sich die Schwingungen mit der Zeit aufaddieren und so den Roboter zum Umfallen bringen.

Besonders beachten muss man die mechanischen Beschränkungen der Roboters. Bedingt durch die Konstruktion der Gelenke und den Operationsradius der Motoren kann es sein, dass manche geplanten Punkte nicht angefahren werden können. Dies wird umso problematischer, wenn solche Punkte erst bei der Interpolation entstehen und somit unbemerkt bleiben können.

Anstatt ein Laufmuster komplett nach oben angegebenen Kriterien neu zu planen, wird oftmals versucht Laufmuster aus der Natur zu übernehmen [15]. Für sechsbeinige Laufroboter war die Stabheuschrecke bereits des Öfteren Vorbild, bei zweibeinigen Laufrobotern ist es der Mensch. Die Laufmuster in der Natur hatten bereits seit Jahrtausenden Zeit, sich optimal zu entwickeln. Insbesondere im Bezug auf Stabilität und Energieeffizienz ist der menschliche Gang weitaus besser als die besten zweibeinigen Laufalgorithmen, die bis heute verfügbar sind. Man kann daher das Laufmuster des Menschen als Grundlage nehmen, und es soweit modifizieren, bis es das gewählte Stabilitätskriterium ausreichend erfüllt.

Nicht zuletzt sollte ein Laufmuster immer so geplant werden, dass es mit Hilfe verschiedener Parameter leicht modifiziert werden kann. Als minimale Anzahl an Parametern sollten Schritthöhe, Schrittweite und Schrittdauer mit in das Laufmuster einfließen. Somit ist es leicht möglich, den Laufalgorithmus mit verschiedenen Parametern zu testen und eventuelle maximale Werte zu finden, die noch ein stabiles Laufen ermöglichen, ohne jedesmal ein neues Laufmuster planen zu müssen. Aber auch für spätere Modifikationen während der Laufzeit sind solche Parameter von Vorteil. Es gibt sogar bereits erste Ansätze für parameterisiertes omnidirektionales Laufen bei zweibeinigen Laufrobotern [6].

Dies ist wichtig, da die maximale Mobilität zweibeinigen Laufens ohne die Fähigkeit, in jede Richtung laufen zu können, nie ausgenutzt werden kann.

## 2.3 Sensorbasierte Modifikation von Laufmustern

Selbst der beste Laufalgorithmus kommt nicht ganz ohne Modifikationen zur Laufzeit aus. Durch viele Faktoren die in der Berechnung des Laufmusters nicht beachtet werden können, beispielsweise ein Rutschen der Füße auf dem Boden oder Spiel im Lager der Motoren, kommt es zu geringen Abweichungen vom geplanten Muster, die sich nach ein paar Schritten schnell aufsummieren können. Dies führt im besten Fall dazu, dass der Roboter ein wenig von der geplanten Richtung abweicht, im schlimmsten Fall können diese Abweichungen zu einer Instabilität des Roboters führen.

Um diese Abweichungen entsprechend ausgleichen zu können, müssen ständig Informationen über den Ist-Zustand des Roboters vorliegen. Hierfür bieten sich eine Reihe von Sensoren an. Nachfolgend werden mögliche Sensoren beschrieben und wie deren Informationen genutzt werden können. Eine gute Übersicht über verschiedene Sensoren bieten auch [10] und [35]. Dabei müssen nicht immer alle Sensorinformationen für ein erfolgreiches Laufen zur Verfügung stehen, oftmals reicht bereits eine kleine Auswahl aus den unten angegebenen Sensoren.

### 2.3.1 Gelenkstellungen

Die Gelenkstellungen bezeichnen die aktuelle Stellung der Motoren in den einzelnen Gelenken. Diese können beispielsweise bei Rotationsgelenken durch die Verwendung von Potentiometern gemessen werden. Moderne Servomotoren haben oftmals schon entsprechende Sensoren zur Positionsmessung eingebaut.

Die aktuellen Gelenkstellungen werden in erster Linie benötigt, um den Massenschwerpunkt bzw. den COG zu berechnen. Mit Hilfe der Gelenkstellungen kann aber auch die Einhaltung eines geplanten Laufmuster überprüft werden. Außerdem können diese zur Hinderniserkennung genutzt werden. Wenn ab einer bestimmten Stelle keine Änderungen mehr in den Stellungen zu messen sind obwohl die Motoren noch angesteuert werden, kann davon ausgegangen werden, dass das Gelenk an ein Hindernis anstößt. Idealerweise sollten Hindernisse aber bereits vor einem Kontakt erkannt werden, daher ist diese Möglichkeit der Hinderniserkennung eher ungeeignet.

### 2.3.2 Neigung und Orientierung

Um die Orientierung eines Roboters entlang einer oder mehrerer Achsen zu bestimmen, stehen eine Reihe verschiedener Sensoren zur Verfügung: Beschleunigungsmesser, Kreisel und Neigungsmesser. Der Beschleunigungsmesser misst die Beschleunigung entlang einer Achse und der Kreisel[1] die Änderung der Lage um ein Achse. Mit beiden kann daher nur die Änderung der Orientierung direkt bestimmt werden. Um die absolute Orientierung zu erhalten, müssen die Messwerte über die Zeit numerisch integriert werden. Hierdurch entstehen Ungenauigkeiten, die sich mit längerer Laufzeit entsprechend verstärken können.

Neigungsmesser hingegen messen die absolute Orientierung um eine Achse. Somit ist auch keine numerische Integration nötig und es entstehen keine Ungenauigkeiten. Neigungsmesser scheinen daher die bessere Wahl zu sein. Allerdings wird bei diesen oft eine Zeitverzögerung beim Messen festgestellt, welche ein ausreichend schnelles Reagieren in manchen Fällen unmöglich macht. Am erfolgsversprechendsten ist daher eine Kombination aus Beschleunigungsmesser oder Kreisel mit einem Neigungsmesser und anschließender Sensordatenfusion.

Alle genannten Sensoren zur Neigungsmessung sind allerdings sehr empfindlich gegenüber Vibrationen. Daher sollte bei Verwendung solcher Sensoren besonders Wert auf eine flüssige Bewegung der Gelenke gelegt werden, um - wie bereits oben erwähnt - die Vibrationen möglichst gering zu halten. Zusätzlich ist es empfehlenswert die Sensorsignale vor der Verwendung entsprechend zu filtern.
Ein völlig anderer Ansatz zur Neigungsmessung kann mit Hilfe von Kameras realisiert werden, genaueres hierzu in Kapitel 2.3.5.

Durch die Neigungsmessung kann in erster Linie festgestellt werden, ob der Roboter beginnt zu fallen. Für die Berechnung des COG kann es ebenfalls notwendig sein, die genaue Neigung zu kennen. Dies ist der Fall, wenn das für die Berechnung des Massenschwerpunktes verwendete Koordinatensystem keine Achse parallel zur Schwerkraft besitzt. Mit der Neigung kann dann die genau Richtung der Schwerkraft bestimmt und somit die vertikale Projektion des Massenschwerpunktes durchgeführt werden.

Die Laufrichtung eines Roboters lässt sich am leichtesten mit Hilfe eines Kompasses ermitteln. Zumindest für Drehungen um die Achse parallel zur Schwerkraft ist dies eine verlässlichere Lösung als die oben angesprochenen Sensoren. Mit dem Wissen über die

[1] Gemeint sind Piezo Kreisel. Mechanische Kreisel können aufgrund der Größe nicht für humanoide Roboter verwendet werden.

Laufrichtung des Roboters kann ein eventuelles Abdriften von einem geplanten Kurs, verursacht durch Rutschen der Füße auf dem Untergrund, verhindert werden. Diese Informationen können zudem auch für eine spätere Navigation des Roboters von Interesse sein.

### 2.3.3 Kraft

Die von unten auf den Fuß wirkenden Kräfte werden meist von Dehnungsmessstreifen, die auf einem Verformkörper aufgebracht sind, ermittelt. Bei entsprechender Konstruktion des Verformkörpers und mit mehreren Dehnungsmessstreifen können alle möglichen Richtungskomponenten in einem Punkt bestimmt werden. Somit ist es auch möglich, die Position des COP bzw des ZMP im Stützpolygon zu ermitteln. Diese Information kann verwendet werden, um mit Hilfe einer Regelung den ZMP immer möglichst nahe in der Mitte des Stützpolygons und dadurch den Roboter möglichst stabil zu halten. Sobald der Roboter allerdings instabil wird, kann aus dieser Methode, wie bereits erwähnt, kein weiterer Nutzen gezogen werden.

### 2.3.4 Momente

Die im Roboter wirkenden Momente können mit Hilfe von Kraft-Moment Sensoren ermittelt werden. Ein Kraft-Moment Sensor misst alle an diesem Punkt wirkenden Kräfte und Momente. Wenn solch ein Sensor entsprechend tief am Fuß angebracht ist, können die Momente, die auf den gesamten Roboter wirken, bestimmt werden, solange man von der Vernachlässigbarkeit der Füße ausgeht. Aus den wirkenden Momenten und Kräften kann der FRI berechnet, wie in [20] gezeigt, und zur dynamischen Stabilisierung des Roboters verwendet werden.

### 2.3.5 Kamera

Eine Kamera ist der wohl am vielseitigsten verwendbare Sensor. Gleichzeitig ist dieser für autonome Laufroboter, die sich selbstständig in einer fremden Umgebung orientieren sollen, unerlässlich. Lange Zeit war es allerdings nicht möglich, eine Kamera zur Steuerung zu verwenden. Insbesondere bei kleineren zweibeinigen Laufrobotern, stand für die Algorithmen zur Bildverarbeitung nicht genug Rechenleistung zur Verfügung. Die heut-

zutage drastische gestiegene Rechenleistung in Kleincomputern, wie beispielsweise bei PDAs oder Mobiltelefonen, macht allerdings auch eine Nutzung von Kameras möglich.

In ersten Linie werden Kameras dazu verwendet, ein genaueres Bild von der Umgebung zu bekommen und so eventuelle Hindernisse oder Ziele zu erkennen. Wie bereits oben erwähnt kann eine Kamera auch dazu genutzt werden, die Neigung und Orientierung des Roboters zu ermitteln. In [35] wurde dies bereits erfolgreich getan. Durch die beschränkte Rechenleistung musste hierfür allerdings vor dem Roboter eine weiße Wand mit schwarzem Strich positioniert werden, um die Algorithmen möglichst einfach halten zu können. Mit einer höheren verfügbaren Rechenleistung wäre es allerdings auch denkbar, beliebige Punkte in der Umgebung zur Neigungsbestimmung zu nutzen.

# 3 Mathematische Grundlagen für die Beinsteuerung

Für die Beschreibung der Beintrajektorien werden oft nur einige markante Punkte im Raum angegeben, die den groben Verlauf der Beine aufzeigen. Diese Punkte müssen noch interpoliert werden, um ausreichend Zwischenschritte für eine flüssige Beinbewegung zu erhalten. Die erste Hälfte dieses Kapitels beschäftigt sich daher mit den in der Robotik üblichen Interpolationsverfahren.

Die aus der Interpolation gewonnenen Punkte liegen immer noch im kartesischen Raum vor. Um die gewünschte Bewegung der Beine zu erhalten, müssen die entsprechenden Stellungen der Gelenke bekannt sein. Daher muss für die Beinsteuerung eine Beziehung zwischen Punkten im kartesischen Raum und den Stellungen der Gelenke hergestellt werden. Dies kann mit einer kinematischen Analyse geschehen, welche in der zweiten Hälfte diese Kapitels erläutert wird.

Die angesprochenen Verfahren haben sich bereits bei vielen Robotern bewährt. Dabei werden sie nicht nur für die Entwicklung von Laufrobotern angewendet, sondern finden bereits seit langem auch in Industrierobotern Anwendung: Die Arme von Industrierobotern bestehen, ebenso wie die Beine von Laufrobotern, aus kinematischen Ketten. Somit sind die Anforderungen an die Steuerung in beiden Fällen ähnlich.

## 3.1 Interpolation

Bei Industrierobotern wird eine Interpolation meist in zwei Stufen durchgeführt [34]: In der ersten Interpolationsstufe werden aus den Punkten im kartesischen Raum, welche die grobe Trajektorie beschreiben, ausreichend Zwischenschritte berechnet. Dies dient zum einen dazu, eine flüssige Bewegung zu erzeugen. Zum anderen sollen hierdurch ruckartige Beschleunigungs- oder Geschwindigkeitsänderungen verhindert werden, um eine unnötige Belastung der Mechanik zu vermeiden. Bei Laufrobotern können solche ruckartigen

Bewegungen zusätzlich die Sensoren stören oder zu Instabilität führen (vgl. Kapitel 2). Desweiteren sollen aus der Interpolation Geschwindigkeiten abgeleitet werden; die Informationen über die Geschwindigkeiten in einem Punkt können hilfreich für die spätere Motorregelung sein.

Die zweite Stufe der Interpolation, auch Feininterpolation genannt, führt eine Interpolation auf Ebene der Gelenke durch. Die Feininterpolation ist meist in der Motorregelung implementiert und dient dazu, Motoreigenschaften wie Beschleunigungs- und Verzögerungsphase mit in die Regelung einzubeziehen. Inhalt dieser Arbeit ist nur die erste Stufe der Interpolation, daher werden die Besonderheiten der zweiten Stufe nicht weiter erläutert. Die vorgestellten Interpolationsverfahren finden allerdings auch in der Feininterpolation Anwendung.

Das bekannteste Interpolationsverfahren, das aus einer Anzahl von Stützpunkten eine stetige Kurve generiert, ist die Polynominterpolation bzw. Lagrange-Interpolation. Hierbei wird für $n$ Stützpunkte ein Polygon vom Grad $n-1$ entwickelt. Dieses Verfahren hat allerdings den entscheidenden Nachteil, dass bei höherem Grad des Polynoms - also bei einer größeren Anzahl verwendeter Stützpunkte - die Kurve beginnt zu oszillieren [34]. Dies führt bereits bei kleinen Änderungen in den Stützpunkten zu großen Ausschlägen der Kurve (Abb. 3.1). In der Praxis haben daher Polynome mit einem Grad größer fünf kaum Bedeutung für die Interpolation.

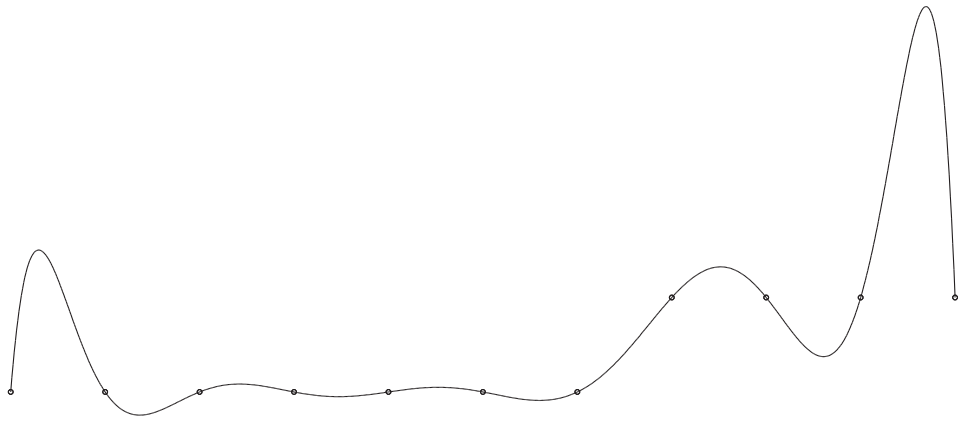


Abbildung 3.1: Lagrange-Interpolation

Ein weiteres für die Interpolation verwendetes Verfahren sind Bézier Kurven [25]. Hierbei entstehen auch bei einer größeren Anzahl an Stützpunkten keine ungewollten Oszillationen. Allerdings bieten diese nur wenig lokale Kontrolle über die Kurve: Änderung an einem Stützpunkt wirken sich auf die ganze Kurve aus. Desweiteren benötigt die In-

terpolation mit Bézier Kurven einen höheren Rechenaufwand im Vergleich zu anderen Verfahren. Aus diesen Gründen sind auch Bézier Kurven für die Interpolation im Bereich der Robotik eher ungeeignet.

Zur Interpolation von Trajektorien am besten geeignet ist die Spline-Interpolation [34]. Bei diesem Verfahren entstehen keine ungewollten Oszillationen in der Kurve und Änderungen an einem Stützpunkt wirken sich nur auf die nähere Umgebung aus. Abb. 3.2 zeigt eine Spline-Interpolation der selben Stützpunkte wie in oben gezeigter Lagrange-Interpolation (Abb. 3.1).

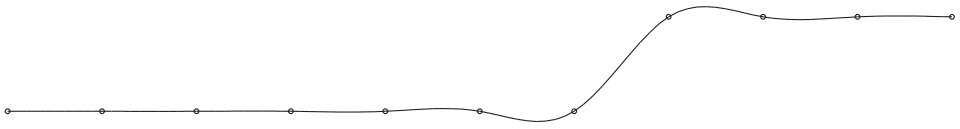


Abbildung 3.2: Spline-Interpolation

Der Begriff Spline kommt aus dem Englischen und bezeichnet dort eine biegsame Latte, die früher im Schiffsbau verwendet wurde. Um Zeichnungen zu erstellen wurde die Latte auf einem Brett entlang einer Reihe von Stütznägeln aufgespannt. Die entstehende Form hat die Eigenschaft, die Biegeenergie der Latte zu minimieren. Übertragen auf eine Kurve bedeutet dies, dass eine Kurve mit minimaler und stetiger Steigung entsteht. Somit treten keine ruckartigen Beschleunigungs- oder Geschwindigkeitsänderungen auf. Spline-Kurven besitzen annähernd die gleichen Eigenschaften wie zuvor erwähnte Latte.

Für die Erzeugung einer Spline-Kurve gibt es verschiedene Ansätze. Nachfolgend werden kubische Polynomsplines und B-Splines genauer erläutert. Weiterführende Informationen sind in [34][25][3] zu finden.

### 3.1.1 Kubische Polynomsplines

Bei Polynomsplines wird für jedes Intervall zwischen zwei Stützpunkten ein Polynom gebildet. Sind diese Polynome vom dritten Grad, so spricht man von kubischen Polynomsplines. Für die Herleitung der Polynomsplines ist es notwendig, dass die zu interpolierenden Punkte $P_i$ in parametrischer Darstellung vorliegen, was bedeutet, dass jedem Punkt ein eindeutiger Parameter zugeordnet sein muss. Für die Interpolation von Beintrajektorien bietet es sich an, die Zeit $t_i$ als Parameter zu wählen. Für die Spline-Kurve $S_i$ im Intervall $i$ ergibt sich somit folgende Definition des Polynoms:

$$S_i(t) = A_i + B_i(t - t_i) + C_i(t - t_i)^2 + D_i(t - t_i)^3 \tag{3.1}$$

Mit Hilfe der Stützpunkte $P_i = (x_i, y_i, z_i)^T$ werden die Koeffizientenvektoren $A_i$, $B_i$, $C_i$ und $D_i$ bestimmt werden. Bei $n$ Stützpunkten ergeben sich $n-1$ Intervalle und somit $n-1$ Polynome. Jedes Polynom besitzt vier Koeffizienten, folglich müssen insgesamt $4 \cdot (n-1)$ Bedingungen aufgestellt werden, um das Gleichungssystem lösen zu können.

Aus der Anforderung, dass die Kurve durch die Stützpunkte laufen soll, ergeben sich die ersten beiden Bedingungen für das Polygon:

$$S_i(t_i) = P_i \tag{3.2}$$

$$S_i(t_{i+1}) = P_{i+1} \tag{3.3}$$

Dies stellt sicher, dass sich die Kurven in den Stützpunkten berühren. Um auch Sprünge in der Beschleunigung und Geschwindigkeit ausschließen zu können, muss zusätzlich die Stetigkeit der ersten beiden Ableitungen der Spline-Kurve in den Stützpunkten sichergestellt werden. Dies kann mit folgenden Bedingungen erreicht werden:

$$S_i'(t_i) = S_{i-1}'(t_i) \tag{3.4}$$

$$S_i''(t_i) = S_{i-1}''(t_i) \tag{3.5}$$

Die Bedingungen (3.4) und (3.5) sind nicht anwendbar für das erste Intervall, daher bleiben zwei freie Bedingungen. Häufig werden diese genutzt, um in den Randpunkten die Krümmung Null zu setzen. Somit werden Sprünge in der Beschleunigung an dieser Stelle verhindert. Splines mit einer solchen Bedingung werden auch als natürliche kubische Splines bezeichnet.

Die Stützpunkte $P_i$ können aus einer beliebigen Anzahl Komponenten bestehen. Es müssen allerdings für jede Komponente eigene Koeffizienten bestimmt werden. Nachfolgend wird die Bestimmung der Koeffizienten am Beispiel der Komponente $x$ dargestellt. Das entsprechende Polynom lautet:

$$x_i(t) = a_{x,i} + b_{x,i}(t - t_i) + c_{x,i}(t - t_i)^2 + d_{x,i}(t - t_i)^3 \tag{3.6}$$

Aus $t = t_i$ ergibt sich:

$$a_{x,i} = x_i \tag{3.7}$$

Nach der Bedingung für natürliche Polynomsplines muss die Krümmung in den Randpunkten Null sein. Aus der zweiten Ableitung des Polynoms ergibt sich daher:

$$c_{x,0} = c_{x,n} = 0 \tag{3.8}$$

Für die restlichen Koeffizienten gelten folgende Gleichungen[1]:

$$h_{i-1}c_{x,i-1} + 2c_{x,i}(h_{i-1} + h_i) + h_i c_{x,i+1} = \frac{3}{h_i}(a_{x,i+1} - a_{x,i}) - \frac{3}{h_{i-1}}(a_{x,i} - a_{x,i-1}) \tag{3.9}$$

$$b_{x,i} = \frac{1}{h_i}(a_{x,i+1} - a_{x,i}) - \frac{h_i}{3}(c_{x,i+1} + 2c_{x,i}) \tag{3.10}$$

$$d_{x,i} = \frac{1}{3h_i}(c_{x,i+1} - c_{x,i}) \tag{3.11}$$

$$\text{Mit: } h_i = t_{i+1} - t_i \tag{3.12}$$

Gleichung (3.9) lässt sich als tridiagonales Gleichungssystem darstellen. Dieses kann mit dem Gaußschen Eliminationsverfahren eindeutig gelöst werden. Nachdem alle Koeffizienten bestimmt wurden, stehen auch alle Polynome für die Intervalle fest. Um den Wert von $x$ an einer beliebigen Stelle $t_s$ zu erhalten, muss zuerst das Intervall gesucht werden, in dem der Punkt $t_s$ liegt. Danach kann der Wert mit dem entsprechenden Polynom bestimmt werden.

Die Geschwindigkeiten können mit der Ableitung des allgemeinen Polynoms bestimmt werden, hierfür ist es nicht nötig neue Koeffizienten zu berechnen. Gleichung (3.13) zeigt die Ableitung für die Komponente $x$.

$$\dot{x_i}(t) = b_{x,i} + 2c_{x,i}(t - t_i) + 3d_{x,i}(t - t_i)^2 \tag{3.13}$$

Die so gewonnene Spline-Kurve geht durch alle angegebenen Stützpunkte. Sie weist kaum ungewollten Oszillationen auf und ist zweimal stetig differenzierbar. Somit sind auch Sprünge in der Geschwindigkeit oder Beschleunigung ausgeschlossen. Jedoch entsteht durch die Lösung des tridiagonalen Gleichungssystem ein relativ hoher Rechenaufwand, insbesondere bei einer größeren Anzahl an Stützpunkten.

### 3.1.2 B-Splines

B-Splines sind eine weitere Möglichkeit zur Interpolation von Trajektorien [30]. Genaugenommen bieten B-Splines allerdings keine Interpolation, sondern nur eine Approximation; eine B-Spline Kurve durchfährt die Stützpunkte nicht direkt, stattdessen wird die Form der Kurve nur durch die Stützpunkte beeinflusst (Abb. 3.3). Hierdurch ergibt

[1] In [26] ist die genaue Herleitung beschrieben. Eine entsprechende Ausführung an dieser Stelle würde den inhaltlichen Rahmen der Arbeit sprengen

sich auch, das die B-Spline Kurve immer innerhalb der konvexen Hülle der Stützpunkte bleibt.

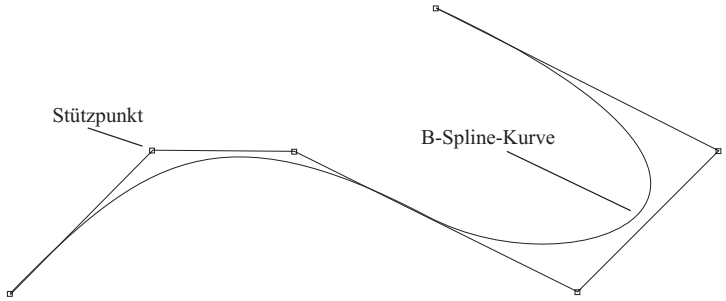



Abbildung 3.3: B-Spline Kurve

B-Splines beruhen auf Basisfunktionen, welche die Gewichtung der Stützpunkte an einer gegebenen Stelle der Kurve bestimmen. Da die Basisfunktionen nur auf einem kleinen Intervall ungleich Null sind, werden ausschließlich Punkte in der näheren Umgebung für die Auswertung der Kurve benutzt. Die bekanntesten Basisfunktionen für B-Splines können mit der Rekursionsformel (3.14) bestimmt werden.

$$\begin{aligned} N_i^0(u) &= \begin{cases} 1 & \text{für } u \in [u_i, u_{i+1}) \\ 0 & \text{sonst} \end{cases} \\ N_i^n(u) &= \frac{u - u_i}{u_{i+n} - u_i} N_i^{n-1}(u) + \frac{u_{i+n+1} - u}{u_{i+n+1} - u_{i+1}} N_{i+1}^{n-1}(u) \end{aligned} \tag{3.14}$$

Um die Funktionen zu bestimmen, ist der Parametervektor $U = (u_0, u_1, \ldots)^T$ notwendig. Im Gegensatz zu den Polynomsplines besitzen diese Parameter jedoch keine eindeutige Bindung an die Stützpunkte.
$n$ entspricht dem Grad der Basisfunktion. Auch hier werden meist kubische Polynome verwendet ($n = 3$). Die dabei entstehenden Kurven sind ebenso zweimal stetig differenzierbar.
Bei der Berechnung kann der Nenner der Brüche Null werden. In diesem Fall wird aber auch der Zähler Null, der ganze Term wird dann mit Null angenommen.

Zum Bestimmen der Kurve an einer gegebenen Stelle $u$ kann Formel (3.15) verwendet werden. $m$ entspricht der Anzahl an Stützpunkten.

$$S_i(u) = \sum_{i=0}^{m-1} P_i N_i^n(u) \tag{3.15}$$

Wie bereits oben erwähnt besitzt der Parametervektor $U$ keine eindeutige Bindung an die Stützstellen, daher ist es nicht möglich die Zeit als Parameter zu verwenden. Stattdessen

wird sie in die Stützpunkte miteinbezogen und mit interpoliert. Dies erzeugt allerdings zwei Probleme: Zum einen können keine Punkte aus der Kurve mit konstantem Zeitabstand gewonnen werden. Für die Feininterpolation in der Motorregelung kann es aber notwendig sein, dass die Punkte der Trajektorie mit konstanten Zeitabständen vorliegen. Zum anderen ist es nicht mehr möglich, eine Ableitung nach der Zeit zu bilden und so die Geschwindigkeiten zu ermitteln. Es ist zwar prinzipiell möglich eine B-Spline Kurve abzuleiten, allerdings nur nach dem Parameter $U$. Ohne eine Bindung an die Stützpunkte können aus dieser Ableitung aber keine nützlichen Informationen gewonnen werden.

