Sturzverminderung von humanoiden Fußballrobotern unter Entwicklung eines passiven Zweikampfes

Masterarbeit

Philip Reichenberg

Matrikelnummer: 4119829

3. März 2021

Fachbereich Mathematik / Informatik
Studiengang Master Informatik

1. Gutachter: Dr. Thomas Röfer
2. Gutachter: Prof. Dr. Rolf Drechsler

Erklärung

Ich versichere, die Masterarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Oyten, den 3. März 2021

.....

(Philip Reichenberg)

Zusammenfassung

Diese Arbeit behandelt die Entwicklung einer Laufmodifikation, neuer Schussbewegungen aus dem Laufen und eines passiven Zweikampfes für Fußball spielende Roboter vom Typ NAO V6. Der NAO ist ein 58 cm hoher, humanoider Roboter der Firma SoftBank Robotics, welcher im RoboCup Verwendung findet. Der RoboCup ist ein weltweiter Robotikwettbewerb, bei dem in verschiedenen Ligen gegeneinander angetreten wird, um Forschungsergebnisse in verschiedensten Bereichen zu evaluieren. Der NAO selber wird in der Standard Platform League verwendet, in der zwei Teams mit jeweils fünf Robotern gegeneinander antreten. Hier ist es wichtig, dass die Roboter schnell und stabil über das Spielfeld laufen und schießen können, trotz häufiger Berührungen mit anderen Robotern. Dies stellt aber eine große Herausforderung dar, da oftmals die Roboter langsamer laufen als sie könnten, um die Umfallrate minimal zu halten. Ebenso sind die Schüsse nicht sehr genau und andere Roboter werden gelegentlich nicht berücksichtigt. In dieser Arbeit wird das aktuelle Laufen mithilfe eines linearisierten invertierten Pendelmodells durch eine Laufschrittanpassung stabilisiert und ein Schussverfahren entwickelt, welches Laufschritte basierend auf Ballpositionen generiert. Aufbauend auf diesen Änderungen wird ein Zweikampfverhalten vorgestellt, in dem der Ball strategisch sinnvoller gespielt und gleichzeitig die Kollision mit anderen Robotern vermieden wird. Insgesamt soll durch die Kombination dieser drei Themen die Umfallrate minimiert und die Spielgeschwindigkeit gleichzeitig erhöht werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Zielsetzung

In den letzten Jahren hat sich die Leistungsspanne zwischen den besten Teams in der RoboCup Standard Platform League verkleinert. So laufen die Roboter der Top-Teams mit einer ähnlichen Geschwindigkeit, agieren vergleichbar schnell am Ball, können sich gut auf dem Spielfeld zurechtfinden und um andere Roboter herum planen. Dadurch unterscheidet sich der Erfolg zwischen den Teams oftmals mehr im strategischen Verhalten und kleineren Optimierungen.

Da alle Teams die gleichen Roboter verwenden müssen, ohne Modifikationen vorzunehmen und sich somit nur in der Software unterscheiden, liegt die Vermutung nahe, dass insbesondere im Bereich der Bewegungen bereits die Grenzen erreicht sind und die Teams dort keine Vorteile mehr erschließen können. Dennoch fallen häufig Roboter im Kampf um den Ball oder sogar ohne weitere sichtbaren Einflüsse auf dem Spielfeld um.

Diese Stürze sind dabei auch bei der relativen kleinen Körpergröße der NAOs nicht zu missachten. Üblicherweise vergrößert sich das Gelenkspiel oder es entstehen interne Schäden über die Wettbewerbe hinweg. Als Folge laufen die Roboter instabiler, können nicht mehr schnell laufen oder fallen aufgrund beschädigter Sensoren komplett aus. Dadurch bekommen jene Teams einen deutlichen Vorteil, die weniger Hardwareschäden vorweisen können, da deren Roboter nicht ausfallen und schneller laufen können.

Es stellt sich somit die Frage, warum umfallende Roboter weiterhin ein Hauptbestandteil der Wettbewerbsspiele sind und ob dies durch Softwarelösungen vermeidbar ist, ohne das Spielgeschehen zu verlangsamen oder sogar mit einer Beschleunigung dessen.

