Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MASTERTHESIS

Reinforcement Learning zum selbstoptimierenden Laufen mit Kraftsensoren im RoboCup

vorgelegt von

Fabian Fiedler

MIN-Fakultät

Fachbereich Informatik

TAMS - Technical Aspects of Multimodal Systems

Studiengang: Informatik

Matrikelnummer: 6208316

Erstgutachter: Prof. Dr. Jianwei Zhang

Zweitgutachter: Dr. Norman Hendrich

Abstract

Humanoid walking has been an active field of research for over 50 years. One of the widespread theories about a stable movement is to control the so called *zero-moment-point (ZMP)* inside the support polygon of the stance leg. However, calculating the exact position of the ZMP is not possible without deep knowledge about the robot, especially about the actuators, and the ground. Therefore it is hardly possible to calculate the trajectory, which achieves a desired position of the ZMP. Current walking algorithms use an approximation for the controlling, which is not accurate enough. To compensate for this inaccuracy, manually tuned parameters are added.

Fortunately the current ZMP can be determined by force sensors between the robot and the ground. This master's thesis intends to develop and integrate force sensors for the robots of the Department of Informatics based RoboCup team and to adapt the information about the ZMP via reinforcement learning to enhance the walking capabilities of the robots on non flat grounds such as artificial turf. This should render the widespread need for a tedious configuration of the walking algorithm unnecessary.

Zusammenfassung

Am Gang von humanoiden Robotern wird seit über 50 Jahren aktiv geforscht. Im Rahmen dieser Forschung ist die Theorie des *zero-moment-point (ZMP)* weit verbreitet. Dieser Punkt wird innerhalb des Support Polygons der Füße gehalten. Allerdings ist die genaue Berechnung ohne tiefergehendes Wissen über den Roboter und seine Motoren nicht möglich. Daher ist im Allgemeinen die Bewegung zum Steuern des ZMP zum gewünschten Punkt nicht berechenbar. Aktuelle Ansätze des humanoiden Gehens nutzen eine ungenaue Annäherung des ZMP für die generierten Bewegungen. Die Fehler dieser Annäherung werden durch Parameter ausgeglichen, die aufwendig durch Ausprobieren optimiert werden müssen.

Die Position des ZMP lässt sich allerdings auch durch Kraftsensoren in den Füßen berechnen. Ziel dieser Masterarbeit ist es, Kraftsensoren für die Füße zu entwickeln und diese in die Roboter der RoboCup-AG des Fachbereichs Informatik zu integrieren. Mit diesen Kraftsensoren soll nun der ZMP bestimmt werden sowie das Laufen der Roboter mittels Reinforcement Learning und dem gemessenen ZMP verbessert werden. Dadurch soll die bisher allgegenwärtige aufwendige Konfiguration des Laufalgorithmus durch den Nutzer vermieden werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

CoM	Center of Mass
ZMP	Zero-Moment-Point
CoP	Center of Pressure
IK	Inverse Kinematik
TD	Temporal Difference
PID	Proportional, Integral und Derivative
ROS	Robot Operating System

1 Einleitung

Die Hamburg Bit-Bots ist eine am Fachbereich Informatik der Universität Hamburg angesiedelte studentische Arbeitsgemeinschaft, die an den Fußball-Roboter-Weltmeisterschaften des RoboCup teilnimmt. Ein umfangreiches Forschungsthema innerhalb der Bit-Bots ist das Gehen der Roboter. In der Einleitung dieser Masterthesis wird erst das Forschungsfeld des Gehens beschrieben und dann eine Einführung in den RoboCup gegeben. Danach werden die genutzten Roboter vorgestellt. Abschließend erfolgt eine Analyse vergleichbarer Arbeiten.

Das zweite Kapitel dieser Thesis erklärt die theoretischen Grundlagen des Laufalgorithmus, die Bewegungen der Füße werden modelliert und das Stabilitätskriterium Zero-Moment-Point (ZMP) wird erklärt. Aus dem ZMP wird der Reward — also das Qualitätsmaß für den Lernalgorithmus — generiert. In den Kapiteln drei und vier werden sowohl die Grundlagen als auch die Umsetzung der Hardware dargestellt, um den ZMP präzise messen zu können. Auf Basis des gemessenen ZMP werden im fünften Kapitel zuerst die verwendeten Techniken zum Lernen des optimalen Ganges erklärt und anschließend auf die Modellierung und Umsetzung dieser Techniken zur eigentlichen Ausführung des Laufens eingegangen. Daran anschließend wird in Kapitel sechs die entwickelte Software evaluiert und dabei die Ergebnisse der Versuche mit dem Roboter vorgestellt. Am Ende erfolgt eine Zusammenfassung der Ergebnisse und ein Ausblick.

1.1 Motivation

Das Laufen von Robotern mit Beinen ist seit langem ein aktives Forschungsfeld. Trotzdem ist stabiles schnelles Laufen bisher selten erreicht und immer stark auf die jeweilige Plattform ausgelegt. Obwohl mit dem *Zero-Moment-Point (ZMP)* seit 1972 [Juricic and Vukobratovic, 1972] eine mathematische Modellierung für die Stabilität von Bewegungsabläufen

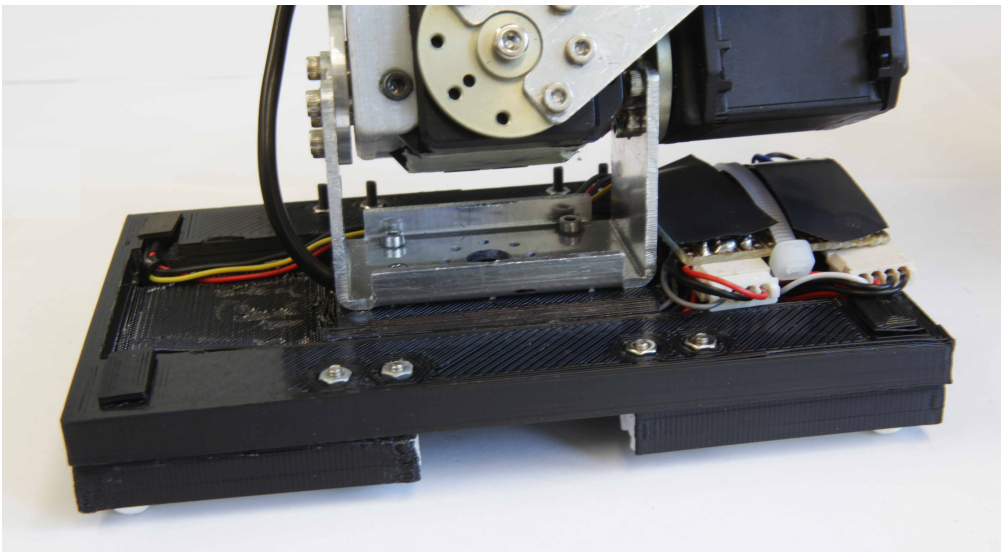

Abbildung 1.1: Im Rahmen der Masterarbeit entwickelter Fuß. In den Ecken sind in weiß die Spitzen der Kraftsensoren sichtbar, welche die beim Laufen entstehenden Kräfte messen. Oben sind Teile der Pitch- und Rollmotoren des Sprunggelenkes sichtbar. Rechts zwischen Fuß und Rollmotor befindet sich die entwickelte Prototyp-Platine, die die Sensoren ausliest.

besteht, ist die Entwicklung stabiler Laufalgorithmen auf Basis des ZMP bis heute schwierig. Um den ZMP während eines Bewegungsablaufes berechnen zu können, wird fundiertes Wissen über alle auftretenden Kräfte benötigt. Insbesondere der Einfluss der Aktoren auf den ZMP ist kaum abzuschätzen, da ihre Performance stark von ihrer Temperatur und weiteren Faktoren abhängig ist.

Der klassische Ansatz, diese Unkenntnis über die tatsächlich auftretenden Kräfte auszugleichen, ist das Einfügen von Parametern. Diese werden solange per Hand modifiziert, bis sich der Roboter stabil fortbewegt. Allerdings sind diese Parameter in der Regel nicht mathematisch herleitbar und ohne genaue Kenntnis des Algorithmus nicht anpassbar. Dieser Ansatz wird häufig in den Weltmeisterschaften des Roboterfußballturniers RoboCup angewendet. Ein Beispiel hierfür ist der Laufalgorithmus des Team DARwIn [Team-Darwin, 2017, Song, 2010], der von etlichen anderen Teams übernommen wurde. Ein anderes Beispiel ist der Laufalgorithmus des Teams NimbRo [Team Nimbro, 2009, Missura, 2005], der im Rahmen einer Promotion entwickelt wurde. Dieser Algorithmus umfasst 77 Parameter,

die stark auf eine spezielle Roboterplattform optimiert sind. Selbst der Autor des Algorithmus hatte es in den Weltmeisterschaften 2014 – 2016 nicht geschafft, dass der Algorithmus ebenso gute Ergebnisse mit der neu entwickelten Plattform erreicht.

Kraftsensoren bieten die Möglichkeit, die aktuelle Stabilität eines Roboters aktiv zu messen und dabei auf der realen Hardware zu bleiben. Es entsteht also keine Abstraktion durch den Einsatz von Simulatoren, die dazu führen, dass der Algorithmus in der Realität deutlich schlechtere Ergebnisse erzielt. Allerdings wird Hardware bei falscher Nutzung schnell beschädigt. Deshalb müssen Algorithmen, die mit Kraftsensoren arbeiten, den Roboter entsprechend vorsichtig steuern. Ferner sollten die Roboter auch mit dem Ausfall eines oder mehrerer Sensoren umgehen können.

Ziel dieser Masterarbeit ist es, einen Laufalgorithmus zu entwickeln, der die Stärken einer inversen Kinematik nutzt und auf dieser aufbauend selbstständig einen stabilen Gang lernt. Dabei soll der Algorithmus möglichst viel aus den Kraftsensoren im Fuß (siehe Abbildung 1.1) lernen und wenig auf Konfiguration oder Vorinitialisierung durch den Nutzer angewiesen sein. Dies soll insbesondere auch die Portierbarkeit des Ansatzes auf andere Plattformen gewährleisten und es soll nicht mehr notwendig sein, bei Änderungen an der Hardware weite Teile des Laufalgorithmus neu zu konfigurieren. Der Algorithmus soll trotzdem robust gegenüber beschädigten Sensoren bleiben, da diese z.B. während der Meisterschaften im RoboCup nicht immer direkt ausgetauscht werden können. Gleichzeitig soll die Möglichkeit gewährleistet bleiben, aufbauende Algorithmen zur Stabilisation anzuwenden.

1.2 RoboCup

Der RoboCup ist eine internationale Forschungsinitiative, die 1993 in Japan gegründet wurde und die 1995 das erste Mal einen internationalen Wettkampf in Montreal (Kanada) veranstaltet hat [RoboCup Federation, 2017]. Seitdem gibt es jährliche Weltmeisterschaften in verschiedenen Ligen und Ländern. Während es ursprünglich nur wenige Ligen gab, darunter Simulationsligen und eine Fußballliga auf kleinen, fahrenden Robotern, hat sich heutzutage ein breites Spektrum an Ligen entwickelt.

Neben den Ligen mit fahrenden Robotern, die heutzutage in der Middle-Size-Liga bereits sehr dynamische Spiele spielen, gibt es die Ligen mit humanoiden Robotern: Zum einen die Standard-Plattform-Liga, in der alle Teams mit Robotern des Typ *Nao* von Aldeberan Robotics spielen. In dieser Liga führt eine starke Fokussierung auf diesen Robotertyp ebenfalls zu Spielen, in denen die Strategie der Roboter spielentscheidend ist. Die Humanoid-Liga ist hingegen stark auf die Entwicklung der Hardware fokussiert. Hier treten die meisten Teams mit selbst entwickelten Robotern an, was den Codeaustausch erschwert, da der Code ausschließlich für die eigene Plattform entwickelt wird. Aber auch hier wird ein größerer Austausch gefördert: Zum Beispiel wird zurzeit unter der Leitung der Bit-Bots eine Standardisierung eines Software-Stacks für die Humanoid RoboCup-Ligen unterstützt [Bestmann, 2016]. Da der Fokus der Liga auf der Hardware liegt, sind insbesondere in dieser Liga die beiden Grundfähigkeiten Laufen und Bildverarbeitung spielentscheidend.

Abbildung 1.2: Links: Minibot (Mitte) beim Versuch über den Rasen zu laufen, was ein trippelndes, instabiles Schlittern ist. Rechts: Hambot (hinten) unbeweglich im Tor, da bisher kein funktionierender Laufalgorithmus existiert.

Neben den Fußballligen haben sich im RoboCup insgesamt zehn weitere Ligen herausgebildet. Beispiele für diese Ligen sind die Rescue-Liga, bei der Roboter entwickelt und programmiert werden, die in schwer zugänglichen Gebieten im Katastrophenfall die Rettungskräfte unterstützen sollen, sowie die Robocup@Home-Liga, deren Roboter als Serviceroboter im Haushalt helfen sollen.

Die Humanoid-Liga besteht aus drei Teilligen, die im Folgenden der Größe der teilnehmenden Roboter nach sortiert aufgelistet sind. Die Liga mit den größten Robotern ist die Adult-Size-Liga, deren Roboter mindestens 130 cm

groß sein müssen. Hier wurde bisher eins gegen eins in einem modifizierten Elfmeterschießen gespielt, bei dem der Ball hinter dem Roboter lag und dieser erst einmal hinter den Ball gehen musste, um ihn dann schießen zu können. Da die Regeln der Subligen aneinander angepasst werden, wird 2017 mit den Robotern erstmals ein reguläres Spiel „eins gegen eins“ gespielt, wobei die Besonderheit bleibt, dass ein sogenannter Robot-Handler hinter dem Roboter stehen darf, um ihn im Notfall aufzufangen, da diese Roboter ansonsten stark beschädigt werden könnten. Die Roboter der Teen-Size-Liga sind kleiner und dürfen zwischen 80 cm und 140 cm groß sein – hier wird „zwei gegen zwei“ gespielt. Die Roboter der Kid-Size-Liga sind die kleinsten mit 40 bis 90 cm und spielen „vier gegen vier“. Die Roboter der Hamburg Bit-Bots während eines Spieles der Kid-Size-Liga sind in Abbildung 1.2 zu sehen. Gerade in der Kid-Size-Liga müssen die Roboter Stürze aushalten und nach einem Sturz selbstständig aufstehen können. Da die Roboter entwickelt werden, um Stürze auszuhalten, ist die Anforderung an den Laufalgorithmus, niemals das Gleichgewicht zu verlieren, nicht zwingend erforderlich.

Neben der Größe gibt es noch andere Anforderungen an die Roboter, wobei die Regeln dem Grundsatz folgen, dass der Roboter möglichst menschenähnlich sein soll. Er darf also nur menschliche Sensorik, wie zum Beispiel (Stereo-)Kameras, Kraftsensoren, Gyroskope oder Accelerometer haben. Sensorik wie Magnetometer, Lasersensoren oder andere Tiefensensoren sind dagegen nicht erlaubt.

Auch die Länge der Arme ist auf die 1.2-fache Körperhöhe beschränkt. Für humanoides Gehen relevant ist insbesondere auch die Größe der Füße, deren Fläche $(2{,}2 \cdot H_{com})^2/32$ nicht überschreiten darf, wobei H_{com} die Höhe des Schwerpunktes des mit durchgestreckten Knien stehenden Roboters ist. Auch darf der Fuß nicht mehr als 2.5-mal so lang wie breit sein. Ferner muss der Roboter sich menschlich fortbewegen; Krabbeln oder Räder sind nicht erlaubt.

Seit 2015 wird in der Humanoid-Liga auf Kunstrasen gespielt, was insbesondere kleinere Roboter vor eine große Herausforderung stellt, da sie nicht schwer genug sind, den sehr nachgiebigen Rasen genug zu komprimieren, um eine glatte und gerade Fläche unter sich zu haben. Sie kippen daher schon beim Stehen leicht nach vorn oder hinten um. Ein großes Hindernis beim Laufen ist auch die in Abbildung 1.3 sichtbare Ausrichtung der Rasenhalme, die sehr prägnant ist. Während ein Roboter mit der Halmrichtung über das Feld rutschen könnte, ist dies gegen die Halmrichtung unmöglich.

Abbildung 1.3: Der Kunstrasen, der auch bei den German Open verwendet wird. Der sichtbare Farbunterschied zwischen links und rechts ist kein Unterschied im Teppich, sondern ausschließlich seine unterschiedliche Halmrichtung. Der Unterschied als Laufunterlage ist ähnlich prägnant wie die Farbunterschiede. Problematisch zum Laufen sind auch die häufig existierenden Falten, sichtbar als dunkle waagerechte Streifen im hellen Teil.

1.3 Roboterplattform

Im Rahmen der Hamburg Bit-Bots wurden zwei Roboterplattformen entwickelt, auf denen der in dieser Arbeit vorgestellte Laufalgorithmus angewendet werden soll. Beide Roboter nutzen die Aktuatoren der MX-Reihe von Robotis (siehe Abschnitt 3.1).

1.3.1 Minibot

Minibot (siehe Abbildung 1.4 links) hat 20 Freiheitsgrade und ist an das Design des DarWIN-OP von Robotis angelehnt [Ha et al., 2013]. Der Roboter hat sechs Motoren pro Bein, welche die Bewegungsfreiheit des Roboters ermöglichen. Je drei Motoren pro Hüfte sind eine Annäherung an ein Kugelgelenk. Diese stellen jeweils eine der drei *Euler-Winkel* dar, also Pitch, Yaw und Roll. Hinzu kommen ein Motor pro Knie und je ein Pitch- und Rollmotor in den Fußgelenken. Diese Struktur ist in Abbildung 1.5 und Abbildung 1.6 zu sehen und ermöglicht, dass der Roboter fast jede menschliche Position ansteuern kann. Verbunden sind die Motoren über ein Skelett aus Aluminium, das zusammen mit den Motoren den Hauptteil des Gewichts von ca. 5 kg ausmachen. Minibot wurde entwickelt, weil die

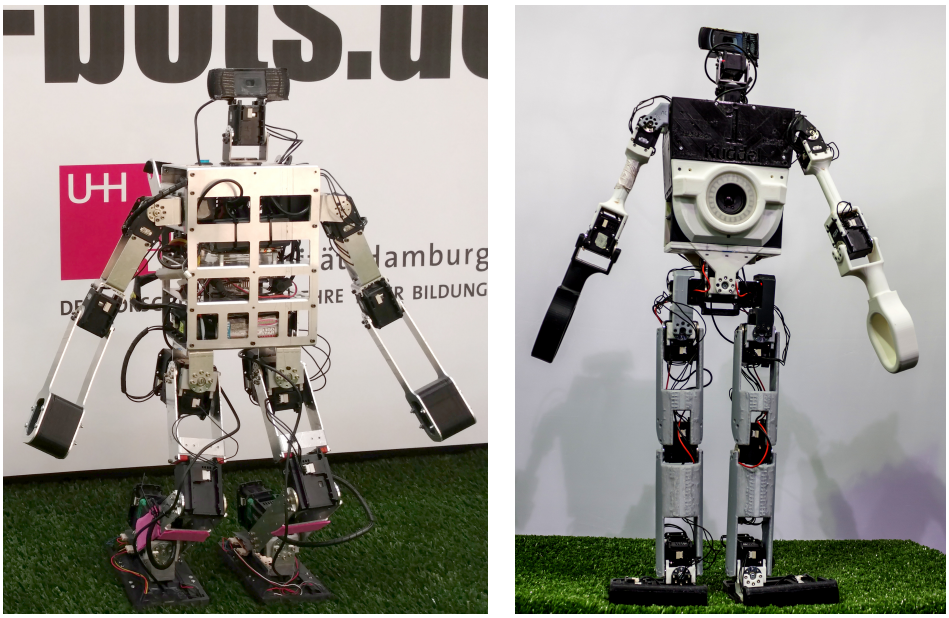

Abbildung 1.4: Minibot (links) und Hambot (rechts), die beiden primären Plattformen für die entwickelten Algorithmen.

vorher genutzten Darwin-OPs auf dem neuen Kunstrasen kaum noch stehen können, da sie zu leicht sind. Minibot ist schwerer und mit etwa 70 cm deutlich größer. Außerdem hat er stärkere Motoren, was die Bewegungen stabiler macht.

1.3.2 Hambot

Hambot [Bestmann et al., 2015] (siehe Abbildung 1.4 rechts) ist die Entwicklungsplattform der Hamburg Bit-Bots. Dieser ist komplett 3D-gedruckt und mit 87 cm und ca. 5,5 kg größer und etwas schwerer als Minibot. Da die komplette Elektronik für Hambot neu entwickelt wird, ist er bis heute in einem experimentellen Zustand, sodass Minibot die Hauptplattform ist. Beim Design der Algorithmen in dieser Arbeit wurde Hambot aber immer mitberücksichtigt.

Größter Unterschied zum Minibot sind neben den Zehen die zwei Freiheitsgrade in der Hüfte: Hambot kann seinen Oberkörper nach vorne und zur Seite neigen. Er hat damit eine Art Lendenwirbelsäule, die mehr Flexibilität liefert. Allerdings wird der Roboter durch das Spiel der weiteren Motoren instabiler (siehe Abschnitt 3.1).

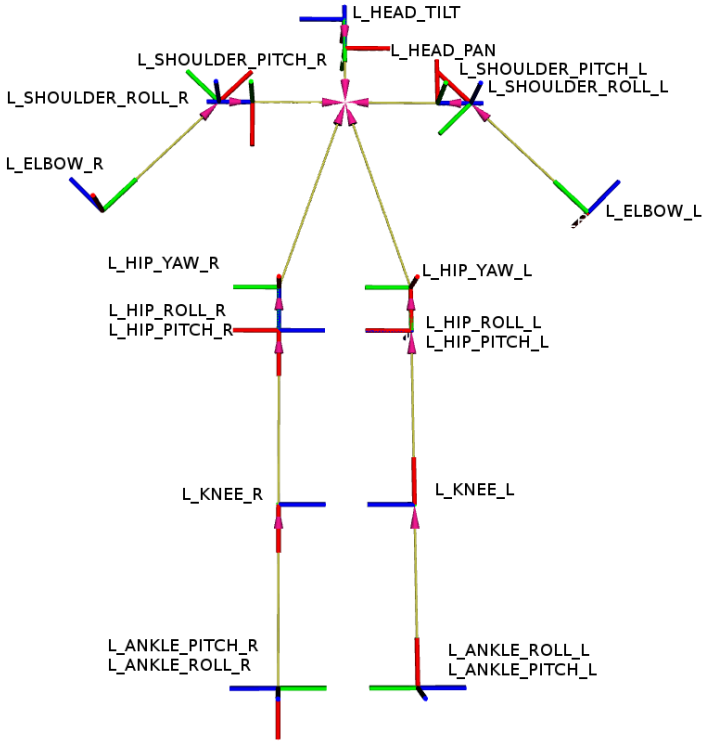

Abbildung 1.5: Das kinematische Modell des Minibot von vorne. Die Drehrichtungen der Motoren werden durch die blauen Achsen visualisiert. In der Hüfte und im Fuß liegen die Pitch und Roll Motoren direkt hintereinander. Deshalb überschneiden sich einige der Achsen der Koordinatensysteme.

1.4 Vergleichbare Arbeiten

Der klassische Ansatz laufender Roboter ist das Modellieren und Abschätzen der im Roboter auftretenden Kräfte.