## 3.2 Beschreibung kinematischer Ketten

Eine kinematische Kette besteht aus mehreren starren Körpern, die jeweils über ein Gelenk miteinander verbunden sind (Abb. 3.4).

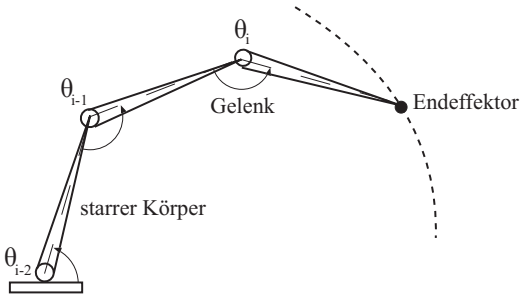
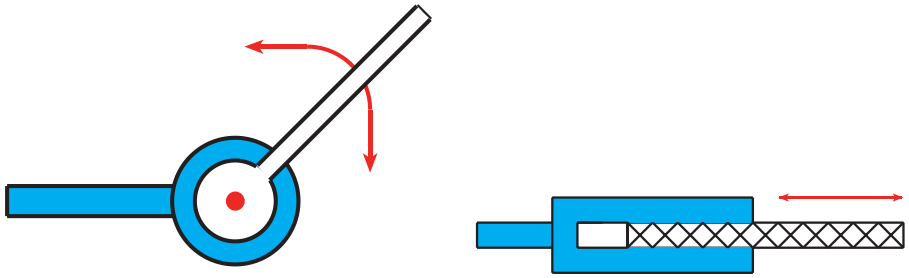



Abbildung 3.4: Kinematische Kette

Die einzelnen Körper können über Rotations- oder Translationsgelenke untereinander verbunden sein (Abb. 3.5).



(a) Rotationsgelenk (b) Translationsgelenk

Abbildung 3.5: Gelenktypen

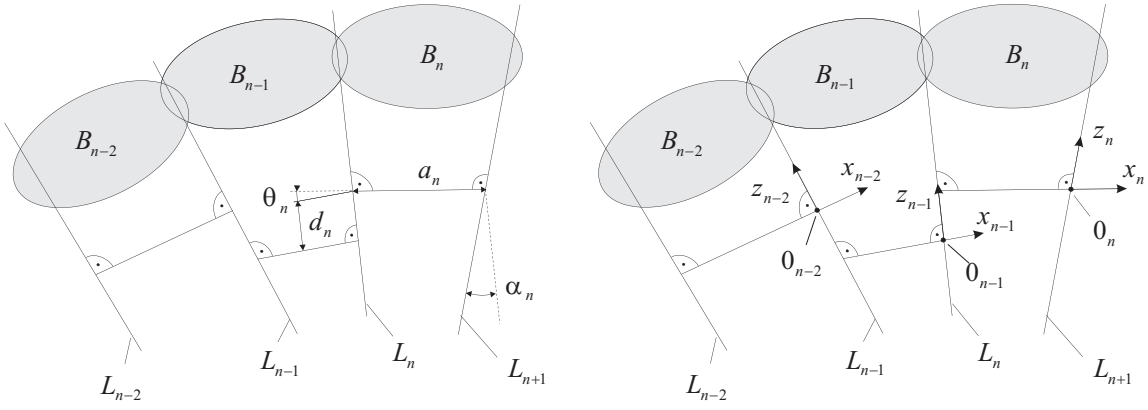
Die Anzahl der Gelenke entspricht der Anzahl der Freiheitsgrade, welche die kinematische Kette besitzt. Bei einem Laufroboter bildet jedes einzelne Bein für sich alleine eine kinematische Kette.

In diesem Abschnitt werden verschiedene Methoden der Kinematik beschrieben. Die Kinematik beschreibt das Verhältnis zwischen den einzelnen Stellungen der Gelenke und der Position im kartesischen Raum des letzten Elements einer kinematischen Kette, auch Endeffektor genannt. Bei den Beinen eines Laufroboters ist der Endeffektor der Fuß.
Mit Hilfe der *direkten Kinematik* kann, bei einer vorgegebenen Stellungen der Gelenke, die Position und Orientierung des Endeffektors im Raum bestimmt werden.
Die *inverse Kinematik* hingegen versucht für eine gesuchte Position und Orientierung des Endeffektors die nötigen Gelenkstellungen zu bestimmen.

Die im Folgenden beschriebenen Methoden wurden bereits in anderer Literatur ausführlich erläutert: [32][31][36][4].

### 3.2.1 Denavit-Hartenberg Verfahren

Denavit und Hartenberg entwickelten ein Verfahren, um kinematische Ketten zu beschreiben [12]. Dieses Verfahren basiert auf der Grundidee, dass die gegenseitige Lage zweier Elemente zueinander durch vier Parameter ausreichend beschrieben werden kann. Man bezeichnet diese auch als DH Parameter.




(a) Parameter (b) Koordinatensysteme

Abbildung 3.6: Denavit-Hartenberg Verfahren

### Parameter

Abb. 3.6(a) zeigt eine kinematische Kette mit drei Elementen $B_{n-2}$ bis $B_n$. Für das Element $S_n$ können die Parameter wie folgt bestimmt werden: Zwischen zwei Achsen - $L_n$ und $L_{n+1}$ - wird eine gemeinsame Normale gesucht. Die Länge dieser Normalen entspricht dem ersten Parameter $a_n$.
Der Versatz zwischen dieser Normalen und der vorhergehenden ergibt den zweiten Parameter $d_n$.
Der dritte Parameter entspricht dem Winkel $\alpha_n$ um den die beiden Achsen $L_n$ und $L_{n+1}$ zueinander verdreht sind.
Der letzte Parameter ergibt sich aus der Drehung um die Achse $L_n$ und wird mit $\theta_n$ bezeichnet.
Drei der vier Parameter bleiben gleich, solange die Konstruktion des Roboters gleich bleibt. Der vierte Parameter, nachfolgend Gelenkvariable genannt, entspricht der Stellung des Gelenks. Welcher Parameter variabel ist, hängt vom Gelenktyp ab. Bei einem Rotationsgelenk (Abb. 3.5(a)) ist dies $\theta_n$, bei einem Translationsgelenk (Abb. 3.5(b)) $d_n$.

### Koordinatensysteme

Zusätzlich zu den Parametern muss in jedem Gelenk ein eigenes Koordinatensystem definiert werden. Die Koordinatensysteme werden zum einen benötigt, um die Transformationsgleichungen aufstellen zu können. Zum anderen sind diese auch wichtig, um später Eigenschaften der einzelnen Elemente wie Massenschwerpunkt oder Trägheitseigenschaften zusammenzufassen.

Abb. 3.6(b) zeigt eine Möglichkeit, wie die Position und Orientierung der Koordinatensysteme bestimmt werden kann. Der Ursprung liegt dabei immer in der Kreuzung der Achse $L_{n+1}$ und der gemeinsamen Normalen zwischen $L_n$ und $L_{n+1}$. Wenn sich $L_n$ und $L_{n+1}$ in einem Punkt schneiden, ist dies der Ursprung. Die Achse $x_n$ hat die selbe Orientierung wie die Normale $a_n$. Die Achse $z_n$ wird auf die Achse $L_{n+1}$ gelegt. Das Koordinatensystem wird mit der Achse $y_n$ so vervollständigt, dass sich ein rechtsdrehendes Koordinatensystem ergibt. Im Prinzip kann für das Koordinatensystem auch eine andere Orientierung gewählt werden. Um die Transformationsgleichungen später aufstellen zu können, ist es allerdings notwendig, dass immer eine Achse des Koordinatensystems auf der Gelenkachse und eine zweite Achse auf der Normalen liegt.

In der praktischen Anwendung des Denavit-Hartenberg Verfahrens ergeben sich oft einige Vereinfachungen [36]. Bei parallel oder senkrecht aufeinander stehenden Achsen werden bestimmte Parameter 0, wodurch die spätere Transformation vereinfacht wird. Allerdings entstehen dadurch auch Probleme im Bezug auf die Lage der Koordinatensysteme: Durch diese Vereinfachungen ist das DH Verfahren nicht mehr eindeutig. Wenn beispielsweise zwei Achsen parallel sind, existieren unendlich viele Normalen und somit auch unendlich viele Positionen für das Koordinatensystem. In diesem Fall ist es egal, welcher Ursprung gewählt wird. Allerdings ist es empfehlenswert, den Ursprung so zu wählen, dass der Versatz $d_n$ zu der Normalen des vorherigen oder folgenden Gelenkes wenn möglich 0 wird, wodurch die Transformationsgleichungen weiter vereinfacht werden.

### 3.2.2 Direkte Kinematik

Mit Hilfe der direkten Kinematik ist es möglich - bei gegebenen Stellungen der Gelenke - die Position und Orientierung des Endeffektors zu berechnen (Abb. 3.7).

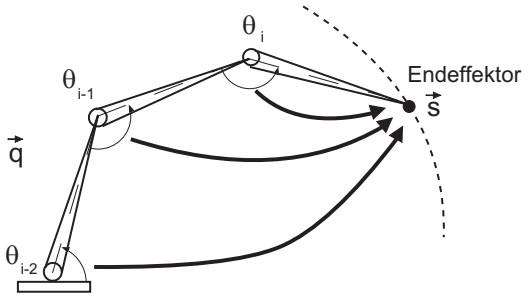



Abbildung 3.7: Direkte Kinematik

Wenn der Vektor $\vec{s} = (x, y, z, \alpha, \beta, \gamma)^T$ der Position und Orientierung des Endeffektors entspricht und der Vektor $\vec{q} = (q_1, ..., q_n)^T$, mit $n$ = Anzahl der Gelenke, die Gelenkvariablen der einzelnen Gelenke enthält, dann kann die direkte Kinematik mit der Formel (3.16) ausgedrückt werden.

$$\vec{s} = f(\vec{q}) \tag{3.16}$$

$f$ ist eine nichtlineare Funktion. Bei einer linearen Änderung der Gelenkvariablen ergibt sich somit keine lineare Änderung der Position des Endeffektors.

Für die Lösung der direkten Kinematik wird eine homogene Transformationsmatrix benötigt, die den Übergang vom Ausgangskoordinatensystem in das Koordinatensystem des Endeffektors, in Abhängigkeit zu den Gelenkvariablen, beschreibt. Dazu werden zu-

nächst die homogenen Transformationsmatrizen für den Übergang zwischen den einzelnen Gelenken ermittelt. Dies ist mit Hilfe der im vorhergehenden Kapitel vorgestellten DH Parametern und der Transformationsgleichung (3.17) möglich.

$$ {}^{n-1}_{n}T = Rot(z_{n-1}, \theta_n) Trans(0, 0, d_n) Trans(a_n, 0, 0) Rot(x_n, \alpha_n) \tag{3.17} $$

Die Transformationsgleichung entspricht den folgenden Einzeltransformationen:

1. Rotation um die Achse $z_{n-1}$ um den Winkel $\theta_n$.
2. Verschiebung entlang der Achse $z_{n-1}$ um $d_n$.
3. Verschiebung entlang der Achse $x_n$ um $a_n$.
4. Rotation um die Achse $x_n$ um den Winkel $\alpha_n$.

Bei der Transformationsgleichung ist zu beachten, dass die Achsen, um die rotiert oder entlang der verschoben wird, direkt mit der Wahl der Koordinatensysteme zusammenhängen. Wie bereits im vorhergehenden Kapitel erwähnt, muss je eine Achse auf der Gelenkachse und eine auf der Normalen liegen, damit die Transformationsgleichung angewendet werden kann. Für die ersten beiden Einzeltransformationen wird die Achse verwendet, die auf der Gelenkachse liegt, während für die letzten beiden Einzeltransformationen die Achse gewählt wird, die auf der Normalen liegt. Die Transformationsgleichung (3.17) ist gültig, wenn die Koordinatensysteme wie im vorhergehenden Kapitel vorgeschlagen bestimmt wurden.

Wenn die in Gleichung (3.17) enthaltenen Einzeltransformationen durch die entsprechenden homogenen Transformationsmatrizen ersetzt und diese dann miteinander multipliziert werden, ergibt sich die Transformationsmatrix (3.18).

$$
\begin{aligned}
{}^{n-1}_{n}T &= \begin{pmatrix} \cos\theta_n & -\sin\theta_n & 0 & 0 \\ \sin\theta_n & \cos\theta_n & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & a_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_n & -\sin\alpha_n & 0 \\ 0 & \sin\alpha_n & \cos\alpha_n & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
{}^{n-1}_{n}T &= \begin{pmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & a_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & a_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned}
\tag{3.18}
$$

Die erhaltene Transformationsmatrix beschreibt den Übergang vom Koordinatensystem des Gelenks $n-1$ in das Koordinatensystem des Gelenks $n$ in Abhängigkeit der DH

Parameter. Um die Transformationsmatrix für die gesamte Kinematische Kette vom Anfangskoordinatensystem bis zum Endeffektor zu erhalten, müssen alle Einzeltransformationen miteinander multipliziert werden:

$$ {}^{0}_{m}T = {}^{0}_{1}T\,{}^{1}_{2}T \ldots {}^{m-1}_{m}T \tag{3.19} $$

Mit Hilfe dieser Transformationsmatrix kann nun ein Punkt im Koordinatensystem des Endeffektors in das Anfangskoordinatensystem transformiert werden. Es ist allerdings auch möglich, Orientierung und Position des Endeffektors relativ zum Anfangskoordinatensystem aus der Transformationsmatrix zu extrahieren. Dazu kann die homogene Transformationsmatrix in Teilmatrizen aufgeteilt werden.

$$ {}^{0}_{m}T = \begin{pmatrix} {}^{0}_{m}R & {}^{0}\vec{r} \\ 0 & 1 \end{pmatrix} \tag{3.20} $$

${}^{n}_{m}R$ ist eine 3x3 Rotationsmatrix, welche die Orientierung der beiden Koordinatensysteme zueinander beschreibt. Der Vektor ${}^{0}\vec{r}$ gibt die Position des Koordinatenursprungs im Endeffektor relativ zum Anfangskoordinatensystem an. Aus diesem Vektor kann somit die Position des Endeffektors abgelesen werden. Um die Orientierung aus der Rotationsmatrix zu extrahieren, muss die Matrix noch entsprechend aufgelöst werden.

Nachdem die allgemeinen Rotationsmatrizen für die drei Winkel $\alpha, \beta, \gamma$ miteinander multipliziert wurden, erhält man folgende Rotationsmatrix:

$$ {}^{0}_{m}R = Rot(z_0, \gamma) Rot(y_0, \beta) Rot(x_0, \alpha) $$

$$ {}^{0}_{m}R = \begin{pmatrix} \cos\gamma\cos\beta & \cos\gamma\sin\beta\sin\alpha - \sin\gamma\cos\alpha & \cos\gamma\sin\beta\cos\alpha + \sin\gamma\sin\alpha \\ \sin\gamma\cos\beta & \sin\gamma\sin\beta\sin\alpha + \cos\gamma\cos\alpha & \sin\gamma\sin\beta\cos\alpha - \cos\gamma\sin\alpha \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha \end{pmatrix} $$

Aus den neun Elementen der Matrix kann ein Gleichungssystem mit neun Gleichungen und drei Unbekannten aufgebaut werden:

$$ \begin{aligned} a_{11} &= \cos\gamma\cos\beta & a_{32} &= \cos\beta\sin\alpha \\ a_{21} &= \sin\gamma\cos\beta & a_{13} &= \cos\gamma\sin\beta\cos\alpha + \sin\gamma\sin\alpha \\ a_{31} &= -\sin\beta & a_{23} &= \sin\gamma\sin\beta\cos\alpha - \cos\gamma\sin\alpha \\ a_{12} &= \cos\gamma\sin\beta\sin\alpha - \sin\gamma\cos\alpha & a_{33} &= \cos\beta\cos\alpha \\ a_{22} &= \sin\gamma\sin\beta\sin\alpha + \cos\gamma\cos\alpha \end{aligned} $$

Nachdem die Gleichungen entsprechend umgestellt und nach den Winkeln aufgelöst wurden, erhält man die Gleichungen (3.21), (3.22) und (3.23).

$$\gamma = \arctan(a_{21}/a_{11}) \tag{3.21}$$

$$\beta = \arctan(-a_{31}/(a_{11}\cos\gamma + a_{21}\sin\gamma)) \tag{3.22}$$

$$\alpha = \arctan(a_{32}/a_{33}) \tag{3.23}$$

Das genaue Vorgehen ist im Buch „Introduction to Robotics“ [32] beschrieben. Es sind allerdings auch andere Lösungswege für das Gleichungssystem möglich.

Bei der Implementierung sollte beachtet werden, dass die Terme in den Nennern 0 werden können. Es empfiehlt sich daher die Verwendung der Funktion *atan2*, welche auch solche Fälle behandeln kann.

### 3.2.3 Inverse Kinematik

Im Gegensatz zu der direkten Kinematik versucht die inverse Kinematik, für eine gegebene Position des Endeffektors mögliche Stellungen der Gelenke zu finden (Abb. 3.8).

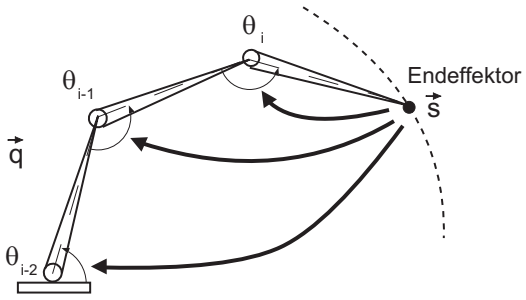



Abbildung 3.8: Inverse Kinematik

Entsprechend der Funktion für die direkte Kinematik (3.16) ist die Suche nach einer Lösung für das Problem der inversen Kinematik gleichbedeutend mit der Suche nach der inversen Funktion (3.24).

$$\vec{q} = f^{-1}(\vec{s}) \tag{3.24}$$

Die meisten Positionen eines Endeffektors können durch unterschiedliche Gelenkstellungen angefahren werden (Abb. 3.9). Die inverse Kinematik besitzt demzufolge in den meisten Fällen mehrere Lösungen. Einige dieser Lösungen sind allerdings aufgrund mechanischer Beschränkungen der kinematischen Kette oder aufgrund von Hindernissen im Arbeitsbereich nicht verwendbar.

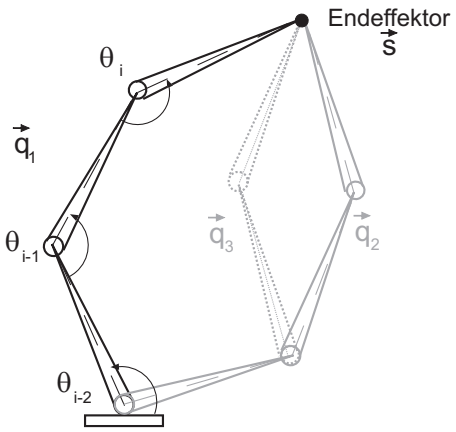



Abbildung 3.9: Lösungen für inverse Kinematik

Es gibt schließlich auch eine unendlich große Menge an Positionen, für die es überhaupt keine Lösung gibt. Jede kinematische Kette besitzt einen begrenzten Arbeitsraum. Der Arbeitsraum enthält die Menge an Positionen, die mit dem Endeffektor noch angefahren werden können. Positionen außerhalb des Arbeitsraums besitzen somit auch keine Lösung der inversen Kinematik. Ein Spezialfall stellt der Rand des Arbeitsraums dar. Positionen nahe am oder direkt auf dem Rand können zwar angefahren werden, allerdings ist es dann nicht mehr in allen Fällen möglich, den Endeffektor mit einer bestimmten Orientierung auszurichten.

Ein weiteres Problem der inversen Kinematik stellen Singularitäten dar. Von Singularitäten spricht man im Bereich der Robotik, wenn die Achsen einer kinematischen Kette so angeordnet sind, dass die Kette Freiheitsgrade verliert. Die Drehung an verschiedenen Achsen ergibt dann dieselbe Bewegung. Dies ist der Fall, wenn Achsen aufeinander liegen oder mehrere Achsen parallel sind. Es gibt somit unendlich viele Lösungen für eine Position und es kann keine eindeutige Lösung mehr bestimmt werden. In der Praxis versucht man diese Probleme zu umgehen, indem man die Bewegungsbahn des Endeffektors so modifiziert, dass die Nähe solcher Singularitäten gemieden wird. Eine andere Möglichkeit besteht darin, in der Nähe von Singularitäten über die Stellungen der Gelenke zu interpolieren, und somit die inverse Kinematik in diesem Bereich zu meiden [31].

Die Funktion $f$ aus Gleichung (3.16) ist eine nichtlineare Funktion. Die Suche nach $f^{-}1$ entspricht somit der Auflösung eines nichtlinearen Gleichungssystems. Es gibt in der Robotik zwei verbreitete Verfahren für die inverse Kinematik: Eine Möglichkeit ist es, eine analytische Lösung für das Gleichungssystem herzuleiten. Das andere Verfahren versucht mit Hilfe numerischer Verfahren eine Näherung der Funktion $f^{-1}$ zu finden.

### Analytische Lösung

Für die Suche nach der analytischen Lösung gibt es keine allgemeingültige Vorgehensweise. Es muss versucht werden, auf Basis der geometrischen Eigenschaften der kinematischen Kette entsprechende Formeln herzuleiten. Je größer die Anzahl an Freiheitsgraden ist, umso komplizierter wird dies. Sobald die Formeln gefunden sind, bietet die analytische Lösung aber mehrere Vorteile: Erstens liefert sie immer alle möglichen Gelenkstellungen für eine gegebene Position. Desweiteren ist die analytische Berechnung der gesuchten Gelenkstellungen performanter als mit einer numerischen Lösung; laut dem Institut für Produktionsanlagen und Konstruktionslehre in Berlin um den Faktor 25 [19]. Alle Lösungen sind stets exakt und eventuelle Singularitäten können bereits während der Herleitung der Formeln identifiziert werden.

Die Herleitung der Formeln für die analytische Lösung einer kinematischen Kette mit sechs Freiheitsgraden ist allerdings nur für bestimmte Spezialfälle möglich [31], beispielsweise wenn sich drei Achsen in einem Punkt im Endeffektor kreuzen. In diesem Fall kann das Problem in zwei Teilprobleme aufgeteilt werden: Mit den ersten drei Achsen wird die Position des Endeffektors berechnet und mit den anderen drei die Orientierung.
Die Beine des Roboters, der Grundlage dieser Arbeit ist, besitzen ebenfalls sechs Freiheitsgrade. Die Anordnung der Gelenkachsen erlaubt allerdings keine Aufteilung in Teilprobleme, womit eine numerische Lösung gewählt werden muss. Aus diesem Grund wird hier auf eine genauere Erklärung der analytischen Methode verzichtet.

### Numerische Lösung

Die numerische Lösung der inversen Kinematik sucht nach einer linearen Annäherung an $f^{-1}$, was mit der Jacobimatrix erreicht werden kann. Um diese zu erhalten, müssen zunächst die Teilfunktionen der Vektorfunktion $f$ bestimmt werden.

$$\begin{aligned} x &= f_1(q_1, q_2, ...q_n) \\ y &= f_2(q_1, q_2, ...q_n) \\ z &= f_3(q_1, q_2, ...q_n) \\ \alpha &= f_4(q_1, q_2, ...q_n) \\ \beta &= f_5(q_1, q_2, ...q_n) \\ \gamma &= f_6(q_1, q_2, ...q_n) \end{aligned}$$

Die Teilfunktionen können aus der Transformationsmatrix für die direkte Kinematik extrahiert werden, wie in Kapitel 3.2.2 beschrieben. Jede dieser Funktionen wird nach

jeder Variable für die Gelenkstellungen partiell abgeleitet und damit die Jacobimatrix (3.25) gebildet.

$$J(\vec{q}) = \left(\frac{\partial f_i}{\partial q_i}\right) = \begin{pmatrix} \frac{\partial f_1}{\partial q_1} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_6}{\partial q_1} & \cdots & \frac{\partial f_6}{\partial q_n} \end{pmatrix} \tag{3.25}$$

Dies ergibt eine $6 \times n$ Matrix. Die Jacobimatrix entspricht, wie in Gleichung (3.26) zu sehen ist, der ersten Ableitung der Funktion $f$ im Punkt $\hat{q}$. Somit beschreibt sie die Änderung der Position des Endeffektors ($\Delta\vec{s}$) bei einer gegebenen Änderung der Gelenkstellungen ($\Delta\vec{q}$) im Punkt $\hat{q}$.

$$\Delta\vec{s} = J(\hat{q})\Delta\vec{q} \tag{3.26}$$

$$\vec{s} = f(\hat{q}) + J(\hat{q})(\vec{q} - \hat{q}) \tag{3.27}$$

Gleichung (3.27) beschreibt die Tangente an der Funktion $f$ im Punkt $\hat{q}$ (Abb. 3.10), sie entspricht also einer linearen Annäherung an $f$. In Abb. 3.10 ist zu erkennen, dass die Annäherung nur im näheren Bereich des Punktes $\hat{q}$ noch akzeptable Werte liefern kann. Mit größerem Abstand steigt die Abweichung zur tatsächlichen Funktion $f$ schnell an. Um den Fehler möglichst gering zu halten, sollte sich die gesuchte Lösung daher immer in der Nähe des Punkts $\hat{q}$ befinden.

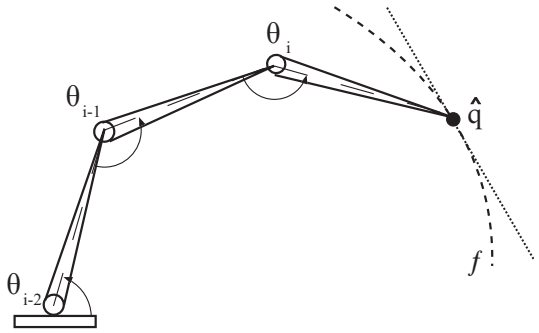



Abbildung 3.10: Annäherung an $f$

Um die Annäherung zur Funktion $f^{-1}$ zu erhalten, muss Gleichung (3.26) entsprechend umgestellt werden. Somit ergibt sich:

$$\Delta\vec{q} = J^{-1}(\hat{q})\Delta\vec{s} \tag{3.28}$$

$$\vec{q} = \hat{q} + J^{-1}(\hat{q})(\vec{s} - f(\hat{q})) \tag{3.29}$$

Mit Gleichung (3.29) ist es möglich, bei gegebener Position $\vec{s}$ die passenden Gelenkstellungen $\vec{q}$ zu berechnen. Da dies wiederum nur eine lineare Annäherung an die Funktion

$f^{-1}$ ist, muss auch hier der oben angesprochene Fehler, der bei großem Abstand zwischen der gesuchten Lösung und dem Punkt $\hat{q}$ entsteht, beachtet werden. Hierzu wird nach der Berechnung von $\vec{q}$ die gefundene Lösung mit Hilfe der direkten Kinematik überprüft. Das Ergebnis wird mit der gesuchten Position verglichen. Sollte die Abweichung zu groß sein, muss die Jacobimatrix neu berechnet und $\vec{q}$ erneut ausgewertet werden. Für die Neuberechnung der Jacobimatrix werden die im vorherigen Durchgang berechneten Gelenkstellungen verwendet. Somit erreicht man eine schrittweise Annäherung an die Lösung, bis der Fehler klein genug ist.