In dieser Arbeit soll dieses Problem insofern gelöst werden, dass die Roboter unabhängig ihrer Abnutzungserscheinungen schnell laufen können und eines der Hauptrisiken vermindert wird, bei dem sie aus Erfahrung des letzten Wettbewerbes häufig umfielen: der Zweikampf. Hierfür sollen Statistiken über die Kontexte gesammelt werden, in denen Stürze geschehen. Basierend darauf werden drei große Bereiche in dieser Arbeit behandelt. Zuerst soll das

aktuell verwendete Laufen von dem Team B-Human, auf dessen Codebasis gearbeitet wird, erweitert werden, um ein schnelleres Laufen zu ermöglichen, welches gleichzeitig resistenter gegenüber äußeren Einflüssen sein soll. Anschließend werden die Schüsse aus dem Laufen neu entwickelt, um diese vollständig in das Laufen zu integrieren. Dies ist aktuell nur teilweise der Fall. Dadurch kann das Zweikampfverhalten nur eingeschränkt handeln. Abschließend wird eine Alternative zum aktuellen Zweikampfverhalten vorgestellt. Diese soll die Möglichkeiten eines voll integrierten Schießens im Laufen vollständig nutzen, dabei aber Risiken minimieren, ohne das Spielgeschehen um den Ball zu verlangsamen.

1.2 Aufbau

Im folgenden Kapitel wird zuerst der Hintergrund und das verwendete Software-Framework dieser Arbeit vorgestellt. Anschließend wird die Problemstellung beschrieben, die in dieser Arbeit behandelt wird, sowie eine Übersicht an relevanten Arbeiten gegeben, die sich mit solchen Problemen beschäftigt haben. Kapitel 4 behandelt daraufhin die wichtigsten Begriffe, die für diese Arbeit relevant sind. Kapitel 5, Kapitel 6 und Kapitel 7 beschreiben jeweils getrennte Lösungsansätze in ihren jeweiligen Themenbereichen für die zuvor erarbeiteten Probleme, bevor schlussendlich in Kapitel 8 ein Fazit der gesamten Arbeit gezogen und mögliche Verbesserungen sowie ein Ausblick gegeben wird.

Kapitel 2

Plattform

2.1 RoboCup[1]

Der RoboCup ist Teil der *RoboCup Federation*, welche das Ziel hat, die Wissenschaft der Robotik mithilfe von weltweiten Wettbewerben, Konferenzen und Messen zu fördern. Der RoboCup ist eine Wettbewerbsveranstaltung, in der Forschungsgruppen gegeneinander antreten. Einer der Wettbewerbe ist die Standard Platform League, kurz *SPL*. In dieser müssen alle teilnehmenden Teams mit Robotern des Typs *NAO* (siehe Abschnitt 2.2) von Softbank Robotics [2021] autonom spielen, ohne die Hardware dabei zu modifizieren. Die SPL ist somit ein reiner Softwarewettbewerb. In der *SPL* spielen zwei Teams mit jeweils fünf Robotern des Typs *NAO* auf einem 9 m × 6 m großen Kunstrasen mit angepassten Regeln Fußball.

Ziel des RoboCups ist es, im Jahre 2050 mit einem Robotikteam im Fußball den amtierenden menschlichen FIFA Fußballweltmeister zu schlagen. Um dieses Ziel zu erreichen, werden die Regeln, welche nicht vollständig dem echten Fußball gleichen, jährlich angepasst. So wurde in der *SPL* noch vor ein paar Jahren mit farbigen Toren und Bällen gespielt, die aber mittlerweile durch weiße Tore und schwarz-weiße Bälle ausgetauscht wurden.

2.2 NAO

Aktuell wird die neueste NAO Generation V6 verwendet (siehe Abbildung 2.1). Diese hat 25 Freiheitsgrade, ein Gewicht von 5.4 kg und eine Höhe von 57.4 cm. Verbaut ist im Torso eine IMU, welche sowohl ein Gyrometer als auch ein Accelerometer mit 3D-Messungen besitzt. Die Positionen der Gelenke können alle 12 ms angesteuert werden. Eine Besonderheit der Gelenke ist dabei, dass für Drehungen in der z-Achse der Beine nur das Gelenk HipYawPitch verantwortlich ist und dabei beide Beine gleichzeitig dreht. Eine getrennte Drehung der Beine in dieser Achse ist daher nicht möglich.

[1] Teile von diesem Text wurden von meiner Bachelorarbeit [vgl. Reichenberg, 2018] übernommen.

Zusätzlich besitzt der NAO V6 Drucksensoren unter den Füßen und zwei Kameras im Kopf. Die Kameras liefern jeweils 30 Bilder pro Sekunde, asynchron zueinander, während alle weiteren Sensoren 83.33 mal pro Sekunde verfügbar sind.