Im RoboCup sind insbesondere Arbeiten von Seungmoon Song [Song et al., 2011, Song, 2010] relevant. In diesen beschreibt er das Design des Team-DARwIn Laufalgorithmus, der nicht nur ein de-facto-Standard für die RoboCup Humanoid-Kid-Size-Liga ist, sondern auch einen Ausgangspunkt für diese Masterthesis darstellt. Die zentrale Stelle des Laufalgorithmus ist die Definition des „Body swing“. Über diesen versucht Song

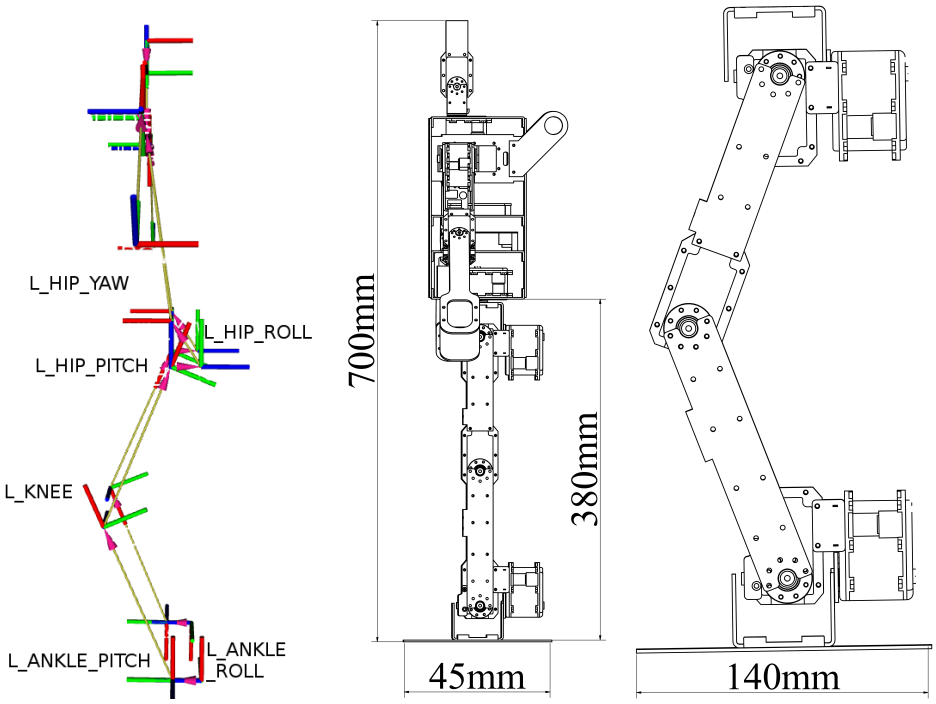

Abbildung 1.6: Das kinematische Modell des Minibot von der Seite (links). Die Drehrichtungen der Motoren werden durch die blauen Achsen visualisiert. In der Mitte ist eine technische Zeichnung des gesamten Roboters von der Seite dargestellt. Rechts ist eine technische Zeichnung eines Beines abgebildet, die eine Beispielkonfiguration zum Anheben des Beines mithilfe der Freiheitsgerade der Pitch-Motoren zeigt.

die optimale Bewegung des Oberkörpers herzuleiten. Sein Ergebnis ist, dass die optimale Bewegung entlang der x- bzw. Vorwärtsachse des Roboters gemäß folgender Formel erfolgt:

$$\begin{aligned} x(t) =& x_0 \cdot \cosh\left(\sqrt{\frac{g}{z_{CoM}}} \cdot t\right) + \\ & \left\{x_T - x_0 \cdot \cosh\left(\frac{g}{z_{CoM}} \cdot T\right)\right\} \cdot \frac{\sinh\left(\sqrt{\frac{g}{z_{CoM}}} \cdot t\right)}{\sinh\left(\sqrt{\frac{g}{z_{CoM]}}} \cdot T\right)} \end{aligned} \tag{1.1}$$

In dieser Formel ist t der aktuelle Zeitpunkt, T der Endzeitpunkt des Schrittes, x_0 die Startposition, x_T die Endposition, g die Gravitation und z_{CoM} die z-Position des Schwerpunktes. Allerdings kommt Song zu dem Ergebnis, dass diese Formel auf echten Robotern nicht funktioniert und fügt deswegen einen nicht hergeleiteten Parameter $tZMP$ ein [Song, 2010].

$$\begin{aligned} x(t) =& x_0 \cdot \cosh\left(\sqrt{\frac{g}{z_{CoM}}} \cdot \frac{t}{tZMP}\right) + \\ & \left\{x_T - x_0 \cdot \cosh\left(\frac{g}{z_{CoM}} \cdot \frac{T}{tZMP}\right)\right\} \cdot \frac{\sinh\left(\sqrt{\frac{g}{z_{CoM}}} \cdot \frac{t}{tZMP}\right)}{\sinh\left(\sqrt{\frac{g}{z_{CoM}}} \cdot \frac{T}{tZMP}\right)} \end{aligned} \tag{1.2}$$

Dieser Parameter $tZMP$ muss manuell durch Ausprobieren auf der Roboterplattform optimiert werden. Da sich der Parameter aber nicht konstant über alle Geschwindigkeiten und Beschleunigungen verhält, müssen weitere Offsets hinzugefügt werden, sodass es am Ende 20 Konfigurationswerte für den Oberkörper gibt, von denen die meisten nicht herleitbar sind, sondern aufwendig manuell optimiert werden müssen.

Ein anderer im RoboCup verbreiteter Algorithmus ist der des Team-NimbRo, der im Rahmen der Doktorarbeit von Marcell Missoura bei Prof. Dr. Behnke an der Universität Bonn entstanden ist [Missura, 2005]. Dieser Laufalgorithmus hat zwei Komponenten: Zum einen die Berechnung der Grundbewegung und zum anderen einen darauf aufbauenden Stabilisator, der auf den sogenannten Capture-Points basiert. Die Capture-Points haben eine einfach nachvollziehbare physikalische Definition als Stabilisator. Die Berechnung der Grundbewegung bleibt allerdings zentraler Bestandteil der Stabilität und ist noch mehr per Hand optimiert als das Team-DARwIn-Walking. So hat der ganze Laufalgorithmus insgesamt 140 Konfigurationswerte, von denen die meisten auf die Grundbewegung fallen. Es hat sich gezeigt, dass es nicht reicht, einen groben Laufalgorithmus zu entwickeln und die eigentliche Stabilität von den Capture-Steps gewährleisten zu lassen [Schmidt, 2015].

Der humanoide Roboter iCub ist eine von der EU geförderte Roboterplattform, die mittlerweile an vielen Universitäten verbreitet ist. iCub ist einem dreieinhalbjährigen Kind nachempfunden und kann offiziell nur krabbeln [Metta et al., 2008]. Allerdings wurden Laufalgorithmen auch auf dem iCub ausprobiert. Ein funktionierender, ebenfalls stark manuell optimierter

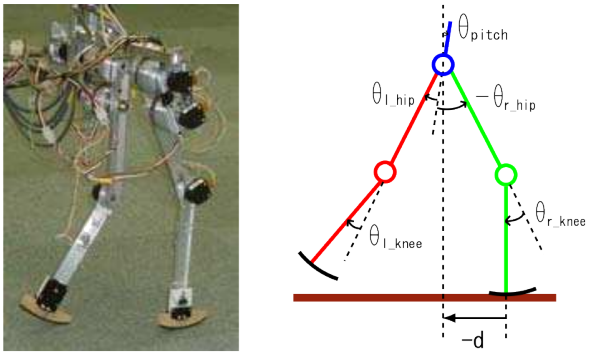

Abbildung 1.7: Ein einfacher passiv-dynamischer Laufroboter mit 4 Motoren: 2 in der Hüfte und 2 im Knie [Morimoto et al., 2005]. Da der Roboter nur Pitch-Motoren hat, kann er nur nach vorne gehen.

Ansatz ist der von Hu et al. [Hu et al., 2016]. Dieser erreicht allerdings nur eine Geschwindigkeit von 0,037 m/s, was im Kontext des Robocup vergleichsweise langsam ist.

Eine weitere verbreitete Strategie ist, das Laufen von zweibeinigen Robotern mithilfe von Reinforcement Learning zu optimieren. Hierbei ist insbesondere der Ansatz von Morimoto et al. [Morimoto et al., 2005] interessant, die mithilfe von einer Poincaré-Map eine Abbildung entwickelt haben, in der man einen zweckmäßigen Suchraum aufspannen kann. Dieser Suchraum kann nun exploriert werden, ohne dass der Roboter schnell in instabile Situationen kommt. Nach einem anfänglichen Lernen im Simulator kann der Roboter erfolgreich sein Laufen auf der echten Hardware verbessern. Diese Roboter haben allerdings eine eingeschränkte, nicht menschliche Bewegungsfreiheit, die häufig auf die Vorwärtsachse beschränkt ist. Sie fallen nicht so leicht um und sind daher einfacher zu optimieren. Auch in der Doktorarbeit von Tedrake [Tedrake, 2004] findet sich diese Art von Robotern. Allerdings ist bei keinem dieser Ansätze sichtbar, wie man sie auf komplexere Roboter ausweiten kann, ohne einen zu großen Suchraum zu erreichen.

Im Rahmen des RoboCups wurde vom Team-EROS ein neuartiger Ansatz [Saputra et al., 2015] vorgeschlagen, der vollständig auf künstlichen neuronalen Netzen basiert. Aber auch dieser wurde noch nicht öffentlich auf Hardware getestet, weshalb seine Performance schwer abschätzbar ist.

Der Ansatz der aktuellen Robocup-Humanoid-Kidsize-Weltmeister ist ein sehr einfacher Open-Loop-Spline-basierter Ansatz, der sehr einfach auf die Zielplattform konfigurierbar ist und dessen Ungenauigkeiten sich durch einen Stabilisator, der Kraftsensoren nutzt, ausgleichen lässt [Rhoban, 2009]. Während Stabilisatoren auf Basis von Kraftsensoren verbreitet sind, gibt es keine Veröffentlichungen über Lernverfahren des eigentlichen Laufalgorithmus mithilfe von Kraftsensoren auf humanoiden Robotern.

2 Theoretische Grundlagen

In diesem Kapitel werden die wichtigsten mathematischen Begriffe erläutert. Zuerst wird auf die direkte und inverse Kinematik eingegangen, die das Verhältnis von Position eines Endeffektors im Raum zu den Gelenkwinkeln berechnen. Wie der Gang humanoider Roboter spezifiziert werden kann, wird im Anschluss erläutert. Danach wird dargestellt, unter welchen Bedingungen ein Roboter in seiner Bewegung stabil ist.

2.1 Direkte und inverse Kinematik

Die direkte Kinematik berechnet aus Gelenkwinkeln eine Position des Endeffektors im Raum. Nach einer anfänglichen Spezifikation der Entfernungen und Rotationen der einzelnen Gelenke zueinander, lässt sich die Position der Endeffektoren durch einfache Matrix-Multiplikation darstellen [Siciliano et al., 2008].

Die inverse Kinematik (IK) hingegen berechnet aus einer gegebenen Position des Endeffektors im Raum die dazugehörigen Gelenkwinkel. Diese Berechnung ist mathematisch deutlich komplexer. Im Allgemeinen gibt es zwei Ansätze zum Lösen der inversen Kinematik. Zum einen kann ein analytischer Ansatz gewählt werden, dieser muss aber speziell für jeden Robotertyp einzeln programmiert und optimiert werden. Analytische IK-Solver haben zusätzlich die Eigenschaft, dass für eine gegebene Position des Endeffektors die Gelenkwinkel immer gleich sind. Zum anderen kann ein numerischer Näherungsalgorithmus genutzt werden. Dieser kann auf beliebigen Robotertypen arbeiten, braucht aber deutlich mehr Rechenzeit und ist im Allgemeinen ungenauer als die analytischen Algorithmen.

Im Rahmen dieser Masterarbeit wurde ein analytischer IK-Solver des RoboCup-Teams NUbots von der Universität von Newcastle in Australien genutzt [NuBots, 2017].

2.2 Menschliches Laufen

Ziel der Robotik im Rahmen des RoboCups ist es Roboter zu entwickeln, welche dem menschlichen Vorbild möglichst ähnlich sind. Daher ist die kinetische Struktur dem Menschen nachempfunden und es ist wünschenswert auch den menschlichen Gang zu erreichen.

Ein medizinisches Modell der Laufabfolge eines Menschen ist in Abbildung 2.1 zu sehen. Die grundsätzliche Unterteilung von belasteter und unbelasteter Phase ist auch für ein humanoides Gehen auf Robotern sinnvoll. Allerdings überschneiden sich die beiden doppelt belasteten Phasen im medizinischen Modell und sind damit Teile beider Belastungsphasen. Um eine Partitionierung und gleichmäßige Aufteilung zu ermöglichen, ist es daher für die Modellierung des Laufalgorithmus ratsam, die doppelt belasteten Phasen nochmals aufzuteilen. Die doppelt belastete Phase hat dann ein Standbein, das die Hauptlast trägt und ein Spielbein, das weniger oder keine Last des Roboters trägt.

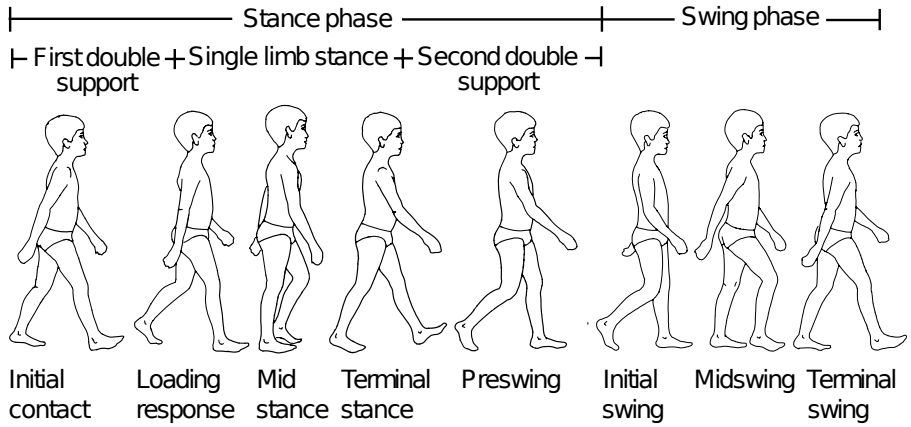

Abbildung 2.1: Das Laufmuster eines Jugendlichen [Vaughan and Brian Davis, 1999]. Die Bezeichnungen der belasteten (stance) und der unbelasteten (swing) Phase beziehen sich auf das rechte Bein. Im medizinischen Modell überschneiden sich die doppelt belasteten (double support) Phasen der beiden Beine.

Mit der gegebenen Hardware nicht hinreichend abbildbar ist allerdings die Abrollbewegung des menschlichen Ganges. In Abbildung 2.2 ist links ein menschlicher Fuß zu sehen und rechts der Fuß von Minibot. Es ist auffällig, dass der menschliche Fuß deutlich komplexer als der des Roboters

ist. Insbesondere die Muskeln und Knochen in Mittelfuß und Zehen sind beim Roboter nicht vorhanden, genaueres zum Design des Fußes findet sich in Abschnitt 4.1. Während die Zehen noch in den Fuß integriert werden können [Bestmann et al., 2015], ist der Platz für Hardware zum aktiven Steuern des Mittelfußes nicht ausreichend. Dieser aktive Mittelfuß ist allerdings für das menschliche Gehen notwendig, wie in Abbildung 2.3 zu sehen ist. Im Teilschritt b ist der Fußaußenrand noch nicht belastet, in Teilschritt c aber vollständig. Diese Druckveränderung kann nur zustande kommen, wenn der Mensch beim Gehen die Haltung des Mittelfußes ändert. Dies führt dazu, dass ein Roboter keine menschenähnliche Abrollbewegung machen kann.

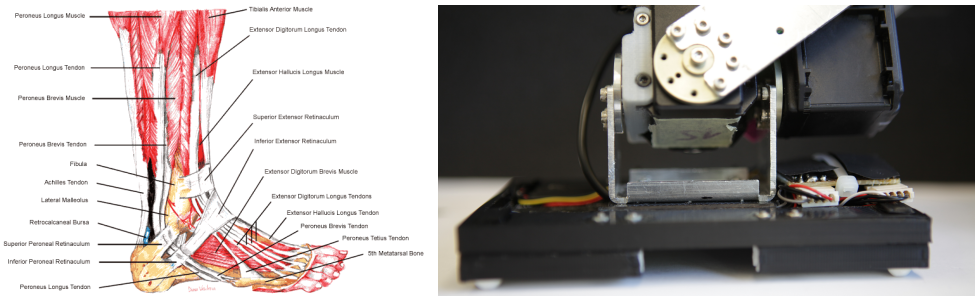

Abbildung 2.2: Fuß eines Menschen (links) [Swierzewski, 2017] und Fuß eines Roboters (rechts). Während der Roboterfuß aufgrund der beiden Motoren nur über zwei Freiheitsgrade verfügt, besteht der menschliche Fuß neben den Knochen aus vielen Sehnen und Muskeln, die zu vielen Freiheitsgraden führen.

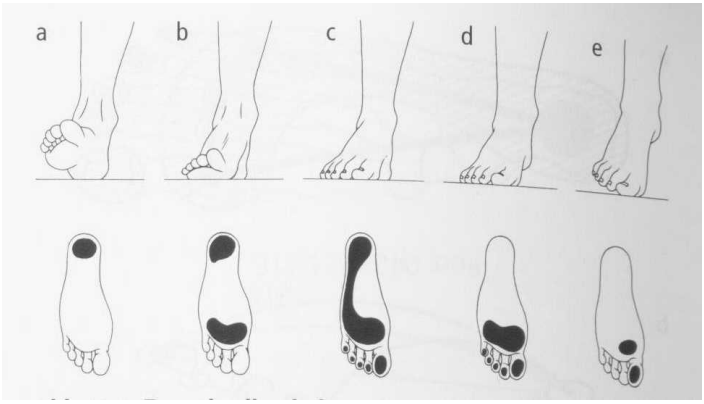

Abbildung 2.3: Visualisierung der Fußbelastung beim Gehen. Oben ist die Fußposition und darunter die dazugehörige Druckverteilung zu sehen [Debrunner, 2002].

2.3 Gang humanoider Roboter

Das generelle Ziel der Laufalgorithmen ist es, sich möglichst nahe an das menschliche Gehen anzunähern (vgl. Abschnitt 2.2). Mithilfe einer inversen Kinematik lassen sich für den Gang relevante Bewegungen der Beine mithilfe der Zielpositionen von Oberkörper, rechtem Fuß und linkem Fuß beschreiben. Da Roboter beim Gehen ihre Füße nicht abrollen können, wird häufig modelliert, dass die Fußsohle immer parallel zum Boden bleibt. Dadurch wird die Berechnung des Gehens deutlich vereinfacht, da so für den unbelasteten Fuß die Zielposition durch Höhe und Vorwärtsbewegung bzw. Seitwärtsbewegung modellierbar ist.

Um den Bewegungsablauf optimal zu gestalten, muss die Eigenschaft der Motoren beachtet werden, dass sie nicht die Position direkt sondern den Antriebsstrom steuern. Der Antriebsstrom korreliert stark mit dem Drehmoment des Motors. Deswegen sollten weder die Position noch die Geschwindigkeit Sprünge in ihren Werten haben, d.h. die Positionsfunktion sollte zweimal differenzierbar sein.

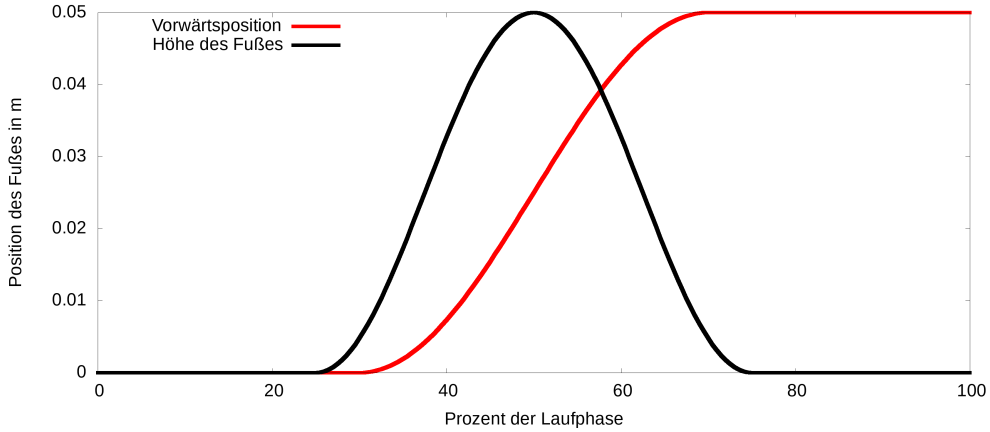

Abbildung 2.4: Zielpositionen des Fußes des Spielbeins. Während das Spielbein den Boden berührt, die Höhe also null ist, befindet sich der Roboter in einer der doppelt belasteten Phasen. Der Roboter bewegt in diesen seinen Oberkörper, um das Standbein und Spielbein zu wechseln. Wenn das Spielbein angehoben wird, liegt die Last vollständig auf dem Standbein. Das nun unbelastete Bein kann nach vorne bewegt werden.

Die Sinusfunktion ist beliebig oft differenzierbar und leicht konfigurierbar, daher eignet sie sich gut als Positionssignal. Für die Vorwärts- bzw. Seitwärtsbewegung eignet sich die Funktion: $\sin(x)+1$ für $x \in [0,\pi)$. Die Höhe hingegen lässt sich durch $\sin(x)+1$ für $x \in [0,2\pi)$ darstellen. Diese beiden Funktionen ergeben die in Abbildung 2.4 dargestellten Bewegungen.

Bewegt der Roboter nur den unbelasteten Fuß, fällt er um, da diese Bewegung nicht stabil ist. Es muss also auch der Oberkörper bzw. der Schwerpunkt des Roboters bewegt werden, um eine stabile Bewegung zu erreichen, die dazu führt, dass der Roboter nicht umkippt. Da aus der Beobachtersicht der belastete Fuß sich nicht bewegt, sollte sich auch die modellierte Position des belasteten Fußes nicht ändern. Die Ausgleichsbewegung sollte also ausschließlich durch die Position des Oberkörpers gesteuert werden. Das Laufen der Roboter kann daher als Tripel (Position des Oberkörpers, Position des linken Fußes, Position des rechten Fußes) aufgefasst werden, von denen die Positionen der beiden Füße leicht definierbar sind. Die Steuerung des Oberkörpers soll die Stabilität des Systems gewährleisten. Dies wird in den nächsten Kapiteln beschrieben. Der Mensch hat dabei seinen Oberkörper und insbesondere auch den Kopf immer aufrecht, was auch im RoboCup eine wichtige Anforderung ist, damit das Sichtfeld des Roboters nicht eingeschränkt wird.

Die Schrittfrequenz des Roboters ist nach Song [Song, 2010] optimal, wenn sie mit der Eigenfrequenz des virtuellen Pendels vom CoM zum Boden korreliert. Mit z_{CoM} als Höhe des CoM und g der Gravitationskonstante ist die Frequenz des Ganges f_{Gang} mit dem wählbaren Faktor a:

$$f_{Gang} = a \cdot \sqrt{\frac{g}{z_{CoM}}} \quad \Big| \quad a \in \left\{ n \cup \frac{1}{n} \mid n \in \mathbb{N} \right\} \tag{2.1}$$

Da die Schrittfrequenz nicht beliebig sein kann, sollte die Geschwindigkeit des Gehens hauptsächlich mit der Schrittweite gesteuert werden.

2.4 Stabilität von Robotern

Damit der Roboter möglichst stabil laufen kann, ist es notwendig herauszufinden, wie stabil der Roboter gerade ist. Hierfür wird der relevante Begriff der *konvexen Hülle* kurz erläutert und anschließend mit dem *Center-of-Mass* (CoM) das Stabilitätskriterium für ein System ohne Kräfte erklärt.

Dieses wird dann mit dem Zero-Moment-Point (ZMP) auf den dynamischen Fall erweitert, um abschließend mit dem messbaren Center-of-Pressure in Bezug gesetzt zu werden.