Wenn eine fortlaufende Folge von Punkten berechnet werden soll - was in der Praxis oft der Fall ist um Bewegungsbahnen auszuwerten -, kann der Rechenaufwand mit geschickter Wahl der Punkte gering gehalten werden. Die Punkte sollten so gewählt werden, dass sie einen ausreichend geringen Abstand zueinander haben. Somit können die Punkte mit nur wenigen Iterationen des oben genannten Algorithmus berechnet werden.

Ein Problem ergibt sich, wenn der gesuchte Punkt außerhalb des Arbeitsbereichs der kinematischen Kette liegt. Es kann mit dem oben genannten Algorithmus nicht direkt entschieden werden, ob dies der Fall ist. Stattdessen lässt man den Algorithmus eine vorgegebene maximale Anzahl an Iterationen durchlaufen. Sollte danach keine ausreichend genaue Lösung vorliegen, kann davon ausgegangen werden, dass die gesuchte Position außerhalb des Arbeitsbereichs liegt. Eine andere Möglichkeit wäre, bereits vor dem Start des Algorithmus zu überprüfen, ob der gesuchte Punkt innerhalb des Arbeitsbereichs liegt. Allerdings ist es schwer, den genauen Arbeitsbereich ohne eine analytische Lösung der inversen Kinematik zu bestimmen.

Die numerische Lösung der inversen Kinematik berechnet im Vergleich zur analytischen Lösung immer nur eine mögliche Lösung. Man bekommt immer die Lösung, die am nähesten zu dem Punkt $\hat{q}$, für den die Jacobimatrix berechnet wurde, ist. Dies hat den Nachteil, dass es keine Wahlmöglichkeit zwischen den Lösungen mehr gibt, um eventuelle Hindernisse zu umfahren. Allerdings muss, wenn eine Bewegungsbahn berechnet wird, nicht bei jedem Punkt entschieden werden welche Lösung am nähesten zu dem letzten Punkt ist. Dies ist bei der analytischen Lösung nötig, um Sprünge in der Bewegung zu vermeiden.

Ein weiteres Problem der numerischen Lösung stellen Singularitäten dar, welche in diesem Kapitel bereits angesprochen wurden. Singularitäten äußern sich in der numerischen Lösung dadurch, dass die Jacobimatrix an Rang verliert und singulär wird. Sie ist somit nicht mehr invertierbar und es kann keine Lösung für die inverse Kinematik bestimmt werden. Desweiteren können sich in der unmittelbaren Umgebung von Singularitäten

sehr hohe Geschwindigkeiten in den Gelenken ergeben, was zu ungewollt schnellen Bewegungen führen kann. Es sollten daher zuvor angesprochene Maßnahmen getroffen werden, um Singularitäten zu meiden.

### 3.2.4 Kinematik der Geschwindigkeiten

In gewissen Fällen kann es notwendig sein, eine Beziehung zwischen der Geschwindigkeit des Endeffektors im kartesischen Raum und den Geschwindigkeiten in den Gelenken zu haben. Beispielsweise um die maximale Geschwindigkeit des Endeffektors zu bestimmen, welche durch die maximale Geschwindigkeit der Gelenke beschränkt wird. Ein weiteres Beispiel wäre die Berechnung der nötigen Geschwindigkeiten der Gelenke, um eine gewünschte Geschwindigkeit des Endeffektors zu erreichen. Dies kann dann insbesondere dazu verwendet werden, um die für eine möglichst flüssige Steuerung der Motoren benötigten Geschwindigkeiten der Gelenke zu bestimmen.

Der erste Fall entspricht der direkten Kinematik, also der Umrechnung von Gelenkstellungen in den kartesischen Raum. Im zweiten Fall wird eine Umrechnung vom kartesischen Raum in die Gelenkstellungen benötigt, was wiederum der inversen Kinematik entspricht.

Um die Beziehung zwischen den Geschwindigkeiten herzustellen, kann die Jacobimatrix verwendet werden, welche bereits für die numerische Lösung der inversen Kinematik in Kapitel 3.2.3 verwendet wurde. Wenn Gleichung (3.26) durch die Zeit dividiert wird, erhält man die Gleichung für die Geschwindigkeiten.

$$\dot{\vec{s}} = J(\hat{q})\dot{\vec{q}} \tag{3.30}$$

Mit Gleichung (3.30) kann bei gegebener Geschwindigkeit der Gelenke $\dot{\vec{q}}$ im Punkt $\hat{q}$ die resultierende Geschwindigkeit des Endeffektors $\dot{\vec{s}}$ im kartesischen Raum berechnet werden. Der Vektor $\dot{\vec{s}} = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)^T$ enthält die Geschwindigkeiten entlang der drei Achsen $(v_x, v_y, v_z)$ und die Winkelgeschwindigkeiten um die Achsen $(\omega_x, \omega_y, \omega_z)$. Je nach Typ der Gelenke enthält der Vektor $\dot{\vec{q}}$ die Geschwindigkeit bei Translationsgelenken und die Winkelgeschwindigkeit bei Rotationsgelenken.

Analog zur Entwicklung der numerischen Lösung für die inverse Kinematik kann Gleichung (3.30) umgestellt werden, um bei gegebener Geschwindigkeit des Endeffektors die entsprechenden Geschwindigkeiten in den Gelenken zu berechnen.

$$\dot{\vec{q}} = J^{-1}(\hat{q})\dot{\vec{s}} \tag{3.31}$$

Um genaue Ergebnisse zu erhalten, muss die Jacobimatrix immer für den Punkt $\hat{q}$, in dem die Geschwindigkeit bestimmt werden soll, neu berechnet werden. Hierzu muss der für die Position des Endeffektors $\vec{s}$ entsprechende Punkt $\hat{q}$ berechnet werden. Dies kann mit der bereits aus der direkten Kinematik bekannten Funktion $f$ geschehen. Somit erhält man Gleichung (3.32), welche bei gegebener Geschwindigkeit des Endeffektors $\dot{\vec{s}}$ im Punkt $\vec{s}$ die entsprechenden Geschwindigkeiten der Gelenke $\dot{\vec{q}}$ bestimmt.

$$\dot{\vec{q}} = J^{-1}(f(\vec{s}))\dot{\vec{s}} \tag{3.32}$$

Bei der Invertierung der Jacobimatrix kann wieder das Problem von Singularitäten auftreten, welches bereits in Kapitel 3.2.3 beschrieben wurde. In diesem Fall kann die Jacobimatrix nicht mehr invertiert werden und eine Lösung der Gleichung ist somit in diesem Punkt nicht möglich. Wie bereits zuvor erwähnt, kann dies durch entsprechende Planung der Bewegungsbahn des Endeffektors oder durch Interpolation im Bereich der Singularitäten vermieden werden.

# 4 Der Roboter

Dieses Kapitel beschreibt den Roboter (Abb. 4.1), der Grundlage für den hier entwickelten Laufalgorithmus ist. In der Diplomarbeit [21] wurde die nachfolgend beschriebene Konstruktion entwickelt. Im Umfang der Arbeit wurden auch detaillierte CAD Zeichnungen erstellt, in denen bereits alle nötigen Einzelteile enthalten sind.

Ziel war die Konstruktion eines zweibeinigen Laufroboters, der die Anforderungen der RoboCup Humanoid Kid Size League 2006 [5] erfüllen soll. Hauptbestandteile der Anforderungen sind die zweibeinige Fortbewegung und ein voll autonomer Betrieb des Roboters. Außerdem gibt es Vorgaben in Bezug auf die Ausmaße des Roboters und die verwendeten Sensoren.

Abgesehen von diesen Anforderungen wurde versucht, die Proportionen und kinematischen Eigenschaften möglichst menschenähnlich zu konstruieren. Der menschliche Bewegungsapparat hatte, wie bereits erwähnt, viele Jahrtausende Zeit, sich optimal zu entwickeln. Daher bildet eine solche Konstruktion die ideale Grundlage für zweibeinige Laufroboter. Nicht zuletzt sollte bei der Konstruktion ein Budget für die Materialien in Höhe von 5000 € nicht überschritten werden.

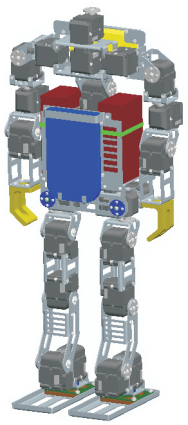


Abbildung 4.1: Roboter

## 4.1 Mechanischer Aufbau

Der Roboter besitzt insgesamt 23 Freiheitsgrade. Jeweils sechs pro Bein, vier pro Arm, zwei zur Neigung und Drehung des Oberkörpers und einer um den Kopf zu drehen. Abb. 4.2 zeigt das kinematische Model des Roboters. Nicht alle Gelenke, die ein Mensch besitzt, wurden in die Konstruktion miteinbezogen. Einige sind nicht relevant für die geplanten Aufgaben des Roboters. Andere Gelenke besitzen nur minimale Bewegungsradien und können daher ebenso vernachlässigt werden.

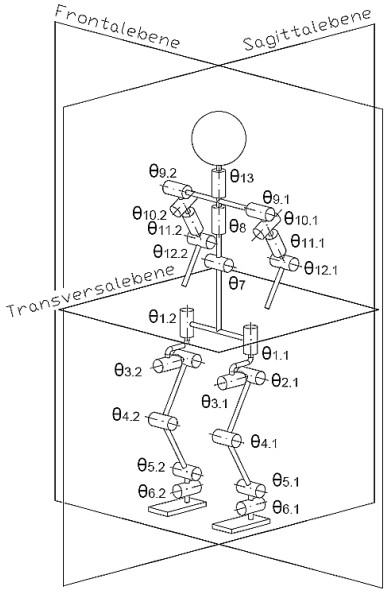



Abbildung 4.2: Kinematisches Model

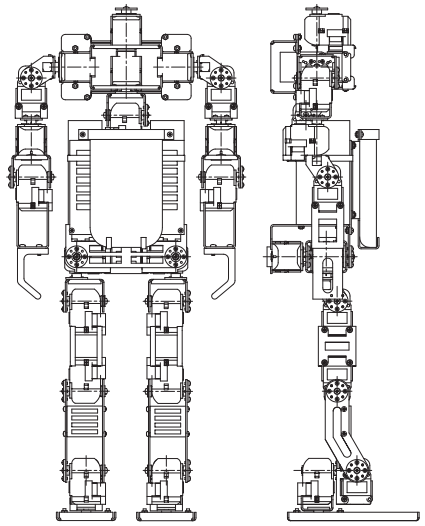


Abbildung 4.3: Konstruktionszeichnung

Die Konstruktionszeichnung ist in Abb. 4.3 zu sehen. Für den Kopf wurde zwar bereits ein Freiheitsgrad vorgesehen, er wurde allerdings noch nicht in die Konstruktion miteinbezogen, da bisher keine geeignete Kamera gefunden werden konnte.

Für diese Arbeit sind insbesondere die beiden Beine von Interesse. Das linke Bein ist in Abb. 4.4 zu sehen, das rechte Bein ist genau spiegelverkehrt dazu aufgebaut.

Bei der Ansteuerung der Motoren muss besonders darauf geachtet werden, die mechanischen Beschränkungen nicht zu überschreiten, um Beschädigungen an der Konstruktion zu verhindern. Tabelle 4.1(a) zeigt die in [21] vorgegebenen maximalen Ausschläge der Achsen für das linke Bein, Tabelle 4.1(b) für das rechte Bein.

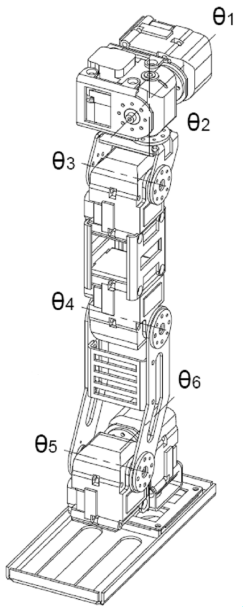



Abbildung 4.4: Konstruktion linkes Bein

Tabelle 4.1: Bewegungsradien der Beine

(a) linkes Bein

| Achse | + | - |
|---|---|---|
| $\theta_1$ | 148° | 20° |
| $\theta_2$ | 150° | 150° |
| $\theta_3$ | 93° | 125° |
| $\theta_4$ | 128° | 86° |
| $\theta_5$ | 50° | 126° |
| $\theta_6$ | 90° | 90° |

(b) rechtes Bein

| Achse | + | - |
|---|---|---|
| $\theta_1$ | 148° | 20° |
| $\theta_2$ | 150° | 150° |
| $\theta_3$ | 125° | 93° |
| $\theta_4$ | 86° | 128° |
| $\theta_5$ | 126° | 50° |
| $\theta_6$ | 90° | 90° |

## 4.2 Steuerungsrechner

Früher wurden für die Steuerung von Robotern aufgrund der Größenbeschränkungen oft Mikrocontroller oder Einplatinen-Industrierechner verwendet. Diese haben allerdings mehrere Nachteile: geringe Rechenleistung, kein Benutzerinterface und nur wenig verfügbare Schnittstellen.

Durch den stark gesunkenen Preis im Bereich der Handheld PCs wie beispielsweise PDAs (Personal Digital Assistant), bieten sich diese heute als Alternative an [7]. Oftmals sind solche Geräte bereits mit einer großen Anzahl an unterschiedlichen Schnittstellen wie

WLAN, CF(CompactFlash), SDIO(Secure Digital Input/Output) und serieller Schnittstelle ausgestattet. Weitere Schnittstellen können durch die vielen verfügbaren Zubehörteile leicht nachgerüstet werden. Außerdem bieten alle Geräte ein großes Display, mit dem relativ leicht eine Benutzerschnittstelle realisiert werden kann. Nicht zuletzt sind auch die heutzutage hohen Rechenleistungen, die sogar komplexere Bildverarbeitungsverfahren erlauben, ein weiteres Argument für Handheld PCs.

Aus diesen Gründen wird auch in dem hier verwendeten Roboter ein PDA als Steuerungsrechner genutzt. Wichtige Eigenschaften für die Auswahl waren, neben einer für den Roboter passenden Größe, ein möglichst geringes Gewicht und ausreichend Schnittstellen für die Ansteuerung der Motoren, den Anschluss einer Kamera und weiterer Sensoren. Letzteres kann über einen CF oder SDIO Port geschehen. Die Auswahl fiel daher auf den HP iPaq hx2490 (Abb. 4.5). Dieser bot, im Vergleich zu PDAs mit ähnlichen Eigenschaften, den geringsten Preis.

Abbildung 4.5: Verwendeter PDA

## 4.3 Sensoren

In die Konstruktion mit eingeplant wurden vorerst nur zwei Kraftsensoren in den Kontaktflächen der Füße. Diese sollen später zur Messung des ZMP genutzt werden. Die Kraftsensoren bestehen jeweils aus einem Verformkörper, welcher mit Dehnungsmessstreifen versehen ist. Da industriell hergestellte Kraftsensoren entweder zu groß oder zu kostspielig sind, wurde der hier verwendete Kraftsensor in oben genannter Arbeit neu entwickelt.

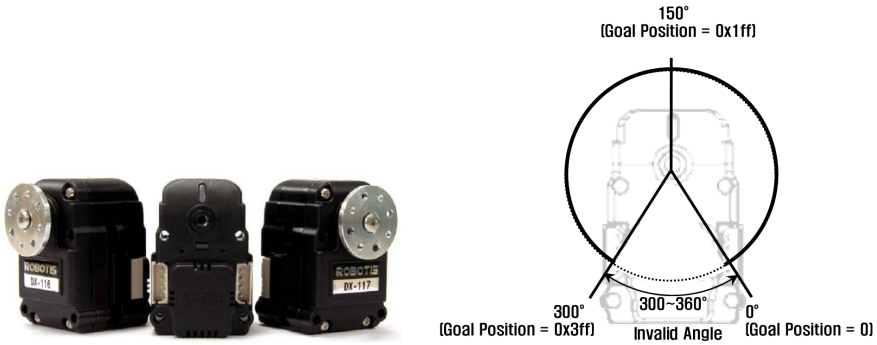
Wie bereits erwähnt, ist auf dem Kopf schon Platz für eine Kamera vorgesehen. Die Suche nach einer passenden Kamera gestaltet sich allerdings schwierig, da diese zwei speziellen

Anforderungen genügen muss: Sie muss mit dem verwendeten PDA kompatibel sein und gleichzeitig eine kabelgeführte Montage im Kopf erlauben. Die meisten Kameras die für PDAs erhältlich sind, erlauben nur eine feste Montage am PDA. Solch eine Konstruktion kommt hier allerdings nicht in Frage, da nach den Regeln der Humanoid League als Position für die Kamera nur der Kopf erlaubt ist.

Es ist vorgesehen, noch weitere Sensoren in den Roboter einzubauen. Allerdings müssen auch hier die Beschränkungen der Humanoid League beachtet werden. Insbesondere Sensoren, die elektromagnetische Felder, Licht oder Schall ausstrahlen, sind nicht erlaubt. Geplant ist daher nur der Einbau von Sensoren zur Positions- und Neigungsbestimmung wie beispielsweise GPS, Beschleunigungssensoren oder Kreisel.

## 4.4 Motoren

Als Motoren wurden von der Firma Tribotix Antriebe der DX Serie [23] ausgewählt (Abb. 4.6(a)). Diese bieten bei geringen Ausmaßen und einem akzeptablen Preis ausreichend Drehmoment, um den Roboter anzutreiben. Darüber hinaus bieten Motoren der Firma Tribotix noch weitere wichtige Eigenschaften: Die Antriebe werden über eine RS485 Bus miteinander verbunden, somit ist nur ein Kontroller nötig, um eine große Anzahl an Motoren ansteuern zu können. Durch die Verwendung eines Bussystems vereinfacht sich auch die Verkabelung des Roboters. Außerdem verfügen die Motoren bereits über eine integrierte Positionsregelung und liefern zusätzlich eine große Zahl an Informationen über den Zustand des Motors. Unter anderem können Informationen über die Position, Temperatur, Drehmoment und Geschwindigkeit abgerufen und überwacht werden.




(a) DX Serie (b) Drehbereich

Abbildung 4.6: Antriebe

Um die Kommunikation mit dem PDA zu ermöglichen, ist ein Schnittstellenwandler notwendig, der die Daten von der RS232 Schnittstelle des PDAs zu dem RS485 Bus überträgt. Hierfür kann die von Tribotix verfügbare Steuerungsplatine verwendet werden. Für alle Gelenke im Roboter werden die stärksten Motoren der Serie - DX 117 - verwendet. Nicht überall sind zwar so hohe Drehmomente nötig wie in den Beinen, allerdings ist es vorteilhaft die Anzahl an unterschiedlichen Komponenten im Roboter möglichst gering zu halten. Die Antriebe können ausgehend von der Mittelstellung jeweils um 150° in beide Richtung gedreht werden, somit ergibt sich ein verfügbarer Drehbereich von 300° (Abb. 4.6(b)).

# 5 Entwicklung des Laufalgorithmus

Da bisher keine Arbeiten vorliegen, die sich mit der Entwicklung eines Laufalgorithmus für den in Kapitel 4 vorgestellten Roboter befassen, war es zunächst notwendig, eine kinematische Analyse des Roboters durchzuführen. Im ersten Teil dieses Kapitels wird daher beschrieben, wie die kinematischen Transformationsgleichungen für die Beine hergeleitet wurden. Ebenso werden hier die verwendeten Koordinatensysteme genauer erläutert, da diese unerlässlich für die Entwicklung des Stabilitätskriteriums sind. Die hier entwickelten Ergebnisse sind grundlegend für jeden Laufalgorithmus, daher können sie auch für alle nachfolgenden Arbeiten in Bezug auf diesen Roboter verwendet werden.

Im nächsten Schritt musste ein Stabilitätskriterium entwickelt werden, um die Planung des Laufmusters zu ermöglichen. Der zweite Teil befasst sich daher mit den Schritten, die für eine Stabilitätsanalyse notwendig sind. Da zum Zeitpunkt der Arbeit noch keine geeigneten Sensoren und Erfahrungen mit dem Roboter im Laufbetrieb vorlagen, wurde zunächst ein statisches Stabilitätskriterium gewählt. Zwar könnte auch ein dynamisch stabiles Laufmuster ohne Sensorinformationen berechnet werden, dies würde jedoch nur zu unwesentlich besseren Ergebnissen führen (vgl. Kapitel 2), die den Mehraufwand bei den Berechnungen nicht aufwiegen.
Wie bereits erwähnt, können mit einem statisch Stabilitätskriterium zwar keine herausragenden Resultate erwartet werden, es soll aber als Grundlage für erste Laufversuche mit dem Roboter dienen. Auf Basis der gesammelten Erfahrungen kann später ein dynamisch stabiler Laufalgorithmus entwickelt werden. Obwohl die Schritte der Stabilitätsanalyse speziell für die Berechnung einer statischen Stabilität entwickelt wurden, können viele Teile später auch bei der Entwicklung eines dynamischen Laufalgorithmus weiterverwendet werden.

Schließlich konnte mit der Planung des Laufmusters auf Basis des gewählten Stabilitätskriteriums begonnen werden. Der letzte Teil beschreibt daher das Laufmuster, welches dem Laufalgorithmus zugrunde liegt. Um spätere praktische Versuche zu vereinfachen, wurde das Laufmuster parameterisiert. Dieser Teil beschreibt somit auch, wie aus gegebenen Parametern die entsprechenden Trajektorien berechnet werden können. Während

der Planung des Laufmusters haben sich bestimmte Eigenschaften der Beinkonstruktion als problematisch herausgestellt. Abschließend werden daher die Probleme genauer erläutert und entsprechende Lösungen vorgeschlagen.

## 5.1 Modellierung der Kinematik

Der in dieser Arbeit entwickelte Laufalgorithmus beschränkt sich auf die Beine. Andere Gelenke im Oberkörper oder in den Armen wurden nicht miteinbezogen. Von den insgesamt verfügbaren 23 Freiheitsgraden wurden deshalb nur die jeweils sechs pro Bein benötigt. Die nachfolgend beschriebenen Ergebnisse beschränken sich daher auf die beiden Beine. Jedes Bein bildet hierbei für sich eine eigene kinematische Kette und muss somit auch gesondert analysiert werden.

### 5.1.1 Bestimmen der DH Parameter

Um die kinematischen Formeln zur Beschreibung der Beine herzuleiten, wurde das Denavit-Hartenberg Verfahren (vgl. Kapitel 3.2.1) verwendet. Dazu mussten zunächst die DH Parameter bestimmt werden. Abb. 5.1 zeigt die Abstände des linken Beins, die hierfür benötigt wurden; Tabelle 5.1 zeigt die zugehörigen Werte. Zusätzlich sind auf der Abbildung die Drehwinkel $\delta_1$ bis $\delta_6$ eingezeichnet.

Zur Bestimmung der Parameter musste zunächst ein Anfangskoordinatensystem für die Kinematik festgelegt werden. Der Ursprung des Koordinatensystems wurde so gewählt, dass er in dem Kreuzungspunkt der ersten beiden Drehachsen, $\delta_1$ und $\delta_2$, liegt. Somit konnten überflüssige Translationen beim Übergang in das erste Gelenk vermieden werden. Die Orientierung wurde so gewählt, dass die z-Achse antiparallel zur Schwerkraft liegt und die x-Achse in Laufrichtung zeigt. Die y-Achse ergänzt das rechtsdrehende Koordinatensystem. Desweiteren wurde festgelegt, dass in jedem Gelenk die x-Achse auf der Drehachse und die y-Achse auf der Normalen liegt. Somit ergibt sich die Transformationsgleichung (5.1) für die direkte Kinematik:

$$ {}^{n-1}_{n}T = Rot(x_{n-1}, \theta_n)Trans(d_n, 0, 0)Trans(0, a_n, 0)Rot(y_n, \alpha_n) \tag{5.1} $$

Die gewählte Ausrichtung der Koordinatensysteme bietet keine besonderen Vor- oder Nachteile. Es kann auch jede beliebige andere Ausrichtung gewählt werden, solange dabei das Vorgehen aus Kapitel 3.2.1 beachtet wird.

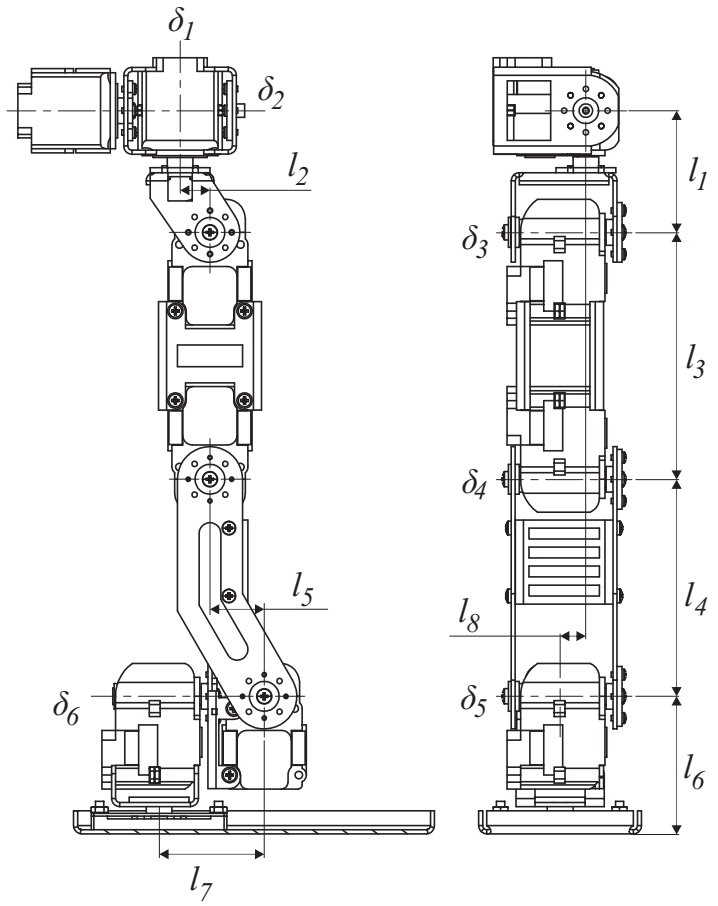



Abbildung 5.1: Abmessungen und Drehwinkel linkes Bein

| | |
|---|---|
| $l_1$ | 44,8 mm |
| $l_2$ | 11,0 mm |
| $l_3$ | 91,0 mm |
| $l_4$ | 80,0 mm |
| $l_5$ | 20,0 mm |
| $l_6$ | 50,7 mm |
| $l_7$ | 38,7 mm |
| $l_8$ | 9,5 mm |

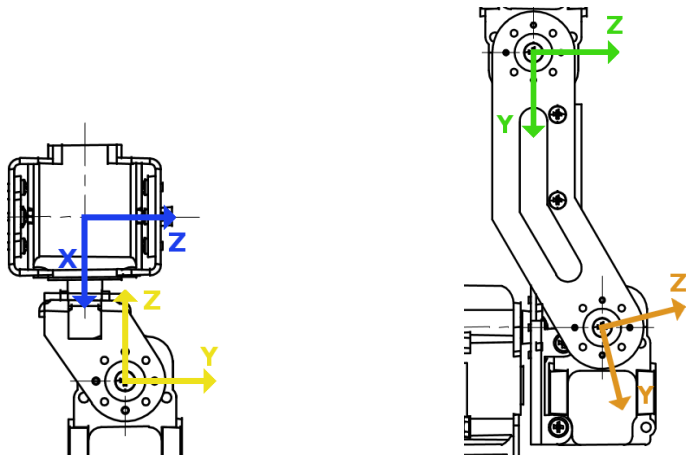
Tabelle 5.1: Maße linkes Bein

Nachdem das Anfangskoordinatensystem festgelegt und die relevanten Achsen für die Transformationsgleichung bestimmt waren, konnten die DH Parameter aufgestellt werden. Bei der Umsetzung ergaben sich allerdings einige Probleme, die besondere Beachtung benötigten. Abb. 5.2 zeigt die Gelenke, in denen die Probleme auftraten.