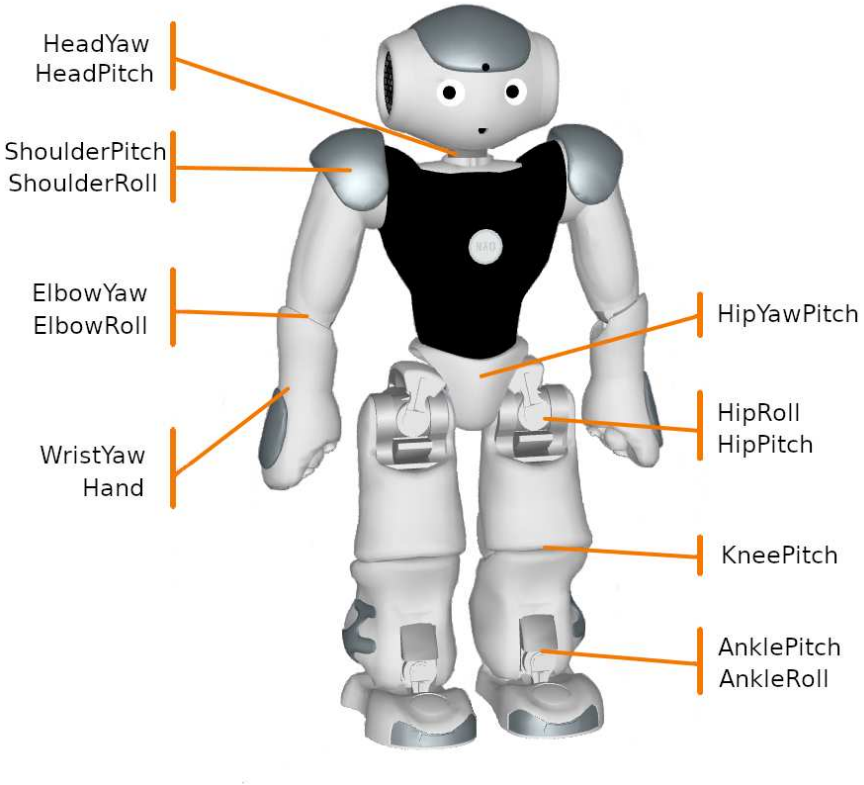

Abbildung 2.1 Der NAO V6. Grafik übernommen vom Team B-Human.

2.3 Framework

Im B-Human-Framework existieren insgesamt fünf Threads, welche auf mehreren Prozessorkernen parallel zueinander laufen. In Abbildung 2.2 sind die Threads abgebildet. In diesen werden sogenannte Repräsentationen ausgetauscht, die von Modulen bereitgestellt werden. Repräsentationen sind hierbei Datenstrukturen, in die Informationen geschrieben werden. Jeweils nur ein Modul aktualisiert eine Repräsentation je Zyklus, jedoch können mehrere Module auf dieselbe Repräsentation zugreifen, um Informationen zu lesen. Eine Repräsentation kann zum Beispiel ein Kamerabild, die Sensordaten oder erkannte Objekte im Kamerabild sein. Module sind für sich getrennte Programmteile, welche auf bereits geschriebene Repräsentationen zugreifen und neue bereitstellen können. Zum Beispiel existieren für die Kamerabilder jeweils separate Module, welche diese Bilder von der Schnittstelle des NAOs entgegennehmen und in Repräsentationen schreiben.

Die Threads laufen dazu zueinander teilweise asynchron mit unterschiedlichen Taktraten. Der Motion-Thread startet seinen Zyklus mit dem Empfangen von neuen Sensordaten, der Upper-

und Lower-Thread mit dem Empfangen von Bildern der jeweiligen oberen und unteren Kamera, während der Cognition-Thread erst startet, sobald der Upper- oder Lower-Thread mit seinem Zyklus fertig ist. Der Cognition-Thread behandelt dabei die Verhaltensebene. In dieser wird auf Basis der Modellierung entschieden, was der Roboter tun soll. Der Motion-Thread behandelt die Ansteuerung der Gelenke und setzt das angefragte Verhalten des Cognition-Threads um. Ein Zyklus wird auch als Frame bezeichnet.