2.4.1 Konvexe Hülle

Für die formale Definition der Stabilität von Robotern ist der Begriff der konvexen Hülle relevant. Eine Menge M von Punkten (z.B. Berührungspunkte des Roboters mit dem Boden) ist konvex, wenn auch stets alle Punkte auf der Verbindungsstrecke von je zwei Punkten zu dieser Menge M gehören [Lau, 2004]. Die konvexe Hülle ist dann die kleinste konvexe Menge $\overline{M}$ um alle Punkte der Menge M:

$$\begin{aligned}\overline{M} := \Big\{ \mathcal{X} \in \mathbb{R}^{n\times 1} \;\Big|\; \exists t \in \mathbb{N}\ \exists \lambda_1, \ldots, \lambda_t \in \mathbb{R}\ \exists \mathcal{X}_1, \ldots, \mathcal{X}_t \in M : \\ (\forall i : 0 \leq \lambda_i \leq 1) \wedge \sum_{i=1}^{t} \lambda_{i=1} \wedge \mathbb{X} = \sum_{t=1}^{t} \lambda_i \cdot \mathcal{X}_i \Big\}\end{aligned} \tag{2.2}$$

2.4.2 Schwerpunkt

Wenn mit Ausnahme der Gravitation keine Kräfte auf ein Objekt wirken, so ist der Schwerpunkt (CoM – vom englischen Center of Mass) die kritische Kenngröße. Das Center of Mass ist für ein Objekt R, das aus n Teilobjekten besteht, wie folgt definiert:

$$CoM(R) = \frac{1}{\sum_{i=1}^{n} m_i} \sum_{i=1}^{n} m_i r_i \tag{2.3}$$

mit r_i als Position vom Teilobjekt i und m_i als Masse des Teilobjekts i [Radi, 2013].

Wenn das CoM innerhalb der konvexen Hülle der Standflächen ist, steht das Objekt stabil, wie in Abbildung 2.5 zu sehen. Diese Fläche wird auch als Support Polygon bezeichnet. Wenn der CoM außerhalb der konvexen Hülle liegt, fällt bzw. dreht sich das Objekt in die Richtung des CoM. Hierbei korreliert die Geschwindigkeit der Drehung mit der Entfernung des CoM von der konvexen Hülle.

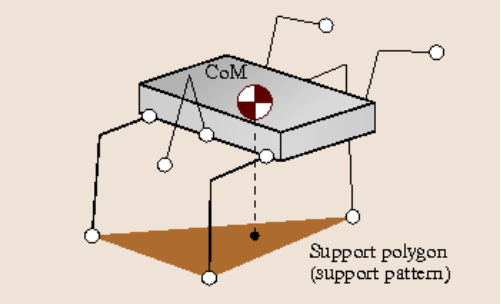

Abbildung 2.5: Visualisierung des Support Polygons eines sechsbeinigen Roboters [Siciliano and Khatib, 2008]. Das in orange eingezeichnete Support Polygon ist durch die Standbeine des Roboters definiert.

Wenn kein Feedback über die aktuellen Kräfte vorliegt, ist es im allgemeinen Fall zweckmäßig, das CoM in die Mitte der konvexen Hülle zu verschieben. Dadurch können möglichst große Interferenzen, wie zum Beispiel Stöße, abgefangen werden.

Das CoM ist aber nur dann ausschlaggebend, wenn keine Kräfte auf den Roboter wirken. Wenn hingegen eine Kraft nach vorne auf den Roboter wirkt, kann das den Roboter aus dem Gleichgewicht bringen, auch wenn die Position des CoM nicht verändert wird. Daher muss ein Stabilitätsmaß entwickelt werden, das die Kräfte beachtet: Der Zero-Moment-Point.

2.4.3 Zero-Moment-Point

Defintion

Der Zero-Moment-Point (ZMP), ist der Punkt auf der Oberfläche des Bodens, an dem die Summe aller anliegenden physikalischen Momente gleich null ist. Der Punkt kann stark vereinfacht als Verallgemeinerung des CoM unter Beachtung der auftretenden Kräfte interpretiert werden. Wenn der ZMP innerhalb der konvexen Hülle des Fußes liegt, steht der Roboter stabil.

Vukobratovic legte die Grundsteine für den ZMP und definierte ihn als:

Definition 1. „The overall indicator of the mechanism behavior is the point where the influence of all forces acting on the mechanism can be replaced by one single force. This point was termed the Zero-Moment-Point.“ [Vukobratović and Borovac, 2004]

In der Literatur bestehen verschiedene Definitionen, die jeweils auf bestimmte Aspekte des ZMP fokussiert sind.

Definition 2. „The Zero-Moment-Point is that point on the ground at which the net moment of the inertial forces and the gravity forces has no component along the horizontal axes.“ [Dasgupta and Nakamura, 1999]

Definition 3. „p is the point that $T_x = 0$ and $T_y = 0$, T_x, T_y represent the moments around x- and y-axis generated by reaction force F_r and reaction torque T_r , respectively. The point p is defined as the Zero Moment Point (ZMP). When ZMP exists within the domain of the support surface, the contact between the ground and the support leg is stable:

$$p_{ZMP} = (x_{ZMP}, y_{ZMP}, 0) \in S,$$

where p_{ZMP} denotes a position of ZMP. S denotes a domain of the support surface. This condition indicates that no rotation around the edges of the foot occurs.“ [Arakawa and Fukuda, 1997]

Definition 4. „ZMP is the point on the ground where the total moment generated due to gravity and inertia equals to zero.“ [Takanishi et al., 1990]

Definition 5. „The ZMP (Zero-Moment Point) is defined to be a point on the ground at which the tangential component of the moment generated by the ground reaction force/moment becomes zero.“ [Harada et al., 2003]

Definition 6. „The ZMP is defined as that point on the ground at which the net moment of the inertial forces, the external disturbance and gravity forces has no component, along the horizontal axes.“ [van Oort and Stramigioli, 2011]

Die Definitionen 3 und 5 weisen auf eine wichtige Bedingung hin, die bei den anderen nur implizit mitschwingt: Die Reibungskraft zwischen Fuß und Boden muss hoch genug sein, um die entstehenden Kräfte entlang des Bodens aufzuheben.

Was die Definitionen nicht deutlich machen, ist eine in der Wissenschaft bis heute geführte Diskussion, wie der ZMP im Falle eines instabilen Systems definiert ist. Während die verbreiteten mathematischen Modellierungen des ZMP sinnvolle Ergebnisse für instabile Systeme liefern, widersprechen einige Forscher, dass diese Berechnungen dann der ZMP seien. Es gibt hauptsächlich drei Interpretationen für den instabilen Fall:

- Der ZMP kann immer dem Druck unter dem Fuß zugeordnet werden. Der ZMP bleibt dann an der Kante des Fußes [Sardain and Bessonnet, 2004].
- Die mathematische Modellierung berechnet einen Punkt außerhalb des Fußes, dieser wird als ZMP gesehen [Vukobratović and Borovac, 2004].
- Der ZMP ist nur im Falle eines stabilen Systems definiert, die Ergebnisse der Berechnungen werden dann häufig als Foot-Rotation-Index (FRI) bezeichnet [Goswami, 1999].

Mathematische Grundlage

Für die Berechnung des Zero-Moment-Point ist das Moment $\boldsymbol{M}_Q^{gi}$ um den beliebigen Punkt Q wichtig [Sardain and Bessonnet, 2004]:

$$\boldsymbol{M}_Q^{gi} = \boldsymbol{QG} \times m\boldsymbol{g} - \boldsymbol{QG} \times m\boldsymbol{a}_G - \dot{\boldsymbol{H}}_G \tag{2.4}$$

Bei dieser Gleichung ist $\boldsymbol{QG}$ der Vektor von Q zu G, m ist die Masse und $\boldsymbol{g}$ ist die Gravitationskraft. Also ist $\boldsymbol{QG} \times m\boldsymbol{g}$ der Einfluss der Gravitationskraft auf den Punkt Q. Wohingegen $\boldsymbol{a}_G$ die Beschleunigung des CoM und $\dot{\boldsymbol{H}}_G$ das Drehmoment des CoM beschreiben. Da der ZMP der Punkt ist, an dem kein Drehmoment existiert, muss er also die Gleichung 2.5 erfüllen.

$$\boldsymbol{M}_Q^{gi} \times n = 0 \tag{2.5}$$

Wobei n die Normale des Untergrunds ist, also im Regelfall die z-Achse des Systems [Sardain and Bessonnet, 2004].

Die Gleichungen 2.4 und 2.5 lassen sich zusammenfassen. Wenn zusätzlich die Normale des Untergrunds der z-Achse gleichsetzt wird, lässt sich der ZMP wie folgt berechnen (vgl. [Sardain and Bessonnet, 2004]):

$$\boldsymbol{OD} = \frac{mgz \times \boldsymbol{OG} \times z + z \times \dot{\boldsymbol{H}}_G}{mg + m\boldsymbol{a}_g \cdot z} \tag{2.6}$$

Der so berechnete Zero-Moment-Point ist nun ein direktes Stabilitätsmaß für den Roboter. Der Gangalgorithmus sollte nun den tatsächlichen ZMP möglichst nahe am optimalen halten. Da der Roboter keine abrollende Bewegung machen kann, ist im Allgemeinen der optimale ZMP in der

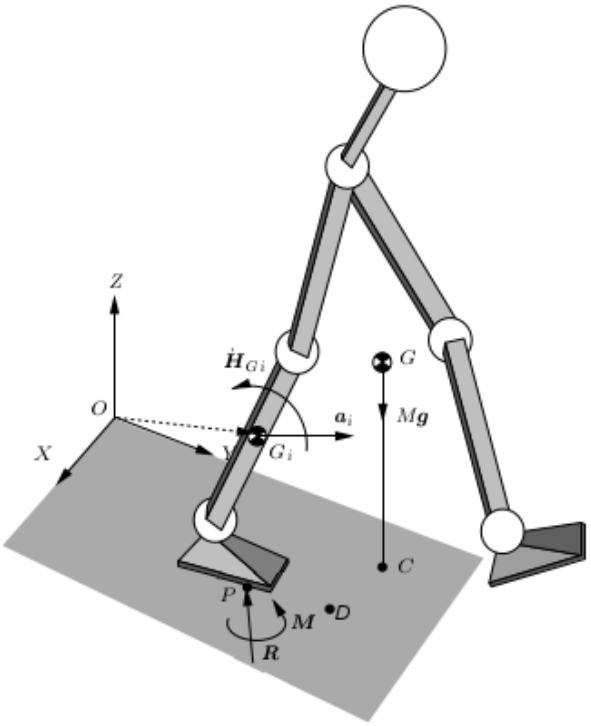

Abbildung 2.6: Visualisierung des ZMP eines instabilen Systems. G ist der Schwerpunkt des Roboters und C seine Projektion auf den Boden. D ist der *ZMP*, P ist der *COP* (modifiziert nach [Goswami, 1999]).

Mitte des Fußes während der einzeln belasteten Phase, da der Roboter auf diese Weise einen möglichst großen Sicherheitsrahmen hat. Er kann so die größtmöglichen Stöße in jede Richtung aushalten ohne umzukippen.

Wenn $\boldsymbol{OD}$ der Vektor von einem beliebigen Ursprung O zum aktuellen ZMP und $\boldsymbol{OZ}$ der Vektor zum optimalen ZMP ist, sollte der Laufalgorithmus nach Dasgupta die mittlere quadratische Abweichung minimieren [Dasgupta and Nakamura, 1999]:

$$\text{minimize} \int ||\boldsymbol{OD} - \boldsymbol{OZ}||^2 \mathrm{d}t \tag{2.7}$$

Da sich über diese Formel die Qualität eines Schrittes quantifizieren lässt, kann ein Lernalgorithmus, der den aktuellen ZMP kennt, das Ergebnis des Integrals für eine festlegbare Zeit als Reward verwenden.

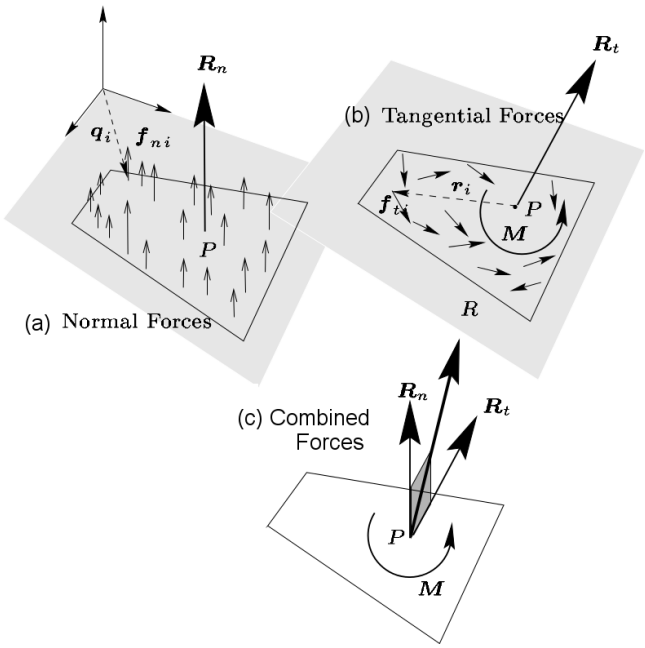

Abbildung 2.7: Visualisierung der auftretenden Kräfte unter einem flachen Fuß und dem dazugehörigen CoP [Goswami, 1999]. Die Kräfte lassen sich in zwei Klassen einteilen, zum einen die Kräfte entlang der Normalen des Fußes und zum anderen die Drehkräfte unter dem Fuß. Die Summe der Kräfte wirken am CoP.

2.4.4 Center of Pressure

> „CoP [Center of Pressure] represents the point on the support foot polygon at which the resultant of distributed foot ground reaction forces acts.“ [Vukobratovic et al., 2001]

Mit Hilfe von Kraftsensoren ist es möglich, die Kräfte zwischen Roboter und Boden zu messen. Diese Kräfte können als eine einzelne Kraft dargestellt werden, die an einem bestimmten Punkt (vgl. Abbildung 2.7 rechts) anliegt: Dem Center of Pressure (CoP). Hierfür reicht es, im Falle eines flachen Bodens, vier Kraftsensoren in die Ecken des Fußes zu integrieren [Vukobratović and Borovac, 2004].

Nun lässt sich der CoP leicht berechnen:

$$\boldsymbol{OP} = \frac{\sum r_i f_i}{\sum f_i} \tag{2.8}$$

wobei r_i die Position des i-ten Sensors und f_i dem Sensorwert entspricht [Goswami, 2015].

Es lässt sich zeigen, dass der CoP und der ZMP im Falle eines stabilen Ganges übereinstimmen [Sardain and Bessonnet, 2004] [Vukobratovic et al., 2001]. Mithilfe von vier einfachen Kraftsensoren lässt sich also direkt ein Stabilitätsmaß für einen humanoiden Gang berechnen.

3 Hardwaregrundlagen

In diesem Kapitel wird auf die genutzten Servomotoren eingegangen. Dabei wird insbesondere ein Fokus darauf gelegt, wie sie angesteuert werden und wie sie intern ihre Position anfahren.

Anschließend wird erklärt, wie mit 3D-Druckern Kraftsensoren hergestellt und wie diese kalibriert werden, um aussagekräftiges Feedback zur aktuellen Druckverteilung zu liefern.

3.1 Motoren

In den Robotern sind Servomotoren der MX-Reihe des Unternehmens Robotis verbaut. Es gibt drei verschiedene Stärken der Motoren: MX-28 mit 2.5 Nm, MX-64 mit 6 Nm und MX-106 mit 8.4 Nm nominellen Stillstands-Drehmoment (stall torque). Die Erfahrungen im RoboCup zeigen aber, dass die Motoren effektiv ein deutlich niedrigeres Stillstands-Drehmoment haben. Betrieben werden die Motoren bei 12 Volt. Unter Last fließen laut Datenblatt bis zu 5 Ampere durch den Motor.

3.1.1 Ansteuerung

Die Motoren haben eine integrierte Positionskontrolle, sie steuern also selbstständig ihre gesetzte Zielposition an. Der Zielwinkel kann in Schritten von 0,088 Grad eingestellt werden, wobei der Motor aber deutlich ungenauer ist. Während der Bewegungen liegt durch die Motorsteuerung die effektive Genauigkeit bei bis zu 8 Grad [Rhoban, 2017].

Die Kommunikation mit den Motoren findet über ein von Robotis definiertes Dynamixel-Protokoll statt, das auf einem TTL-Bus basiert und damit ein asynchrones Half-Duplex-Verfahren ist. Bei diesem Protokoll werden die

Binärdaten über eine einzelne Ader mit 1-Megabaud übertragen. Jeder Motor muss einzeln nacheinander angesteuert werden, sodass eine effektive Updaterate der Motoren zwischen 100 und 150 Hertz erzielt werden kann.

3.1.2 PID-Regler

Die Motoren von Robotis steuern ihre Zielposition über einen PID-Regler an. PID-Regler basieren darauf, ein Stellglied in Abhängigkeit der Abweichung eines Zielwertes anzupassen. PID steht dabei für *proportional*, *integral* und *derivative*. Der P-Regler steuert das Signal proportional, er steuert also im Verhältnis zur Abweichung vom Sollwert. Der I-Regler summiert den Fehlerwert über die Zeit auf und steuert so über die Zeit einem Fehler entgegen. Der D-Regler leitet den Fehlerwert über die Zeit ab und regelt so in Abhängigkeit von der Veränderung des Fehlerwertes.

Der PID-Regler ist nicht nur mathematisch einfach, sondern auch sehr flexibel einsetzbar [Bennett, 1984]. Er wird daher für einfache Motorsteuerungen häufig genutzt. Allerdings hat sich bei Tests gezeigt, dass Robotis den PID-Regler in mancher Hinsicht ungewöhnlich umgesetzt hat. Zum einen hat Robotis nicht dokumentiert, mit welcher Maßeinheit sie steuern. So dokumentieren sie nur, dass der Standardwert des P-Reglers als 4 ins Motormodell übertragen wird, aber geben keine Informationen über die Dimension dieser 4 [Robotis, 2017]. Ein Hindernis ist z.B., dass die Motoren anscheinend den Fehlerwert ständig integrieren und er nicht zurücksetzbar ist — dadurch kann man ihn nicht beliebig aus- und einschalten, da er dann fehlerhaft vorinitialisiert ist. Andererseits ist es aber nicht ratsam, den I-Regler die ganze Zeit mitlaufen zu lassen, da der Fehlerwert je nach Belastungssituation deutlich unterschiedlich ist (vgl. Abschnitt 6.2). Deswegen ist der I-Regler der Motoren nicht effektiv einsetzbar.

3.2 3D-gedruckte Kraftsensoren

3D-Druck ist ein additives Herstellungsverfahren, das sich seit dem Aufkommen der RepRap Drucker sehr stark verbreitet hat [Moilanen and Vadén, 2013]. Bei diesen Druckern handelt es sich um Fused-Deposition-Modeling Drucker, die das Plastik-Filament schmelzen und dabei eine Kunststoffschicht erzeugen. Das Zielobjekt wird so Schicht für Schicht erstellt. Diese Technologie ermöglicht es, insbesondere Prototypen schnell und einfach zu erstellen und so Hardware in iterativen Schritten zu entwickeln.

Da sich gleiche 3D-gedruckte Objekte vom selben Drucker unter Last sehr ähnlich verformen, lassen sich mithilfe von 3D-Druckern einfache, auf den Anwendungsfall zugeschnittene Kraftsensoren entwickeln, die darauf basieren, die Verformung des Objektes unter Last zu messen. Als flexibler Ansatz wurde das Messen der Verformung über Näherungssensoren gewählt. Hierbei wird unter Krafteinfluss der Abstand zwischen dem 3D-gedruckten Objekt und einem Näherungssensor verringert. Dadurch verändert sich dann der gemessene Wert des Näherungssensors. Diese Abbildung von Kraft zu Sensorwert muss kalibriert werden, um sinnvolle Informationen zu bekommen.

Für die Kalibrierung sind zwei Faktoren relevant: Zum einen die Verformung des Objektes durch die Kraft, zum anderen die unterschiedlichen Sensorwerte durch den Abstand des Objektes vom Sensor. Insbesondere die Veränderung des Sensorwertes ist nicht linear und daher aufwendig zu kalibrieren. Um die Kalibrierung möglichst generell zu halten, können die zwei Faktoren unabhängig voneinander kalibriert werden. Es wird dann zweistufig der Sensorwert auf eine Entfernung vom Sensor abgebildet und anschließend die Kraft aus der Entfernung berechnet. Diese doppelte Abbildung hat den Vorteil, dass die aufwendigere Kalibrierung der Entfernung zum Sensorwert nur einmal durchgeführt werden muss und dann beim Wechseln des 3D-gedruckten Objektes nur die Verformung kalibriert werden muss.

3.2.1 Kalibierung von Näherungssensoren

Optische Näherungssensoren können die Verformung und damit Kräfte bestimmen, indem sie den Abstand zwischen dem Sensor und einem Objekt messen. Im Rahmen der Masterarbeit wurden die Reflexlichtschranken „ITR8307 “ von Everlight genutzt [Everlight Electronics, 2010]. Diese enthalten eine Infrarotdiode und einen NPN-Fototransistor, dessen Schaltplan in Abbildung 3.1 gezeigt ist. Der Fototransistor kann für den Anwendungsfall als variabler Widerstand gesehen werden, dessen effektiver Widerstandswert von dem einfallenden Infrarotlicht und damit der Entfernung des reflektierenden Objektes abhängt. Die Infrarotdiode sendet das Licht aus, dessen Reflexion den Widerstand des Fototransistors bestimmt.

Optische Näherungssensoren sind im Allgemeinen nicht linear bezüglich ihrer Reaktion auf veränderte Entfernungen, daher muss eine Kalibrierung über die gesamte Sensorkurve durchgeführt werden. Hierfür wird eine

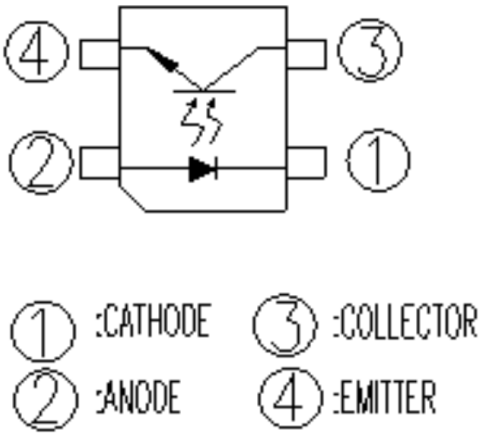

Abbildung 3.1: Der Schaltplan des ITR8307: Unten die Infrarot-LED und oben der Fototransistor [Everlight Electronics, 2010].

Menge von Kalibrierungspunkten aufgenommen, zwischen denen dann interpoliert wird, um die effektive Position zu bestimmen.

Im Verlauf dieser Masterthesis wurde ein Testsetup erstellt, das den Sensor an einem handelsüblichen 3D-Drucker befestigt. Dieses Setup ist in Abbildung 3.2 links zu sehen, rechts in dem Bild ist die Düse des 3D-Druckers, an dessen Halterung der Sensor befestigt wurde. Das Testsetup lässt sich dadurch sehr präzise bewegen. Der Sensor ist in der Mitte des Bildes sichtbar und befindet sich knapp oberhalb der Oberfläche des Balkens, gegen die gemessen wird. Teile des Sensors verdeckend ist ein Lichtschutz zu sehen, der vor einfallendem Infrarotlicht der Sonne oder anderen Quellen schützt.

Da die Reflexionseigenschaften jedes Materials unterschiedlich sind, sollte die Kalibrierung für den späteren Einsatz mit dem Zielobjekt durchgeführt werden. Der schichtweise Aufbau des 3D-gedruckten Objektes führt zu unterschiedlichen Reflexionseigenschaften in Abhängigkeit davon, ob die Achse der Oberfläche den Schichten entspricht. Dieser Unterschied macht bis zu 15% der Reflexion aus, daher sollte die Kalibrierung auch die Orientierung des Objektes beachten.