Das erste Problem bestand im Übergang vom ersten in das zweite Gelenk, welche in Abb. 5.2(a) zu sehen sind. Die erste Translation entlang der x-Achse verschiebt das Koordinatensystem nach unten. Nun müsste noch eine Translation entlang der y-Achse das Koordinatensystem nach rechts verschieben, um den Ursprung auf die Kreuzung der Drehachsen $\delta_2$ und $\delta_3$ zu setzen. Dies ist allerdings nicht möglich, da die y-Achse nicht nach rechts zeigt. Um dies zu erreichen, muss bei der ersten Rotation die y-Achse, entlang derer die zweite Translation verschiebt, bereits nach rechts ausgerichtet werden. Hierfür wurde zum Drehwinkel $\delta_2$ noch $\frac{\pi}{2}$ addiert. Man kann dies als eine Änderung der Grundposition der Gelenke auffassen; das Bein ist in diesem Fall, im Gegensatz zu der Position

aus 5.1, nach links verdreht. Da diese Änderung aber innerhalb der DH Parameter selbst vorgenommen wurde, muss sie bei der späteren Verwendung der Transformationsmatrix nicht mehr beachtet werden. Man kann folglich bei der Interpretation der Drehwinkel von der in Abb. 5.1 gezeigten Grundposition ausgehen.

Zwischen Gelenk drei und vier ergab sich ein ähnliches Problem: Abb. 5.2(b) zeigt die beiden Gelenke. Hier ist die y-Achse wiederum nicht passend ausgerichtet, um auf die nächste Drehachse zu verschieben. Wie im vorherigen Beispiel wurde hier zu dem Drehwinkel $\delta_4$ $atan(\frac{l_5}{l_4})$ addiert, um die y-Achse entsprechend auszurichten. Da die Achse im nächsten Schritt wieder auf die ursprüngliche Ausrichtung gebracht werden musste, wurde der Drehwinkel $\delta_5$ um den selben Wert wieder zurück gedreht.




(a) Gelenk 1 und 2 (b) Gelenk 3 und 4

Abbildung 5.2: Spezialfälle der Beinkinematik

Tabelle 5.2(a) zeigt die DH Parameter für das linke Bein. Zusätzlich zu den sechs Transformationen, die für die sechs Gelenke nötig sind, wurde eine siebte hinzugefügt. Diese dient dazu, das Koordinatensystem im letzten Gelenk wieder auf die selbe Ausrichtung wie das Anfangskoordinatensystem zu bringen. Somit kann aus der Transformationsmatrix direkt abgelesen werden, ob und wieviel der Fuß relativ zum Anfangskoordinatensystem verdreht ist, ohne dass weitere Umrechnungen nötig sind. Da im siebten Schritt keine neue Variable für $\theta$ eingeführt wurde, wird auch die spätere Berechnung der direkten und inversen Kinematik nicht aufwendiger.

Die DH Parameter für das rechte Bein zeigt Tabelle 5.2(b). Aufgrund der fast identischen mechanischen Struktur beider Beine gibt es allerdings keinen großen Unterschied zu den DH Parametern des linken Beins. $d_5$ ist der einzige Parameter, der sich bei beiden Beinen

unterscheidet. Hier muss, bedingt durch den spiegelverkehrten Aufbau der Beine, im rechten Bein in die entgegengesetzte Richtung verschoben werden.

Tabelle 5.2: DH Parameter

(a) linkes Bein

| n | $a_n$ | $d_n$ | $\alpha_n$ | $\theta_n$ |
|---|---|---|---|---|
| 1 | 0 | 0 | $\frac{\pi}{2}$ | $\delta_1$ |
| 2 | $l_2$ | $l_1$ | $-\frac{\pi}{2}$ | $\delta_2 + \frac{\pi}{2}$ |
| 3 | $l_3$ | 0 | 0 | $\delta_3 - \frac{\pi}{2}$ |
| 4 | $\sqrt{l_5^2 + l_4^2}$ | 0 | 0 | $\delta_4 + \text{atan}(\frac{l_5}{l_4})$ |
| 5 | 0 | $l_8$ | $-\frac{\pi}{2}$ | $\delta_5 - \text{atan}(\frac{l_5}{l_4})$ |
| 6 | $l_6$ | $-l_7$ | 0 | $\delta_6$ |
| 7 | 0 | 0 | 0 | $\frac{\pi}{2}$ |

(b) rechtes Bein

| n | $a_n$ | $d_n$ | $\alpha_n$ | $\theta_n$ |
|---|---|---|---|---|
| 1 | 0 | 0 | $\frac{\pi}{2}$ | $\delta_1$ |
| 2 | $l_2$ | $l_1$ | $-\frac{\pi}{2}$ | $\delta_2 + \frac{\pi}{2}$ |
| 3 | $l_3$ | 0 | 0 | $\delta_3 - \frac{\pi}{2}$ |
| 4 | $\sqrt{l_5^2 + l_4^2}$ | 0 | 0 | $\delta_4 + \text{atan}(\frac{l_5}{l_4})$ |
| 5 | 0 | $-l_8$ | $-\frac{\pi}{2}$ | $\delta_5 - \text{atan}(\frac{l_5}{l_4})$ |
| 6 | $l_6$ | $-l_7$ | 0 | $\delta_6$ |
| 7 | 0 | 0 | 0 | $\frac{\pi}{2}$ |

### 5.1.2 Transformationsgleichungen für direkte und inverse Kinematik

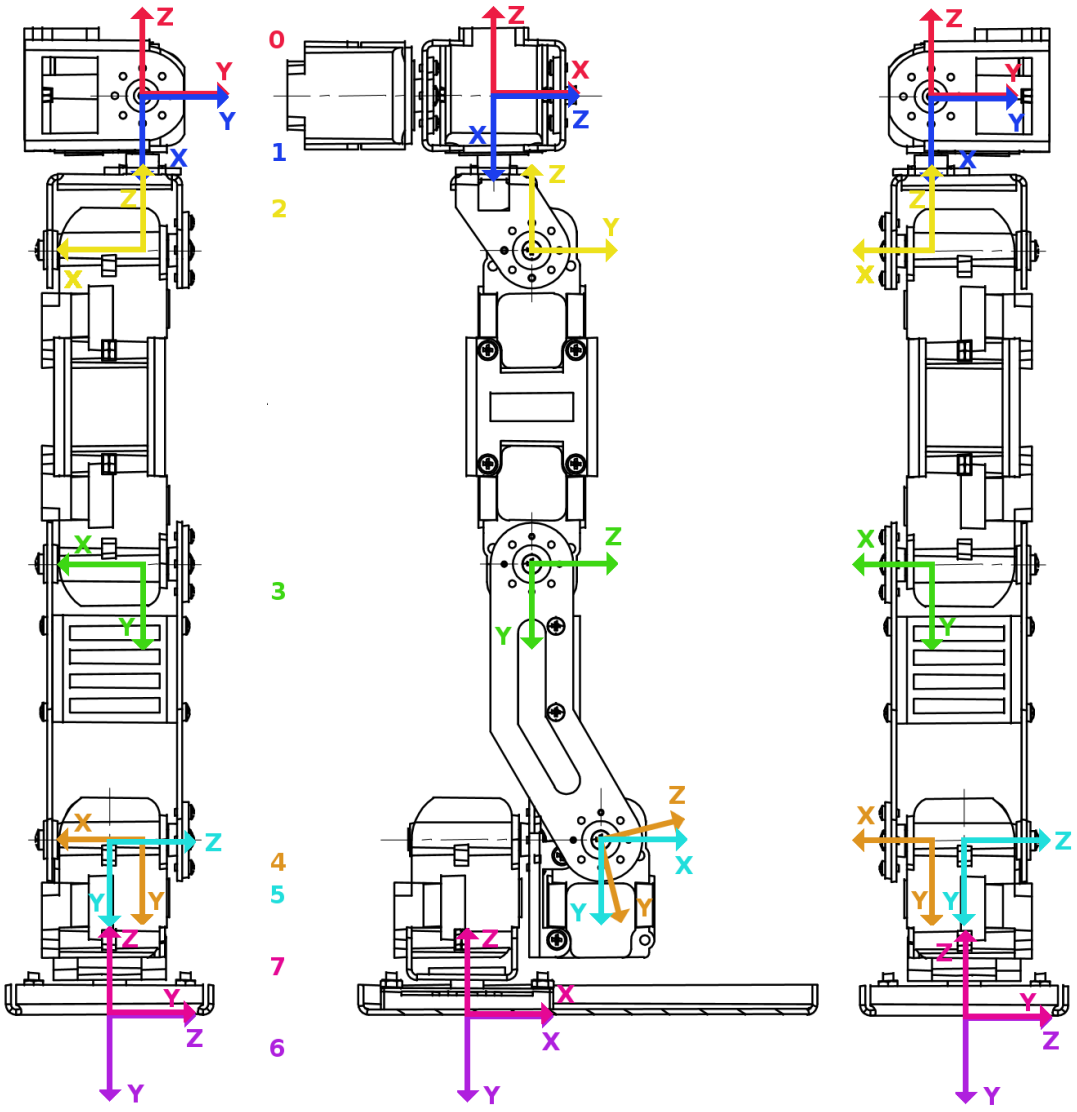
Um die Transformationsmatrix für die direkte Kinematik zu erhalten, müssen die zuvor bestimmten DH Parameter für jedes Gelenk einzeln in Gleichung (5.1) eingesetzt und die entstehenden Matrizen miteinander multipliziert werden. Dies muss für beide Beine getrennt berechnet werden. Durch die große Komplexität der Beine ergibt sich dabei eine entsprechend große Transformationsmatrix, deren Abbildung den verfügbaren Platz bei weitem übersteigt. Aus diesem Grund wird hier auf die explizite Abbildung der Matrix verzichtet, das genaue Vorgehen wurde bereits in Kapitel 3.2.2 ausführlich erläutert.

Für die Lösung der inversen Kinematik müssen die entsprechenden Jacobimatrizen der beiden Beine hergeleitet werden. Hierfür müssen die Teilfunktionen, wie in Kapitel 3.2.3 beschrieben, aus der Transformationsmatrix extrahiert werden. Nach partieller Ableitung der Teilfunktionen können die Jacobimatrizen aufgestellt werden. Mit Gleichung (3.29) kann dann die Lösung der inversen Kinematik berechnet werden.

### 5.1.3 Koordinatensysteme der Beine

Aus der Herleitung der DH Parameter ergibt sich für jedes Gelenk ein eigenes Koordinatensystem. Diese Koordinatensysteme sind wichtig, um später Masseneigenschaften des

Roboters wie Massenschwerpunkt, Trägheit und Momente zu berechnen. Abb. 5.3 zeigt alle Koordinatensysteme in den Beinen. Die Position und Orientierung der Koordinatensysteme in der Seitenansicht (Abb. 5.3(b)) ist für beide Beine gleich. Wie bereits an den DH Parametern zu erkennen war, gibt es keinen großen Unterschied zwischen den Beinen. Die Orientierung aller Koordinatensysteme ist gleich. Der einzige Unterschied der Position in den Gelenken zwei bis vier wird durch den Versatz $l_8$ verursacht.




(a) Linkes Bein (b) Seitenansicht linkes Bein (c) Rechtes Bein

Abbildung 5.3: Koordinatensysteme

Das mit 0 gekennzeichnete Koordinatensystem ist das Anfangskoordinatensystem, auch Beinbezugskoordinatensystem (BBKS) genannt. BBKSL und BBKSR bezeichnen folglich das Beinbezugskoordinatensystem des linken und rechten Beins. Für bestimmte Berechnungen, unter anderem für den Massenschwerpunkt und das Stützpolygon, ist es notwendig, ein gemeinsames Referenzkoordinatensystem für beide Beine zu definieren. Das Referenzkoordinatensystem hat die Bezeichnung Unterkörperkoordinatensystem (UKKS). Die Orientierung ist gleich zu BBKSL und BBKSR und der Ursprung des UKKS liegt genau in der Mitte zwischen beiden (Abb. 5.4). Um vom BBKSR in das UKKS zu transformieren, genügt eine Translation entlang der y-Achse um 50mm. Beim BBKSL sind es entsprechend -50mm entlang der y-Achse.

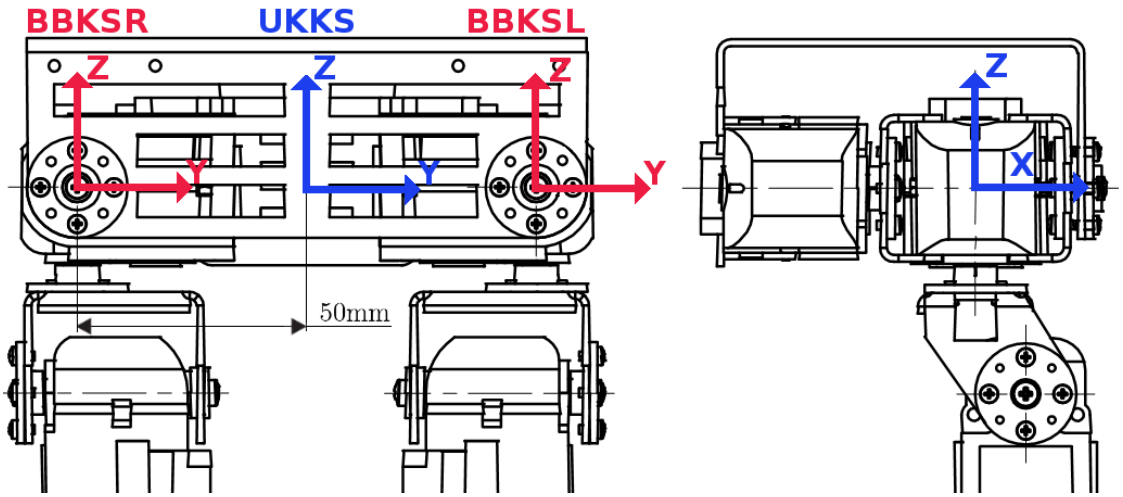



Abbildung 5.4: Unterkörperkoordinatensystem - UKKS

Üblicherweise wäre es sinnvoll, noch weitere Referenzkoordinatensysteme im Oberkörper des Roboters zu definieren. Da in dieser Arbeit jedoch nur die beiden Beine betrachtet werden, wurde darauf verzichtet. Außerdem liegt noch kein ausreichendes Wissen über die Kinematik im Oberkörper vor, um geeignete Koordinatensysteme zu bestimmen.

### 5.1.4 Umrechnung der Drehwinkel für die Motoren

Der Drehsinn der Winkel $\delta_1$ bis $\delta_6$ der beiden Beine ist bereits durch die rechtsdrehenden Koordinatensysteme festgelegt. Der Drehsinn der Motoren wiederum ist durch den Einbau festgelegt. In den meisten Fällen ist der Drehsinn in den Motoren und den Koordinatensystemen nicht gleich, daher müssen die Winkel für die Steuerung der Motoren entsprechend umgerechnet werden.

Außerdem sind in der Grundstellung der Beine (vgl. Abb. 5.1) die Winkel $\delta_1$ bis $\delta_6$ in Stellung 0°. Die Motoren sind in dieser Stellung aber in der Mitte ihres Bereichs (0° bis 300°) also bei 150° ( $\frac{5}{6}\pi$ rad) montiert. Dies muss in der Umrechnung ebenfalls beachtet werden.

Wenn für die Winkel $\delta_1$ bis $\delta_6$ die Motorwinkel entsprechend mit $\chi_1$ bis $\chi_6$ bezeichnet werden, ergeben sich folgende Formeln für die Umrechnung:

| Für das linke Bein: | Für das rechte Bein: |
|---|---|
| $\delta_1 = -\chi_1 + \frac{5}{6}\pi$ | $\delta_1 = -\chi_1 + \frac{5}{6}\pi$ |
| $\delta_2 = -\chi_2 + \frac{5}{6}\pi$ | $\delta_2 = -\chi_2 + \frac{5}{6}\pi$ |
| $\delta_3 = -\chi_3 + \frac{5}{6}\pi$ | $\delta_3 = \chi_3 - \frac{5}{6}\pi$ |
| $\delta_4 = -\chi_4 + \frac{5}{6}\pi$ | $\delta_4 = \chi_4 - \frac{5}{6}\pi$ |
| $\delta_5 = -\chi_5 + \frac{5}{6}\pi$ | $\delta_5 = \chi_5 - \frac{5}{6}\pi$ |
| $\delta_6 = -\chi_6 + \frac{5}{6}\pi$ | $\delta_6 = -\chi_6 + \frac{5}{6}\pi$ |

## 5.2 Stabilitätskriterium

In dieser Arbeit soll ein statisch stabiler Laufalgorithmus entwickelt werden. Wie bereits in Kapitel 2 erwähnt, werden bei einem statisch stabilen Laufalgorithmus nur statische Kräfte beachtet.

Für die Berechnung der statischen Stabilität sind drei Schritte notwendig:

1. Massenschwerpunkt des gesamten Roboters berechnen.
2. Das von den Füßen aufgespannte Stützpolygon ermitteln.
3. Testen, ob die vertikale Projektion des Massenschwerpunktes (COG) innerhalb des Stützpolygons liegt.

Nachfolgend werden die drei Schritte genau erklärt. Die vorgestellten Methoden lassen sich direkt auf andere Roboter zur Entwicklung statisch stabiler Laufalgorithmen anwenden. Aber auch dynamische Laufalgorithmen können die meisten der vorgestellten Methoden verwenden; insbesondere die Bestimmung des Stützpolygons und der Punkt-in-Polygon Test werden auch für fast alle dynamischen Laufalgorithmen benötigt.

### 5.2.1 Masseneigenschaften des Roboters zusammenfassen

In Kapitel 2.1.2 wurde bereits erklärt, wie man einzelne Massenschwerpunkte zusammenfassen kann. Um die Formel (2.1) anwenden zu können, mussten zunächst die Masseneigenschaften der einzelnen Gelenke der Beine bestimmt werden. Aus der Diplomarbeit [21] ist bereits ein CAD Modell des kompletten Roboters, erstellt in Pro/ENGINEER, vorhanden. Um möglichst genaue Daten zu erhalten, mussten noch die Masseneigenschaften der Motoren, des PDAs und der Akkus nachgetragen werden. In Pro/ENGINEER wurden die Beine dann so zerteilt, dass für jedes Gelenk ein eigenes CAD Modell existiert. Nur so ist es möglich, eine Analyse der Masseneigenschaften der einzelnen Gelenke mit Pro/ENGINEER durchzuführen. Jedes Gelenk wurde wiederum mit dem entsprechenden Koordinatensystem aus dem DH Verfahren versehen (vgl. Abb. 5.3), um Berechnungen relativ zu diesen Koordinatensystemen zu ermöglichen. Danach konnten mit Pro/ENGINEER die Masseneigenschaften der einzelnen Gelenke berechnet werden. Die Tabellen 5.3(a) und 5.3(b) zeigen die Positionen der Massenschwerpunkte in den Beinen. Die angegebenen Koordinaten sind immer relativ zum Koordinatensystem des jeweiligen Gelenks

Zusätzlich zu den Beinen wurden noch die Masseneigenschaften des restlichen Roboters benötigt. Der Oberkörper wird noch nicht für Bewegungen genutzt, daher können die restlichen Teile des Roboters vereinfacht als ein Massenpunkt betrachtet werden. Dies setzt jedoch voraus, dass alle Gelenke des Oberkörpers in ihrer Ausgangsposition bleiben. Ansonsten wäre das hier entwickelte Stabilitätskriterium nicht mehr aktuell. Der Massenschwerpunkt des Oberkörpers, relativ zum Koordinatensystem UKKS, ist in Tabelle 5.3(c) zu sehen.

Um Formel (2.1) anzuwenden, ist es weiterhin notwendig alle Massenschwerpunkte in ein gemeinsames Koordinatensystem zu transformieren. Dazu können die Transformationsmatrizen der direkten Kinematik verwendet werden. Für jedes Gelenk $i$ muss die entsprechende Transformationsmatrix ${}^{0}_{i}T$ berechnet werden. Zusätzlich müssen die Massenschwerpunkte aus dem Beinbezugskoordinatensystem in das Unterkörperkoordinatensystem transformiert werden. Die Masseneigenschaften des Oberkörpers ($m_{ok}$ und $\vec{s}_{ok}$) wurden bereits relativ zum UKKS berechnet und müssen daher nicht mehr transfor-

miert werden. Die Formel (2.1) kann zur Berechnung des Massenschwerpunktes $\vec{s}$ somit umgeformt werden zu:

$$\vec{s} = \frac{m_{ok} \cdot \vec{s}_{ok} + \sum_{i=1}^{n} m_i \cdot ({}^{UKKS}_{BBKS}T \cdot {}^{0}_{i}T \cdot \vec{s}_i)}{m_{ok} + \sum_{i=1}^{n} m_i} \tag{5.2}$$

Tabelle 5.3: Masseneigenschaften

(a) linkes Bein

| Joint | Masse (g) | COM x (mm) | COM y (mm) | COM z (mm) |
|---|---|---|---|---|
| 1 | 92,7215300 | -4,4234198 | -14,0188990 | 0,3597805 |
| 2 | 15,3023720 | 1,2132379 | -6,4567850 | 11,8765470 |
| 3 | 164,5129700 | 14,9749620 | -45,5000000 | 0,0000000 |
| 4 | 29,8616050 | 4,5686886 | -43,2604700 | -1,7063145 |
| 5 | 86,5480690 | -2,7180418 | 13,5969630 | -5,2014226 |
| 6 | 179,8308000 | 2,8558632 | -17,3958130 | 0,1439557 |

(b) rechtes Bein

| Joint | Masse (g) | COM x (mm) | COM y (mm) | COM z (mm) |
|---|---|---|---|---|
| 1 | 92,7215300 | -4,4234198 | 14,0188990 | 0,3597805 |
| 2 | 15,3023720 | -1,2132379 | -6,4567850 | 11,8765470 |
| 3 | 164,5129700 | -14,9749620 | -45,5000000 | 0,0000000 |
| 4 | 29,8616050 | -4,5686886 | -43,2604700 | -1,7063145 |
| 5 | 86,5480690 | -2,7180418 | 13,5969630 | 5,2014226 |
| 6 | 179,8308000 | 2,8558632 | -17,3958130 | -0,1439557 |

(c) Oberkörper

| Masse (g) | COM x (mm) | COM y (mm) | COM z (mm) |
|---|---|---|---|
| 1879,0305000 | -5.5616935 | -0.26180284 | 111.41225 |

### 5.2.2 Ermitteln des Stützpolygons der Füße

Das Stützpolygon bezeichnet das konvexe Polygon, das von den Beinen des Roboters aufgespannt wird. Für das Stützpolygon sind nur die Beine von Interesse, die auch tatsächlich Bodenkontakt haben. Um statische Stabilität zu erhalten ist es notwendig, die vertikale Projektion des Massenschwerpunktes (COG) immer in diesem Stützpolygon zu halten. Bei vier- oder sechsbeinigen Laufrobotern ist es leicht ein ausreichend großes Stützpolygon zu erhalten, wenn sich immer mindestens drei Beine auf dem Boden befinden.

Bei einem zweibeinigen Laufroboter hingegen können nur maximal zwei Beine Bodenkontakt haben, während dem Laufen oftmals nur eines. Es genügt daher nicht, wie bei vier- oder sechsbeinigen Laufrobotern üblich, jedes Bein als einen Punkt im Stützpolygon aufzufassen. Um überhaupt Laufen bei zweibeinigen Laufrobotern zu ermöglichen, müssen die Füße mit verhältnismäßig großen Kontaktflächen konstruiert werden. Das Stützpolygon eines zweibeinigen Laufroboters kann und muss daher durch die Eckpunkte der Kontaktflächen der beiden Füße gebildet werden, wie in Abb. 5.5 zu sehen ist.

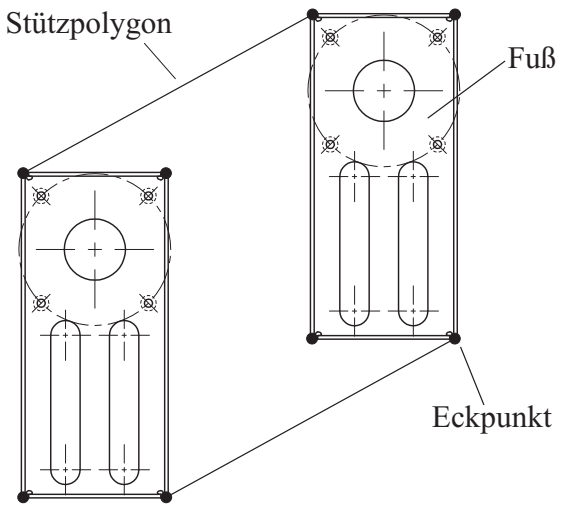



Abbildung 5.5: Stützpolygon der Füße

Bevor die Eckpunkte allerdings für weitere Berechnungen verwendet werden können, müssen diese in ein gemeinsames Koordinatensystem transformiert werden. Tabelle 5.4 zeigt die Position der vier Eckpunkte, relativ zum jeweiligen Koordinatensystem des Fußes. Da in beiden Füßen das Koordinatensystem die gleiche Position hat, sind auch die Koordinaten der vier Punkte gleich.

| Eckpunkt | x (mm) | y (mm) |
|---|---|---|
| 1 | 98,5 | -27,0 |
| 2 | 98,5 | 27,0 |
| 3 | -28,5 | 27,0 |
| 4 | -28,5 | -27,0 |

Tabelle 5.4: Position der Eckpunkte im rechten und linken Fuß

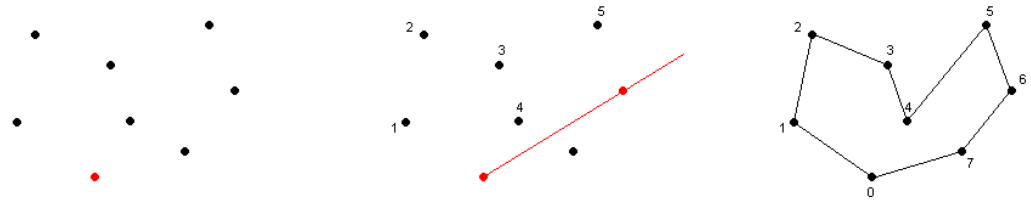
Da der Massenschwerpunkt bereits im UKKS berechnet wurde, liegt es nahe auch die Eckpunkte in das UKKS zu transformieren und die Berechnung für das Stützpolygon dort durchzuführen. Für die Stabilitätsanalyse genügt es, den COG - also die Projektion des Massenschwerpunktes auf die horizontale Ebene - zu betrachten. Da auch das Stützpolygon eine Fläche auf der horizontalen Ebene ist, können die folgenden Berechnungen für das Stützpolygon und den Punkt-In-Polygon Test auf ein zweidimensionales Problem reduziert werden.