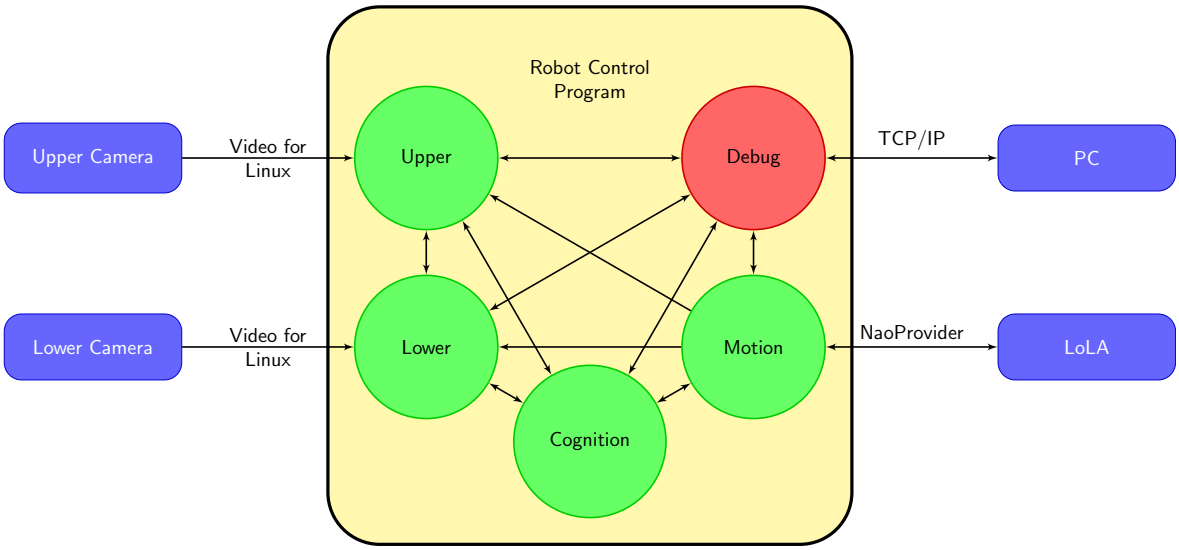

Abbildung 2.2 Die Threads auf dem NAO beim Team B-Human. [vgl. Röfer u. a., 2019]

Kapitel 3

Problemanalyse

3.1 Wo liegen die Probleme?

Das Spielverhalten, wodurch die Roboter umfallen, wird durch Videoaufnahmen der Spiele vom RoboCup 2019 und den German Open 2019 analysiert. Diese sind von den Teams Nao Devils Dortmund (Dortmund [2021]) und Berlin United - NaoTH (NaoTH [2021]) auf YouTube bereitgestellt und öffentlich zugänglich. Zusätzlich werden die drei Top-Teams neben B-Human verglichen, bewertet durch deren Platzierung beim RoboCup 2019. Der Vergleich besteht dabei aus Zählungen, wie oft Roboter in welchem Kontext umgefallen sind. Diese Aufzählungen für beide Wettbewerbe sind in Tabelle 3.1 und Tabelle 3.2 veranschaulicht. Die drei Top-Teams sind dabei, absteigend aufgezählt, Nao-Team HTWK von der Hochschule für Technik, Wirtschaft und Kultur Leipzig, rUNSWift von der University of New South Wales und Nao Devils von der TU Dortmund.

Im Vergleich der vier Teams fällt auf, dass die Roboter vom Team B-Human auf beiden Wettbewerben signifikant weniger umfielen als die der anderen Teams. 29.4 % - 40.8 % weniger auf den German Open respektive 34.5 - 40 % auf dem RoboCup. Die Roboter vom Team rUNSWift liefen sehr nahe an anderen Robotern vorbei und stürzten dadurch häufig durch Schulter-an-Schulter-Berührungen. Die Roboter vom Team Nao-Team HTWK liefen auf den German Open häufig entweder in andere Roboter oder traten beim Dribbeln des Balles auf die Füße gegnerischer Roboter. Nao-Team HTWK hat dieses Problem auf dem RoboCup dadurch reduziert, beurteilend aus den Videoaufnahmen und aus Gesprächen mit einigen derer Teammitglieder, indem deren Roboter unter anderem in Zweikämpfen auf der Stelle liefen statt in die Gegenspieler zu laufen. Sie haben also aktiv gewartet. Dadurch sind sie aber häufig, da der Ball im Zweikampf öfter zur Seite rollte, seitlich gegen den Gegenspieler gelaufen und folglich umgefallen. Inwiefern eine Verbesserung an deren Hinderniserkennung dabei geholfen hat die Umfallrate zu reduzieren, kann nicht beurteilt werden.