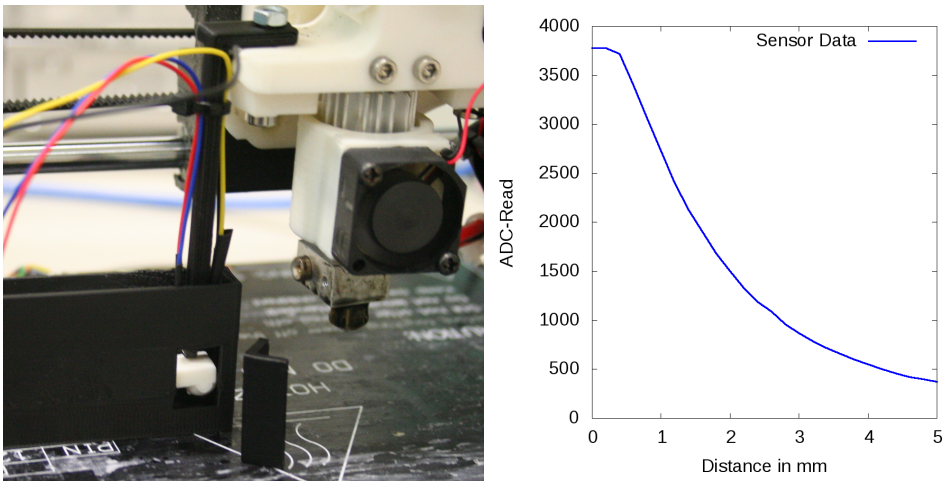

Abbildung 3.2: Kalibrierungssetup für den Entfernungsmesser (links). Der Näherungssensor ist über einen Stab am Drucker befestigt und nahe an den weißen Balken gefahren, der in weiten Teilen von einem Infrarotschutz umgeben ist. Beim Hochfahren des Sensors werden die Sensorwerte aufgenommen. Die dabei entstehende Abbildungskurve von Sensorwert zu Distanz ist rechts visualisiert.

Der Sensor wird beim Setup auf Sicht an die Messoberfläche herangefahren und der Wert des Sensors ausgelesen. Da das Druckbett des 3D-Druckers bei Druck nachgibt, wird der Sensor nicht direkt beschädigt, falls es zum Kontakt mit der Oberfläche des Balkens kommt. Der Sensor wird nun präzise in 0,1 mm Schritten von der Oberfläche weggefahren und dabei werden weitere Sensorwerte aufgenommen. Mithilfe der so erstellten Tabelle (Abbildung 3.2 rechts) lassen sich gegebene Sensorwerte durch Interpolation zwischen den gemessenen Werten sehr genau auf die tatsächliche Entfernung abbilden.

3.2.2 Kalibrierung von Verformungen

Wenn die Verformung des Objektes bekannt ist, muss noch die Verformung auf die Kraft abgebildet werden. Während die generelle Verformung von Objekten nur über die Finite-Elemente-Methode berechenbar und damit sehr komplex ist, lässt sich das Problem vereinfachen, indem das verformende Objekt speziell gestaltet wird. Der einfachste Fall ist ein sich verbiegender Balken, bei dem das Hooke'sche Gesetz gilt [Wasserfall et al., 2017].

Die anliegende Kraft F ist durch die Federkonstante k und die Verformung δl berechenbar:

$$F = k \cdot \delta l \tag{3.1}$$

Um die Verformung nun zu kalibrieren, reicht es den Sensor gegen eine einfache Waage zu drücken und die Kraft und Verformung zu messen. Die Federkonstante ist dann:

$$k = \frac{F}{\delta l} \tag{3.2}$$

Zusammengesetzt lässt sich das Prinzip in Abbildung 3.3 sehen.

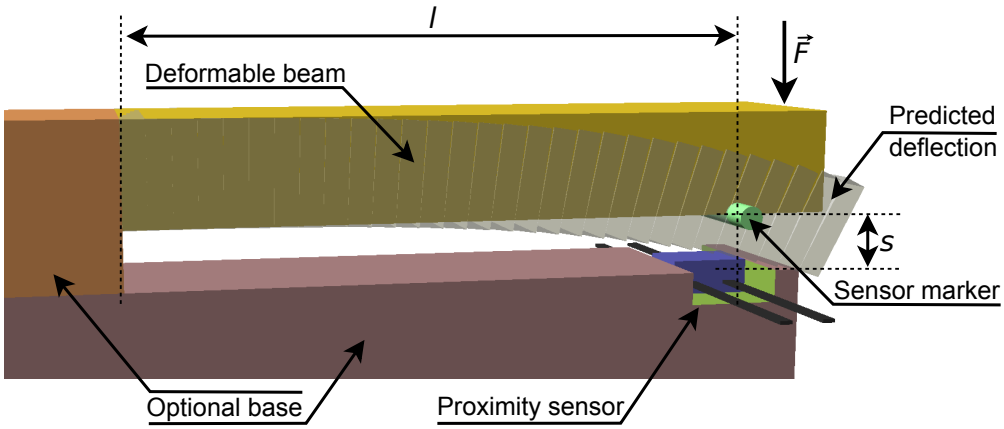

Abbildung 3.3: OpenSCAD-Rendering eines einfachen Kraftsensors. Oben in gelb ist der verformbare Balken. In blau und grün sind der Entfernungssensor beziehungsweise der Messpunkt angegeben, der benutzt wird, um die entsprechenden Kräfte zu messen [Wasserfall et al., 2017].

4 Hardwaredesign

Im Rahmen dieses Kapitels wird die Entwicklung der Hardware beschrieben. Zuerst wird der Prototyp erklärt, bei dem die Sensoren nur an einem bestehenden Fuß befestigt wurden. Danach wird der neu entwickelte 3D-gedruckte Fuß vorgestellt. Im anschließenden Abschnitt wird über den Entwurf des kleinen Prototypboards, das die für die Sensoren benötigte Elektronik platzsparend integriert, berichtet und abschließend die Entwicklung des professionellen Boards erläutert, das die Kraftsensoren in Füßen zukunftsfähig machen soll.

4.1 3D-Druck-Fuß

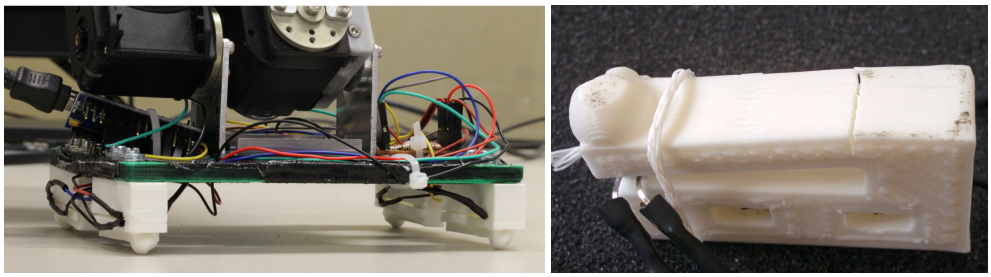

Abbildung 4.1: Links ein Foto des ersten Prototypen. Rechts ein gebrochener Kraftsensor, dessen Bruchstelle am rechten Rand sichtbar ist.

Beim ersten Prototypen wurden die Kraftsensoren an einem bestehenden Fuß befestigt, dieser ist in Abbildung 4.1 zu sehen. Hier waren die Sensorkabel zur Seite herausgeführt. An diesen Kabeln konnte sich der Roboter jedoch schnell verhaken, sodass sie leicht abgerissen wurden. Zusätzlich war der Näherungssensor fest mit dem verformbaren Balken verbunden, sodass der Balken im Falle von Beschädigungen (siehe Abbildung 4.1 rechts) nicht einfach austauschbar war. Die feste Verbindung führte auch dazu, dass keine Modifizierung der Auflösung durch unterschiedliche Dicke des Balkens möglich war.

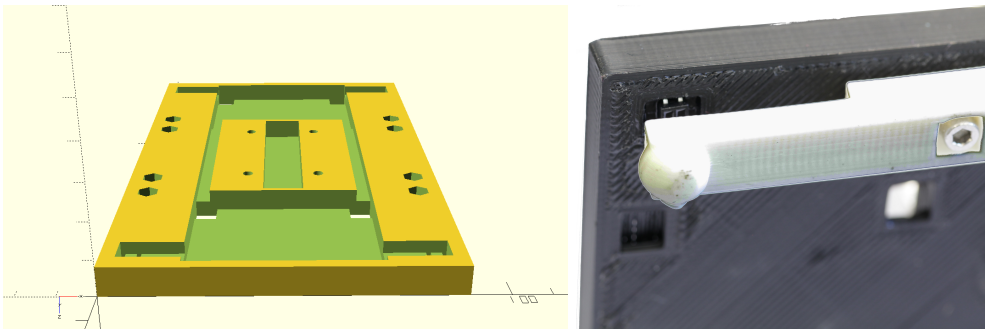

Abbildung 4.2: Links: Das Fußmodell in OpenSCAD. Rechts: Eine Nahaufnahme des Kraftsensors. Die umgebende Lichtschutzhülle wurde für das Foto entfernt. In weiß ist der sich verbiegende Balken zu sehen. Die große schwarze Fläche ist ein Teil des Fußes, in die unterhalb des Balkens der Näherungssensor geklebt wurde. Dieser Sensor ist zur Hälfte sichtbar, inklusive Fototransistor neben der Infrarot-LED mit ihren jeweiligen Anschlüssen.

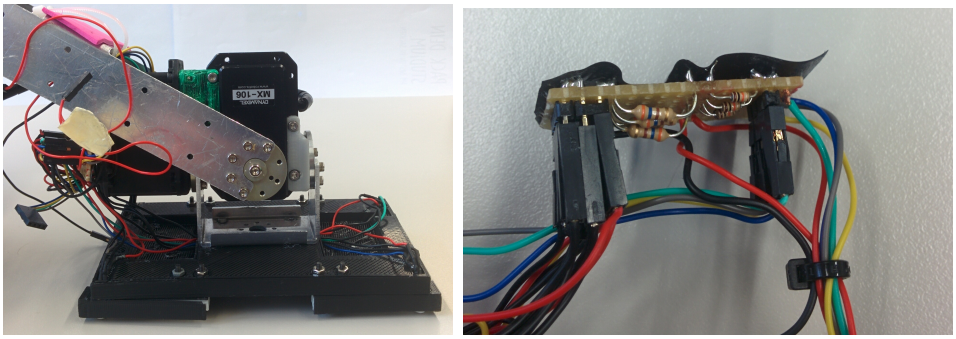

Abbildung 4.3: Links: Die zweite Version des Fußes mit den integrierten Sensoren. Rechts das dazugehörige Board mit den Widerständen. Insbesondere die Kabelführung ist bei diesem Prototypen problematisch.

Um eine gute Integration der Sensoren in den Fuß zu ermöglichen, wurde daher der Fuß in OpenSCAD neu entworfen. Das Design ist Abbildung 4.2 von oben zu sehen. In den Ecken sind die Aussparungen für die Anschlüsse der Sensoren zu sehen. Am Rand der Längsseiten sind die Schraubenlöcher für die sich verformenden Balken zu erkennen. Ferner gibt es Aussparungen für Platinen und Kabel sowie die Befestigung für die Ferse in der Mitte des Fußes. Der ausgedruckte Fuß ist in Abbildung 4.2 rechts zu sehen. Die Näherungssensoren werden von unten in den Fuß geklebt und die Kabel nach oben weggeführt.

4.2 Prototyp-Platine

Die Kabelführung war, wie in Abbildung 4.3 erkennbar, immer noch nicht ausreichend. Zum einen war es nicht ersichtlich, welches stromführende Kabel zu welchem Sensorkabel gehörte. Zum anderen hatten die Standard-Prototypstecker nicht genügend Halt und verloren daher die Verbindung. Schließlich gab es keinen Ort, um die Platine anzubringen. Daher wurde eine kleine integrierte Platine entwickelt.

Hierfür wurde als Microcomputer-Board ein Teensy3.2 benutzt, der auf einem ARM Cortex-M4 basiert und in weiten Teilen kompatibel zur Arduino-Plattform ist [PJRC, 2010]. Dieser wurde an eine Lochrasterplatine gelötet und die dazugehörigen Widerstände möglichst platzsparend verlötet.

Der Schaltplan ist in Bild 4.4 zu sehen. Das Ausmessen des Sensorwertes basiert auf dem Prinzip des Spannungsteilers [Bernstein, 2012]. Von der $3{,}3\,V$ Spannungsversorgung ist ein $10.000\,\Omega$ Widerstand vor dem Fototransistor geschaltet. Zwischen dem Widerstand und dem Fototransistor wird die Spannung gemessen. Ist dabei der Widerstand des Transistors niedrig, fällt die ganze Spannung am Widerstand ab: Die gemessene Spannung ist ebenfalls niedrig. Ist der Widerstand des Transistors hoch, ist hingegen die gemessene Spannung hoch, da am Transistor die meiste Spannung abfällt.

Die LEDs sind mit einem $360\,\Omega$ Vorwiderstand ausgestattet, um das nötige Infrarotlicht für den Fototransistor zu emittieren. In dem Schaltplan befindet sich das eigentliche Board auf der linken Seite. Auf der rechten Seite sind die im Fuß verklebten Sensoren zu sehen. Zwischen den beiden Teilen befinden sich 4er Stecker.

Der Teensy liest über seine integrierten ADC-Pins die Spannung an den Sensoren und interpoliert eine Entfernung mithilfe der aufgenommenen Daten (vgl. Abbildung 3.2). Bei der Kalibrierung wurde eine Federkonstante von $D = 17{,}95\,\mathrm{N/mm}$ gemessen. Der Teensy berechnet dann mit der Anfangsentfernung des Balkens d_0 und der aktuellen Entfernung d_t die gemessene Kraft F (siehe Gleichung 4.1) und überträgt diese dann über den integrierten USB-Chip als serielles Signal.

$$F = 17{,}95 \cdot (d_0 - d_t) \tag{4.1}$$

4.3 PCB-Platine

Im Rahmen der Hardwareentwicklung wurde in Kooperation mit anderen Mitgliedern der Hamburg Bit-Bots eine komplette Neuentwicklung des Mikrocontroller-Boards (siehe Bild 4.6 links) durchgeführt. Dieses Board basiert auf einem STM32F103 und beinhaltet unter anderem einen TTL-Treiber und einen 24-Bit Analog-Digital-Converter. Der TTL-Treiber unterstützt eine Kommunikation über den TTL-Bus der Dynamixel Motoren, deswegen benötigt der Chip kein eigenes USB-Kabel zum Rechner. Die 24-Bit des ADC-Chip ermöglichen eine deutlich höhere Auflösung der gemessenen Spannung. Außerdem kann der Chip auch differenzielle Signale

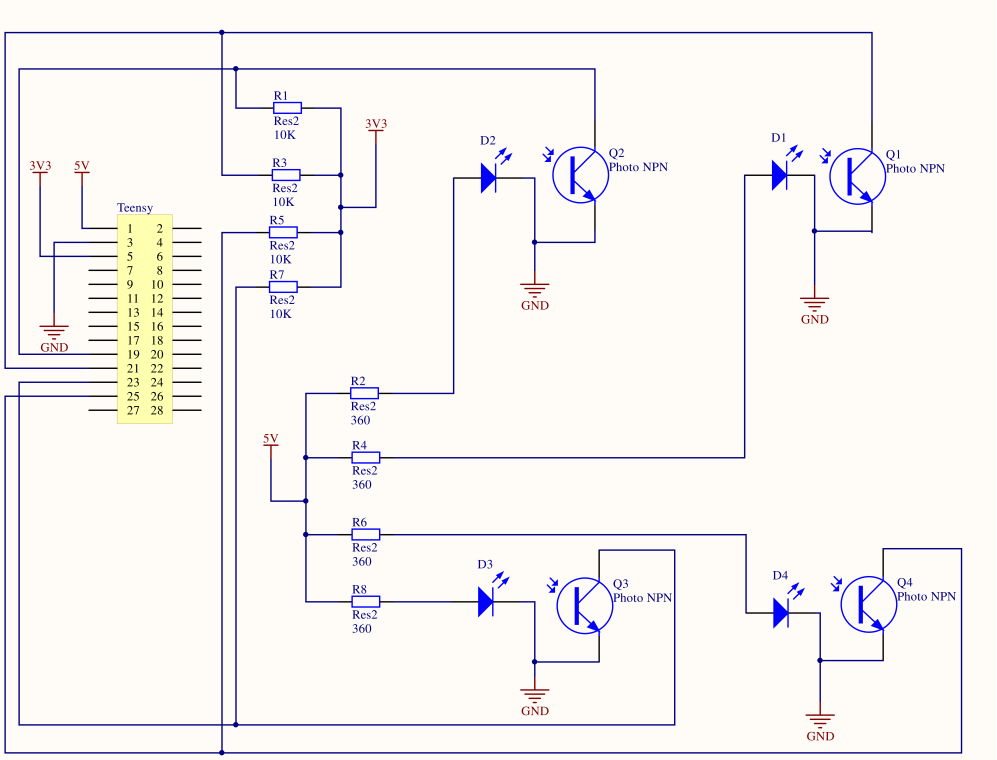

Abbildung 4.4: Schaltplan der Prototyp-Platine. Links befindet sich das Pinout des Teensy, mit den 3,3 und 5 V Pegeln. In der Mitte sind die Widerstände für die LEDs und für die Spannungsteiler. Rechts sind die vier Drucksensoren, die jeweils aus einer Infrarot-LED und einem Phototransistor bestehen.

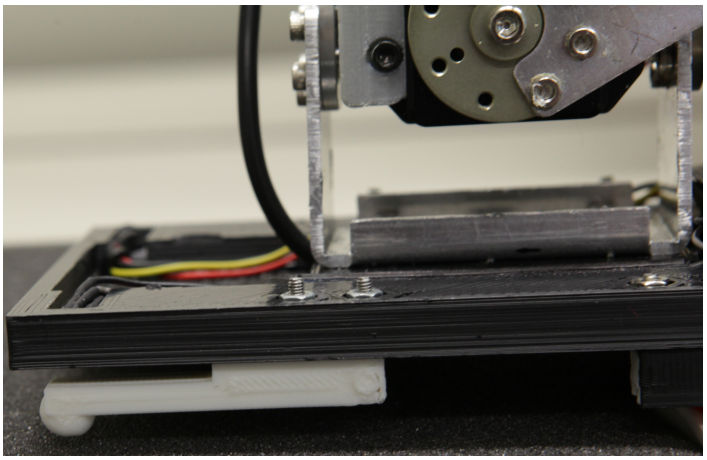

Abbildung 4.5: Ausschnitt vom entwickelten Fuß. Links in weiß ist der verformbare Balken des Kraftsensors gut zu sehen. Im Fuß oberhalb des Spalts zum Balken ist der Näherungssensor verklebt, dessen Kabel auf der Oberseite des Fußes sichtbar sind.

von Dehnungsmessstreifen verarbeiten. Das Board ist damit nicht nur in der Lage, die auf Näherungssensoren basierenden Kraftsensoren, sondern auch industriell gefertigte Wägezellen auszuwerten.

Industrielle Wägezellen basieren genauso wie die gedruckten Kraftsensoren darauf, dass die Verformung eines Messbalkens gemessen wird. Diese Wägezellen sind wie in Abbildung 4.6 rechts erkennbar meist aus Metall gefertigt. Der messbare Hub ist bei der gleichen Kraft deshalb deutlich kleiner als bei 3D-gedruckten Sensoren. Genauso wie auch die gedruckten Sensoren müssen die verwendeten Dehnungsmessstreifen ebenfalls kalibriert werden, wobei auch bei diesen Sensoren die Veränderung des Widerstands linear zur Verformung ist. Deswegen lässt sich die Verformung e mit folgender Formel berechnen [Fraden, 2010]:

$$e = \frac{dR}{R \cdot S_e} \tag{4.2}$$

wobei S_e ein materialspezifischer Faktor ist, der sich zusammen mit der Federkonstante einfach gegen eine Waage kalibrieren lässt.

Programmiert ist der STM32F103 in C++ mithilfe der STM32Plus Library von Andy Brown [Brown, 2017]. Die Software des Chips (vgl. Code auf der beiliegenden CD) kommuniziert in Dauerschleife mit dem LTC-4422-ADC und liest auf dem Motorbus durch Hardware-Interrupts die Pakete mit,

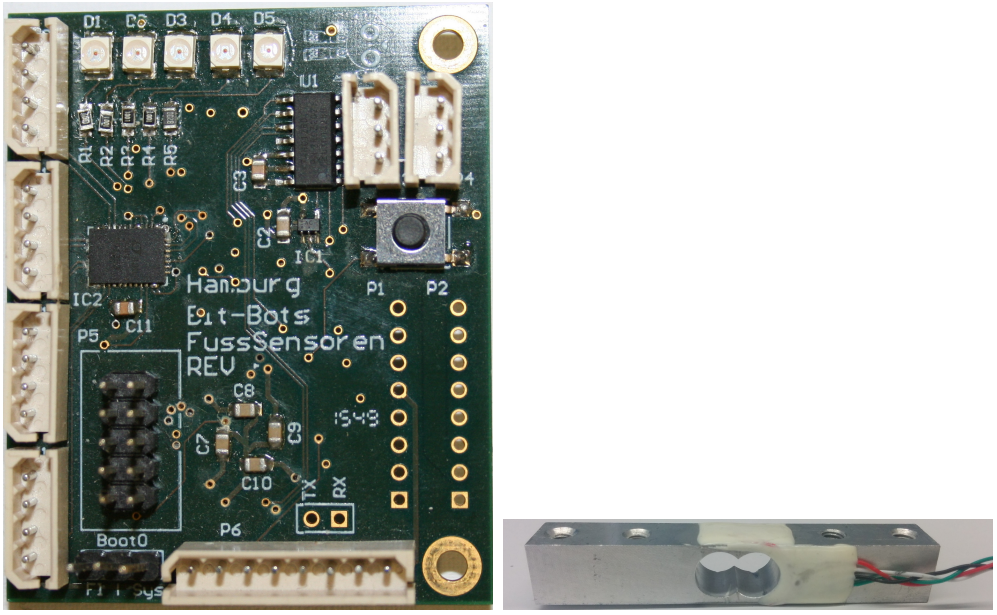

Abbildung 4.6: Die neu entwickelte Platine (links) und eine einsetzbare Wägezelle (rechts). Die Platine enthält sowohl alle benötigten Komponenten, um die Kommunikation über den Motorbus zu ermöglichen, als auch einen genauen Analog-Digital-Konverter, um die Sensoren präzise auszuwerten.

um dem Master antworten zu können. Allerdings gibt es bisher keinen vollständig einsatzfähige Prototypen, sodass die Platine nicht evaluiert werden konnte.

5 Software

Im Rahmen des Kapitels zur entwickelten Software wird zunächst die Technik des Reinforcement Learnings eingeführt, die genutzt wird, um das Gehen auf dem physischen Roboter während der Bewegungen zu optimieren und damit zu stabilisieren. Danach wird der Einsatzbereich des Reinforcement Learning im Kontext der bestehenden Klassen und Abläufe eingeordnet.

Im Anschluss wird genauer spezifiziert, was der Algorithmus lernen soll und wie die dazugehörige Policy strukturiert bzw. visualisiert werden kann. Abschließend wird dargestellt, wie die Policy sinnvoll vorinitialisiert werden kann, um das anfängliche Lernen zu beschleunigen.

5.1 Reinforcement Learning

> „Reinforcement Learning is learning what to do - how to map situations to actions - so as to maximize a numerical reward signal. [...] The learner [...] must discover which actions yield the most reward by trying them.“ [Sutton and Barto, 1998]

Durch Reinforcement Learning kann Software Probleme lösen, ohne dass die exakte Lösung im Vorwege programmiert wird. Der Algorithmus probiert verschiedene Strategien aus und beobachtet anschließend das numerische Feedback bzw. den Gewinn R, um langfristig das bestmögliche Feedback zu erhalten.

Die aktuelle Strategie, die von einem Reinforcment Learning Algorithmus ausgeführt wird, ist die Policy π, die bestimmt, dass in dem Zustand s die Aktion a ausgeführt wird. Ein Zustand kann die Phase des Laufalgorithmus sein und die Aktion die Position des Oberkörpers. Das anschließende Rewardsignal kann nun die Abweichung des ZMP vom Zielwert sein. Dadurch

ist es möglich, dass die Bewegung des Oberkörpers nicht explizit modelliert ist, sondern sich in Abhängigkeit der Performance des Roboters verändert.