Wenn das Stützpolygon aus den Eckpunkten der Kontaktflächen bestimmt werden soll, ergeben sich allerdings einige Spezialfälle, die beachtet werden müssen. Zum einen ist das von allen Eckpunkten aufgespannte Polygon oftmals noch nicht konvex. Für die Stabilität ist allerdings nur die konvexe Hülle des Polygons von Interesse. Einige der Eckpunkte müssen daher entfernt werden, um eine konvexe Hülle zu erhalten. In Abb. 5.5 ist bereits zu erkennen, dass einige Punkte nicht mehr für das Stützpolygon relevant sind. Welche Punkte dies sind, hängt von den Positionen der beiden Kontaktflächen ab. Weitere Spezialfälle ergeben sich, da sich die beiden Kontaktflächen um jeweils 150° in beide Richtungen verdrehen können. Es kann somit keine allgemeingültige Aussage darüber gemacht werden, in welcher Reihenfolge die Eckpunkte zum Aufspannen des Stützpolygons verwendet werden müssen. Obwohl eine so extreme Verdrehung der Füße in der Praxis eher selten der Fall sein wird, ist es für einen korrekten Algorithmus notwendig, diese Fälle zu beachten.

Aufgrund der beiden genannten Spezialfälle wird also ein Algorithmus benötigt, der zunächst aus einer Menge von Punkten ein Polygon erstellt und danach die konvexe Hülle des Polygons bildet. Hierfür bietet sich der aus der Bildverarbeitung bekannte Graham Algorithmus an, da er alle geforderten Eigenschaften erfüllt. Nachfolgend wird eine kurze Einführung in die Funktionsweise gegeben[1].

[1] Im Buch „Computerorientierte Geometrie“ [2] wird der Algorithmus ausführlich erläutert.

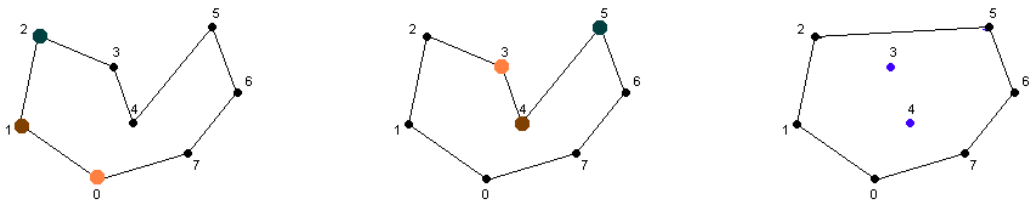
Zu Beginn muss ein Extrempunkt gefunden werden (Abb. 5.6(a)). Dies kann beispielsweise der Punkt mit dem kleinsten y Wert sein. Die restlichen Punkte müssen, ausgehend von dem Extrempunkt, sternförmig sortiert werden (Abb. 5.6(b)). Nun sind die Punkte bereits so sortiert, dass sich ein Polygon ergibt (Abb. 5.6(c)).




(a) Extrempunkt (b) Punkte sortieren (c) Polygon

Abbildung 5.6: Graham Algorithmus - Polygon bilden

Es ist allerdings hierbei noch nicht sichergestellt, dass das so erhaltene Polygon bereits konvex ist. Um dies zu erreichen, werden nacheinander immer drei folgende Punkte betrachtet, angefangen bei dem zuvor gewähltem Extrempunkt. Bilden die drei Punkte eine Rechtskurve (Punkte 0,1,2), wie in Abb. 5.7(a), oder sind sie kollinear, kann mit der Betrachtung der nächsten Dreierfolge (Punkte 1,2,3) weitergemacht werden. Sollten die drei Punkte allerdings, wie in Abb. 5.7(b), eine Linkskurve bilden (Punkte 3,4,5), muss der mittlere Punkt entfernt werden. Die somit neu entstandene Dreierfolge (Punkte 2,3,5) muss erneut getestet werden. Nach dieser Methode müssen alle Dreierfolgen überprüft werden, bis der Extrempunkt wieder erreicht ist. Die Liste mit Punkten, welche nach einem Durchgang noch übrig sind, entspricht der konvexen Hülle der zuvor angegebenen Punktmenge (Abb. 5.7(c)).




(a) Rechtskurve (b) Linkskurve (c) Konvexe Hülle

Abbildung 5.7: Graham Algorithmus - Konvexe Hülle

Es ist jedoch nur dann nötig, den Graham Algorithmus anzuwenden, wenn beide Beine Bodenkontakt haben. Sollte nur ein Bein auf dem Boden stehen, kann das Stützpolygon relativ einfach bestimmt werden: Das von den vier Eckpunkten aufgespannte Polygon ist immer konvex, daher entfällt die Suche nach der konvexen Hülle. Außerdem kann sich die Position der Eckpunkte zueinander nicht ändern, daher ist es auch leicht möglich die Reihenfolge der Punkte festzulegen, mit der das Polygon aufgespannt werden muss. Die oben genannten Spezialfälle sind somit in diesem Fall nicht relevant.

### 5.2.3 Punkt-in-Polygon Test

Voraussetzung für die statische Stabilität eines Laufroboters ist es, den COG stets innerhalb des Stützpolygons der Beine zu halten. In den beiden vorhergehenden Kapiteln wurde beschrieben, wie der COG berechnet und das Stützpolygon bestimmt werden kann. Letzter Schritt zur Bestimmung der Stabilität ist die Überprüfung, ob der COG innerhalb des von den Beinen aufgespannten Stützpolygons liegt.

In der Computergrafik gibt es bereits ein verbreitetes Verfahren, mit dem überprüft werden kann, ob ein Punkt in einem Polygon liegt [2]. Dazu wird, ausgehend von dem zu überprüfenden Punkt, in eine beliebige Richtung eine Linie, auch „Scan Line“ genannt, gebildet. Entlang dieser gedachten Linie werden alle Kreuzungen mit Polygonkanten gezählt. Bei einer geraden Anzahl von Kreuzungen liegt der Punkt außerhalb des Polygons, bei einer ungeraden Anzahl innerhalb.

Mit diesem Verfahren ist es möglich herauszufinden, ob der Roboter noch statisch stabil ist oder nicht. Für einen Laufalgorithmus ist es allerdings vorteilhaft, genauere Informationen über die relative Position des COG zu den Kanten des Stützpolygons zu haben. Je näher der COG an einer Kante liegt, umso empfindlicher wird der Roboter gegenüber äußeren Einflüssen und dynamischen Kräften. Hinzu kommt, dass eventuelle Ungenauigkeiten bei der Berechnung des Massenschwerpunktes und des Stützpolygons den Roboter schon zum Fallen bringen können, obwohl er nach dem Stabilitätskriterium noch stabil sein müsste.

Bei der Planung des Laufmusters sollte daher darauf geachtet werden, dass der COG immer möglichst nahe an der Mitte des Stützpolygons liegt und somit eine Stabilitätsreserve gegeben ist. Um die dafür nötigen Informationen zu erhalten, wurde im Verlauf dieser Arbeit ein modifizierter Algorithmus für den Punkt-in-Polygon Test entwickelt.

Ausgehend vom COG werden in vier Richtungen Linien gebildet. Jeweils eine in positiver und negativer Richtung parallel zur x-Achse sowie in positiver und negativer Richtung parallel zur y-Achse (Abb. 5.8). Die Endpunkte der Linien müssen so gewählt werden, dass sie garantiert außerhalb des Stützpolygons liegen und somit in jedem Fall die Polygonkanten kreuzen wenn der COG innerhalb des Polygons liegt. Entlang jeder dieser Linien wird nach einer Kreuzung mit einer Polygonkante gesucht und der Abstand zu dieser bestimmt. Sollte auf mindestens einer dieser Linien keine Kreuzung gefunden werden, liegt der Punkt außerhalb. Da das Stützpolygon in jedem Fall konvex ist, liegt der COG innerhalb des Polygons, wenn auf jeder Linie eine Kreuzung gefunden wurde. Mit Hilfe der Abstände kann dann genau bestimmt werden, wie nahe der COG auf der x-Achse und y-Achse an der Mitte ist.

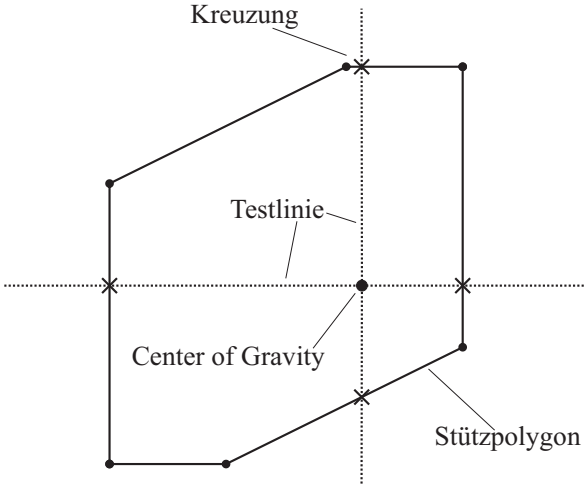



Abbildung 5.8: Erweiterter Punkt-in-Polygon Test

## 5.3 Planung der Beintrajektorien

Zur Planung der Beintrajektorien müssen Stützpunkte im Raum definiert werden, welche die gewünschten Beintrajektorien ausreichend beschreiben. Hierzu genügt es, die Position und Orientierung der Füße festzulegen. Es müssen dabei nur die wichtigsten Extremwerte der Trajektorien vorgegeben werden, mit Hilfe einer Interpolation können daraus später ausreichend Zwischenpunkte für eine flüssige Bewegung bestimmt werden. Die entsprechenden Stellungen der Gelenke werden nach der Interpolation mit der inversen Kinematik berechnet.

Um einen stabilen Laufalgorithmus zu erzeugen, muss jeder geplante Stützpunkt das gewählte Stabilitätskriterium erfüllen. Im Fall dieser Arbeit wird hierzu das im vorhergehenden Kapitel entwickelte statische Stabilitätskriterium verwendet. Um dieses zu erfüllen, muss der COG stets innerhalb des Stützpolygons der Füße gehalten werden. Es ist weiterhin empfehlenswert, den COG möglichst nahe in der Mitte bzw. weit entfernt von den Kanten zu halten. Hierdurch kann eine optimale Stabilität erreicht werden; zusätzlich ergeben sich Spielräume für die spätere Interpolation.

Desweiteren müssen mechanische Beschränkungen der Beine beachtet werden. Hierzu ist es notwendig zu überprüfen, ob die in der inversen Kinematik berechneten Gelenkstellungen innerhalb der durch die Konstruktion vorgegebenen Beschränkungen liegen. Da bei der Interpolation unter Umständen Zwischenschritte entstehen können, welche die mechanischen Beschränkungen überschreiten, sollten nicht nur die Stützpunkte sondern auch die interpolierten Zwischenschritte überprüft werden.

Im Allgemeinen ist es zu empfehlen, auch die Beschleunigungen und Drehmomente in den Gelenken zu analysieren, damit die Toleranz der Motoren nicht überschritten wird. Da mit dem hier entwickelten statisch stabilen Laufalgorithmus wegen der auftretenden dynamischen Kräfte jedoch keine hohen Geschwindigkeiten zu erwarten sind, wurde auf diese Kontrolle verzichtet. Bei der Konstruktion des Roboters [21] wurden die Motoren bereits so bestimmt, dass eine Laufgeschwindigkeit ähnlich der des Menschen unter Beachtung der Größenunterschiede möglich ist. Eine solch hohe Laufgeschwindigkeit kann mit einem statisch stabilen Laufalgorithmus nicht erreicht werden. Somit ist auch nicht damit zu rechnen, dass Beschleunigungen und Drehmomente die Werte der Motoren übersteigen.

Im ersten Teil dieses Kapitels wird das verwendete Laufmuster genauer erläutert. Bei der Entwicklung des Laufalgorithmus wurde besonderen Wert darauf gelegt, das Laufmuster mit Parametern leicht anpassen zu können. Der zweite Teil erklärt daher, wie aus gegebenen Parametern die entsprechenden Stützpunkte des Laufmusters berechnet werden können. Im dritten Teil werden Probleme in der Konstruktion der Roboterbeine beschrieben, welche es schwierig machen einen optimalen Laufalgorithmus zu entwickeln. Die Ergebnisse dieses Teils können als Anstoß für Verbesserungen im Rahmen nachfolgender Arbeiten dienen.

### 5.3.1 Grundlegendes Laufmuster

Abb. 5.9 zeigt alle Zwischenschritte des entwickelten Laufmusters für insgesamt zwei Schritte. Ein Schritt kann in vier Phasen zerteilt werden:

1. Gewicht auf ein Bein verlagern (c)&(h)
2. Das andere Bein anheben und den Oberkörper nach vorne schieben (d)&(i)
3. Den Oberkörper weiter nach vorne schieben und das Bein wieder absetzt (e)&(j)
4. Gewicht zurück in die Mitte verlagern (k)&(p)

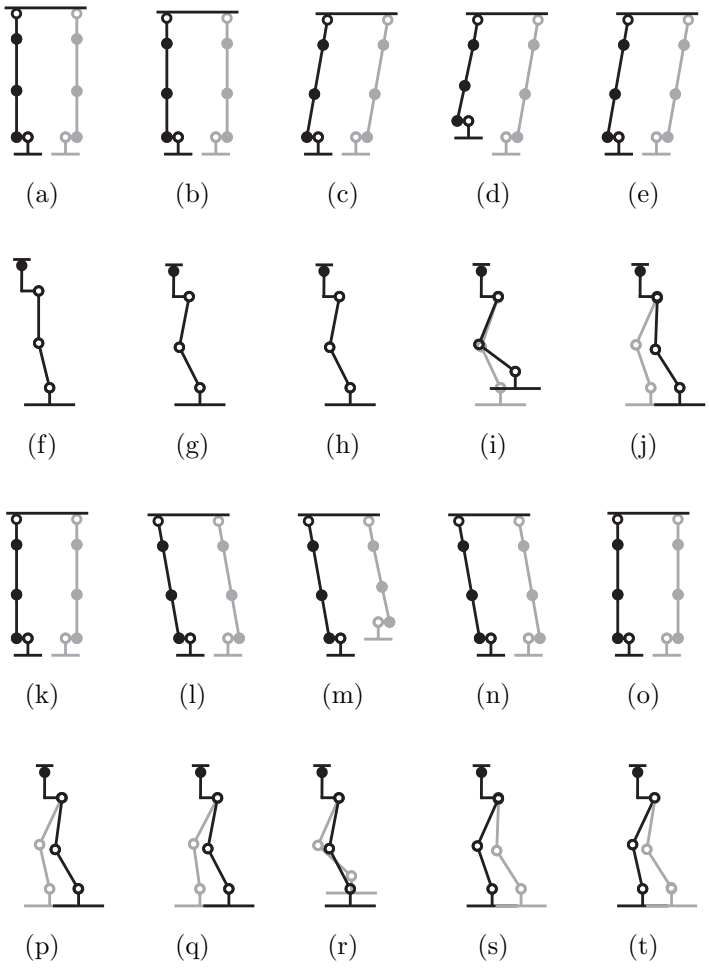



Abbildung 5.9: Das entwickelte Laufmuster: (a) - (e) und (k) - (o) zeigen die Frontalansicht des Roboters. (f) - (j) und (p) - (t) zeigen die Ansicht von der rechten Seite.

Der zweite Schritt läuft analog, nur wird diesmal das andere Bein angehoben. Phase 2 und 3 entsprechen der Single Support Phase (vergleiche Kapitel 2.2), in der sich nur ein Bein am Boden befindet. Vor dem ersten Schritt ist es empfehlenswert, den Roboter ein wenig abzusenken (Abb. 5.9(b)&(g)). Dies gibt den Beinen einen größeren Spielraum und ermöglicht somit größere Schrittlängen.

Die statische Stabilität in der Single Support Phase ist gegeben, da vor dem Anheben eines Beins das Gewicht auf das jeweils andere Bein verlagert wird. Dies wird erreicht, indem der Oberkörper so weit seitlich verschoben wird, dass der COG über der Kontaktfläche des entsprechenden Fußes liegt. Die Double Support Phase wird dazu genutzt, den COG von der Kontaktfläche des einen Fußes auf den anderen zu verschieben. Um auch hier die Stabilität nicht zu verlieren, muss darauf geachtet werden, dass der COG das Stützpolygon während der Gewichtsverlagerung nicht verlässt.

Auffällig bei oben gezeigtem Laufmuster ist das nach hinten einknickende Knie. Dies erinnert mehr an einen vogelähnlichen als an einen menschenähnlichen Gang, kann aber aufgrund der verwendeten numerischen Lösung der inversen Kinematik (vergleiche Kapitel 3.2.3) nicht verhindert werden. Wenn das Knie in der Grundposition nach vorne einknickt, würde sich das Bein zunächst aufgrund der Konstruktion ein wenig verlängern. Da die numerische Lösung der inversen Kinematik nur der Tangente in diesem Punkt entspricht, ergibt sich bei einem nach vorne einknickenden Knie somit eine stetige Verlängerung des Beins, bei einem Knick nach hinten dagegen eine Verkürzung. Daher ergibt auch ein verkürztes Bein in der inversen Kinematik als Lösung einen Knick nach hinten.

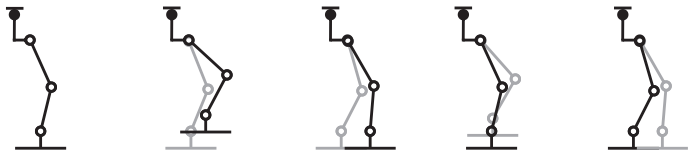



Abbildung 5.10: Modifiziertes Laufmuster

Um dennoch ohne Änderung der Konstruktion ein nach vorne einknickendes Bein zu erhalten, kann die Grundposition der Gelenke vor dem Start des Laufmusters entsprechend modifiziert werden. Hierfür muss das Kniegelenk bei gleichbleibender Position des Oberkörpers soweit nach vorne verschoben werden, dass die Drehachse des Knies vor der Drehachse des Fußes liegt (Abb. 5.10(a)). Dann ist oben genannter Fall genau um-

gedreht und eine Bewegung des Knies nach hinten würde eine Verlängerung des Beins ergeben. Somit ergibt die Lösung der inversen Kinematik einen Knick nach vorne. Das entstehende Laufmuster ist in Abb. 5.10 zu sehen.

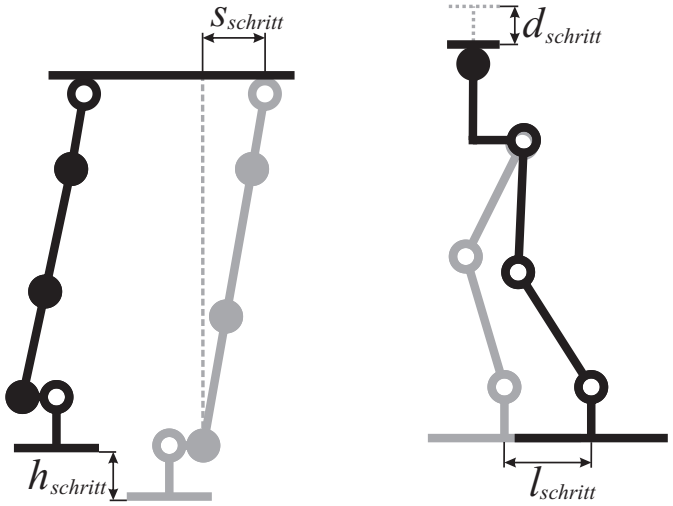
### 5.3.2 Bestimmen der Stützpunkte nach gegebenen Parametern

Die Möglichkeit, das Laufmuster mit Parametern anzupassen, erleichtert spätere praktische Versuche mit dem Laufalgorithmus. Insbesondere wegen des Einflusses dynamischer Kräfte, aber auch aufgrund anderer Faktoren, wie beispielsweise dem Spiel in den Motorgetrieben oder Ungenauigkeiten in den Berechnungen, ist es unmöglich, ohne praktische Versuche optimale Werte für Geschwindigkeit, Schrittweite, -höhe und andere Eigenschaften zu bestimmen. Somit ist es unerlässlich, mit späteren praktischen Versuchen die Leistungsfähigkeit des Laufalgorithmus zu überprüfen und das Laufmuster zu optimieren. Desweiteren können die Parameter auch verwendet werden, um während der Laufzeit das Laufmuster anzupassen. So kann beispielsweise die Schritthöhe zum Überschreiten eines Hindernisses vergrößert werden.

Der hier entwickelte Laufalgorithmus erlaubt eine Modifikation des Laufmusters mit folgenden fünf Parametern:

- Die Schritthöhe $h_{schritt}$ gibt den höchsten Punkt an, welchen das in der Luft befindliche Bein erreicht (Abb. 5.11(a)).
- Die Schrittlänge $l_{schritt}$ legt den maximalen Abstand in Laufrichtung zwischen den beiden Füßen fest (Abb. 5.11(b)).
- Der Parameter $s_{schritt}$ gibt den Wert an, um den der Oberkörper zur Gewichtsverlagerung zur Seite verschoben wird (Abb. 5.11(a)).
- Der Parameter $d_{schritt}$ legt die Höhe fest, um die der Oberkörper abgesenkt wird (Abb. 5.11(b)).
- Die Schrittdauer $t_{schritt}$ gibt an, wie lange ein Schritt dauert.

In dem beschriebenen Laufmuster bleiben die Füße zu jedem Zeitpunkt parallel zum Boden, somit entstehen keine Rotationen um die x- oder y-Achse. Desweiteren soll das Laufmuster zunächst auf eine gerade Vorwärtsbewegung beschränkt werden, daher ist es auch nicht notwendig, einen der Füße um die z-Achse zu rotieren. Hieraus ergibt sich, dass in jedem Stützpunkt die Orientierung entlang der drei Achsen immer unverändert bleibt.




(a) (b)

Abbildung 5.11: Parameter für Laufmuster

Die Position des $i$-ten Stützpunktes der Beine relativ zum jeweiligen BBKS wird nachfolgend mit $x_{L,i}, y_{L,i}, z_{L,i}$ für das linke Bein und $x_{R,i}, y_{R,i}, z_{R,i}$ für das rechte Bein bezeichnet.

Wie zuvor erwähnt ist es empfehlenswert, den gesamten Roboter ein wenig abzusenken, um die Reichweite der Beine zu erhöhen. Wie weit abgesenkt wird, kann mit dem Parameter $d_{schritt}$ bestimmt werden. Für den ersten Stützpunkt ergibt sich somit:

$$x_{L,0} = -7,7 \qquad x_{R,0} = -7,7 \tag{5.3}$$

$$y_{L,0} = -9,5 \qquad y_{R,0} = 9,5 \tag{5.4}$$

$$z_{L,0} = -266,5 + d_{schritt} \qquad z_{R,0} = -266,5 + d_{schritt} \tag{5.5}$$

Die angegebenen Werte ergeben sich aus der Position der Füße relativ zum jeweiligen BBKS in der Grundstellung.

In Phase 1 des Laufmusters müssen die Füße um den entsprechenden Wert entlang der y-Achse verschoben werden:

$$y_{L,1} = y_{L,0} - s_{schritt} \qquad y_{R,1} = y_{R,0} - s_{schritt} \tag{5.6}$$

Die restlichen Werte werden vom vorhergehenden Schritt übernommen.

In Phase 2 wird das rechte Bein angehoben. Zusätzlich muss der Oberkörper nach vorne bewegt werden. Hierzu wird das linke Bein nach hinten verschoben und das rechte Bein nach vorne. Da beim ersten Schritt beide Füße noch nebeneinander stehen, darf das

rechte Bein nur um die Hälfte der Schrittlänge $l_{schritt}$ nach vorne bewegt werden. Für die zweite Phase ergeben sich somit folgende Formeln:

$$x_{L,2} = x_{L,1} - \frac{l_{schritt}}{4} \qquad x_{R,2} = x_{R,1} + \frac{l_{schritt}}{4} \tag{5.7}$$

$$y_{R,2} = y_{R,1} + \left( \frac{s_{schritt}}{266,5 - d_{schritt}} \cdot h_{schritt} \right) \tag{5.8}$$

$$z_{R,2} = z_{R,1} + h_{schritt} \tag{5.9}$$

In Phase 3 wird der Oberkörper weiter nach vorne geschoben, damit dieser weit genug über der Kontaktfläche des rechten Beins ist und somit das Gewicht im nächsten Schritt auf dieses Bein verlagert werden kann. Außerdem muss das rechte Bein wieder abgesetzt werden:

$$x_{L,3} = x_{L,2} - \frac{l_{schritt}}{4} \qquad x_{R,2} = x_{R,1} + \frac{l_{schritt}}{4} \tag{5.10}$$

$$y_{R,3} = y_{R,2} - \left( \frac{s_{schritt}}{266,5 - d_{schritt}} \cdot h_{schritt} \right) \tag{5.11}$$

$$z_{R,3} = z_{R,2} - h_{schritt} \tag{5.12}$$

In der letzten Phase wird das Gewicht wieder in die Mitte verlagert:

$$y_{L,4} = y_{L,3} + s_{schritt} \qquad y_{R,4} = y_{R,3} + s_{schritt} \tag{5.13}$$

Im nächsten Schritt müssen die Formeln des linken und rechten Beins vertauscht werden. Zusätzlich muss das Vorzeichen des Parameters $s_{schritt}$ geändert werden, um eine Gewichtsverlagerung auf das rechte Bein zu erreichen.