Kontext / Team	B-Human	rUNSWift	Nao-Team HTWK	Nao Devils
Kollisionen rückwärts	11	2	6	3
Kollisionen vorwärts	16	22	92	40
Kollisionen seitlich	23	50	28	18
Schüsse aus dem Laufen	14	-	-	-
Sonstiges	13	35	4	38
Gesamt	77	109	130	99
Durchschnitt pro Spiel	15.4	21.8	26	24.75

Tabelle 3.1 Die Umfallhäufigkeiten in verschiedenen Kontexten von Robotern der Teams B-Human, rUNSWift, Nao-Team HTWK und Nao Devils auf den German Open 2019. Verglichen werden Kollisionen mit anderen Robotern, aufgeteilt aus Sicht des umgefallenen Roboters, wie er mit denen kollidierte. Im Vergleich stehen neben Stürzen als Folge von Kollisionen mit anderen Robotern oder Torpfosten auch solche nach einem Schuss aus dem Laufen oder ohne jeglichen Einfluss anderer. Eine Kollision ist dabei die Berührung zweier Roboter, sei es Schulter an Schulter, Fuß gegen Bein, Fuß auf Fuß oder ähnliches. B-Human, rUNSWift und Nao-Team HTWK spielten fünf Spiele, Nao Devils vier.

Kontext / Team	B-Human	rUNSWift	Nao-Team HTWK	Nao Devils
Kollisionen rückwärts	2	9	10	6
Kollisionen vorwärts	42	32	41	45
Kollisionen seitlich	21	57	48	37
Schüsse aus dem Laufen	9	-	11	-
Sonstiges	4	21	20	40
Gesamt	78	119	130	128
Durchschnitt pro Spiel	11.14	17	18.57	18.29

Tabelle 3.2 Die Umfallhäufigkeiten in verschiedenen Kontexten von Robotern der Teams B-Human, rUNSWift, Nao-Team HTWK und Nao Devils auf dem RoboCup 2019. Verglichen werden Kollisionen mit anderen Robotern, aufgeteilt aus Sicht des umgefallenen Roboters, wie er mit denen kollidierte. Im Vergleich stehen neben Stürzen als Folge von Kollisionen mit anderen Robotern oder Torpfosten auch solche nach einem Schuss aus dem Laufen oder ohne jeglichen Einfluss anderer. Eine Kollision ist dabei die Berührung zweier Roboter, sei es Schulter an Schulter, Fuß gegen Bein, Fuß auf Fuß oder ähnliches. Alle Teams spielten sieben Spiele.

Von den German Open zum RoboCup ist bei den Robotern aller Teams ist eine Reduktion im durchschnittlichen Umfallen pro Spiel zu erkennen. Die Gründe dafür können aus den Tabellen nicht entnommen werden, da unbekannt ist, inwiefern die Bildverarbeitung und das Verhalten Einfluss genommen haben. Lediglich für die sonstigen Fälle, welche umfallende Roboter beinhalten, die ohne Einfluss anderer umfielen, lassen sich Vermutungen anstellen. Diese sollen im Folgenden untersucht werden. So ist auffällig, dass die Roboter vom Team B-Human auf den German Open 13-mal ohne Einflüsse stürzten, auf dem RoboCup aber nur 4 Mal. rUNSWift sank dabei von 35 auf 21 Fälle, während Nao-Team HTWK von nur 4 auf 20 stieg und Nao Devils fast gleich viele Fälle behielten. Alle Teams mussten dabei auf dem RoboCup mehr Spiele absolvieren, wodurch sich die Nao Devils, bezogen auf Stürze ohne