Die klassischen Ansätze von Sutton et. al. speichern eine Value-Funktion, die eine Qualitätsabschätzung des jeweiligen Zustandes darstellt. Im Allgemeinen berechnet die Value-Funktion also eine Abschätzung des zukünftigen Gewinns. Diese Ansätze werfen aber im Kontext von humanoiden Robotern, insbesondere wenn die Aktionen a kontinuierlich sind, viele Probleme auf [Deisenroth et al., 2013], die bei Algorithmen ohne Value Funktion nicht in dem Maße auftreten [Peters et al., 2003].

Aufgabe des Reinforcement Learnings ist es nun, die optimale Policy herauszufinden. Die Policy ist optimal, wenn sie den Gewinn maximiert. Das Lernverfahren darf dabei den Roboter nicht beschädigen und sollte nicht zu viel Zeit benötigen. Aus diesem Grund ist eine randomisierte Suche nicht tolerierbar, auch wenn sie theoretisch ein optimales Ergebnis erzielen würde. Es müssen aufwendigere Ansätze wie die Policy Gradient Algorithmen angewendet werden, um schnell und vergleichsweise sicher ein lokales Optimum zu finden. Eine Schwäche dieser Algorithmen ist, dass dieses Optimum weit vom globalen entfernt sein kann.

Die Aktion a ist dabei ein Tupel von zwei Gleitkommazahlen, welche die Position in Metern vom Startpunkt des Oberkörpers darstellt. Gesteuert wird die x- (vorwärts) und y- (seitwärts) Position des Oberkörpers.

Im Folgenden werden die Techniken des Reinforcement Learning eingeführt, die für die Evaluation (vgl. Kapitel 6) der Software verwendet wurden.

5.1.1 Policy Gradient Algorithmen

> „Policy search is a subfield in reinforcement learning which focuses on finding good parameters for a given policy parametrization. It is well suited for robotics as it can cope with high-dimensional state and action spaces, one of the main challenges in robot learning.“ [Deisenroth et al., 2013]

Während klassische Reinforcement Learning Algorithmen in ihrer aktiven Lernphase eine randomisierte Suche durchführen, die den gesamten Suchraum evaluiert, suchen Policy Gradient Algorithmen nur in der Nähe der aktuellen Policy. Sie schätzen dabei den Gradienten des zu erwartenden Gewinnes ab, um damit entlang des Gradienten die Policy zu verbessern.

Ausschließlich entlang des Gradienten zu suchen, erhöht zwar deutlich das Risiko, in einem lokalen aber nicht globalen Minimum zu landen, allerdings reduziert die Abwesenheit von randomisierter Suche die Gefahr, den Roboter zu beschädigen. Wenn der Roboter bereits einen vergleichsweise stabilen Gang aufweist, dann führt die Policy Gradient Suche dazu, dass der Roboter sich nicht aus dem stabilen Gang entfernt. Hat der Roboter noch keinen stabilen Gang gelernt, ist im Allgemeinen ein menschlicher Beobachter anwesend, der notfalls eingreifen kann. Der Roboter macht durch eine Gradientensuche im Normalfall keine unerwarteten Bewegungen, was das Eingreifen des Beobachters erleichtert. Um die Wahrscheinlichkeit von gefährlichen Bewegungen noch weiter zu reduzieren, kann die Steigung des Gradienten begrenzt werden: Sobald der Gradient diese Grenze erreicht hat, steigt er nicht mehr. So führen einzelne massive Abweichungen im Rewardsignal zu keinen großen Problemen.

5.1.2 Rewardsignal

Da der Roboter den Fuß nicht abrollen kann und die optimale Position für den ZMP sich deswegen im Rahmen dieser Thesis in der Mitte des Fußes befindet, wird in Anlehnung an Gleichung 2.7 das Rewardsignal R_x für die x-Policy in Gleichung 5.1 und R_y für die y-Policy in Gleichung 5.2 definiert. Der gemessene CoP ist 0, wenn er exakt in der Mitte des Fußes ist. Das Rewardsignal ist damit eine Minimierungsaufgabe. Der Reward ist optimal, wenn er null ist. Daraus ergibt sich:

$$\min(R_x(|CoP_x|)) \tag{5.1}$$

$$\min(R_y(|CoP_y|)) \tag{5.2}$$

Dieser Reward kann nach jeder Kommunikation mit den Motoren und Sensoren gegeben werden. Da es auch keine Extrabelohnung am Ende eines Schrittes gibt, kann der Algorithmus online und durchgängig beim Gehen lernen.

5.1.3 Finite Differenzen Policy Gradient

Die einfachste Variante eines Policy Gradient Algorithmus basiert auf Finiten Differenzen. Dieser Algorithmus nutzt eine Episoden-basierte Evaluationsstrategie. Der Algorithmus schätzt dabei den Gradienten anhand der Erfahrungen ab, die er durch mehrere Durchgänge gewonnen hat. Dafür fügt er kleine Störungen $\delta\Theta$ zur ausgeführten Bewegung hinzu. Dann können die Unterschiede im Feedback $\delta R = R(\Theta + \delta\Theta) - R(\Theta)$ beobachtet werden. Anschließend kann der Gradient wie folgt abgeschätzt werden [Deisenroth et al., 2013]:

$$\nabla E(R(\pi))_\Theta = (\delta\Theta^T \ \delta\Theta)^{-1}\delta\Theta^T \ \delta R \tag{5.3}$$

5.1.4 ZMP-Position als Gradientenabschätzung

Während die Finite Differenzen Policy Gradient Methode eine Blackbox-Methode ist, um den Gradienten zu bestimmen, lässt sich aus der Position des ZMP direkt eine Abschätzung des Policy Gradienten ableiten. Wenn der ZMP zu weit vorn ist, muss der Roboter sich weiter nach hinten lehnen, um auch den ZMP nach hinten zu verschieben. Beim ZMP als Gradientenabschätzung wird das Rewardsignal um die Richtung der Abweichung erweitert:

$$R_x(\text{Richtung,Reward}) = \begin{cases} R_x(1,|CoP_x|) & \text{falls } CoP_x < 0 \\ R_x(-1,|CoP_x|) & \text{falls } CoP_x \geq 0 \end{cases} \tag{5.4}$$

$$R_y(\text{Richtung,Reward}) = \begin{cases} R_x(1,|CoP_y|) & \text{falls } CoP_y < 0 \\ R_x(-1,|CoP_y|) & \text{falls } CoP_y \geq 0 \end{cases} \tag{5.5}$$

Beim Lernen wird die Zielposition der Policy in Abhängigkeit der Richtung und des Rewards verändert.

5.1.5 Temporal Differences „TD(λ)“

Beim Temporal Differences (TD) Learning wird nicht nur das aktuelle Reward-Signal betrachtet, sondern auch, wie gut der erreichte Zustand ist. Hierfür gibt es bei dem TD(λ)-Ansatz sogenannte „eligibility traces“, bei denen für die nächsten n Zustände der Reward jeweils um dem Faktor $\lambda \in (0,1)$ reduziert mit einfließt [Sutton and Barto, 1998]. Der Reward ist dann:

$$R_t(\pi) = r_t + \lambda \cdot r_{t+1} + \lambda^2 \cdot r_{t+2} + \ldots + \lambda^n \cdot r_{t+n} \tag{5.6}$$

5.1.6 Funktionsapproximation mit Tile-Coding

Eine Herausforderung im Reinforcement Learning für Roboter ist die Kontinuität des Zustandsraums. Ein Beispiel hierfür ist die Zeit seit dem Anfang des Schrittes. Aus dieser Kontinuität folgt, dass es nicht möglich ist, alle erreichbaren Zustände aufzulisten, da es unendlich viele von ihnen gibt. Die möglichen Zustände müssen daher über Funktionen approximiert werden. Eine der einfachsten und wichtigsten Ansätze ist das Tile-Coding [Sutton and Barto, 1998].

Beim Tile-Coding wird der Zustandsraum in Partitionen aufgeteilt, in denen die Funktion als konstant angenommen wird. Der Zustandsraum wird also künstlich diskretisiert.

5.1.7 Gauß-Filter

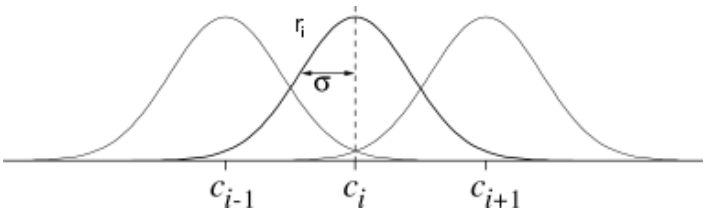

Abbildung 5.1: Einfluss des Feedbacksignal r_i auf die benachbarten Tiles (modifizierte Grafik aus [Sutton and Barto, 1998] über Radial Basis Functions).

Das Rewardsignal kann mit einem Gauß-Filter über adjazente Tiles gefiltert werden. Dieser glättet die Auswirkungen des Lernsignals auf mehrere Tiles. Dadurch wird eine gleichmäßigere Bewegung in der erlernten Policy erreicht.

Die Veränderung an den Tiles ist durch den Gauß-Filter eine ableitbare Funktion. Der Einfluss des Rewardsignal r_i hat in den benachbarten Tiles c_i in Abhängigkeit von der Breite σ in folgender Form Einfluss:

$$c_i(r_i) = \exp\left(\frac{||s - c_i||^2}{2\sigma^2}\right) \tag{5.7}$$

5.2 Softwarearchitektur

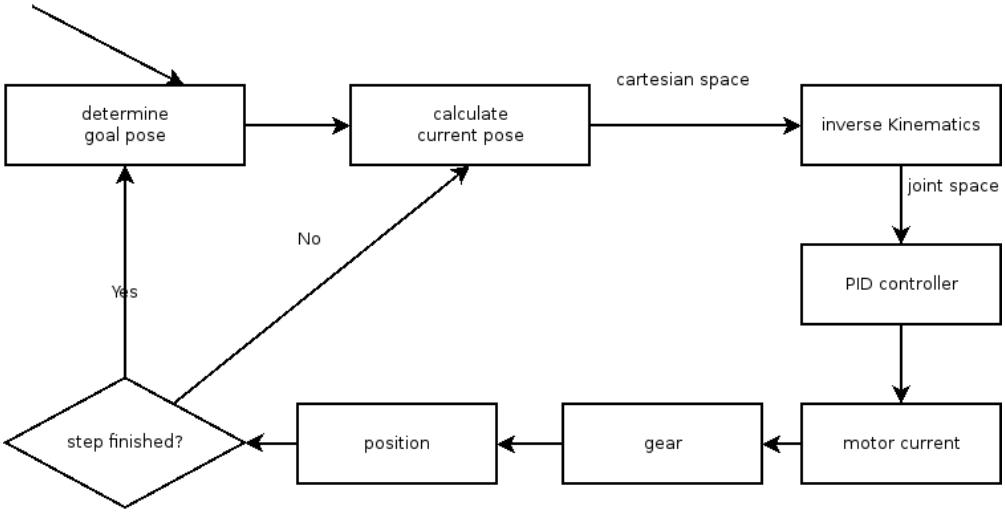

Abbildung 5.2: Abstrahiertes Diagramm des Laufalgorithmus in dem die Abfolge von beteiligten Steuerelementen dargestellt ist, die aus einer Richtung bzw. Geschwindigkeit des Laufalgorithmus die physische Bewegung auf dem Roboter generiert.

Die generelle Struktur des Laufalgorithmus ist in Abbildung 5.2 dargestellt. Zunächst wird die Zielposition für das Ende des Schrittes bestimmt, anschließend die aktuelle Position des Roboters im kartesischen Koordinatensystem. Diese wird mithilfe der inversen Kinematik in die entsprechenden Gelenkwinkel übersetzt. Diese Zielwinkel werden dann an die Motoren übertragen, auf denen ein PID-Regler das Drehmoment steuert. Wenn die Kommunikation mit den Motoren abgeschlossen ist, wird der aktuelle Schritt weitergeführt oder wenn nötig ein neues Schrittziel berechnet.

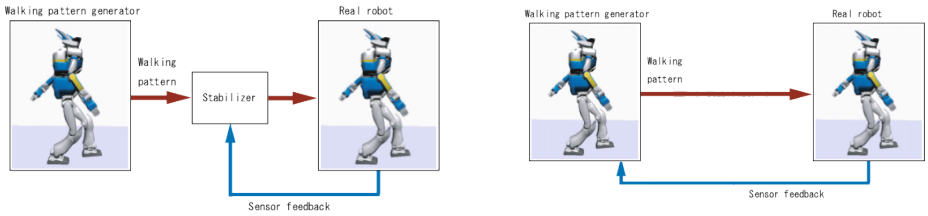

Abbildung 5.3: Laufgenerator mit und ohne Stabilisator. Das Sensorfeedback soll nicht wie in den Standardansätzen in einen Stabilisator integriert werden (links), der kaum Möglichkeit hat, aus dem Feedback für die Zukunft zu lernen. Stattdessen wird das Sensorfeedback direkt in den Generator der Zielpositionen integriert (rechts). Dadurch kann die generierte Trajektorie des Roboters dauerhaft verbessert werden. Die linke Grafik wurde aus [Siciliano and Khatib, 2008] übernommen.

Im Rahmen der Software werden die Informationen über den ZMP im zweiten Teilschritt genutzt: Es wird gelernt, wie die aktuelle Zielposition sein muss, um ein möglichst stabiles Laufen zu ermöglichen. Im Gegensatz zu klassischen Ansätzen mit Sensoren kommt kein Stabilisator (vgl. Abbildung 5.3) zum Einsatz. Ein solcher Stabilisator erhält die generierte Abfolge des Laufalgorithmus und modifiziert diese in Abhängigkeit vom aktuellen Feedback. Dadurch, dass diese Stabilisatoren kein Wissen über den Zustand des Laufalgorithmus haben, sind sie nicht in der Lage, durch aktuelles Feedback die Bewegungsabfolge langfristig zu verbessern. Das Sensorfeedback wird wie in Abbildung 5.3 rechts zu sehen, direkt in den Generator der Bewegungsabfolge integriert, damit das Feedback die eigentliche Abfolge verbessert und so eine langfristige Verbesserung zeigt. Dabei bleibt der Generator der Abfolge aber von den Sensoren weitestgehend unabhängig: Die Bewegungen des Oberkörpers sind ausschließlich von der Position und Geschwindigkeit der Füße abhängig. Dies ermöglicht es, auf diesem Algorithmus aufbauende Stabilisatoren zu entwickeln, solange diese selten in die ausgeführte Bewegung eingreifen.

Ein Überblick über die relevanten Klassen zum Lernen mit dem ZMP ist in Abbildung 5.4 zu sehen. Die zentrale Klasse Walking stellt zum einen die Schnittstelle zum Rest des Systems dar. Zum anderen werden in der Klasse die mathematisch einfach modellierbaren Teile des Laufens berechnet, unter anderem die Phase des Schrittes, die Schrittabfolge, aber auch die Positionen der Beine. Insbesondere strukturell ist diese Klasse noch an den ursprünglichen Code vom Team DARwIn angelehnt [Team-Darwin, 2017].

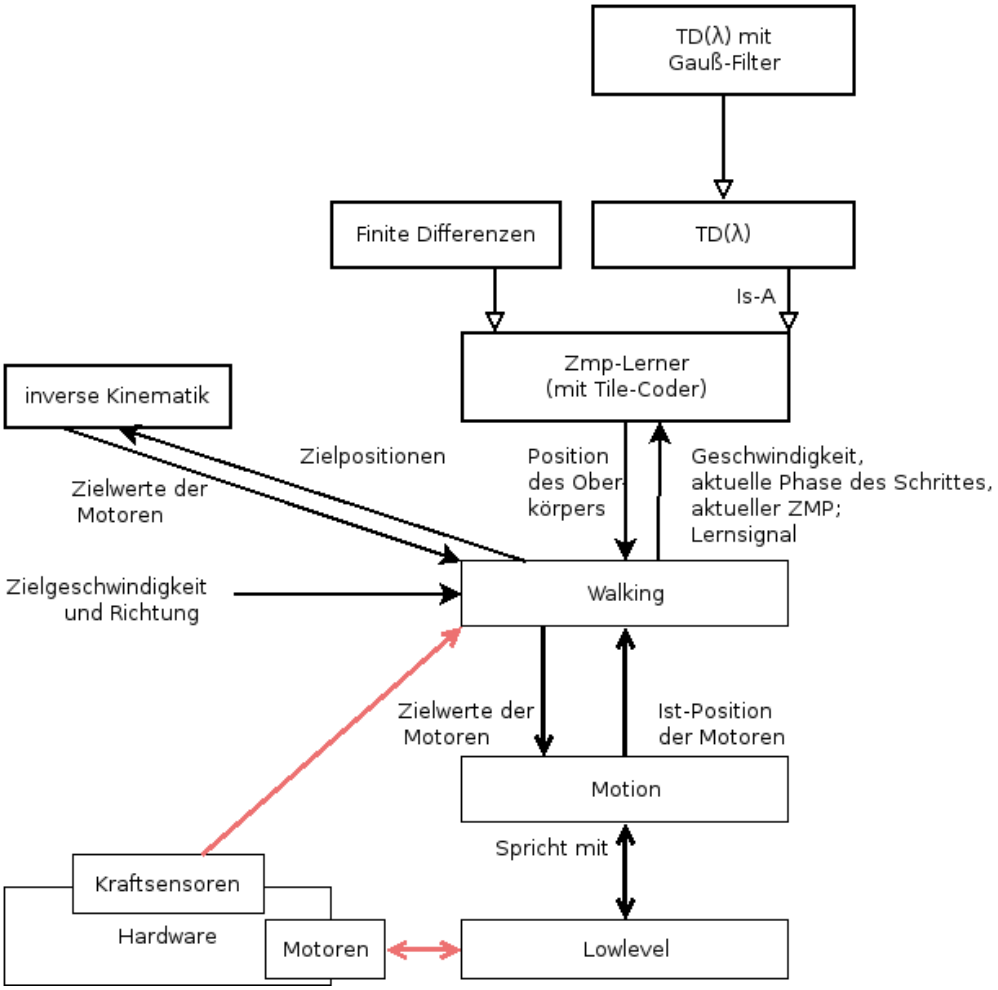

Abbildung 5.4: Abstrahiertes Klassendiagramm. In der Mitte ist die Klasse Walking dargestellt, welche die Schnittstelle zum restlichen System definiert. Die Berechnung der Oberkörperposition ist in den ZMP-Lerner und dessen Subklassen ausgelagert. Die Kommunikation mit den Motoren geschieht über die Klassen Motion und Lowlevel.

Die Klassen Motion und Lowlevel stellen die wichtigsten Funktionen für den Laufalgorithmus bereit. Das Lowlevel organisiert die Kommunikation mit dem Motorbus, bildet also aus den Zielwinkeln die entsprechen Motorbefehle. Die Klasse Motion kümmert sich um Aufgaben, wie das Aufstehen nach einem Sturz des Roboters. Die Klasse steuert auch, ob der Roboter nach den aktuellen Regeln des Spieles gerade laufen darf. Die inverse Kinematik berechnet aus den Zielpositionen die Zielwinkel der Motoren, welche

das Walking direkt an die Motion weiterleitet. Hierdurch erfolgt sämtliche Kommunikation mit dem restlichen Softwaresystem ausschließlich durch die Walking Klasse.

Die klare und übersichtliche Schnittstelle führte dazu, dass der Laufalgorithmus nach ROS portiert und erfolgreich ausgeführt werden konnte [Bestmann, 2016].

Das Ziel des ZMP-Lerner ist das Lernen der Oberkörperbewegung, sodass der ZMP möglichst nahe am Optimum ist (vgl. Gleichung 2.7). Da der Roboter mit seinem Fuß nicht abrollen kann, wird im Rahmen dieser Thesis der allgemeine Fall angenommen, dass der optimale ZMP sich in der einzeln belasteten Phase immer in der Mitte des Fußes befindet.

5.3 Policy Struktur

Um die Zielposition des Oberkörpers zu bestimmen, bekommt der ZMP-Lerner die aktuelle Position der Füße übergeben. Da die Trajektorie der Füße mathematisch fest modelliert ist (vgl. Abschnitt 2.3 „Gang humanoider Roboter“) und daher konstant über die Durchgänge bleibt, wird diese Information aufbereitet und in Form eines Prozentwertes des Schrittes übergeben. Die Bewegungsrichtung des Roboters wird mit übergeben, da diese die optimale Trajektorie des Oberkörpers beeinflusst. Zusätzlich muss auch die Geschwindigkeit beachtet werden: Wenn der Roboter sich schneller bewegt, werden auch die entstehenden Bewegungskräfte größer, der Oberkörper muss also anders bewegt werden.

Wünschenswert wäre es, wenn der aktuelle ZMP vom Lerner beachtet werden würde. Allerdings ist dies mit dem verwendeten Ansatz des Tile-Codings schwer vereinbar, da viele Kombinationen von ZMP und Position selten erreicht werden. Tile-Coding hingegen benötigt genau diese Eigenschaft, um jede ihrer Tiles zu lernen. Man müsste daher eine andere Art der Funktionsapproximation wählen, um den aktuellen ZMP im Lerner zu beachten.

Aktuell interagiert die Klasse Walking noch direkt mit den Kraftsensoren. Mit der überarbeiteten Platine (vgl. Abschnitt 4.3) hingegen wird das Interface der Klasse etwas übersichtlicher, da die Kraftsensoren über die Klassen des „Lowlevel“ ausgelesen werden sollen, da dann die Kraftsensoren über den Motorbus angeschlossen sind.

Die zu lernende Abbildungsfunktion ist also:

$$f_{com}(Speed_x, Speed_y, Speed_\alpha, belastetesBein, Phase) \rightarrow (x,y) \quad (5.8)$$

Die Variable x steht hier für die Vorwärts-Position und die Variable y für die Seitwärts-Position des Oberkörpers relativ zum Startpunkt. Es wird ausschließlich die Position gelernt. Der Roboter soll beim Gehen nicht geneigt werden, da hierdurch das Blickfeld der Kamera beeinträchtigt wird.

Der kontinuierliche Input wird mithilfe von Tile-Coding diskretisiert. Die Zielgeschwindigkeit des Laufalgorithmus wurde während eines Spieles auch vorher schon diskret in fünf Geschwindigkeitsstufen gesetzt. Es entstehen im Rahmen des bestehenden Systems also keine problematischen Einschränkungen durch die Diskretisierung der Zielgeschwindigkeiten. Da der Roboter sich nicht exakt symmetrisch verhält, ist das belastete Bein relevant. Dadurch kann die Phase im Intervall [0,1) modelliert werden.

5.4 Policy Darstellung

Beim Lernen kann die Gleichung 5.8 aufgeteilt werden. Während mehrerer aufeinander folgender Schritte ändert sich die Geschwindigkeit im allgemeinen nicht, d.h. das Lernsignal bezieht sich auf feste Werte von $Speed_x, Speed_y, Speed_\alpha$. Wenn die Zielgeschwindigkeit geändert wird, lernt der Algorithmus diese neue Geschwindigkeit unabhängig von der alten. Deshalb sind zum Verständnis des Lernverfahrens die Parameter $belastetesBein$, $Phase$ wichtig. Dadurch, dass der Roboter zwei Beine besitzt, lässt sich die Policy durch zwei Abbildungen mit dem Parameter Phase, also $f(Phase) = (x,y)$, darstellen.

Die Policy, also die Bewegung des Oberkörpers, ist in Abbildung 5.5 für $(Speed_x = 1, Speed_y = 0, Speed_z = 0)$ visualisiert. Das obere Teilbild visualisiert die Bewegung, während das linke Bein das Standbein ist, das untere Teilbild entsprechend mit dem rechten Bein als Standbein.