Für einen beliebigen Schritt $j$ ergeben sich somit folgende Formeln für die Berechnung der Stützpunkte:

Phase 1 ($i = j * 4 - 3$)

$$x_{B1,i} = x_{B1,i-1} \qquad x_{B2,i} = x_{B2,i-1} \tag{5.14}$$

$$y_{B1,i} = y_{B1,i-1} + (-1)^j \cdot s_{schritt} \qquad y_{B2,i} = y_{B2,i-1} + (-1)^j \cdot s_{schritt} \tag{5.15}$$

$$z_{B1,i} = z_{B1,i-1} \qquad z_{B2,i} = z_{B2,i-1} \tag{5.16}$$

Phase 2 ($i = j * 4 - 2$)

$$x_{B1,i} = x_{B1,i-1} - \frac{l_{schritt}}{2} \qquad x_{B2,i} = x_{B2,i-1} + \frac{l_{schritt}}{2} \tag{5.17}$$

$$y_{B1,i} = y_{B1,i-1} \qquad y_{B2,i} = y_{B2,i-1} + \left( \frac{(-1)^j \cdot s_{schritt}}{266,5 - d_{schritt}} \cdot h_{schritt} \right) \tag{5.18}$$

$$z_{B1,i} = z_{B1,i-1} \qquad z_{B2,i} = z_{B2,i-1} + h_{schritt} \tag{5.19}$$

Phase 3 ($i = j * 4 - 1$)

$$x_{B1,i} = x_{B1,i-1} - \frac{l_{schritt}}{2} \qquad x_{B2,i} = x_{B2,i-1} + \frac{l_{schritt}}{2} \tag{5.20}$$

$$y_{B1,i} = y_{B1,i-1} \qquad y_{B2,i} = y_{B2,i-1} + \left( \frac{(-1)^{j+1} \cdot s_{schritt}}{266,5 - d_{schritt}} \cdot h_{schritt} \right) \tag{5.21}$$

$$z_{B1,i} = z_{B1,i-1} \qquad z_{B2,i} = z_{B2,i-1} - h_{schritt} \tag{5.22}$$

Phase 4 ($i = j * 4$)

$$x_{B1,i} = x_{B1,i-1} \qquad x_{B2,i} = x_{B2,i-1} \tag{5.23}$$

$$y_{B1,i} = y_{B1,i-1} + (-1)^{j+1} \cdot s_{schritt} \qquad y_{B2,i} = y_{B2,i-1} + (-1)^{j+1} \cdot s_{schritt} \tag{5.24}$$

$$z_{B1,i} = z_{B1,i-1} \qquad z_{B2,i} = z_{B2,i-1} \tag{5.25}$$

Die Indizes $B1$ und $B2$ entsprechen abwechselnd dem linken und dem rechten Bein. Für die $x$ Koordinate ergibt sich somit beispielhaft:

$$x_{B1,i} := \begin{cases} x_{R,i} & \text{für } j \text{ gerade} \\ x_{L,i} & \text{für } j \text{ ungerade} \end{cases} \tag{5.26}$$

$$x_{B2,i} := \begin{cases} x_{L,i} & \text{für } j \text{ gerade} \\ x_{R,i} & \text{für } j \text{ ungerade} \end{cases} \tag{5.27}$$

Beim letzten Schritt sollte wie beim ersten die Schrittweite halbiert werden. Somit ist sichergestellt, dass am Ende beide Beine wieder nebeneinander stehen.

Der Parameter $t_{schritt}$ wird nicht für die Berechnung der Stützpunkte benötigt. Stattdessen fließt dieser in die spätere Interpolation ein und entscheidet darüber, wieviele Zwischenpunkte zu berechnen sind. Desweiteren spielt es für die Berechnung der Stützpunkte keine Rolle, welches Laufmuster - nach vorne oder hinten einknickendes Knie - verwendet wird. Es muss lediglich darauf geachtet werden, dass der Parameter $d_{schritt}$ im Falle des modifizierten Laufmusters nicht zu klein gewählt wird. Ansonsten könnte das

Knie zu weit nach hinten rutschen und somit wieder in das ursprüngliche Laufmuster verfallen.

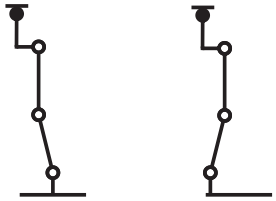
Die Wahl der Parameter ist entscheidend für die Funktionsfähigkeit des Laufalgorithmus: Sollte beispielsweise bei einer relativ geringen Absenkung des Oberkörpers $d_{schritt}$ eine große Schrittweite $l_{schritt}$ verwendet werden, kann es sein, dass die berechneten Beintrajektorien außerhalb des Arbeitsraums liegen. Wenn andererseits $d_{schritt}$ und die Schritthöhe $h_{schritt}$ zu groß gewählt werden, so kann dies die mechanischen Beschränkungen im Kniebereich überschreiten. Der Parameter $s_{schritt}$ wiederum hat großen Einfluss auf die Stabilität: Sollte dieser zu klein oder zu groß gewählt werden, liegt der Massenschwerpunkt nicht mehr über der Kontaktfläche des stützenden Fußes. Aus diesem Grund sollten die berechneten Stützpunkte vor einem praktischen Versuch interpoliert und mit der inversen Kinematik umgewandelt werden. Die so erhaltenen Trajektorien können dann dahingehend überprüft werden, ob diese noch innerhalb der mechanischen Beschränkungen und der statischen Stabilität liegen.

### 5.3.3 Probleme in der Konstruktion des Roboters

Während der Planung des Laufmusters haben sich insbesondere zwei Eigenschaften der Beinkonstruktion als ungünstig herausgestellt: Zum einen ist dies, wie bereits zuvor erwähnt, das nach hinten einknickende Knie. Durch die Biegung im unteren Bereich der Beine ergeben die Lösungen der inversen Kinematik ein nach hinten einknickendes Knie. Dies wird durch die Lösungsmethode auf Basis der Jacobimatrix (vgl. Kapitel 3.2.3) verursacht. Hierbei wird mit Hilfe einer linearen Annäherung die Lösung gesucht. Durch die Biegung im Bein entsteht, wenn das Knie nach vorne eingeknickt wird, eine kleine Verlängerung des Beins. Die in diesem Punkt angelegte Tangente hat somit auch die Eigenschaft, nur bei einem nach hinten einknickenden Knie das Bein zu verkürzen. Daher ergibt diese Lösungsmethode der inversen Kinematik bei Verkürzung des Beins ein nach hinten einknickendes Knie. Eine andere Methode ist aufgrund der kinematischen Komplexität der Bein nicht möglich.

Wie bereits in Kapitel 5.3.1 erklärt, kann dieses Problem durch eine abgeänderte Ausgangsposition des Roboters umgangen werden. Allerdings darf in diesem Fall das Bein nicht zu weit gestreckt werden, da ansonsten das Knie wieder in einen Knick nach hinten verfallen würde. Besser wäre es daher, die Biegung im Bein entsprechend abzuändern. Die Biegung ganz zu entfernen, ist hingegen keine gute Lösung. Alle drei Achsen liegen dann übereinander und es ist nicht mehr vorherzusehen, in welche Richtung das Knie

einknickt. Die beste Lösung wäre daher, die Biegung nur umzudrehen. Um die Position des Massenschwerpunktes relativ zum Stützpolygon nicht zu beeinflussen, sollte auch die Position der Achse im Fuß entsprechend nach hinten verlegt werden (Abb. 5.12(b) ).



(a) Original (b) Modifiziert

Abbildung 5.12: Änderungsvorschlag der Biegung im Bein.

Ein weiteres Problem entsteht durch den Versatz zwischen den beiden für die seitliche Verschiebung verantwortlichen Achsen (Abb. 5.13). Um ein zum Boden paralleles Verschieben des Oberkörpers zu ermöglichen, müssten die Verbindungslinien zwischen den Drehachsen ein Parallelogramm bilden. Da dies nicht der Fall ist, würde eine Verschiebung in eine Richtung das Bein der entgegengesetzten Seite länger werden lassen als das andere Bein. Auf die Berechnung des Laufmusters hat dies keinen Einfluss, die Positionen der Füße sind so vorgegeben, dass sich kein Bein verlängert. Allerdings ergibt die inverse Kinematik in diesem Fall eine Verkürzung des entsprechenden Beins, um die ungleiche Länge beider Beine auszugleichen. Dies führt zu unnötigen Bewegungen in den Kniegelenken der Beine.

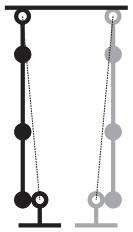


Abbildung 5.13: Achsversatz

Um dieses Problem auszuschließen, wäre eine Änderung in der Konstruktion zu empfehlen. Hierzu müsste entweder der obere oder der untere Motor der entsprechenden Drehachsen soweit verschoben werden, dass beide Achsen keinen Versatz mehr zueinander besitzen.

# 6 Implementierung des Laufalgorithmus

Diese Kapitel beschreibt die Umsetzung des im vorhergehenden Kapitel vorgestellten Konzepts. Der Laufalgorithmus wurde in C++ programmiert, um zum einen die Umsetzung des Programms auf den PDA zu ermöglichen und zum anderen um eine höchstmögliche Geschwindigkeit der Berechnungen zu erreichen.

Für kinematische Berechnungen sind bereits Bibliotheken in C++ vorhanden. Im ersten Teil des Kapitels werden zunächst die für die Berechnungen verwendeten Bibliotheken dargestellt. Dieser Teil beschreibt auch eine an der Hochschule Mannheim entwickelte Motorschnittstelle, welche für die Steuerung der Motoren bei der späteren Umsetzung des Laufalgorithmus auf den Roboter verwendet werden soll.
Der zweite Teil beschreibt die im Umfang dieser Arbeit entwickelte `Humanoid` Klasse. Die Klasse enthält alle für die Kinematik nötigen Berechnungen. Desweiteren sind auch die Methoden für die Stabilitätsanalyse in dieser Klasse enthalten.
Die Berechnung des Laufmusters und die Interpolation werden im dritten Teil beschrieben. Für die so gewonnenen Daten wurde zusätzlich ein Programm zur Auswertung der Trajektorien im Bezug auf Stabilität und mechanische Beschränkungen erstellt.
Im letzten Teil werden Maßnahmen beschrieben, die nötig sind, um das Programm auf den verwendeten PDA zu portieren.

## 6.1 Verwendete Komponenten

Nachfolgend werden die bei der Implementierung verwendeten Komponenten erläutert. ROBOOP wurde für die kinematischen Berechnungen verwendet, Newmat ist eine Bibliothek, welche die Arbeit mit Matrizen vereinfacht und die Motorschnittstelle wird zum ansteuern der Motoren verwendet.

### 6.1.1 ROBOOP - Robotics Object Oriented Package

Die ROBOOP[1] Bibliothek ist eine Sammlung von Algorithmen für die Simulation von Industrierobotern. Viele der verwendeten Algorithmen, beispielsweise für die Kinematik, basieren allerdings auf den selben Konzepten wie bei Laufrobotern. Daher bietet sich diese Bibliothek auch für die Entwicklung von Laufrobotern an. ROBOOP wurde bereits seit 1996 bis heute stetig weiterentwickelt. Die verwendeten Algorithmen sind somit schon vielmals getestet und optimiert worden. Desweiteren ist die Bibliothek objektorientiert aufgebaut und deshalb leicht in neue oder bestehende Projekte einzubinden.

Der Einstiegspunkt für die kinematischen Berechnungen ist die Klasse `Robot`. Diese repräsentiert eine kinematische Kette. Um entsprechende Berechnungen durchführen zu können, ist lediglich die Kenntnis der DH Parameter notwendig. Beim Erstellen des Objekts muss dem Konstruktor eine $n \times 23$ Matrix übergeben werden, welche alle notwendigen Parameter der kinematischen Kette enthält. $n$ entspricht der Anzahl an Freiheitsgraden der kinematischen Kette. Für die Kinematik relevante Parameter für ein Gelenk sind in Tabelle 6.1 aufgelistet:

| Spalte | Beschreibung |
|---|---|
| 1 | Gelenktyp (0=Rotationsgelenk, 1=Translationsgelenk) |
| 3 | DH Parameter $d$ |
| 4 | DH Parameter $a$ |
| 5 | DH Parameter $\alpha$ |
| 8 | DH Parameter $\theta$ |
| 23 | Bewegliches Gelenk (0=beweglich, 1=unbeweglich) |

Tabelle 6.1: Parameter für die Klasse `Robot`

Für die Entwicklung des Laufalgorithmus wurden folgende Methoden verwendet:

```
ReturnMatrix Robot::get_q();
void Robot::set_q(const ColumnVector  q);
ReturnMatrix Robot::kine();
ReturnMatrix Robot::kine(const int j);
ReturnMatrix Robot::inv_kin(const Matrix & Tobj, const int mj,
                            const int endlink, bool & converge);
ReturnMatrix Robot::jacobian(const int ref=0);
```

[1] http://www.cours.polymtl.ca/roboop/

Mit den Methoden `get_q()` und `set_q(.)` können die Gelenkvariablen gesetzt werden. Zur Berechnung der entsprechenden Position des Endeffektors mit der direkten Kinematik wird die Methode `kine()` verwendet. Soll die Position eines bestimmten Gelenks innerhalb der Kette berechnet werden, kann zusätzlich mit dem Parameter `j` das gewünschte Gelenk angegeben werden.
Die inverse Kinematik kann mit der Methode `inv_kin(.)` berechnet werden. Für die Lösung wird das numerische Verfahren, welches in Kapitel 3.2.3 erläutert wurde, verwendet. Bei der ersten Iteration des Algorithmus werden für $\hat{q}$ die aktuell gesetzten Gelenkvariablen verwendet. Der Methode muss eine homogene Transformationsmatrix mit der gewünschten Position und Orientierung übergeben werden. Der Referenzparameter `converge` ist nach Ausführung `true`, wenn eine Lösung gefunden wurde.
Mit der Methode `jacobian()` kann die Jacobimatrix zur Geschwindigkeitsberechnung bestimmt werden.

### 6.1.2 Newmat - Matrix Bibliothek

Die Newmat[2] Bibliothek stellt eine große Zahl an Algorithmen für Matrixoperationen bereit. Da ROBOOP auf der Newmat Bibliothek (Version 11(beta)) basiert, ist es zwingend notwendig, die entsprechende Library mit einzubinden. Allerdings werden auch im Laufalgorithmus selbst viele Matrixoperationen benötigt, weshalb Newmat auch hier Verwendung findet.

Eine neue Matrix kann mit dem Konstruktor `Matrix::Matrix(r,c)` erstellt werden, hierbei gibt `r` die gewünschte Anzahl an Zeilen und `c` die Anzahl an Spalten an. Der Zugriff erfolgt mit `Matrix(x,y)`, wobei der Index - im Gegensatz zu normalen C++ Arrays - bei eins beginnend gezählt wird. Viele Operatoren werden für Matrizen überschrieben. Eine Matrixmultiplikation, wie sie auch bei homogenen Transformationen benötigt wird, kann daher einfach mit dem Multiplikationsoperator durchgeführt werden.

Eine weitere häufig benötigte Operation ist die Inversion. Diese kann mit der Methode `Matrix::i()` durchgeführt werden. Die Inversion wird bei der inversen Kinematik und der inversen Geschwindigkeit für die Invertierung der Jacobimatrix benötigt. Wie bereits erwähnt kann bei einer Singularität die Jacobimatrix nicht mehr invertiert werden. Sollte dies der Fall sein, wird von Newmat die Exception `SingularException` ausgelöst und kann somit bei Bedarf abgefangen werden.

[2] http://www.robertnz.net/nm_intro.htm

### 6.1.3 Schnittstelle zur Motorsteuerung

Im der Fakultät Nachrichtentechnik an der Hochschule Mannheim wird momentan eine Schnittstelle zur Steuerung der Motoren entwickelt. Diese Schnittstelle soll den Zugriff auf den RS485 Bus abstrahieren. Zusätzlich ist eine Feininterpolation der Trajektorien auf Gelenkebene vorgesehen. Abb. 6.1 zeigt die geplante Architektur der Schnittstelle.

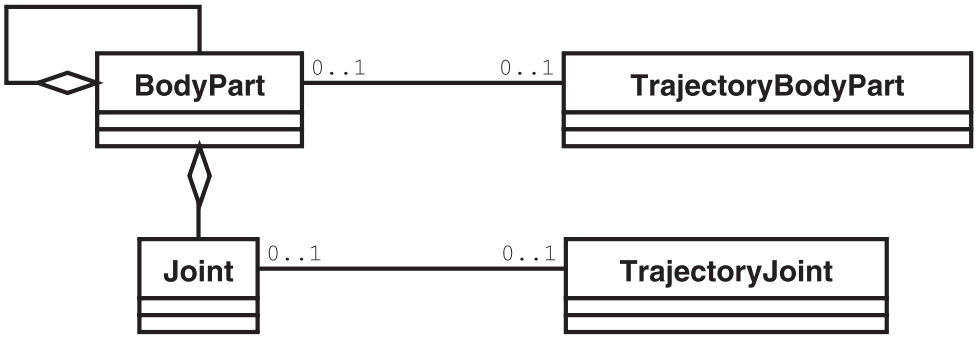



Abbildung 6.1: Architektur der Motorschnittstelle

Der Zugriff auf die Motoren ist auf verschiedenen Ebenen möglich. Die unterste Ebene stellt die Klasse `Joint` dar. Sie bildet genau einen Motor ab. Auf dieser Ebene ist es möglich, die meisten Eigenschaften eines Motors zu nutzen. Es können Werte zur Steuerung, wie beispielsweise Geschwindigkeit, Position und Drehmoment, direkt gesetzt werden. Außerdem ist es möglich, die Betriebsgrenzen des Motors festzulegen und Informationen über den Zustand abzufragen.

Die Klasse `BodyPart` bildet die nächste Ebene. Mit dieser Klasse ist es möglich, mehrere `Joint` Klassen zusammenzufassen. Auch `BodyPart` Klassen können Teil einer anderen `BodyPart` Klasse sein. Somit ist es möglich, einen kompletten Roboter in verschiedene Teilbereiche aufzuteilen: Die Motoren werden zusammengefasst in Beine und Arme, und diese wiederum zum gesamten Roboter. Die `BodyPart` Klasse erlaubt ein gleichzeitiges Ausführen von Befehlen aller enthaltener Motoren. Hiermit können synchronisierte Bewegungen einzelner Teilbereiche oder des ganzen Roboters realisiert werden.

Zum Abfahren ganzer Trajektorien können die beiden Klassen `TrajectoryBodyPart` und `TrajectoryJoint` verwendet werden. Einem `Joint` oder einem `BodyPart` muss dazu die entsprechende Klasse zugeordnet werden. Als Trajektorien können Position-Zeit und Position-Zeit-Geschwindigkeits Vektoren übergeben werden. Die Positionen werden hierbei als Gelenkvariablen angegeben. Die Angabe der Geschwindigkeit dient nur zur Unterstützung der Motorsteuerung, da diese Daten auf der Gelenkebene nicht mehr

direkt bestimmt werden können. Wahlweise kann noch eine Feininterpolation auf Gelenkebene durchgeführt werden.

## 6.2 Humanoid Klasse

Die im Verlauf dieser Arbeit entwickelte `Humanoid` Klasse stellt ein mathematisches Modell des Roboters dar. Hiermit ist es möglich, die verschiedenen kinematischen Berechnungen des Roboters durchzuführen. Desweiteren ist auch die Stabilitätsanalyse mit dieser Klasse möglich. Nachfolgend wird eine Übersicht der wichtigsten Methoden der Klasse gegeben, die genaue Deklaration ist in Anhang A.1 zu finden.

### 6.2.1 Parameter des Roboters

Alle die Konstruktion betreffenden Parameter wurden in der Header Datei *RobotParameter.hpp* zusammengefasst, um das Programm bei einer Modifikation der Roboterkonstruktion leicht anpassen zu können. Die DH Parameter $(a, d, \alpha, \theta)$ sind hier als Konstanten definiert. Außerdem werden in dieser Datei die Werte für die mechanischen Beschränkungen der einzelnen Gelenke, die Positionen der Eckpunkte der Füße und die Masseneigenschaften des ganzen Roboters festgelegt.

Die beiden Funktionen `getLeftParameterMatrix()` und `getRightParameterMatrix()` erstellen aus den angegebenen Daten eine Matrix, die zur Initialisierung der `Robot` Klasse genutzt werden kann. Die `Robot` Klasse bietet nicht nur Platz für die DH Parameter, es können vielmehr noch weitere Eigenschaften der kinematischen Kette dort abgelegt werden. Daher enthalten die Matrizen auch die Masseneigenschaften der einzelnen Gelenke und die Grenzwerte der Gelenkvariablen $\theta$. Auf die entsprechenden Daten wird in der `Humanoid` Klasse nur über Zugriffsfunktionen der Klasse `Robot` zugegriffen. Desweiteren existieren noch Funktionen für den Massenschwerpunkt des Oberkörpers (`getTopCom()`) und die Eckpunkte der Füße (`getFootPoint()`).

Für zukünftige Erweiterungen sollte oben genanntes Vorgehen weiterverfolgt werden, um die Wartbarkeit des Quelltextes zu erhalten: Neue Parameter, die auf der Konstruktion des Roboters basieren, sollten als Konstanten in die Header Datei *RobotParameter.hpp* eingetragen werden. Desweiteren ist es empfehlenswert diese Daten sofern möglich in die Matrizen zur Initialisierung aufzunehmen. Der Zugriff auf die in der `Robot` Klas-

se gespeicherten Daten sollte generell nur über die entsprechenden Zugriffsfunktionen geschehen.

### 6.2.2 Kinematische Berechnungen

Wie bereits erwähnt, wird für die kinematischen Berechnungen die ROBOOP Bibliothek verwendet. Da der Laufalgorithmus beide Beine einbezieht, müssen für die entsprechenden kinematischen Ketten auch zwei `Robot` Objekte vorhanden sein. Erstellt werden diese mit den in `getLeftParameterMatrix()` und `getRightParameterMatrix()` aufgebauten Matrizen.

ROBOOP benutzt, im Gegensatz zu dem in Kapitel 5.1 vorgestellten Konzept, vertauschte Bezeichnungen für die Achsen im Koordinatensystem. Um dennoch dem geplanten Konzept entsprechen zu können, werden die Koordinaten innerhalb der `Humanoid` Klasse für die kinematischen Berechnungen entsprechend umgewandelt.

Für das linke Bein stehen folgende kinematische Methoden zur Verfügung; entsprechende Methoden sind auch für das rechte Bein vorhanden:

```
bool setLeftLegAngles(ColumnVector & pos);
ReturnMatrix getLeftLegAngles() const;
ReturnMatrix getLeftFootPosition() const;
int setLeftFootPosition(ColumnVector & pos);
ReturnMatrix getSpeedLeft(ColumnVector & karSpeed);
```

Die Funktion `setLeftLegAngles(.)` überprüft, ob die mechanischen Beschränkungen von den übergebenen Gelenkstellungen eingehalten wird. Sollte dies nicht der Fall sein, werden die Änderungen nicht übernommen und die Methode liefert `false` als Ergebnis. Ähnliches gilt für die Methode der inversen Kinematik - `setLeftFootPosition(.)`. Wird erfolgreich eine Lösung gefunden, welche auch die mechanischen Beschränkungen nicht überschreitet, werden die entsprechenden Gelenkstellungen übernommen und als Ergebnis wird `0` zurückgeliefert. Wenn es in der inversen Kinematik keine Lösung für die gegebene Position gibt, ist das Ergebnis `1` und wenn mechanische Beschränkungen mit der Lösung überschritten werden liefert die Methode `2` zurück.

Die Jacobimatrix in den Funktionen `setLeftFootPosition(.)` und `getSpeedLeft(.)` wird immer mit den momentan gespeicherten Gelenkstellungen ausgewertet. Bei der inversen Kinematik gilt dies allerdings nur für die erste Iteration.

### 6.2.3 Stabilitätsanalyse

Die Stabilitätsanalyse kann mit der Methode

```
bool isStaticStable(const bool & left, const bool & right,
                    double & xStable, double & yStable);
```

durchgeführt werden. Die Parameter `left` und `right` geben an, welches Bein Bodenkontakt hat. Solange der Roboter stabil ist, enthalten die Referenzparameter `xStable` und `yStable` Informationen darüber, wie nahe sich der COG in x und y Richtung am Mittelpunkt des Stützpolygons befindet. Die Werte sind normalisiert, wobei `0` dem Fall entspricht, dass der COG auf dem Mittelpunkt, und `1`, dass der COG genau auf einer Kante des Stützpolygons liegt. Sollte die Methode `false` als Ergebnis liefern, liegt der COG bereits außerhalb des Stützpolygons, dann enthalten auch `xStable` und `yStable` keine verwertbaren Informationen mehr.

Wie in Kapitel 5.2 erläutert, kann die Stabilitätsanalyse in drei Teilschritte aufgeteilt werden: Schwerpunkt berechnen, Stützpolygon bestimmen und Punkt-in-Polygon Test durchführen. Eine Deklaration der dabei verwendeten Hilfsfunktionen ist in Anhang A.2 nachzulesen.

#### Schwerpunkt

Der Schwerpunkt wird mit der Methode `getRobotCenterOfMass()` berechnet. Hierfür werden zunächst die Massenschwerpunkte der Beine mit `getLeftLegCenterOfMass()` und `getRightLegCenterOfMass()` bestimmt. Für die Berechnung werden die momentan in der Klasse gespeicherten Gelenkstellungen verwendet. Die so berechneten Massenschwerpunkte werden anschließend mit dem Massenschwerpunkt des Oberkörpers, welcher in der Header Datei *RobotParameter.hpp* definiert ist, zu einem Massenschwerpunkt zusammengefasst.

#### Stützpolygon

Das Stützpolygon wird mit der Methode

```
ReturnMatrix getSupportingConvexHull
                    (const bool & left, const bool & right);
```

bestimmt. Auch hierbei geben die Parameter `left` und `right` den Bodenkontakt der beiden Beine an. Sollte nur ein Bein auf dem Boden sein, werden nur die vier Eckpunkte der Fußkontaktfläche als Stützpolygon zurückgegeben. Haben beide Beine Bodenkontakt,

muss die konvexe Hülle aus allen Eckpunkten berechnet werden. Hierfür wird der zuvor beschriebene Graham Algorithmus verwendet.

#### Punkt-in-Polygon Test

Schließlich wird in der Funktion

```
bool pointInPolygon(ColumnVector & point, Matrix & polygon,
                    int delta, double & stableX , double & stableY)
```

die relative Position des Massenschwerpunktes zur Mitte des Stützpolygons ermittelt. Hierbei wird das in Kapitel 5.2.3 vorgestellte Verfahren angewandt. Wichtig für den Test ist die Angabe des `delta` Wertes. Dieser entscheidet darüber, wie lange die verwendeten Testlinien ausgehend vom Mittelpunkt des Polygons sein sollen. Sollte der Wert nicht groß genug sein, liefert die Funktion eventuell fehlerhafte Ergebnisse. Die Linie muss im schlechtesten Fall ausreichend lang sein, um die maximal erreichbare Größe des Stützpolygons vollständig durchqueren zu können. Die maximale Länge des Stützpolygons kann nie größer werden als die zweifache Länge der Beine, dieser Wert bietet somit ausreichend Spielraum um immer exakte Ergebnisse aus dem Algorithmus zu erhalten.

## 6.3 Laufalgorithmus

Der eigentliche Laufalgorithmus setzt sich aus der Berechnung der Stützpunkte nach den Parametern und der nachfolgenden Interpolation zusammen. Da der Roboter bei Abschluss dieser Arbeit noch nicht gebaut war, konnten keine praktischen Versuche durchgeführt werden. Stattdessen wurde ein Programm entwickelt, welches ein berechnetes und interpoliertes Laufmuster auf Stabilität und Einhaltung der mechanischen Beschränkungen überprüft. Anhang A.3 enthält die Deklaration der unten beschriebenen Funktionen und Klassen.