Die rote bzw. schwarze Linie stehen für die Seitwärts- respektive Vorwärtsachse des Oberkörpers, wobei ein Wert von 0 bedeutet, dass der Oberkörper des Roboters sich genau in der Mitte der Anfangsposition der beiden Beine befindet. Da der Roboter sich vorwärts bewegt, startet die Vorwärtsposition des Oberkörpers im negativen Bereich in der Mitte zwischen den beiden Füßen. Danach bewegt sich der Oberkörper nach vorne über das vordere

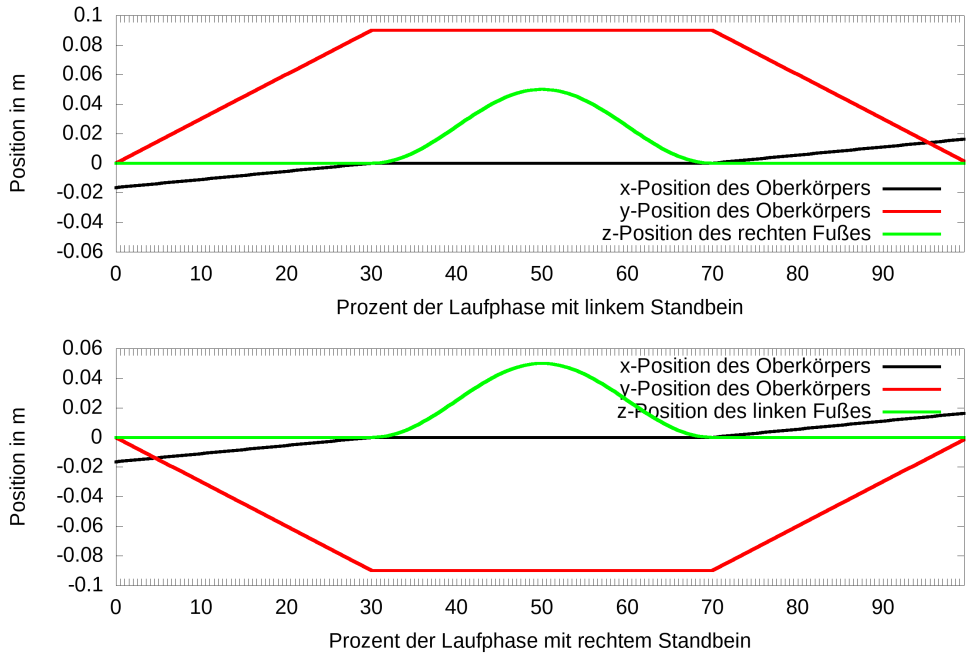

Abbildung 5.5: Initiale Policy: Oben ist die Policy mit links als Standbein und unten die Policy mit rechts als Standbein dargestellt. Zur Unterteilung in doppelt und einfach belastete Phasen ist die Höhe des jeweiligen Spielbeins in grün eingezeichnet. Die x- und y-Positionen sind die Vorinitialisierung der Policy. Beim Wechsel des Standbeins ändert sich die Nullposition des Oberkörpers um eine Schrittweite, wodurch der Start und der Endpunkt der x-Policy den gleichen Ort im Raum beschreiben.

Bein, dann wird das hintere Bein angehoben. Dieser Bewegungsablauf ist durch die grüne Linie angedeutet. Der Oberkörper befindet sich währenddessen über dem Standbein. Das Spielbein wird dabei nach vorne gesetzt und der Oberkörper abschließend wieder in die Mitte zwischen den beiden Beinen bewegt.

Bei der Seitwärtsposition startet der Oberkörper ebenfalls in der Mitte zwischen seinen Beinen, bewegt sich aber über das Schrittbein, um das Gleichgewicht zu halten und bewegt sich am Ende wieder zurück.

5.5 Policy Initialisierung

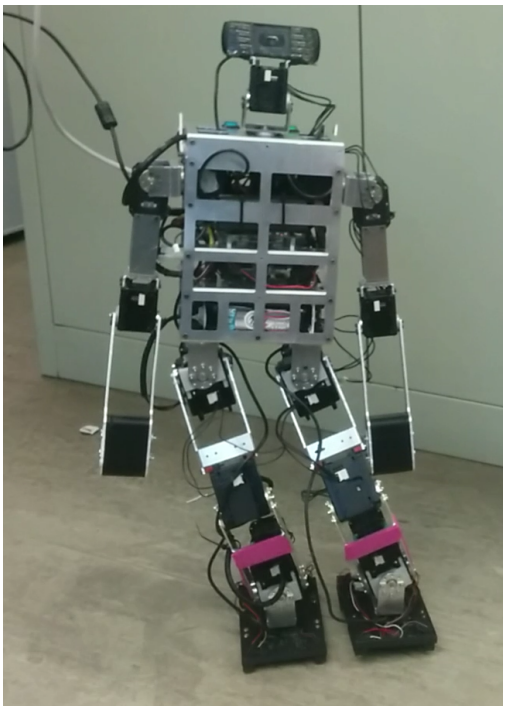

Abbildung 5.6: Minibot beim Austarieren seiner stabilen Position auf einem Bein. Er bewegt dabei seinen Oberkörper langsam nach rechts und hebt dabei das linke Bein, bis er die Position des Oberkörpers gefunden hat, in der er das linke Bein beliebig anheben kann, ohne umzufallen.

Um den Roboter beim anfänglichen Lernen nicht zu beschädigen, muss die Policy initialisiert werden, z.B. wie in Abbildung 5.5. Es ist am einfachsten, eine Startpolicy für ein sehr langsames Gehen zu definieren, da hier die dynamischen Effekte vernachlässigt werden können.

Für die Vorwärtsbewegung sind die obengenannten Positionen im Intervall $\left[-\frac{\text{Schrittweite}}{2}, \frac{\text{Schrittweite}}{2}\right]$ hinreichend. Da die Füße parallel zueinander sind, ist die Stabilität eines stehenden Roboters für die Vorwärtsachse in der Nullpose auch auf einem Bein gegeben. Die Seitwärtsbewegung hingegen muss auf der Hardware exploriert werden, da im Allgemeinen nicht bekannt ist, wann der Roboter auf einem Bein stehend zur Seite umkippt.

Aus diesem Grund wurde eine Anwendung entwickelt, die den Oberkörper des Roboters zur Seite bewegt, dann versucht ein Bein anzuheben und dabei detektiert, ob der Roboter umkippt und entsprechend mit dem Oberkörper gegensteuert. Diese Position wird dann zusammen mit der Schrittweite genutzt, um die Policy zu initialisieren. Die Anfangsinitialisierung, die auf dem Roboter Minibot errechnet wurde, ist in Abbildung 5.5 zu sehen.

Wenn der Laufalgorithmus bei h_{start} Prozent den Fuß anhebt, bei h_{ende} den Fuß wieder auf den Boden setzt und y_{stabil} die detektierte stabile Position ist, lässt sich die initiale Policy für das linke Standbein durch die folgenden Formeln darstellen. Das rechte Standbein folgt mit einem negativen y_{stabil} analog.

$$\pi_x(phase) = \begin{cases} -\frac{\text{Schrittweite}}{2} + \frac{\text{Schrittweite}}{2} \cdot \frac{phase}{h_{start}} & \text{für } 0 \leq phase \leq h_{start} \\ 0 & \text{für } h_{start} < phase \leq h_{ende} \\ \frac{\text{Schrittweite}}{2} \cdot \frac{phase - h_{ende}}{(1 - h_{ende})} & \text{für } h_{ende} < phase \leq 100 \end{cases}$$

$$\pi_y(phase) = \begin{cases} y_{stabil} \cdot \frac{phase}{h_{start}} & \text{für } 0 \leq phase \leq h_{start} \\ y_{stabil} & \text{für } h_{start} < phase \leq h_{ende} \\ y_{stabil} - y_{stabil} \cdot \frac{phase - h_{ende}}{(1 - h_{ende})} & \text{für } h_{ende} < phase \leq 100 \end{cases}$$

6 Evaluation

Im Folgenden werden die Ergebnisse des Ansatzes evaluiert. Hierbei wird zuerst die Qualität der Sensoren getestet. Danach wird auf die durch die Motoren entstehenden Einschränkungen eingegangen sowie die Versuche erläutert, diese auf Softwareebene zu beheben. Anschließend werden die verschiedenen Reinforcement Algorithmen miteinander verglichen.

6.1 Sensorgenauigkeit

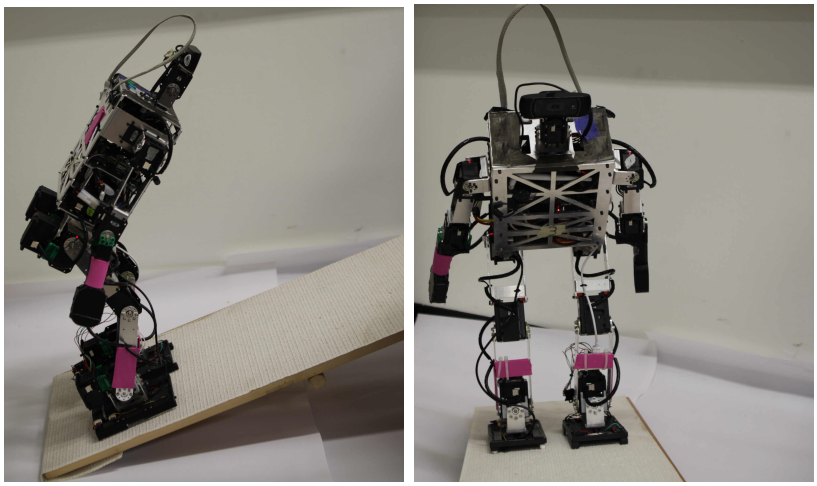

Abbildung 6.1: Minibot beim Balancieren auf einem geneigten Untergrund. Insbesondere die Hüftmotoren sind stark geneigt, damit der Roboter nicht nach hinten umkippt.

Um die Genauigkeit der Sensoren zu evaluieren, wurde ein einfacher Test entwickelt. Hierfür wurde Minibot auf einen kippbaren Untergrund gestellt. Ziel ist es nun, mithilfe der Sensoren und einem einfachen PID-Regler die Winkel der Pitch-Motoren so zu steuern, dass sich der *CoP* in der Mitte des Fußes befindet. Die Zielwerte der Motoren sowie die Winkel der Motoren

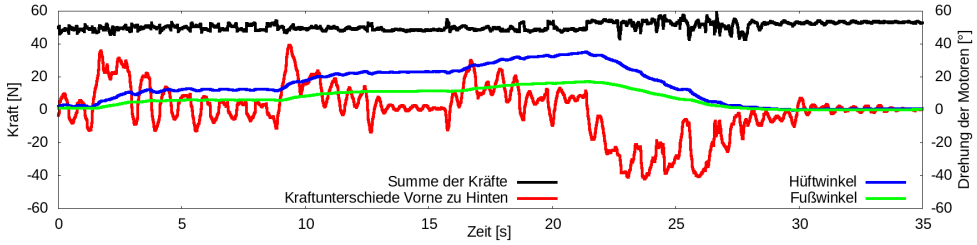

Abbildung 6.2: Kraftverteilung und Motorwinkel während des Balancierens auf einem geneigten Untergrund.

sind in Abbildung 6.2 zu sehen. Die Sprünge in der Kraftverteilung in den Sekunden 2,9 und 16 sind durch ein schnelles Kippen des Untergrundes zustande gekommen, während der Untergrund in den Sekunden von 22 bis 27 langsam geneigt wurde. Es ist bemerkenswert, dass obwohl sich die auf den Sensoren anliegenden Kräfte während der Zeit deutlich unterscheiden, die Gesamtsumme der Kräfte aber sehr konstant ist. Das Oszillieren des Kraftunterschiedes zwischen den vorderen und den hinteren Sensoren kam neben der ungenauen Steuerung mit dem PID-Regler insbesondere dadurch zustande, dass die Daten mit dem ersten Protoypen des Fußes aufgenommen wurde, der sehr nachgiebig war.

6.2 Hardwareeinschränkungen

Die eingesetzten Dynamixel Motoren haben deutlich Spiel in der Steuerung und im Getriebe. Das Team Rhoban, das eine alternative Firmware für die Motoren geschrieben hat, hat eine Abweichung von bis zu 8 Grad pro Motor gemessen [Rhoban, 2017]. Während die vollen 8 Grad im Motor im Rahmen der Versuche nicht gemessen wurden, waren jedoch 2,5 Grad wie in Abbildung 6.3 die Regel. Allerdings ist die Richtung des Ausschlags belastungsabhängig: Während das Bein belastet ist, wird der Zielwinkel nicht erreicht. Unbelastetet hingegen überzieht der Motor. Da diese Abweichungen von der Belastung abhängig sind, summieren sich die Fehler der einzelnen Motoren auf. Zusätzlich gibt es noch eine Verformung in dem tragenden Aluminiumskelett des Roboters, welche die Abweichung vom Sollwert noch verstärkt.

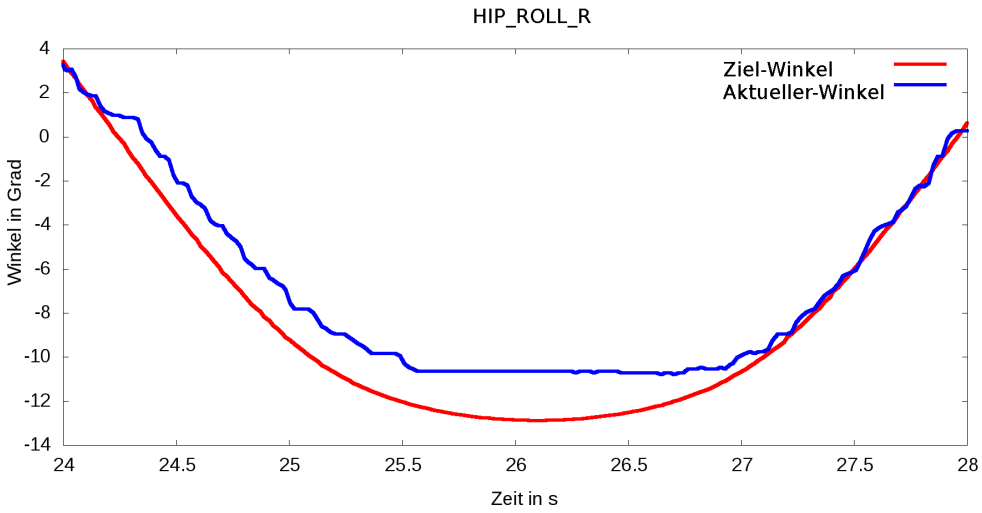

Abbildung 6.3: Beispiel des Unterschiedes zwischen Zielwert und dem erreichtem Winkel des rechten Hüftmotors während des Gehens. Ein sehr negativer Winkel des Motors bedeutet, dass der Oberkörper sich sehr weit rechts befindet und damit das Gewicht des Roboters hauptsächlich auf dem rechten Bein lastet. Durch das hohe Gewicht des Roboters arbeitet der Motor nicht ausreichend, um den Zielwinkel zu erreichen, wodurch der Unterschied zwischen dem Ziel- und dem aktuellen Winkel entsteht.

Dieses Problem mindert insbesondere die Seitwärtsstabilität des Roboters. Die Beine sind bezogen auf die x-Achse in der Mitte des Roboters. Deswegen führt eine Anhebung des einen Beines zwar dazu, dass das ganze Gewicht des Roboters auf dem anderen Bein lastet, aber diese Last ist hauptsächlich entlang der Achse des Beines. Bezogen auf die y-Achse sind die Beine sehr weit am Rand des Roboters. Dieser Abstand der Motoren zum Mittelpunkt führt für die einzeln belasteten Phasen in sehr kurzer Zeit zu einen sehr hohen Drehmoment am Motor. Diesen Anstieg des Drehmoments kann die Steuerung des Motors nicht ausgleichen. Deswegen zeigen insbesondere die Roll-Motoren des Roboters die starken Abweichung und weniger die Pitch-Motoren.

Dies führt dazu, dass der Roboter beim Laufen umkippt. Er versucht sich stabil auf einem Bein zu halten und erreicht nach ein paar Lerndurchgängen eine stabile Position für die einzeln belastete Phase. Hierbei entstehen zwei Probleme. Erstens: Wenn er nun das andere Bein senkt, dann trifft

der Fuß mit relativ hoher Geschwindigkeit auf den Boden und gibt dem Roboter dadurch einen Stoß. Dieser Stoß führt dazu, dass der Roboter sein Gleichgewicht verliert und umkippt. Zweitens: Wenn der Roboter eine stabile Position erreicht hat, wirkt die Hebelkraft der Gravitation am Motor schlagartig in die andere Richtung. Die Position des Motors kann sich dann sprunghaft verändern.

Ein erster Ansatz, diese Probleme zu beheben, war es, den Zielwert der Motorwinkel zu korrigieren. Dafür wurde über verschiedene Schritte die Abweichung errechnet, um dann entsprechend das Ziel anzupassen, wie in Abbildung 6.4 zu sehen. Dadurch entstand allerdings eine hohe Varianz in den Zielwerten der Motoren, was zu einer vermehrten Oszillation des ganzen Roboters führte. Durch diese Oszillation und die Varianz wurde der Roboter sehr instabil. Außerdem hat der Algorithmus bei 70,25 s nicht ausreichend und bei 71 s hingegen zu viel gegengesteuert. Während der Ansatz also meist richtig gegensteuert, führten die Fehlstellen und die erhöhte Oszillation dazu, dass der Ansatz die Stabilität kaum verbesserte.

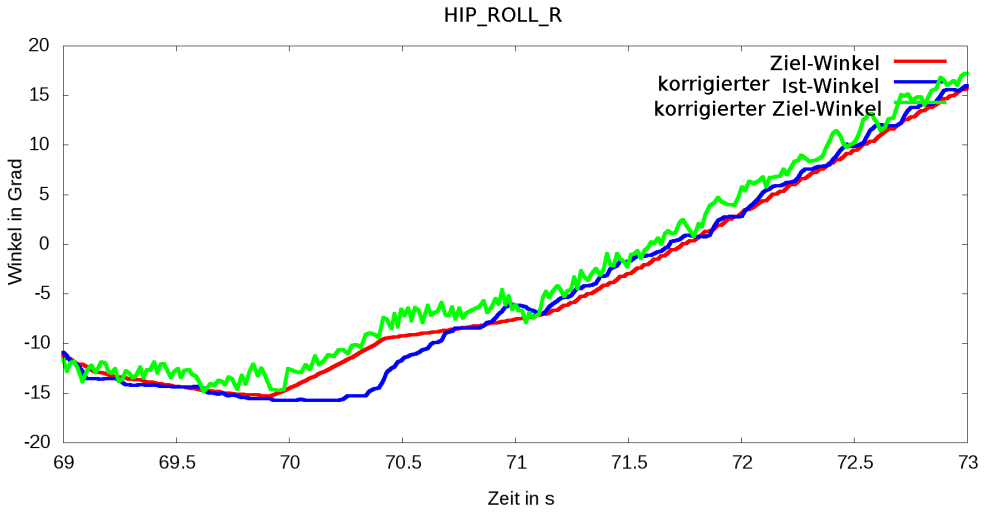

Abbildung 6.4: Ausschnitt aus einem Versuch, den Offset der Motoren durch ein übergeordnetes Steuersystem zu beheben. Hierbei wurden die Abweichungen der vorherigen Durchgänge als Grundlage der korrigierten Zielwerte genutzt. Die Abweichungen des Ist-Winkels vom Ziel-Winkel war in diesem Fall aber nicht wiederholbar genug, um diese Abweichung so sinnvoll auszugleichen.

Ein zweiter Ansatz, das Problem des Zusammenstoßes mit dem Boden zu beheben, entsprach dem Versuch, die aktuelle Abweichung der Ist-Position zu berechnen und diese bei der Soll-Position zu beachten. Die Berechnung der aktuellen Position ist über die direkte Kinematik errechenbar, allerdings ist es schwierig diese Information zu nutzen. Es ist nicht möglich, die Zielposition des Standbeins auf die Realposition zu ändern, in diesem Fall würde eine andere Zielposition den Motoren übergeben werden, wodurch diese weiter nachgeben und noch weiter entfernt von der ursprünglichen Zielposition wären. Es ist also nur möglich, die Zielposition des Spielbeins soweit anzupassen, dass sie sich nicht unterhalb der aktuellen Position des Standbeins befindet, um den oben genannten Stoß zu vermeiden. Allerdings führt das Spiel in den Motoren im Spielbein dazu, dass ein leichter Impuls kaum verhinderbar ist. Dieser Impuls führt zu Nachregelungen in den Motoren, sodass auch dieser Ansatz nicht ausreicht, zu verhindern, dass der Roboter umkippt.

Auch ein weiteres Optimieren des PID-Reglers führt nicht zum gewünschten Erfolg. Der I-Anteil des Reglers ist nicht vollständig dokumentiert. Insbesondere lässt sich der Integrationspart nicht aktiv zurücksetzen und es wirkt so, als ob der Motor ihn auch bei ausgeschaltetem I-Regler berechnet. Durch die ständige Berechnung ist der I-Regler bei Aktivierung in einem fehlerhaften Zustand, wodurch der Motor unkontrollierbare Bewegungen ausführt. Auch ein ständig aktivierter I-Regler führt zu keinem positiven Ergebnis, da die starke Belastungsphase im Vergleich zu den anderen Phasen relativ kurz ist. Ein Verstärken des P-Glieds ist auch problematisch, da der Roboter dann zu oszillieren beginnt.

Ebenso reduzierte ein Umbau von Minibot mit stärkeren Motoren die Abweichungen vom Sollwert nur wenig. Dies verdeutlicht, dass das Problem hauptsächlich in der Ansteuerung innerhalb der Motoren liegt und nicht durch fehlende Kraft in den Motoren entsteht. Am geeignetsten wäre deshalb ein Korrigieren dieser Abweichungen innerhalb der eigentlichen Motorsteuerung und nicht auf einer höheren Software-Ebene. Die Firmware der Motoren ist allerdings nicht veröffentlicht worden und daher nicht anpassbar.

6.3 Gelernte Policy

Im Folgenden wird für die angewendeten Strategien die jeweils gelernte Policy erklärt und die Vor- und Nachteile des Ansatzes diskutiert. Nach Morimoto et al. können die *Sagittalachse* (x-Achse) und die *Lateralachse* (y-Achse) unabhängig voneinander gelernt werden [Morimoto et al., 2005]. Dies ermöglicht, dass im ersten Ansatz des Finite Difference Policy Gradient ausschließlich die x-Achse gelernt wird und erst in den späteren Ansätzen auch die y-Achse.

6.3.1 Finite Difference Policy Gradient

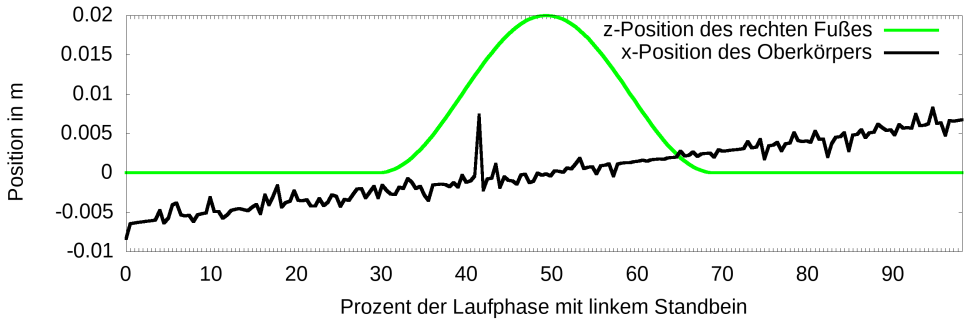

Abbildung 6.5: Gelernte Policy mit der Finite Difference Methode. Zur Unterteilung in doppelt und einfach belastete Phase ist die Höhe des jeweiligen Spielbeins in grün eingezeichnet. In schwarz ist die gelernte Policy visualisiert.

In Abbildung 6.5 ist die Visualisierung der gelernten Policy für die Finite Difference Methode abgebildet. Der Gradient wurde evaluiert, sobald für den Tile fünf mal Feedback vorlag. Die Abweichungen $\delta\Theta$ für das Lernen entstammten aus der Gleichverteilung $[-0.0025,0.0025]$.

Es ist keine sinnvolle Lerntendenz in Abbildung 6.5 sichtbar, obwohl der Roboter beim Laufen deutlich nach hinten geneigt war. Auch führen die gelernten Sprünge zu einem instabilen Laufen, da die Motoren entsprechend stark ihren Torque verändern.