### 6.3.1 Berechnung des Laufmusters nach gegebenen Parametern

Zur Berechnung des Laufmusters steht die Funktion

```
int createWalking(double downP, double heightP, double lengthP,
                  double sideP, int steps,
                  Matrix & trajLeft, Matrix & trajRight)
```

zur Verfügung. Hiermit können nach gegebenen Parametern (`downP`, `heightP`, `lengthP`,

`sideP`, vgl. Kapitel 5.3.2) die entsprechenden Stützpunkte des Laufmusters bestimmt werden. Nach der Ausführung enthalten die beiden Referenzparameter `trajLeft` und `trajRight` die Stützpunkte der Beintrajektorien. Als Ergebnis liefert die Funktion die Anzahl der generierten Punkte zurück.

Mit dem Parameter `steps` wird die Anzahl an Schritten angegeben, die generiert werden sollen. Die Funktion verwendet im ersten und letzten Schritts des Laufmusters jeweils nur die halbe Schrittlänge. Somit ist sichergestellt, dass nach Ausführung der kompletten Trajektorie beide Beine wieder genau nebeneinander stehen.

### 6.3.2 Interpolation

Die Interpolation wird mit der Funktion

```
ReturnMatrix interpolTrajectory(const Matrix & trajLeft,
                  const Matrix & trajRight, ColumnVector & time,
                  int interpol, int intCount, int ts )
```

durchgeführt. Die Parameter `trajLeft` und `trajRight` enthalten die zu interpolierenden Punkte. Der zugehörige Zeitvektor wird als `time` angegeben. Im Normalfall sind die Zeitpunkte äquidistant; um gewisse Teile des Laufmusters zu beschleunigen oder zu verlangsamen, können in Teilbereichen aber auch beliebige andere Zeitabstände angegeben werden. Einzige Voraussetzung für eine erfolgreiche Interpolation ist eine streng monotone Steigung der Zeitpunkte.

Mit dem Parameter `intCount` wird die Anzahl an Zwischenpunkten angegeben, die über das gesamte Laufmuster berechnet werden sollen. Der Parameter `ts` gibt die Dauer des gesamten Laufmusters in Millisekunden an. Die gewünschte Interpolationsart wird mit dem Parameter `interpol` ausgewählt: `0` für keine Interpolation, `1` für kubische Spline-Interpolation und `2` für Interpolation mit B-Splines.

#### Kubische Splines

Für die kubische Spline-Interpolation wird die Klasse `KubSpline` verwendet. Die Methoden wurde nach dem in [26] vorgestellten Algorithmus entwickelt. Dem Konstruktor müssen die beiden Vektoren für Zeit und Position übergeben werden. Für jede zu interpolierende Komponente der Trajektorie muss dabei ein eigenes Objekt angelegt werden. Danach können mit den Methoden `getVal(.)` und `getDerivative(.)` die Kurven an einer beliebigen Stelle ausgewertet werden.

#### B-Splines

Eine B-Spline Interpolation wird mit der Klasse `BSpline` durchgeführt. Die Funktionen basieren auf den Quelltexten aus [24]. Im Gegensatz zu den kubischen Splines genügt ein Objekt für alle Komponenten, inklusive der Zeit. Dem Konstruktor wird hierfür nur eine Matrix mit allen Komponenten und Stützpunkten übergeben. Bei einer B-Spline Interpolation ist es nicht möglich, die Kurve für einen bestimmten Zeitpunkt auszuwerten. Stattdessen wird nach der Vorgabe einer gewünschten Anzahl an Zwischenpunkten diese in einem Durchgang berechnet. Die Methode `interpol(.)` liefert nach gegebener Anzahl die interpolierte Trajektorie zurück. Informationen über die Geschwindigkeiten können aus der B-Spline Kurve nicht gewonnen werden, da ein Ableiten nach der Zeit nicht möglich ist.

### 6.3.3 Überprüfen des Laufmusters

Ein Laufmuster kann mit der Funktion `calculateTrajectory(.)` überprüft werden. Die Funktion übernimmt als Parameter eine Matrix mit den Positionen und Geschwindigkeiten der Beine. Mit Hilfe der inversen Kinematik wird versucht, die einzelnen Punkte in Gelenkkoordinaten umzuwandeln. Diese werden auf die Einhaltung der mechanischen Beschränkungen überprüft. Daraufhin wird für jeden Punkt eine Stabilitätsanalyse erstellt. Abschließend werden die Gelenkgeschwindigkeiten berechnet. Mit diesen Werten können eventuelle Überschreitungen der Motorwerte erkannt werden.

## 6.4 Kompilieren für PDA

Auf dem verwendeten PDA läuft als Betriebssystem Windows Mobile 5.0, welches auf Windows CE basiert. Die Umsetzung von Standard C++ Code auf den PDA stellt daher keine große Hürde dar. Um die benötigten Bibliotheken - ROBOOP und Newmat - verwenden zu können, müssen diese zuvor für Windows Mobile kompiliert werden. Hierzu genügt es, bei Verwendung eines entsprechenden Cross-Compilers, zusätzliche Präprozessordefinitionen anzugeben: `WINCE`; `UNDER_CE`; `UNICODE`. Die selben Definitionen müssen auch bei der Kompilierung des Hauptprogramms verwendet werden, damit die passenden Header Dateien der Standard Bibliotheken verwendet werden.

Unter Windows CE ist eine verkleinerte Version der Windows API verfügbar. Viele bekannte Funktionen können somit ohne Änderungen weiterverwendet werden. In einigen Fällen gibt es jedoch geringfügige Unterschiede. Insbesondere bei der Programmierung einer Benutzerschnittstelle müssen einige Besonderheiten beachtet werden. Einen guten Einstieg in die Windows CE Programmierung bietet das Buch [9].

# 7 Bewertung der Ergebnisse

Um den entwickelten Laufalgorithmus zu bewerten, wurden zunächst die Laufzeiten der Berechnungen auf dem PDA gemessen. Die Ergebnisse sind im ersten Teil dieses Kapitels zu finden. Im zweiten Teil werden die aus dem Laufalgorithmus gewonnenen Daten analysiert und bewertet.

## 7.1 Benötigte Rechenzeit auf dem PDA

Um zu bewerten, wieviele Berechnungen zur Laufzeit auf dem PDA durchgeführt werden können, wurden die benötigten Rechenzeiten der verschiedenen Operationen auf dem PDA gemessen. Tabelle 7.1 zeigt die Resultate der Messungen.Wegen der fehlenden Gleitkommarecheneinheit des PDAs hat insbesondere die Wahl der Gleitkommagenauigkeit einen starken Einfluss auf die Laufzeiten. Die Messungen wurden daher jeweils mit einfacher und doppelter Genauigkeit durchgeführt. Der Geschwindigkeitsvorteil bei einfacher Genauigkeit ist eindeutig zu erkennen. Inwieweit dies die geringere Genauigkeit rechtfertigt, bzw. ob die einfache Genauigkeit ausreichend für den Laufalgorithmus ist, müssen spätere praktische Versuche zeigen. Auffällig hoch sind die Rechenzeiten für die Auswertung einer kompletten Trajektorie mit 100 Punkten. Im Vergleich zu einer einzelnen Operation der inversen Kinematik liegt diese aber trotzdem noch relativ niedrig. Dies liegt an der Auswertung einer fortlaufenden Trajektorie mit geringem Abstand zwischen den Punkten. Hierbei genügt meist eine Iteration des Algorithmus (vgl. Kapitel 3.2.3). Für die Messung einer einzelnen Operation der inversen Kinematik hingegen wurden zufällige Werte verwendet. Somit ist eine mehrfache Auswertung der Jacobi-Matrix nötig, bis eine ausreichend genaue Lösung gefunden wurde.

Für die komplette Berechnung der Trajektorien während der Laufzeit sind die hier entwickelten Algorithmen aufgrund der benötigten Rechenzeit ungeeignet. Besser wäre es, die Trajektorien vor dem Start berechnen zu lassen und während der Laufzeit nur auf Abweichungen in der Stabilität zu achten. Die Rechenzeit der Stabilitätsanalyse macht

| Aktion | Laufzeit (double) | Laufzeit (float) |
|---|---|---|
| Direkte Kinematik (beide Beine) | 1,18799 ms | 0,979077 ms |
| Inverse Kinematik (beide Beine) | 28,7495 ms | 19,8542 ms |
| Geschwindigkeiten (beide Beine) | 1,47979 ms | 1,07159 ms |
| Stabilitätsanalyse:<br>Zwei Beine Bodenkontakt | 4,24144 ms | 3,16021 ms |
| Stabilitätsanalyse:<br>Ein Bein Bodenkontakt | 3,45877 ms | 2,37221 ms |
| Massenschwerpunkt berechnen | 2,82041 ms | 2,07487 ms |
| Konvexe Hülle berechnen:<br>Zwei Beine Bodenkontakt | 1,44072 ms | 1,21621 ms |
| Laufmuster erstellen für zwei Schritte | 0,331692 ms | 0,210872 ms |
| Interpolation mit kubischen Polynomsplines:<br>9 Stützpunkte und 100 Zwischenschritte | 17,9589 ms | 11,8866 ms |
| Interpolation mit B-Splines:<br>9 Stützpunkte und 100 Zwischenschritte | 19,4293 ms | 13,1294 ms |
| Auswertung von zwei Schritten:<br>100 Punkte mit inverser Kinematik,<br>Stabilitätsanalyse, Geschwindigkeiten | 2471,03 ms | 1666.83 ms |

Tabelle 7.1: Laufzeiten auf dem PDA

es möglich, die Stabilität mehrmals in der Sekunde zu überprüfen und bei Bedarf entsprechende Gegenmaßnahmen einzuleiten.

Wie zu sehen ist, stellen beim Vergleich der implementierten Interpolationsarten die kubischen Polynomsplines eindeutig die schnellere Lösung dar. Zusätzlich können mit Polynomsplines, wie in Kapitel 3.1 erläutert, die Ableitungen der Kurve berechnet werden. Somit ist die Interpolation mit kubischen Polynomsplines vermutlich geeigneter für die Verwendung in einem Laufalgorithmus als die Interpolation mit B-Splines.

## 7.2 Auswertung des Laufmusters

Praktische Versuche mit dem Roboter waren zum Zeitpunkt dieser Arbeit noch nicht möglich. Daher wurde versucht, mit Hilfe einer Analyse der berechneten Werte des Laufmusters den Laufalgorithmus zu bewerten. Da hierbei viele äußere Einflüsse nicht

beachtet werden können, sind praktische Versuche zur Evaluierung des Laufalgorithmus dennoch zwingend notwendig.

Für die Analysen wurden folgende Parameter bei der Berechnung des Laufmusters verwendet (vgl. Kapitel 5.3.2):

| | | |
|---|---|---|
| $h_{schritt} = 20$ mm | $l_{schritt} = 30$ mm | $s_{schritt} = 50$ mm |
| $d_{schritt} = 40$ mm | $t_{schritt} = 2000$ ms | Anzahl Schritte $= 4$ |

Bei der Interpolation mit kubischen Polynomsplines wurden Zwischenwerte im Abstand von 20 ms berechnet.

### 7.2.1 Bewertung der Stabilität

Wichtigste Eigenschaft des Laufmusters ist die Stabilität. Sollte bereits bei den Berechnungen die Stabilitätsanalyse negativ ausfallen, werden vermutlich auch praktische Versuche mit diesem Muster scheitern. Daher wurde zunächst die Stabilität beurteilt. Abb. 7.1 zeigt den entsprechenden Verlauf. Die Kurven X und Y geben den normalisierten Abstand des COG zum Mittelpunkt des Stützpolygons wieder.

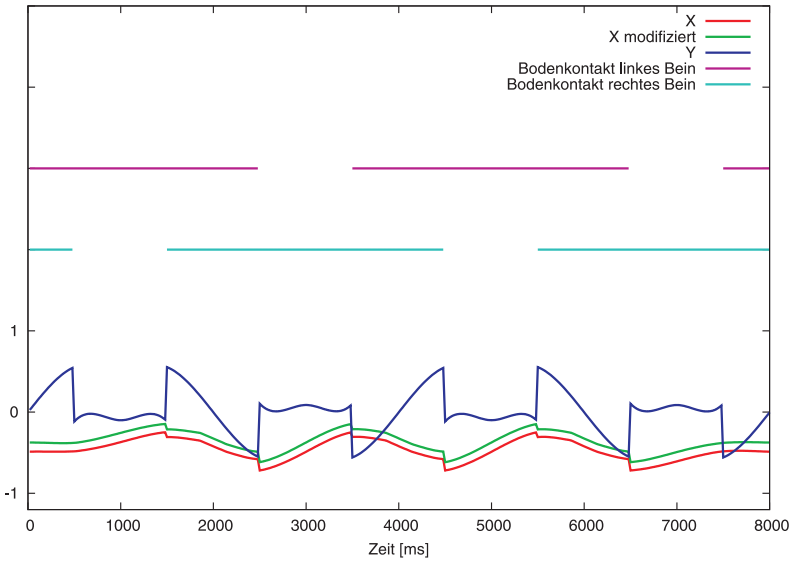



Abbildung 7.1: Verlauf der Stabilität

Nach mehreren Versuchen hat sich für die seitliche Gewichtsverlagerung ein Wert von 50 mm als ideal herausgestellt. Hierbei schwingt die Stabilität entlang der Y Achse genau

um den Nullpunkt. Die Ausschläge betragen maximal ca. 0,5 in der Double Support Phase. In der Single Support Phase liegt der Abstand sogar nur im Bereich von 0,1.

Die Stabilität entlang der X Achse liegt immer im negativen Bereich. Dies ist konstruktionsbedingt, da der Roboter so entworfen wurde, dass der Schwerpunkt in der Grundstellung in etwa über dem Kraftsensor liegt, welcher sich im hinteren Drittel des Fußes befindet. Die modifizierte X Kurve zeigt die Stabilität im Falle des modifizierten Laufmusters (vgl. Kapitel 5.3.1). Durch das nach vorne einknickende Knie wird mehr Masse und somit auch der COG nach vorne verlagert. Die Unterschiede sind aber relativ gering, somit ist fraglich ob dies einen signifikanten Einfluss auf die Stabilität hat.

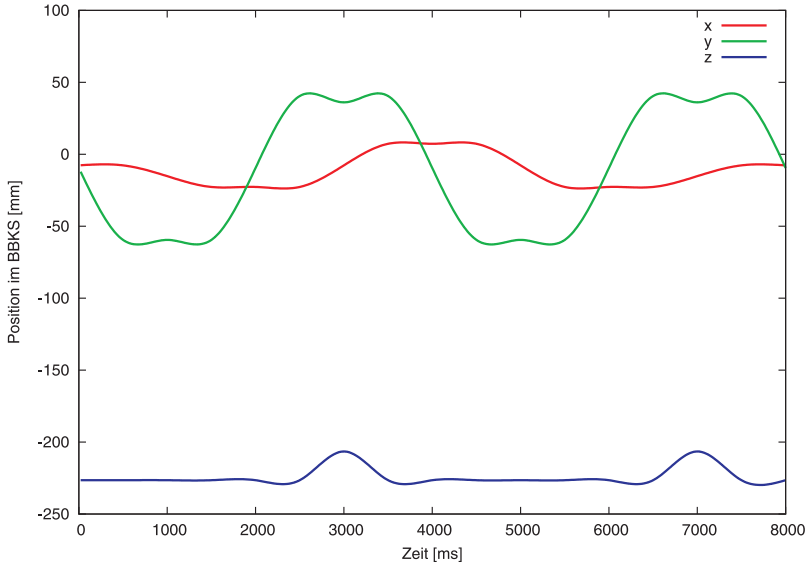
Die Parameter $h_{schritt}$, $l_{schritt}$ und $d_{schritt}$ haben kaum Einfluss auf die statische Stabilität des Roboters. Nur bei Schrittlängen größer 70 mm verlässt der COG das Stützpolygon während der Gewichtsverlagerung. Um auch größere Schrittlängen zu ermöglichen, müsste das Laufmuster entsprechend abgeändert werden. Nur durch das Anpassen der restlichen Parameter kann dies jedoch nicht erreicht werden. Solch große Schrittlängen sind allerdings bei einem statisch stabilen Laufalgorithmus kaum zu erwarten. Welche Schrittlängen noch sinnvoll sind und inwieweit ein kurzzeitiges Verlassen des COG den Roboter zum Umfallen bringen würde, muss in praktischen Versuchen überprüft werden.

Auch die Zeit $t_{schritt}$ hat keinen Einfluss auf die statische Stabilität. Jedoch wirkt sie sich umso stärker auf die auftretenden dynamischen Kräfte aus. Geeignete Werte für die Zeit alleine mit einer Analyse des Laufmusters abzuschätzen, ist unmöglich. Somit ist es auch hier notwendig, mit praktischen Versuchen optimale Werte zu finden.

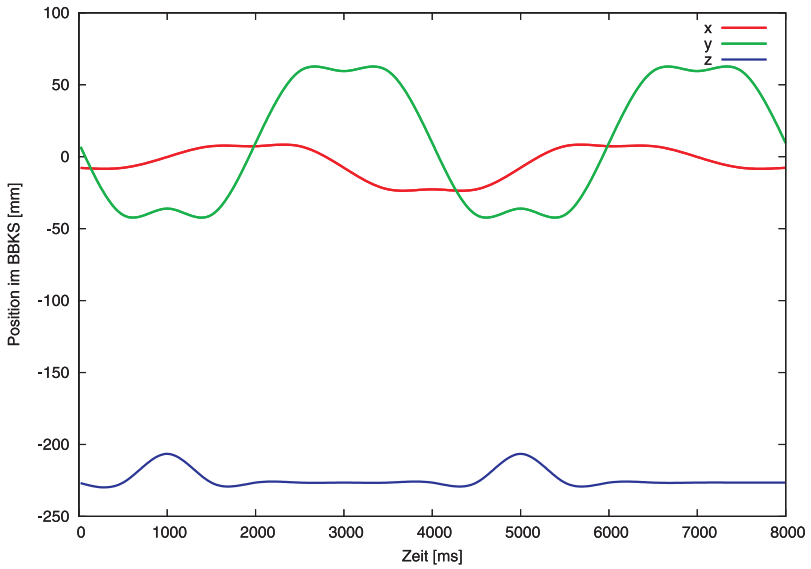
### 7.2.2 Trajektorien der Beine

In Abb. 7.2 sind die Trajektorien im Raum beider Beine, bei Interpolation mit Polynomsplines, zu sehen. An den Y Kurven sind deutlich die Gewichtsverlagerungen zu erkennen. Die Z Kurven zeigen das Anheben der Beine. Die Vorwärtsbewegung kann in der X Kurve entdeckt werden. Hierbei bewegt sich das in der Luft befindliche Bein stets nach vorne, das Bein am Boden hingegen nach hinten. Die leichten Schwingungen, insbesondere in der Y Kurve, werden durch die Interpolation verursacht.

Abb. 7.3 zeigt die entsprechenden Bewegungen in den Gelenken. Auch hier kann wieder deutlich die Gewichtsverlagerung in den Kurven Theta 1 und Theta 6 erkannt werden. Da das Laufmuster nur eine gerade Vorwärtsbewegung bewirkt, bleibt das zweite Gelenk unverändert in der Grundposition. Die ungleichmäßigen Schwingungen in den Gelenken




(a) Linkes Bein

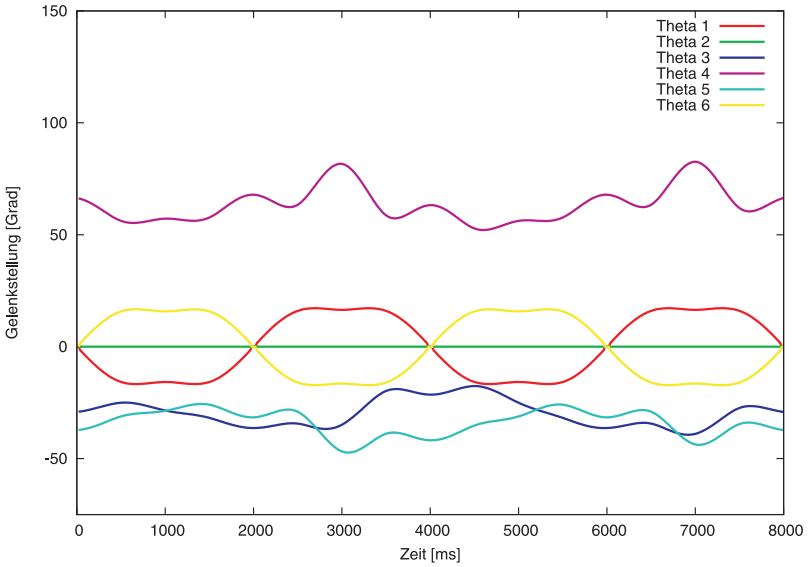



(b) Rechtes Bein

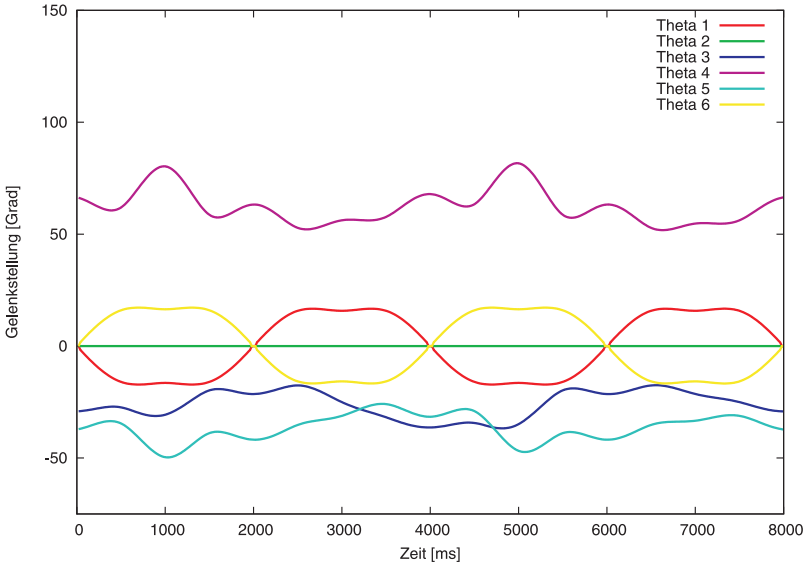
Abbildung 7.2: Raumtrajektorien der Beine

drei bis sechs werden hauptsächlich durch die zuvor angesprochenen Probleme in der Konstruktion (vgl. Kapitel 5.3.3) verursacht. Einzig die beiden großen Ausschläge in der Kurve Theta 4 zeigen den Zeitpunkt, an dem das Bein angehoben wird. Die Diagramme

veranschaulichen somit deutlich, dass eine Änderung der Konstruktion zu empfehlen ist, um die unnötigen Bewegungen zu vermeiden.




(a) Linkes Bein

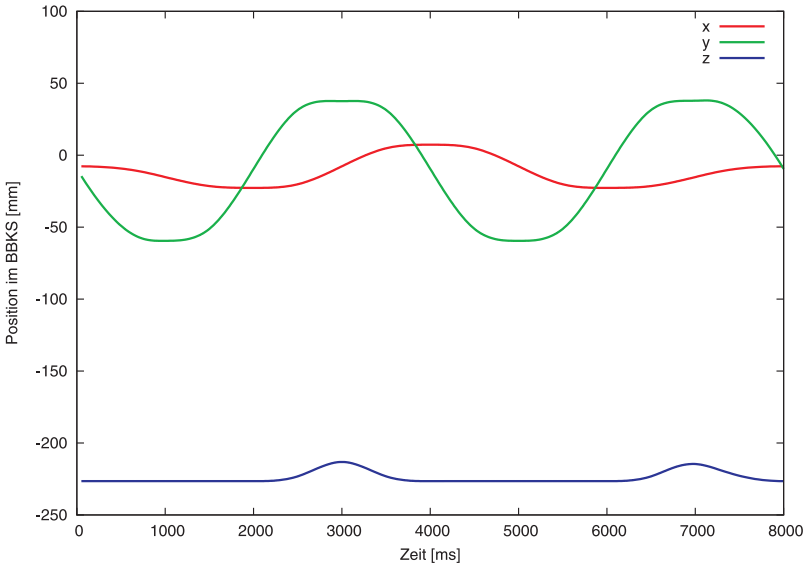



(b) Rechtes Bein

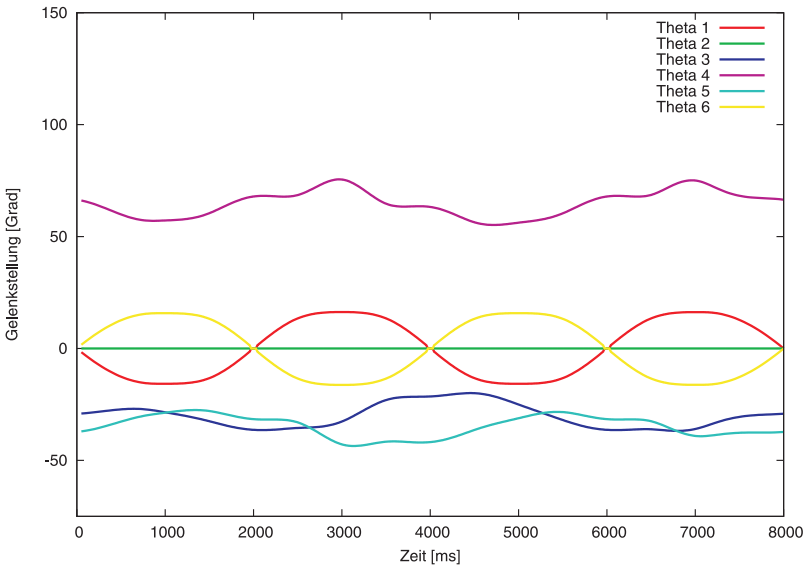
Abbildung 7.3: Gelenktrajektorien der Beine

### 7.2.3 Interpolation mit B-Splines

Abb. 7.4 zeigt die Raum- und Gelenktrajektorien des linken Beins bei Interpolation mit B-Splines.




(a) Raumtrajektorien im linken Bein




(b) Gelenktrajektorien im linken Bein

Abbildung 7.4: Interpolation mit B-Splines

Die Trajektorien sind deutlich glatter im Vergleich zu den Trajektorien die mit Polynomsplines interpoliert wurden (vgl. 7.2(a) und 7.3(a)). Dies wirkt sich insbesondere positiv auf die Bewegungen in den Kniegelenken aus. Aber auch die restlichen Gelenke werden deutlich weniger von Schwingungen beeinflusst.

Die glätteren Trajektorien wirken sich auch positiv auf die Stabilität aus. Abb. 7.5 zeigt den entsprechenden Verlauf. Die maximalen Ausschläge der Stabilität entlang der Y Achse sind geringer als bei der Interpolation mit Polynomsplines (vgl. 7.1). Allerdings sind die Abweichungen vom Nullpunkt in der Single Support Phase größer.