Während es verbesserte Blackboxabschätzungen des Gradienten wie REINFORCE von Williams [Williams, 1992] gibt, die vermutlich besser abschneiden würden, scheint die Variabilität des Roboters bei aufeinanderfolgenden

Schritten zu stark für Blackboxabschätzungen des Gradienten zu sein. Da es mit der Abweichung des ZMP vom Zielwert auch eine direkte Abschätzung des Gradienten gibt, wurden keine weiteren Blackboxabschätzungen implementiert.

6.3.2 TD(λ)

Die Finite Difference Methode zeigt zwei deutliche Probleme. Neben der oben genannten Schwäche von Blackbox-Gradientenabschätzungen, die durch den ZMP abgelöst wurde, ist auch die hohe Varianz der Policy ein Problem: Die einzelne Auswertung jedes Tiles führt zu Sprüngen in der Policy. Diese Sprünge verursachen sprunghafte Bewegungen in den Motoren, die deutlich schlechter steuerbar und sehr viel ungenauer sind. Eine gleichmäßigere Policy ist deutlich vorteilhafter. Der Einfluss der zukünftigen Zustände in dem TD(λ) Ansatz ermöglicht eine gleichmäßigere Policy. In Abbildung 6.6 ist die gelernte Policy mit TD(λ) und dem ZMP als Gradientenabschätzung zu sehen, λ hatte beim Lernen den Wert 0.2 und n den Wert 10.

Beim Lernen brach ein Sensor, was während des Versuches nicht bemerkt wurde. Dieser gebrochene Sensor wurde nur belastet, wenn das andere Bein angehoben wurde. Der Bruch führte zum Sprung in der Policy bei 75%. Während die grundsätzliche Policy nun nicht mehr die unerwünschten Schwankungen hat, kann es immer noch zu Sprüngen führen, die für die Stabilität des Ganges ebenfalls massive Nachteile haben.

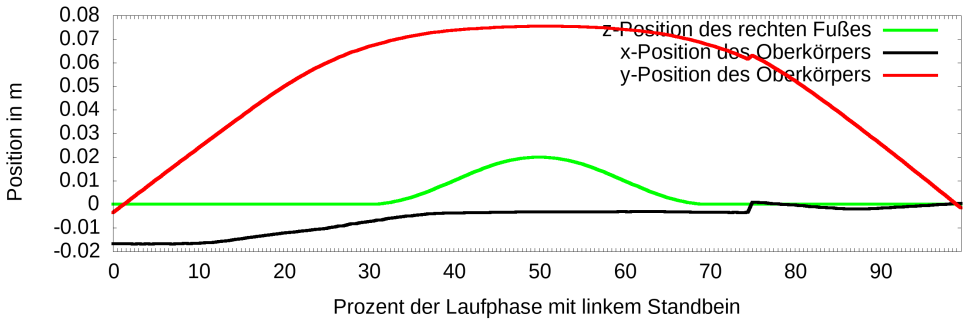

Abbildung 6.6: Gelernte Policy mit TD(λ). Zur Unterteilung in doppelt und einfach belastete Phase ist die Höhe des jeweiligen Spielbeins in grün eingezeichnet. In schwarz ist die x-Position der Policy und in rot die y-Position eingezeichnet.

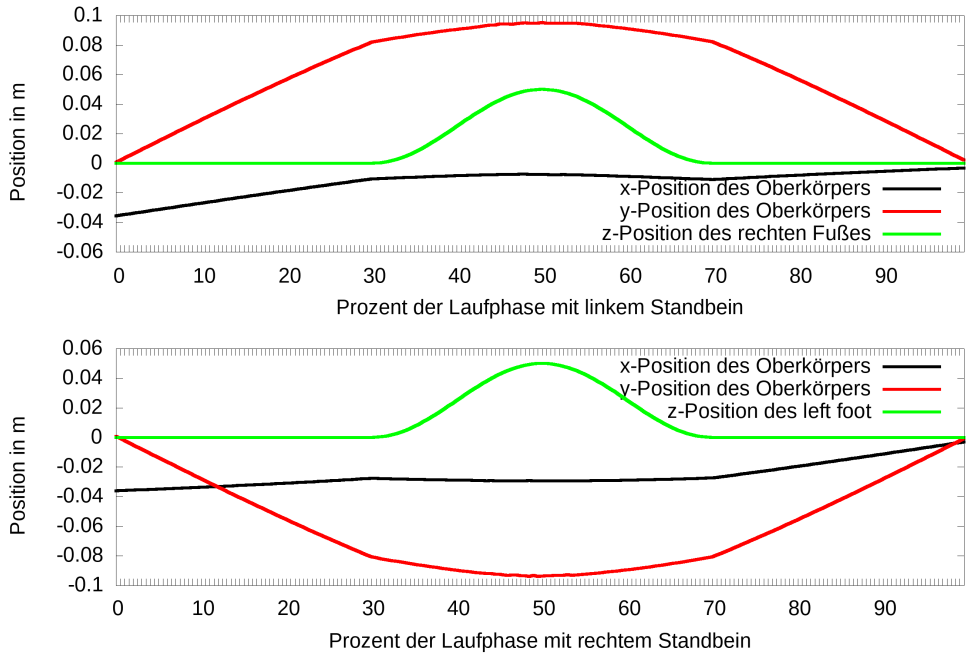

Abbildung 6.7: Gelernte Policy mit TD(λ) und Gauß-Filter. Obwohl die Policy vom linken und rechten Standbein im Optimalfall nahezu gespiegelt sind, bestehen genug kleine Unterschiede, die dazu führen, dass es sinnvoll ist, sie getrennt voneinander zu lernen.

6.3.3 TD(λ) mit Gauß-Filter

Um die Sprünge in der gelernten Policy zu vermeiden, lässt sich das Feedback mithilfe eines Gauß-Filters glätten. Das Strecken des Feedbacks über viele Tiles ermöglicht das Feedback-Signal besonders auf die einzeln belasteten Phasen zu optimieren. Der Roboter steht deutlich stabiler in der doppelt belasteten Phase, da die konvexe Hülle der beiden Füße zusammen deutlich größer als die eines Fußes ist. Da der Oberkörper sich immer in Bewegungsrichtung bewegt, verschiebt sich auch der ZMP während des Schrittes in Bewegungsrichtung. Da am Ende des vorherigen Schrittes sich der ZMP in der Mitte des hinteren Fußes befindet und der ZMP von diesem Punkt aus sich in Bewegungsrichtung mit bewegt, reicht es nun, ausschließlich Feedback zu geben, wenn der ZMP über den Mittelpunkt des vorderen Fußes hinausgeht. Der Rest der Zieltrajektorie wird über TD(λ) und dem Gauß-Filter definiert.

Die gelernte Policy mit TD(λ) und Gauß-Filter ist in Abbildung 6.7 zu sehen. Bei dem Versuch gab es 200 Tiles pro Standbein. Die Breite σ war mit 70 sehr groß, um eine möglichst gleichmäßige Bewegung zu erreichen, wobei die Breite des Einflusses zusätzlich auf 150 Tiles beschränkt war. Die y-Policy ist nur begrenzt aussagekräftig, da der Roboter aufgrund der Hardwareprobleme gleichwohl umfällt, wenn er nicht gestützt wird. Die x-Policy zeigt eine Verschiebung der Kurve in den negativen Bereich, der Roboter scheint also beim Laufen eine Neigung nach vorne zu haben. Das Feedback erreichte nach der anfänglichen Verschiebung ihr Minimum und die x-Policy veränderte sich anschließend kaum. Das dazugehörige aufsummierte Reward-Signal ist in Tabelle 6.1 zu sehen. Insbesondere die x-Policy des linken Fußes ist bereits nach acht Schritten in einem vergleichsweise stabilen Gang. Die x-Policy mit dem rechten Standbein bekommt bereits anfänglich einen schlechteren Reward, verbessert sich aber ebenfalls deutlich über die Zeit. Die Schwankungen in den Motoren führen dazu, dass der Algorithmus bei der y-Policy nicht in der Lage ist, den Reward signifikant zu verbessern.

Der ZMP unter dem linken Fuß während eines Schrittes ist in Abbildung 6.8 eingezeichnet. Da das Gewicht des Roboters nur während der einzeln belasteten Phase vollständig auf dem linken Bein lastet, ist hier der ZMP besonders relevant. Diese Phase ist auch am kritischsten bezüglich der Stabilität. Der ZMP war hier im Bereich von $\pm 15\%$ vom Fußmittelpunkt entfernt — ein solcher Unterschied entsteht bereits durch das ungenaue Stützen des Roboters.

Da der Roboter nicht exakt symmetrisch ist, gibt es Unterschiede in den gelernten Policys. Diese Unterschiede sind in Abbildung 6.9 dargestellt. Die gelernten Policys der y-Achse unterscheiden sich nicht besonders stark, da es softwareseitig eine Begrenzung der maximalen Veränderung pro Schritt gibt, die regelmäßig von dem Algorithmus erreicht wird. Bei der x-Policy hingegen ist ein Unterschied zwischen der linken und der rechten Policy sichtbar, während das Plateau der Position des Roboters über dem Standbein beim rechten Fuß bei $-0.035\,\mathrm{m}$ ist, befindet sich das Plateau bei $-0.015\,\mathrm{m}$. Der Unterschied entsteht unter anderem dadurch, dass die Nullpositionen der Motoren nicht exakt gleich sind. Deswegen ist der Oberkörper des Roboters bei gleicher Zielstellung entsprechend leicht anders gekippt und dadurch das Gewicht anders verteilt.

	Linker Fuß x-Reward	Linker Fuß y-Reward
Schritt 1	0.1391	0.2254
Schritt 2	0.1324	0.2585
Schritt 3	0.1184	0.2659
Schritt 4	0.0864	0.2186
Schritt 5	0.1109	0.2674
Schritt 6	0.0896	0.2679
Schritt 7	0.0715	0.1747
Schritt 8	0.0406	0.3412

	Rechter Fuß x-Reward	Rechter Fuß y-Reward
Schritt 1	0.2352	0.2277
Schritt 2	0.2354	0.2540
Schritt 3	0.2062	0.2742
Schritt 4	0.1724	0.2535
Schritt 5	0.1562	0.2566
Schritt 6	0.1560	0.2710
Schritt 7	0.1366	0.2675
Schritt 8	0.1367	0.2464

Tabelle 6.1: Der durchschnittliche Reward für das linke und rechte Standbein. Da die Updatefrequenz der Motoren und davon abhängig die Menge an Feedback pro Schritt nicht konstant ist, ist der Durchschnitt des Reward aussagekräftiger als die Summe. Der Reward ist dabei ein Fehlerwert, der perfekte Reward wäre null. Während der Fehlerwert der x-Policy am konvergieren ist, führen die Probleme der Motoren zusammen mit dem Stützen des Roboters dazu, dass die y-Policy nicht konvergiert.

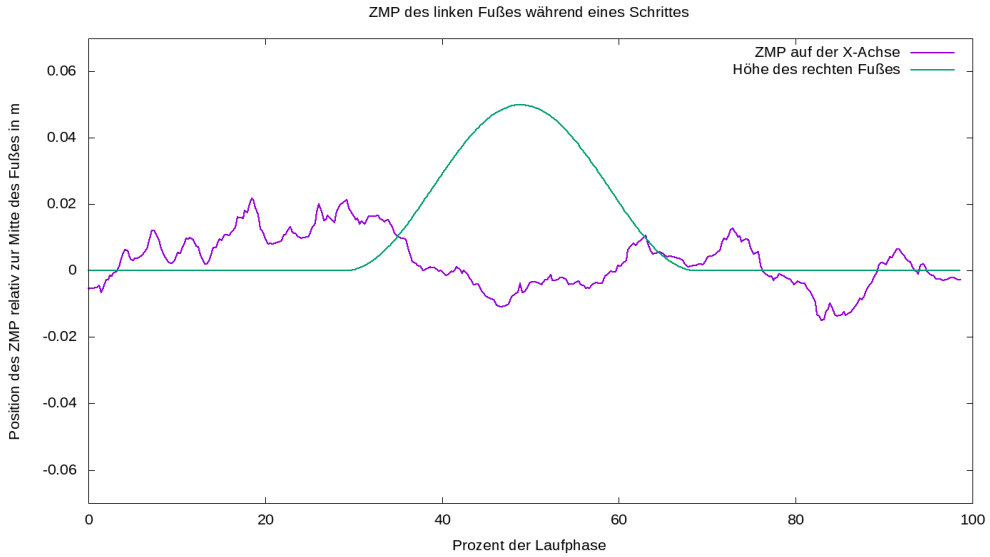

Abbildung 6.8: ZMP des linken Fußes auf der x-Achse. Die Länge des Fußes ist 0.14 m, deshalb ist die Höhe des Plots ± 0.07 m. Die Position des ZMP unter dem linken Fuß während der doppelt belasteten Phase — wenn die Höhe des rechten Fußes gleich 0 ist – definiert nur kaum die Stabilität des Roboters, da die Last auf dem rechten Fuß ebenfalls relevant ist. Während der kritischen einzeln belasteten Phase liegt der ZMP im ± 0.01 m.

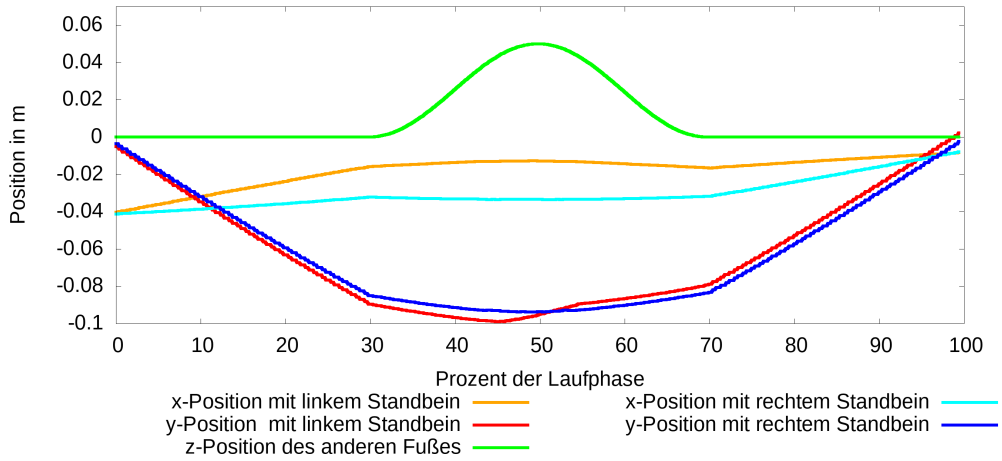

Abbildung 6.9: Vergleich der linken und rechten Policy jeweils während des Schrittes 8. Damit der Unterschied besser sichtbar wird, ist die y-Policy mit rechten Standbein horizontal gespiegelt.

6.3.4 Veränderungen der Policy beim Lernen

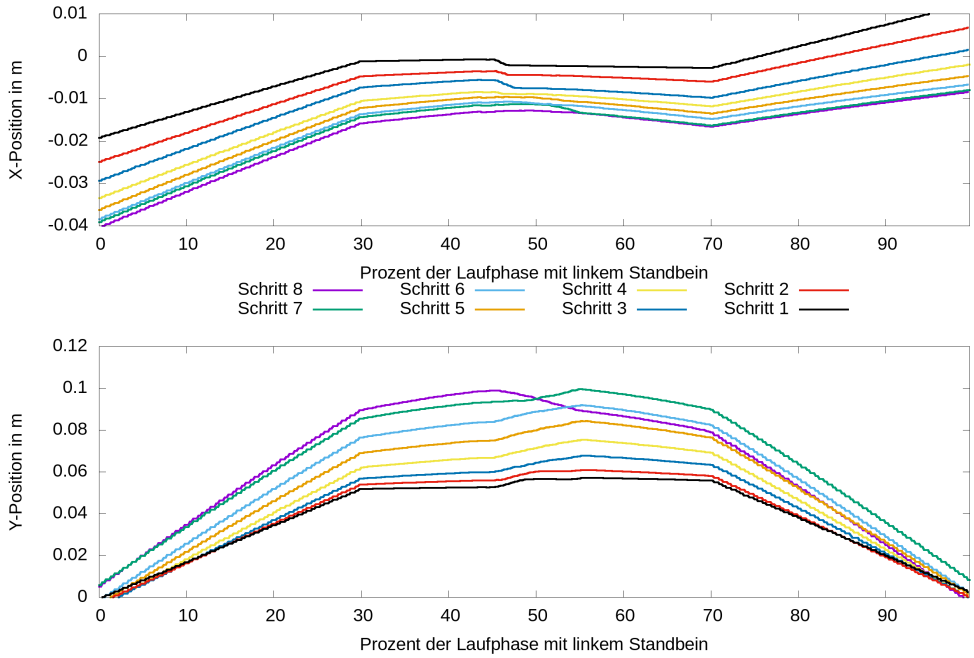

Abbildung 6.10: Ausgeführte Policy mit TD(λ) und Gauß-Filter über mehrere Schritte. Oben ist die x-Policy unten ist die y-Policy jeweils für das linke Standbein visualisiert. Die starken Steigungen in der Mitte der jeweiligen Policy sind Artefakte des Lernens und bedeuten, dass der Lernalgorithmus noch nicht konvergiert ist.

In Abbildung 6.10 sind die Zielpositionen des Oberkörpers während mehrerer Schritte vom TD(λ) mit Gauß-Filter eingezeichnet. Schon während des ersten Schrittes bekommt der Algorithmus genug Feedback um die Policy zu verbessern. Die starken Steigungen in der Mitte der jeweiligen Bewegung entstehen dadurch, dass der Lernalgorithmus bereits in der Mitte des Schrittes genug Feedback bekommen hat, um zu lernen. Dieses Lernsignal verändert über den Gauß-Filter auch den Wert der Tiles, die während dieses Schrittes noch nicht angewendet wurden. Diese Online Veränderung der Policy während ihrer Ausführung führt zu dem sichtbaren Artefakt und entsteht nur, wenn die Policy durch das Lernsignal stark verändert wird. Der Roboter ist dann nicht in einem stabilen Gang und es besteht die Gefahr, dass der Roboter entgegen der Richtung des Artefaktes umkippt. Um die Artefakte zu vermeiden, wäre es möglich, immer erst am Ende eines Schrittes und dann für den Schritt als Episode und nicht kontinuierlich

zu lernen. Die y-Policy wurde nicht mit der Position aus der Austarierung sondern mit einer niedrigeren Zielposition vorinitialisiert, wodurch die deutliche Steigerung der maximalen y-Position zustande kommt.

6.3.5 Seitwärtstrajektorie

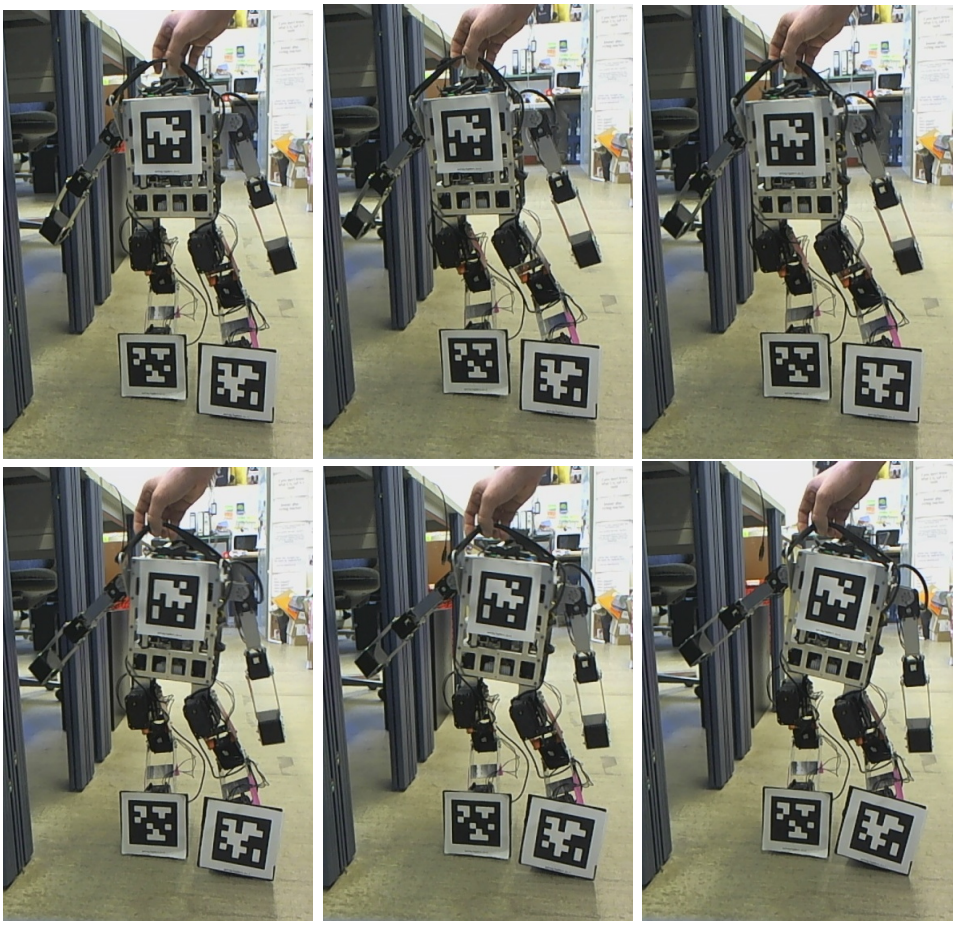

Abbildung 6.11: Minibot während eines Schrittes. Das Testsetup mit den drei verklebten Apriltags ist deutlich erkennbar, mit dessen Hilfe die realen Positionen des Oberkörpers und der Beine zueinander gemessen wurden. Besonders ab dem Bild unten links geben die Motoren des Roboters beim Anheben des rechten Beins nach, wodurch der Roboter sich nach rechts neigt. Dies entspricht den starken Abweichungen der realen von der Zielposition in Abbildung 6.12.

Die ausgeführte Seitwärtstrajektorie während des Gehens ist in Abbildung 6.12 zu sehen. Die in rot eingezeichnete y-Zielposition ist die ausgeführte Policy, die von der inversen Kinematik in die Motorwinkel übersetzt wird. Die Zielwinkel der Motoren unterscheiden sich von den real angefahrenen. Die Position des Oberkörpers wird aus den Motorpositionen mithilfe der direkten Kinematik berechnet und ist in grün visualisiert. Während des Gehens wurden mithilfe von Markern die realen Positionen des Oberkörpers gemessen. Der Versuchsaufbau von den Aufnahmen von Minibot beim Laufen ist in Abbildung 6.11 dargestellt. Die verklebten Marker sind von einer Bildverarbeitung gut erkennbare „Apriltags“, die für das Detektieren von Positionen im Raum entwickelt worden sind [Olson, 2011]. Die unter den Füßen wirkenden Kräfte in Abhängigkeit der Zielposition sind in Abbildung 6.13 eingezeichnet.

Während der doppelt belasteten Phasen, bei denen in Abbildung 6.12 keine Fußpositionen eingezeichnet sind, ist die ausgeführte Trajektorie relativ genau. Innerhalb der einzeln belasteten Phase, zu den Zeitpunkten sind die gelben Fußpositionen eingezeichnet, knickt der Roboter ein und die Position des Oberkörpers bewegt sich schnell und stark zur Mitte. Über weite Zeitspannen sind die gemessene und die über die Odometrie berechneten Positionen sehr ähnlich, einen Großteil der Abweichungen erkennen die Motoren also selber. Ob die Abweichungen zwischen der gemessenen und durch die Odometrie berechneten Trajektorie in den Sekunden bis 82, 90 bis 92 und ab 99 Ungenauigkeiten in der Positionierung und Detektion der Apriltags sind oder durch ungenaue Kodierung der aktuellen Motorpositionen entstehen, ist nicht ersichtlich.