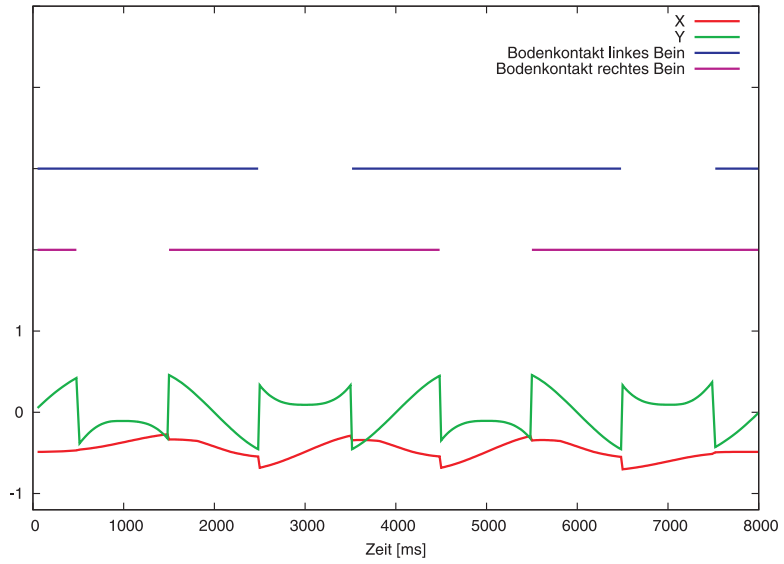



Abbildung 7.5: Verlauf der Stabilität bei B-Spline Interpolation

Die Verbesserungen in den Trajektorien werden durch die Eigenschaft der B-Splines verursacht, die Stützpunkte nur anzunähern. Hierdurch ergeben sich geringere Ausschläge in der Kurve. Dennoch sind diese ausreichend, um eine gute Stabilität zu erreichen, wie in Abb. 7.5 zu erkennen ist. Außerdem ist der Einfluss weit entfernter Stützstellen bei B-Splines ausgeschlossen, somit entstehen auch weniger ungewollte Schwingungen.

Trotz der Nachteile von B-Splines im Bezug auf die Laufzeit und die fehlenden Ableitungen, stellen diese also eine erwähnenswerte Alternative zu den kubischen Polynomsplines dar. Welche Eigenschaften letztlich in einen besseren Laufalgorithmus resultieren, kann mit der hier durchgeführten Analyse kaum beurteilt werden. Nicht zuletzt müssen also auch in praktischen Versuchen die beiden Interpolationsarten miteinander verglichen werden.

# 8 Zusammenfassung und Ausblick

## 8.1 Zusammenfassung

Im Verlauf dieser Arbeit wurde ein statisch stabiler Laufalgorithmus für den an der Hochschule Mannheim konstruierten zweibeinigen Laufroboter konzipiert und implementiert. Der entwickelte Laufalgorithmus soll für erste praktische Laufversuche dienen. Die so gewonnenen Erfahrungen können als Grundlage für die Entwicklung eines dynamischen Laufalgorithmus für diesen Roboter genutzt werden.

Zum Zeitpunkt der Arbeit war nur eine mathematische Analyse des Laufalgorithmus möglich, da der Roboter momentan noch an der Hochschule Mannheim gebaut wird. Die Ergebnisse der Analyse lassen ein positives Ergebnis, soweit dies im Rahmen eines statisch stabilen Laufalgorithmus zu erwarten ist, bei den praktischen Versuchen erwarten.
Die benötigten Rechenzeiten der Algorithmen wurden auf dem zur Steuerung des Roboters verwendeten PDA gemessen, um herauszufinden, inwieweit diese später für Berechnungen zur Laufzeit verwendet werden können. Hierbei stellte sich heraus, dass es zu aufwendig ist, erst während der Laufzeit die Berechnung eines kompletten Laufmusters durchzuführen. Die Überprüfung der Stabilität hingegen ist schnell genug, um mehrmals in der Sekunde durchgeführt zu werden. Somit könnte beispielsweise ein Regelkreis entwickelt werden der versucht, den Roboter möglichst stabil zu halten.

Für die Entwicklung des statischen Stabilitätskriteriums mussten zunächst die genauen Masseneigenschaften der einzelnen Gelenke bestimmt werden, um den Massenschwerpunkt für jede Lage des Roboters berechnen zu können. Außerdem ist es für das Stabilitätskriterium notwendig, genaue Kenntnisse über die Position des Stützpolygons, welches von den Kontaktflächen der Füße aufgespannt wird, zu haben. Der Massenschwerpunkt und das Stützpolygon müssen schließlich auf die gegenseitige Lage hin überprüft werden. Hierfür wurde im Verlauf dieser Arbeit ein modifizierter Punkt-In-Polygon Test

entwickelt. Dieser bietet zusätzliche Informationen über den Abstand des Punktes zu den Polygonkanten und erleichtert somit die Optimierung des Laufmusters.

Auf der Basis des entwickelten Stabilitätskriteriums wurde ein entsprechendes Laufmuster geplant. Das Muster besteht im Prinzip aus einer Gewichtsverlagerung auf ein Bein, woraufhin das andere nach vorne bewegt wird. Bei der Entwicklung wurde besonderer Wert darauf gelegt, das Laufmuster mit Parametern modifizieren zu können. Dies erleichtert die Suche nach optimalen Eigenschaften des Laufmusters in späteren praktischen Versuchen. Außerdem zeigten sich während der Arbeit am Laufmuster zwei problematische Eigenschaften der Konstruktion - das nach vorne geknickte Bein und der Versatz zwischen den Achsen im Fuß und im Oberkörper - und es wurden entsprechende Lösungen vorgestellt.

Um aus den geplanten Stützpunkten des Laufmusters eine flüssige Bewegung der Beine zu erhalten, muss zunächst eine Interpolation der Trajektorien im kartesischen Raum durchgeführt werden. Hierfür wurden zwei Interpolationsarten - kubische Polynomsplines und B-Splines - umgesetzt und bewertet. Kubische Polynomsplines benötigen zwar weniger Rechenzeit und können zusätzlich abgeleitet werden, um Informationen über die Geschwindigkeiten zu erhalten, jedoch zeigen die mit B-Splines interpolierten Trajektorien deutlich weniger ungewollte Schwingungen. Inwieweit sich die eine oder andere Eigenschaft als geeigneter herausstellt, muss in praktischen Versuchen überprüft werden.

Bei der Umsetzung der Trajektorien vom kartesischen Raum in die Gelenkkoordinaten der Beine, wurde das in der Robotik bekannte Denavit-Hartenberg Verfahren verwendet. Hierfür musste eine kinematische Analyse der Roboterbeine durchgeführt werden, um entsprechende Parameter aufstellen zu können. Dabei wurden auch entsprechende Koordinatensysteme in den Gelenken definiert, welche unter anderem für die Berechnungen des Stabilitätskriteriums benötigt werden.

## 8.2 Ausblick

Obwohl bereits eine umfassende theoretische Analyse des Laufalgorithmus vorgenommen wurde, kann diese keine praktischen Versuche ersetzen. Erst hierbei können geeignete Parameter für das Laufmuster bestimmt werden. Außerdem können mit diesen Versuchen erste Erfahrungen mit den Eigenschaften der Konstruktion gesammelt werden. Sobald die ersten Schritte erfolgreich absolviert wurden, kann auch das Verhalten der Sensoren

während des Laufens genauer untersucht werden. Somit können geeignete Sensoren für einen späteren, verbesserten Laufalgorithmus ausgewählt werden.

Wie bereits erwähnt, sind auch die Laufzeiten der Algorithmen vermutlich noch nicht ideal. Bei der Entwicklung wurde nicht im Besonderen auf optimale Geschwindigkeit bei der Programmierung eines PDAs geachtet. Es sollte somit untersucht werden, ob und wie sich die Laufzeit optimieren lässt, da trotz der hohen Rechenleistung des PDAs diese immer noch beschränkt ist. Von den Ergebnissen dieser Untersuchung können auch zukünftige Arbeiten, die weitere Algorithmen implementieren, profitieren.

In dieser Arbeit wurden einige Eigenschaften der Konstruktion identifiziert, die eher ungeeignet für die Entwicklung eines Laufalgorithmus sind. Inwieweit sich die vorgeschlagenen Lösungen umsetzen lassen, oder ob eventuell andere Lösungen sinnvoller sind, muss noch genauer analysiert werden. Bei einer Änderung der Konstruktion müssen auch die hier entwickelten Algorithmen angepasst werden. Alle die Konstruktion betreffenden Eigenschaften wurden aus diesem Grund bereits an einer zentralen Stelle zusammengefasst. Es sollte daher möglich sein, die meisten Konstruktionsänderungen leicht in das Programm zu übernehmen.

# Tabellenverzeichnis

# Abbildungsverzeichnis

# Literaturverzeichnis


[1] Asano, Fumihiko ; Luo, Zhi-Wei ; Yamakita, Masaki: Biped Gait Generation and Control Based on a Unified Property of Passive Dynamic Walking. In: *IEEE Transactions on Robotics* Bd. 21(4), August 2005, S. 754–762

[2] Aumann, Günter ; Spitzmüller, Klaus: *Computerorientierte Geometrie*. 1. Aufl. Bibliographisches Institut, 1993. – ISBN 3–411–16021–7

[3] Baker, Kirby A.: *The Mathematics of Computer Graphics: Cubic Spline Curves*. Vorlesung an der University of California, USA. 2002. – `http://www.math.ucla.edu/~baker/149.1.02w/handouts/dd_splines.pdf` Stand: 30.03.2007

[4] Bařinka, Lukáš ; Berka, Roman: Inverse Kinematics - Basic Methods. In: *Central European Seminar on Computer Graphics* Slowakei, April 2002

[5] Behnke, Sven: *Humanoid League Rules and Setup*. 2006. Bremen: RoboCup. – `http://www.humanoidsoccer.org` Stand: 10.04.2007

[6] Behnke, Sven: Online Trajectory Generation for Omnidirectional Biped Walking. In: *IEEE International Conference on Robotics and Automation* Orlando, FL, USA, Mai 2006, S. 1597–1603

[7] Behnke, Sven ; Müller, Jürgen ; Schreiber, Michael: Using Handheld Computers to Control Humanoid Robots. In: *International Conference on Dextrous Autonomous Robots and Humanoids* Yverdon-les-Bains, Schweiz, Mai 2005

[8] Blow, Michael P. ; Dautenhahn, Kerstin ; Appleby, Andrew ; Nehaniv, Chrystopher ; Lee, David: Perception of Robot Smiles and Dimensions for Human-Robot Interaction Design. In: *IEEE International Symposium on Robot and Human Interactive Communication* University of Hertfordshire, Hatfield, UK, September 2006, S. 469–474

[9] Boling, Douglas: *Programming Microsoft Windows CE. NET. The definitive guide to programming the Windows CE API*. 3. Aufl. Microsoft Press Books, 2003. – ISBN

0–735–61884–8

[10] Bräunl, Thomas: *Embedded Robotics - Mobile Robot Design and Applications with Embedded Systems*. 1. Aufl. Springer, 2003. – ISBN 3–540–03436–6

[11] Chevallerau, Cambrini ; Sardin, Philippe: Design and actuation optimization of a 4 axes biped robot for walking and running. In: *IEEE International Conference on Robotics and Automation* Bd. 4 San Francisco, CA, USA, April 2000, S. 3365–3370

[12] Denavit, J ; Hartenberg, R S.: A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. In: *ASME Journal of Applied Mechanics* Bd. 22, 1955, S. 215–221

[13] Giese, Martin. *Kombination Lehrbasierter und Dynamikbasierter Animation*. Vorlesung an der Universität Tübingen. Januar 2006

[14] Goswami, Ambarish: Foot Rotation Indicator (FRI) Point: A New Gait Planning Tool to Evaluate Postural Stability of Biped Robots. In: *IEEE International Conference on Robotics and Automation* Detroit, MI, USA, Mai 1999, S. 47–52

[15] Hardt, Michael W. ; Stelzer, Maximilian ; Stryk, Oskar von: Modellierung und Simulation der Dynamik des Laufens bei Roboter, Tier und Mensch. In: *thema Forschung* (2002), Nr. 2, S. 56–63

[16] Hauser, Helmut. *Stability Criteria for Humanoid Robots*. Seminar am Institut für theoretische Informatik an der TU Graz, Österreich. Juni 2005

[17] Huang, Qiang ; Yokoi, Kazuhito ; Kajita, Shuuhi ; Kaneko, Kenji ; Arai, Hirohiko ; Koyachi, Noriho ; Tanie, Kazuo: Planning walking patterns for a biped robot. In: *IEEE Transactions on Robotics and Automation* Bd. 17(3), Juni 2001, S. 280–289

[18] Ihme, Thomas: *Steuerung von sechsbeinigen Laufrobotern unter dem Aspekt technischer Anwendungen*, Otto-von-Guericke-Universität Magdeburg, Diss., 2002

[19] Kovács, Peter: *Rechnergestützte symbolische Roboterkinematik*. 1. Aufl. Vieweg, 1993. – ISBN 3–528–06544–3

[20] Li, Qinghua ; Takanashi, Atsuo ; Kato, Ichiro: A Biped Walking Robot Having A ZMP Measurement System Using Universal Force-Moment Sensors. In: *IEEE/RSJ International Workshop on Intelligent Robots and Systems* Bd. 3 Osaka, Japan, November 1991, S. 1568–1573

[21] Myronov, Oleksandr: *Konstruktion eines zweibeinigen Laufroboters mit Sensor-, Aktor- und Steuerungskomponenten*, Hochschule Mannheim, Diplomarbeit, 2006

[22] Nicholls, Elliot: *Bipedal Dynamic Walking in Robotics*, University of Western Australia, Diss., 1998

[23] Robotis: *Dynamixel - DX-113, DX-116, DX-117*. 2. Auflage. 2005. – `http://www.tribotix.com/Products/Robotis/Robotis.htm` Stand: 11.04.2007

[24] Rogers, David F.: *An Introduction to NURBS*. 1. Aufl. Morgan Kaufmann, 2003. – ISBN 1–558–60669–6

[25] Salter, Andy: Spline Curves and Surfaces / Imperial College London. 2007. – Forschungsbericht. `http://www.doc.ic.ac.uk/~dfg/AndysSplineTutorial/index.html` Stand: 30.03.2007

[26] Scraff, Heidi ; Anderson, Ryan: Cubic Splines, Natural and Clamped. 2004. – Forschungsbericht. `http://www.michonline.com/ryan/csc/hw.html` Stand: 31.03.2007

[27] Shih, Chih-Long: Analysis of the dynamics of a biped robot with seven degrees of freedom. In: *IEEE International Conference on Robotics and Automation* Bd. 4 Minneapolis, MN, USA, April 1996, S. 3008–3013

[28] Siegert, Hans-Jürgen ; Bocionek, Siegfried: *Robotik. Programmierung intelligenter Roboter*. 1. Aufl. Springer, 1996. – ISBN 3–540–60665–3

[29] Toussaint, Godfried ; Aloupis, Greg ; Kalzuny, Bohdan. *The 3-Coins Algorithm for Convex Hulls of Polygons*. `http://cgm.cs.mcgill.ca/~beezer/cs507/3coins.html` Stand: 14. März 2007

[30] Ude, Aleš ; Atkeson, Christopher G. ; Riley, Marcia: Planning of Joint Trajectories for Humanoid Robots Using B-Spline Wavelets. In: *IEEE International Conference on Robotics and Automation* Bd. 3 San Francisco, CA, USA, April 2000, S. 2223–2228

[31] Vukobratović, Miomir: *Kinematics and Trajectory Synthesis of Manipulation Robots*. 1. Aufl. Springer, 1986. – ISBN 3–540–13071–3

[32] Vukobratović, Miomir: *Introduction to Robotics*. 1. Aufl. Springer, 1989. – ISBN 3–540–17452–4

[33] Vukobratović, Miomir ; Borovac, Branislav: Zero-Moment-Point - Thirty Five Years Of Its Life. In: *International Journal of Humanoid Robotics* Bd. 1, 2004, S. 157–173

[34] Weck, Manfred ; Brecher, Christian: *Werkzeugmaschinen - Automatisierung von Maschinen und Anlagen*. Bd. 4. 6. Aufl. Springer, 2006. – ISBN 3–540–22507–2

[35] Wicke, Martin: *Bipedal Walking*, Universität Kaiserslautern, Studienarbeit, 2001

[36] Wloka, Dieter W.: *Roboter Systeme 1.* 1. Aufl. Springer, 1992. – ISBN 3–540–54739–8

# A Deklarationen

## A.1 Humanoid Klasse

```
/*Enthält das mathematische Modell des Roboters und bietet verschieden Operationen,
* wie direkte und indirekte Kinematik, auf diesem Modell an.
*/
class Humanoid {
public:
        /*Erstellt ein Modell mit den Parametern aus RobotParameter.hpp
        */
        Humanoid();
        ~Humanoid();

        /*Setzt die Winkel der Motoren.
        * Parameter: Vektor(theta1,theta2,theta3,theta4,theta5,theta6)
        */
        void setAngles(const ColumnVector& right, const ColumnVector& left);

        /*Setzt die Winkel für das linke Bein
        * Die neuen Winkel werden nur übernommen, wenn die Werte nicht die
        * mechanischen Beschränkungen des Roboters überschreiten.
        * Parameter: Vektor(theta1,theta2,theta3,theta4,theta5,theta6)
        * Return: true, wenn Winkel erfolgreich übernommen
        */
        bool setLeftLegAngles(const ColumnVector & pos);

        /*Setzt die Winkel für das rechte Bein
        * Die neuen Winkel werden nur übernommen, wenn die Werte nicht die
        * mechanischen Beschränkungen des Roboters überschreiten.
        * Parameter: Vektor(theta1,theta2,theta3,theta4,theta5,theta6)
        * Return: true, wenn Winkel erfolgreich übernommen
        */
        bool setRightLegAngles(const ColumnVector & pos);

        /*Gibt die momentanen Winkel der Motoren für das linke Bein im Modell zurück
        * Return: Vektor(theta1,theta2,theta3,theta4,theta5,theta6)
        */
        ReturnMatrix getLeftLegAngles() const;

        /*Gibt die momentanen Winkel der Motoren für das rechte Bein im Modell zurück
        * Return: Vektor(theta1,theta2,theta3,theta4,theta5,theta6)
        */
        ReturnMatrix getRightLegAngles() const;
```

```
/*Direkte Kinematik
* Gibt die Position des linken Fusses zurück.
* Die Position ist im BBKS des linken Beins und beschreibt den
* Kontaktpunkt des Kraftsensors.
* Return: Vektor(x,y,z,alpha,beta,gamma)
*/
ReturnMatrix getLeftFootPosition() const;

/*Direkte Kinematik
* Gibt die Position des rechten Fusses zurück.
* Die Position ist im BBKS des rechten Beins und beschreibt den
* Kontaktpunkt des Kraftsensors.
* Return: Vektor(x,y,z,alpha,beta,gamma)
*/
ReturnMatrix getRightFootPosition() const;

/*Inverse Kinematik
* Versucht für die gegebene Position im BBKS des linken Beins eine
* geeignete Stellung der Motoren des linken Beins zu finden.
* Wenn erfolgreich, enthält das Modell automatisch die neuen Stellungen
* der Motoren.
* Parameter: Vektor(x,y,z,alpha,beta,gamma)
* Return: 0:passende Stellung gefunden; 1:ausserhalb des Arbeitsbereichs;
* 2: mechanische Beschränkungen überschritten
*/
int setLeftFootPosition(const ColumnVector & pos);

/*Inverse Kinematik
* Versucht für die gegebene Position im BBKS des rechten Beins eine
* geeignete Stellung der Motoren des rechten Beins zu finden.
* Wenn erfolgreich, enthält das Modell automatisch die neuen Stellungen
* der Motoren.
* Position Vektor(x,y,z,alpha,beta,gamma)
* Return: 0:passende Stellung gefunden; 1:ausserhalb des Arbeitsbereichs;
* 2: mechanische Beschränkungen überschritten
*/
int setRightFootPosition(const ColumnVector & pos);

/*Ermittelt den Massenschwerpunkt des linken Beins im BBKS des linken
* Beins mit den momentan im Modell gesetzten Winkeln.
* Return: Vektor(x,y,z,masse)
*/
ReturnMatrix getLeftLegCenterOfMass();

/*Ermittelt den Massenschwerpunkt des rechten Beins im BBKS des rechten
* Beins mit den momentan im Modell gesetzten Winkeln.
* Return: Vektor(x,y,z,masse)
*/
ReturnMatrix getRightLegCenterOfMass();

/*Ermittelt den Massenschwerpunkt des gesamten Roboters im UKKS.
* Return: Vektor(x,y,z,masse)
*/
ReturnMatrix getRobotCenterOfMass();
```

```
        /*Überprüft ob der Roboter das Stabilitätskriterium erfüllt.
        * Parameter: left und right geben an, welcher Fuß Bodenkontakt hat.
        * Return: true, wenn Stabilitätskriterium erfüllt
        */
        bool isStaticStable(const bool & left, const bool & right);

        /*Überprüft ob der Roboter das Stabilitätskriterium erfüllt
        * Parameter: left und right geben an, welcher Fuß Bodenkontakt hat.
        *            xStable und yStable enthalten den normalisierten
        *               Abstand zum Mittelpunkt des Stützpolygons.
        * Return: true, wenn Stabilitätskriterium erfüllt
        */
        bool isStaticStable(const bool & left, const bool & right, double & xStable,
            double & yStable);

        /*Ermittelt das Stützpolygon des Roboters
        * Parameter: left und right geben an, welcher Fuß Bodenkontakt hat.
        * Return: Matrix mit Spaltenvektoren(x,y,z)
        */
        ReturnMatrix getSupportingConvexHull(const bool & left, const bool & right);

        /*Berechnet für eine gegebene Geschwindigkeit des linken Fußes
        * die entsprechende Geschwindigkeit in den Gelenken.
        * Zur Auswertung werden die aktuelle angegebenen Gelenkstellungen benutzt.
        * Parameter: karSpeed kartesische Geschwindigkeit des EE
        * Return: ColumnVector mit den Gelenkgeschwindigkeiten
        */
        ReturnMatrix getSpeedLeft(const ColumnVector & karSpeed);

        /*Berechnet für eine gegebene Geschwindigkeit des rechten Fußes
        * die entsprechende Geschwindigkeit der Gelenke.
        * Zur Auswertung werden die aktuelle angegebenen Gelenkstellungen benutzt.
        * Parameter: karSpeed kartesische Geschwindigkeit des EE
        * Return: ColumnVector mit den Gelenkgeschwindigkeiten
        */
        ReturnMatrix getSpeedRight(const ColumnVector & karSpeed);

private:
        Robot * leftLeg;
        Robot * rightLeg;
};
```

## A.2 Hilfsfunktionen

```
/*Überprüft ob sich zwei Linien schneiden
* Parameter: (x1,y1) - (x2,y2) Erste Linie
*            (x3,y3) - (x4,y4) Zweite Linie
*            (x,y) Kreuzungspunkt der Linien
* Return: 0:schneiden sich nicht; 1:schneiden sich; 2:liegen aufeinander
*/
int linesIntersect(const double & x1, const double & y1,
                   const double & x2, const double & y2,
```

```
                    const double & x3, const double & y3,
                    const double & x4, const double & y4,
                    double & x, double & y);

/*Punkt-In-Polygon Test
* Parameter: point: zu überprüfender Punkt
*            polygon: Stützpolygon
*            delta: Länge der Testlinien
*            xStable und yStable enthalten den normalisierten
*               Abstand zum Mittelpunkt des Stützpolygons.
* Return: true, wenn Punkt innerhalb des Polygons
*/
bool pointInPolygon(const ColumnVector & point, const Matrix & polygon, const int &
    delta, double & stableX , double & stableY);

/*Vereinigt die Positionen mehrerer Massenschwerpunkte in einen Massenschwerpunkt
* Parameter: coms: Matrix mit Spaltenvektoren(x,y,z,masse)
* Return: Spaltenvektor(x,y,z,masse)
*/
ReturnMatrix mergeCenterOfMass(const Matrix & coms);

/*Führt den Graham Algorithmus auf dem übergebenen Polygon durch.
* Parameter: Matrix mit Spaltenvektoren(x,y)
```

## A.3 Laufalgorithmus

```
/*Interpoliert die vorgegebenen Trajektorien für zwei Beine.
* Parameter: trajLeft und trajRight enthalten die jeweiligen Trajektorien der Beine
*            time: zu den Trajektorien passender Zeitvektor
*            interpol: Typ der Interpolation 0=keine; 1=kubisch; 2=B-Spline
*            intcount: Anzahl an Zwischenschritten, die interpoliert werden sollen
*            ts: Zeit zum Abfahren der kompletten Trajektorien
* Return: Matrix mit kompletten Trajektorien, Geschwindigkeit und Zeit für jeden Punkt
*/
ReturnMatrix interpolTrajectory(const Matrix & trajLeft, const Matrix & trajRight,
    ColumnVector & time, int interpol, int intCount, int ts );

/*Berechnet die Stützpunkte des Laufmusters nach gegebenen Parametern
* Parameter: downP: Absenken des Oberkörpers
*            heightP: Schritthöhe
*            lengthP: Schrittweite
*            sideP: Gewichtsverlagerung
*            steps: Schrittanzahl
*            trajLeft und trajRight enthalten die berechneten Stützpunkte
* Return: Anzahl an berechneten Stützpunkten
*/
int createWalking(double downP, double heightP, double lengthP, double sideP, int steps
    , Matrix & trajLeft, Matrix & trajRight);

/*Führt eine Anaylse der Trajektorien durch.
* Parameter: traj: Enthält die Positionen und Geschwindigkeiten der Beine
*                  die analysiert werden sollen
*            logFunc: Funktion an die Analyseergebnisse gesendet werden
```

```
*            h: Roboter mit dem die Analyse durchegführt wird
*/
void calculateTrajectory(Matrix & traj, void (*logFunc)(string), Humanoid & h);

/*Kubische Spline-Interpolation
*/
class KubSpline
{
public:
        /*Spline anlegen
        * Parameter: t: Zeitvektor
        *            p: Positionsvektor
        */
        KubSpline(ColumnVector t, ColumnVector p);

        // Wert der Kurve an der Stelle x bestimmen
        double getVal(double x);

        // Ableitung der Kurve an Stelle x bestimmen
        double getDerivative(double x);

private:
        ColumnVector t; //Zeitvektor
        ColumnVector p; //Positionsvektor

        // Koeefizienten der Spline-Kurve
        ColumnVector kA;
        ColumnVector kB;
        ColumnVector kC;
        ColumnVector kD;

        int n; // Anzahl an verwendeten Stützpunkten
};

/*B-Spline Interpolation
*/
class BSpline
{
public:
        /*Spline anlegen
        * Parameter: p: Matrix mit Punktvektoren
        *            deg: Grad der Wichtungsfunktion
        */
        BSpline(Matrix p, int deg);

        /*Interpoliert die komplette Kurve
        * Parameter: pointCount: Anzahl an Intervallen
        * Return: Interpolierte Werte mit pointCount Punkten
        */
        ReturnMatrix BSpline::interpol(int pointCount);

private:
        int c;      //Grad der Wichtungsfunktion
        int npts;   //Anzahl an Stütpunkten
        int dim;    //Dimension der Stützpunkte
```

```
        int range; //Wertebereich der Parameter

        Matrix p;        // Stützpunkte
        ColumnVector n; // Basisfunktionen
        ColumnVector x; // Parametervektor

        // Berechnet die Basisfuktionen an Stelle xT
        void calcBasis(double xT);

        // Generiert Parametervektor
        void generateKnots();
};
```

# Eidesstattliche Erklärung

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

Mannheim im April 2007

_____

Jörg Fellmann