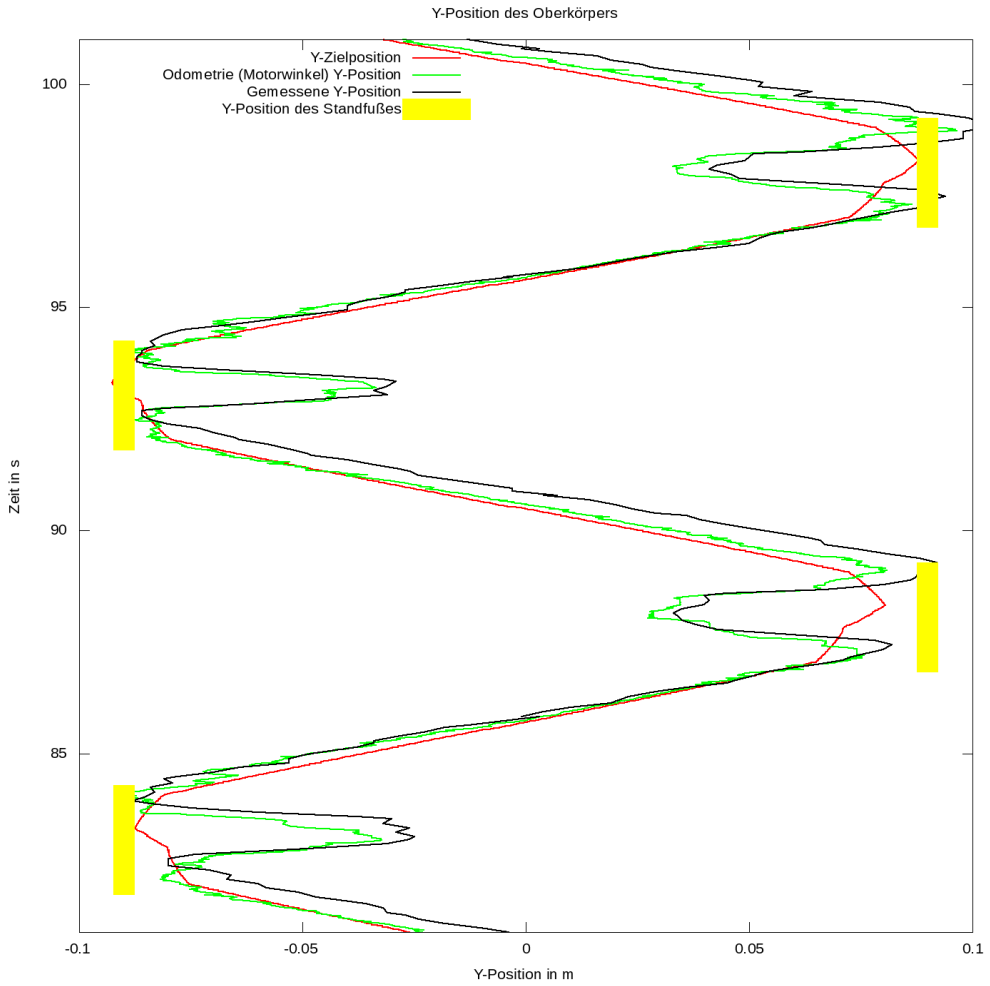

Abbildung 6.12: Ausgeführte Seitwärtstrajektorie der Schritte 7 und 8 aus der Abbildung 6.10. In gelb ist die Position des jeweiligen Standbeines eingezeichnet. Die Positionsangaben sind immer im Bezug zum Standbein, damit die Odometrie und die gemessenen Positionen vergleichbar sind. Die messbare Rotation des Fußes ist nicht dargestellt.

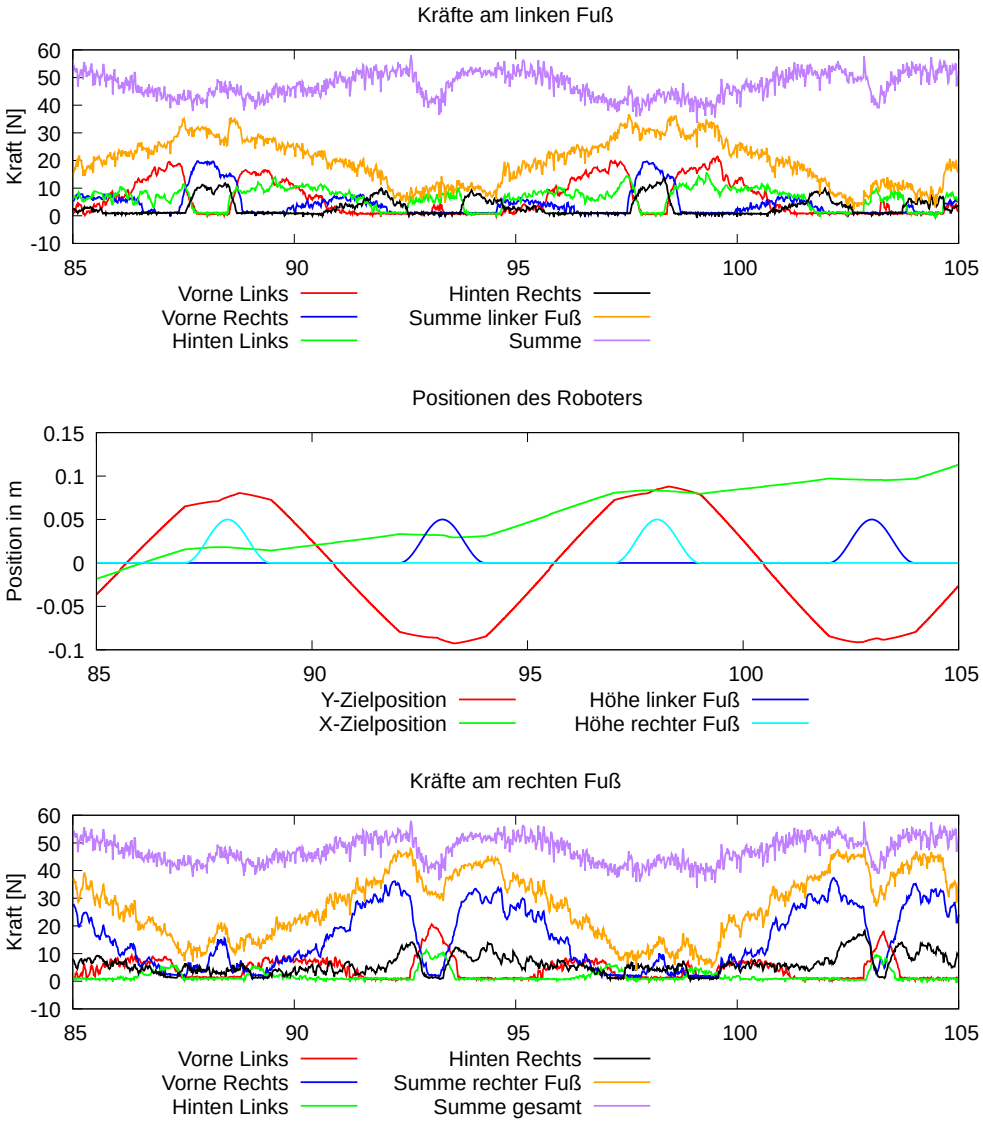

Abbildung 6.13: Entstehende Kräfte während des Gehens. Oben sind die Kräfte an den Kraftsensoren unter dem linken und unten die unter dem rechten Fußes eingezeichnet. Die wichtigsten Zielpositionen des Roboters sind in der Mitte visualisiert. Der Roboter wurde während des Gehens gestützt, wodurch nicht immer das ganze Gewicht des Roboters auf den Sensoren lastete.

6.4 Portierbarkeit

Der Algorithmus ist mit dem Ziel entwickelt worden, möglichst leicht auf andere Plattformen portiert werden zu können, deswegen sollte möglichst wenig angepasst und für die neue Plattform ausschließlich eine inverse Kinematik implementiert werden.

Insbesondere durch die Integration in ROS ist der Algorithmus leicht auf viele Roboterplattformen portierbar, da es mit beispielsweise KDL oder IKFast Implementationen für eine inverse Kinematik gibt, die leicht auf verschiedene Plattformen anwendbar sind. Allerdings muss sich die Abfolge der Motorwinkel bei gleicher Trajektorie wiederholen, diese Anforderung wird nicht von allen numerischen Näherungsalgorithmen erfüllt. Neben der inversen Kinematik müssen zum Portieren nur vergleichsweise wenige Einstellungen angepasst werden:

Höhe des Oberkörpers	Durch diese Einstellung wird bestimmt, inwieweit der Roboter beim Laufen in die Knie geht.
Z_{com}	Die Position des Schwerpunktes auf der z-Achse. Z_{com} ist leicht ausmessbar.
Schrittgeschwindigkeit	Sollte nach Gleichung 2.1 mit der natürliche Frequenz des Pendels der Länge Z_{com} korrelieren.
Schrittlänge	Dieser Parameter bestimmt die Grundlänge eines Schrittes. Während des Gehens ist die tatsächliche Schrittlänge ein in Abhängigkeit der Zielgeschwindigkeit Vielfaches dieser Grundlänge.
Geschwindigkeitsstufen	Durch die Geschwindigkeitsstufen wird die Endgeschwindigkeit des Roboters gesteuert. Zur Zeit ist die maximale Geschwindigkeit: $Geschwindigkeitsstufen \cdot Schrittgeschwindigkeit \cdot Schrittlänge$.

Schritthöhe	Die Höhe des Fußes am höchsten Punkt der Bewegung.
Belastungsphasen	Das Verhältnis der Belastungsphasen, also den einfachen und doppelt belasteten Phasen. Die voreingestellten Werte sollten im Allgemeinen ausreichend sein.
Fuß-Trajektorie	Hierüber kann insbesondere eingestellt werden, wie hoch der Fuß bereits gehoben sein muss, bevor er nach vorne bewegt werden kann. Die voreingestellten Werte sollten im Allgemeinen ausreichend sein.
Lern-Faktoren	Die Lern-Faktoren, wie λ des TD(λ), entscheiden über die Konvergenzgeschwindigkeit. Die voreingestellten Werte sollten im Allgemeinen ausreichend sein.

Keine der oben genannten Eigenschaften muss aufwendig durch Ausprobieren optimiert werden, was einen signifikanten Unterschied zu allen bekannten Algorithmen darstellt. Die Lern-Faktoren sind direkt mit dem Feedback des Reinforcement Learnings verbunden. Die aktuelle Position des ZMP wird in der Vorverarbeitung auf den Bereich -1 bis $+1$ normiert. Durch diese Normierung wird auf unterschiedlichen Robotern der Algorithmus mit den gleichen Faktoren ähnlich gut konvergieren. Die Fuß-Trajektorie ist insbesondere im RoboCup relevant, da der Rasen unterschiedlich hoch sein kann. Bei besonders hohen Halmen des Rasens muss sich der Fuß daher anders bewegen, um sich nicht zu verhaken. Dieser Faktor hat aber auf normalem Untergrund einen geringen Einfluss auf die Stabilität des Roboters.

Nicht trivial ist einzig das Verhältnis der Belastungsphasen, also die relative Zeit, die sich der Roboter auf einem bzw. beiden Beinen befindet. Hier sollten aber im Allgemeinen die voreingestellten Werte einen guten Richtwert liefern.

Die restlichen Konfigurationen sind (mit Ausnahme des messbaren Z_{com}) Designentscheidungen des Entwicklers, die relativ frei gewählt werden können.

Auch die Anforderung einer inversen Kinematik ist wenig problematisch, da kaum ein Roboter ausschließlich gehen können soll. Für viele der anderen Aufgaben des Roboters, wie im RoboCup das Schießen oder andere Ganzkörperbewegungen, wird ebenfalls eine inverse Kinematik benötigt.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

Im Rahmen der Masterarbeit wurde ein neuer Algorithmus entwickelt, der mithilfe von Kraftsensoren die Steuerung des Oberkörpers beim Gehen lernt. Er benötigt dabei, im Gegensatz zu den bekannten Algorithmen, keine aufwendigen Simulationen, ist gleichzeitig aber leicht portierbar.

Während die genutzte Plattform Schwächen zeigt — insbesondere bezüglich der Seitwärtsstabilität — wurde die Vorwärtsbewegung durch das Lernen deutlich stabilisiert. Dabei funktioniert die Repräsentation anschließend auch ohne Kraftsensoren. Dies lässt die Möglichkeit offen, die Sensoren zu deaktivieren, wenn sie beschädigt sind oder unrealistische Werte liefern — z.B. weil der Lichtschutz des Sensors das Infrarotlicht der Umgebung nicht vollständig blockiert.

Mit dem Algorithmus wurde ein Ansatz erprobt, der die Grundbewegung des Roboters optimiert und dabei den Einsatz übergeordneter Stabilisatoren ermöglicht, solange der Roboter stabile Phasen hat. Wenn das System nur selten aus dem Gleichgewicht gerät, lassen sich alle bekannten Stabilisatoren anwenden, z.B. Strategien auf Basis von Capture-Points [Pratt et al., 2006] oder die Algorithmen von Yi et al. [Yi et al., 2011]. Auch ein PID-Regler ist implementierbar. Dieser modifiziert die Zielposition des Oberkörpers, wenn der ZMP sich außerhalb eines Toleranzbereichs befindet. Allerdings muss, nachdem der Stabilisator eingegriffen hat, im Allgemeinen das Lernen kurz ausgesetzt werden, damit keine fehlerhaften Werte erlernt werden.

Durch die leichte Portierbarkeit der implementierten Software und ihre Integration in ROS besteht die Möglichkeit, dass die Software schnell und effektiv auch auf anderen Robotern eingesetzt werden kann, solange es eine inverse Kinematik und Kraftsensoren gibt.

7.2 Ausblick

Um ein stabiles Laufen der Roboter zu ermöglichen, muss evaluiert werden, wie die genutzte Hardware stabilisiert werden kann. Dies erfordert aber wahrscheinlich tiefergehende Veränderungen an den Roboterplattformen. Eine Möglichkeit ist ein zusätzliches Zahnrad, damit die Motoren mehr Torque produzieren und dabei weniger Spiel haben, wie es beispielsweise von dem iranischen Team Baset für ihren Adult-Size Roboter umgesetzt worden ist [Hosseini et al., 2016].

Eine mögliche Softwarelösung ist die alternative Firmware „Dynaban“ vom französichen Team Rhoban [Rhoban, 2017], bei der die Möglichkeit besteht, nicht nur aktuelle Positionen anzugeben, sondern eine Trajektorie, die von den Motoren angefahren werden soll. Dieses Feature ist allerdings bislang noch nicht ausführlich getestet worden.

Neben der Motorproblematik wäre es sinnvoll die Prototyp-Platinen gegen die professionell hergestellten PCB-Platinen zu tauschen, bei denen jedoch die noch bestehenden Fehler behoben werden müssten.

Schließlich sollte ein funktionierender Stabilisator auf Basis der Kraftsensoren implementiert werden. Wenn die angesprochenen Aufgaben umgesetzt worden sind, kann der implementierte Algorithmus in der nächsten RoboCup Weltmeisterschaft in Nagoya eingesetzt werden. Dies wird zu einem stabileren Gehen führen.

Literaturverzeichnis

[Arakawa and Fukuda, 1997] Arakawa, T. and Fukuda, T. (1997). Natural motion generation of biped locomotion robot using hierarchical trajectory generation method consisting of ga, ep layers. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 1.

[Bennett, 1984] Bennett, S. (1984). Nicholas minorsky and the automatic steering of ships. *IEEE Control Systems Magazine*.

[Bernstein, 2012] Bernstein, H. (2012). *Elektrotechnik/Elektronik für Maschinenbauer : Grundlagen und Anwendungen*. Vieweg+Teubner Verlag.

[Bestmann, 2016] Bestmann, M. (2016). Towards Using the Robot Operating System in RoboCup Humanoid League. Master's thesis, Universität Hamburg.

[Bestmann et al., 2015] Bestmann, M., Reichardt, B., and Wasserfall, F. (2015). Hambot: An open source robot for robocup soccer. In *RoboCup 2015: Robot World Cup XIX on RoboCup 2015: Robot World Cup XIX - Volume 9513*, New York, NY, USA. Springer-Verlag New York, Inc.

[Brown, 2017] Brown, A. (Retrieved: 03.01.2017). Stm32plus-framework. https://github.com/andysworkshop/stm32plus.

[Dasgupta and Nakamura, 1999] Dasgupta, A. and Nakamura, Y. (1999). Making feasible walking motion of humanoid robots from human motion capture data. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2.

[Debrunner, 2002] Debrunner, A. (2002). *Orthopädie. Orthopädische Chirurgie. Patientenorientierte Diagnostik und Therapie des Bewegungsapparates*. Bern, Göttingen, Toronto.

[Deisenroth et al., 2013] Deisenroth, M. P., Neumann, G., and Peters, J. (2013). A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2.

[Everlight Electronics, 2010] Everlight Electronics (2010). Datenblatt des everlight itr8307. http://www.everlight.com/file/ProductFile/ITR8307.pdf. Retrieved 2016-10-23.

[Fraden, 2010] Fraden, J. (2010). *Handbook of modern sensors*, volume 3. Springer.

[Goswami, 1999] Goswami, A. (1999). Postural stability of biped robots and the foot-rotation indicator (fri) point. *The International Journal of Robotics Research*, 18.

[Goswami, 2015] Goswami, A. (2015). *Walking Robots*. Springer London, London.

[Ha et al., 2013] Ha, I., Tamura, Y., and Asama, H. (2013). Development of open platform humanoid robot darwin-op. *Advanced Robotics*, 27.

[Harada et al., 2003] Harada, K., Kajita, S., Kaneko, K., and Hirukawa, H. (2003). Zmp analysis for arm/leg coordination. In *IROS*.

[Hosseini et al., 2016] Hosseini, M., Mohammadi, V., Jafari, F., and Bamdad, E. (2016). Baset adult - size 2016 team description paper. In *Robot Soccer World Cup*. Springer.

[Hu et al., 2016] Hu, Y., Eljaik, J., Stein, K., Nori, F., and Mombaur, K. (2016). Walking of the icub humanoid robot in different scenarios: Implementation and performance analysis. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*.

[Juricic and Vukobratovic, 1972] Juricic, D. and Vukobratovic, M. (1972). Mathematical modeling of biped walking systems. *ASME Publication*.

[Lau, 2004] Lau, D. (2004). *Algebra und Diskrete Mathematik 2*. Springer Spektrum.

[Metta et al., 2008] Metta, G., Sandini, G., Vernon, D., Natale, L., and Nori, F. (2008). The icub humanoid robot: An open platform for research in embodied cognition. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '08, New York, NY, USA. ACM.

[Missura, 2005] Missura, M. (2005). *Analytic and Learned Footstep Control for Robust Bipedal Walking*. PhD thesis, Rheinischen Friedrich-Wilhelms-Universität Bonn.

[Moilanen and Vadén, 2013] Moilanen, J. and Vadén, T. (2013). 3d printing community and emerging practices of peer production. *First Monday*.

[Morimoto et al., 2005] Morimoto, J., Nakanishi, J., Endo, G., Cheng, G., Atkeson, C. G., and Zeglin, G. (2005). Poincare-map-based reinforcement learning for biped walking. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*.

[NuBots, 2017] NuBots (Retrieved: 03.01.2017). Github page. https://github.com/NUbots/NUbots.

[Olson, 2011] Olson, E. (2011). Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*.

[Peters et al., 2003] Peters, J., Vijayakumar, S., and Schaal, S. (2003). Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*.

[PJRC, 2010] PJRC (2010). Die Dokumentation des Teensy 3.2. https://www.pjrc.com/store/teensy32.html. Retrieved 2016-10-23.

[Pratt et al., 2006] Pratt, J., Carff, J., Drakunov, S., and Goswami, A. (2006). Capture point: A step toward humanoid push recovery. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*.

[Radi, 2013] Radi, H. A. (2013). *Principles of Physics: For Scientists and Engineers*. Springer, Berlin Heidelberg.

[Rhoban, 2009] Rhoban (2009). Rhobans IKWalker. https://github.com/Rhoban/IKWalk. Retrieved 2017-01-13.

[Rhoban, 2017] Rhoban (Abgerufen am: 31.01.2017). Dynaban README. https://github.com/RhobanProject/Dynaban.

[RoboCup Federation, 2017] RoboCup Federation (2017). A brief history of robocup. http://robocup.org/a_brief_history_of_robocup. Retrieved 03.02.2017.

[Robotis, 2017] Robotis (Aberufen am: 07.02.2017). Dokumentation der Motoren, PID-Regler. http://support.robotis.com/en/product/actuator/dynamixel/mx_series/mx-106.htm#Actuator_Address_1A.

[Saputra et al., 2015] Saputra, A. A., Khalilullah, A. S., and Kubota, N. (2015). *Development of Humanoid Robot Locomotion Based on Biological Approach in EEPIS Robot Soccer (EROS)*. Springer International Publishing, Cham.

[Sardain and Bessonnet, 2004] Sardain, P. and Bessonnet, G. (2004). Forces acting on a biped robot. center of pressure-zero moment point. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 34.

[Schmidt, 2015] Schmidt, R. (2015). Development of a stable robot walking algorithm using center-of-gravity control. Master's thesis, University of Hamburg.

[Siciliano and Khatib, 2008] Siciliano, B. and Khatib, O., editors (2008). *Springer Handbook of Robotics*. Springer.

[Siciliano et al., 2008] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2008). *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition.

[Song, 2010] Song, S. (2010). Development of an Omni-directional Gait Generator and a Stabilization Feedback Controller for Humanoid Robots. Master's thesis, Virginia Polytechnic Institute and State University.

[Song et al., 2011] Song, S., Ryoo, and DW, H. (2011). Development of an omnidirectional walking engine for full-sized lightweight humanoid robots. In *35th Mechanisms and Robotics Conference.*, volume Volume 6.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[Swierzewski, 2017] Swierzewski (Retrieved: 03.01.2017). www.healthcommunities.com/foot-anatomy/muscles-tendons-ligaments.shtml.

[Takanishi et al., 1990] Takanishi, A., Takeya, T., Karaki, H., and Kato, I. (1990). A control method for dynamic biped walking under unknown external force. In *Intelligent Robots and Systems 90. Towards a New Frontier of Applications, Proceedings. IROS 90. IEEE International Workshop on*, volume 2.

[Team-Darwin, 2017] Team-Darwin (Retrieved: 27.01.2017). https://github.com/UPenn-RoboCup/UPennalizers.

[Team Nimbro, 2009] Team Nimbro (2009). Code release. https://github.com/NimbRo/nimbro-op-ros. Retrieved 03.02.2017.

[Tedrake, 2004] Tedrake, R. L. (2004). *Applied Optimal Control for Dynamically Stable Legged Locomotion*. Doctor's thesis, Massachusetts Institute of Technology.

[van Oort and Stramigioli, 2011] van Oort, G. and Stramigioli, S. (2011). Geometric interpretation of the zero-moment point. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*.

[Vaughan and Brian Davis, 1999] Vaughan, C. and Brian Davis, J. C. O. (1999). *Dynamics of Human Gait*.

[Vukobratović and Borovac, 2004] Vukobratović, M. and Borovac, B. (2004). Zero-moment point - thirty five years of its life. *International Journal of Humanoid Robotics*, 1.

[Vukobratovic et al., 2001] Vukobratovic, M., Borovac, B., and Surdilovic, D. (2001). Zero moment point-proper interpretation. *Proc. IEEE-RAS Int. Conf. Humanoid Robots Tokyo, Japan*.

[Wasserfall et al., 2017] Wasserfall, F., Hendrich, N., Fiedler, F., and Zhang, J. (2017). 3d-printed low-cost modular force sensors. *CLAWAR 2017: Proceedings of the 20th International Conference on Climbing and Walking Robots and Support Technologies for Mobile Machines*.

[Williams, 1992] Williams, R. J. (1992). *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning*. Springer US, Boston, MA.

[Yi et al., 2011] Yi, S. J., Zhang, B. T., Hong, D., and Lee, D. D. (2011). Practical bipedal walking control on uneven terrain using surface learning and push recovery. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 6. April 2017 _____

Fabian Fiedler

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 6. April 2017 _____

Fabian Fiedler