Einfluss, deutlich verbessert haben, von 9.5 auf 5.7 Stürze pro Spiel. Für Nao-Team HTWK lässt sich deren Anstieg von 0.8 auf 2.9 Stürze pro Spiel damit erklären, beurteilend aus den Videoaufnahmen, dass die Roboter häufig falsche Schrittwechsel durchführten, während sie auf der Stelle liefen und als Folge dessen umfielen. Die Roboter vom Team rUNSWift hingegen fielen auf beiden Wettbewerben häufig um, während die Roboter schnell geradeaus oder eine Kurve liefen. Dieselben Ursachen ließen sich auch beim Team B-Human auf den German Open erkennen. Die einzige signifikante Änderung, die B-Human zum RoboCup-Wettbewerb vornahm, war eine Verschärfung der Restriktion, wie viel Translation ein Laufschritt haben darf, abhängig von der Rotation, die ausgeführt werden soll. Je mehr Rotation in einem Laufschritt vorhanden ist, desto kleiner muss dieser sein. B-Human lief somit effektiv langsamer. Bei den Robotern vom Team Nao Devils hingegen besteht ein anderes Problem: Diese fielen hauptsächlich seitlich um, während sie normal herumliefen. Bei B-Human und rUNSWift fielen die Roboter hauptsächlich nach vorne oder nach hinten um. Die Ursachen für Nao Devils können aus den Videoaufnahmen alleine nicht entnommen werden.

Da die Roboter aller vier Teams vergleichsweise gleich schnell und nur unterschiedlich große Kurven liefen, d.h. unterschiedlich große omnidirektionale Laufschritte ausführten, beurteilend aus deren Veröffentlichungen und den Videoaufnahmen (rUNSWift [2019]; HTWK [2019]), liegt die Vermutung nahe, dass bei größeren Laufschritten und mehr Freiheiten in der Kombination von Rotation und Translation die Roboter instabiler laufen und somit häufiger umfallen. Möglicherweise lassen die Top-Teams ihre Roboter also nicht schneller laufen, um die Umfallrate minimal zu halten.

Ebenfalls kann im Bezug auf Störungen im Laufen, resultierend durch Kollisionen mit anderen Robotern, angenommen werden, dass die Roboter eigentlich schneller laufen könnten als sie es aktuell tun, darauf aber verzichtet wird, da bereits leichte Unebenheiten im Feldboden die Wahrscheinlichkeit für ein instabiles Laufen stark erhöhen.

Auch fällt auf, dass auf dem RoboCup von den 78 Stürzen bei B-Human 31 davon ein Resultat des Zweikampfes am Ball waren. Dies entspricht knapp 40 %. Es ist schwer zu sagen, ob die anderen Teams ein ähnliches Verhältnis haben. Für B-Human konnte dies sehr leicht aus den Videoaufnahmen entnommen werden, da die Roboter in solchen Situationen immer Schüsse aus dem Laufen ausführen und diese durch Sprachausgabe ansagen. Dies ist für die anderen Teams nicht der Fall. Hier stellt sich die Frage, inwiefern diese Stürze vermeidbar sind, ohne die Spielgeschwindigkeit zu reduzieren.

3.2 Verwandte Arbeiten

Um die Stabilität eines humanoiden Roboters zu beschreiben, wird häufig der Zero Moment Point (ZMP) verwendet [vgl. Vukobratović u. Borovac, 2004]. In Kombination mit einem linearisierten invertierten Pendelmodell [vgl. Kajita u. a., 2001], kurz LIPM, können Regler verwendet werden, um ein stabiles Laufen zu generieren. Diesen Ansatz findet man in vie-

len Laufbewegungen, mit einem älteren Vertreter wie Kajita u. a. [2010], oder neueren wie Schwarz u. a. [2019] und Tsogias [2016].

Eines der bekanntesten Verfahren ist die Verwendung eines ZMP-Preview-Controllers [vgl. Katayama u. a., 1985]. Hier wird der Schwerpunkt bewegt, damit dieser einer Trajektorie für zukünftige ZMPs folgt. Dies wird durch passende Fußpositionen erreicht.

Eine der neuesten Umsetzungen aus dem RoboCup Umfeld ist das *Capture Step* Verfahren von Missura u. a. [2019], welches sehr resistent gegenüber äußeren Einflüssen ist. Hier werden die Fußtrajektorien des ausgeführten Laufschrittes angepasst, um die Stabilität der Roboter zu bewahren.

Auch üblich ist eine Laufschrittplanung basierend auf der Umgebung. [Sabe u. a., 2004] und [Belter, 2019] modellieren die Umgebung, sodass Laufschritte, die aufgrund des Untergrundes den Roboter destabilisieren würden, nicht ausgeführt werden. Dadurch sind diese Roboter in der Lage, auf unterschiedlichem Terrain zu laufen. Um ein solches Umgebungsmodell zu erhalten, wird üblicherweise auf Stereokameras zurückgegriffen, mit denen räumliche Bilder berechnet werden können.

Für Bewegungen wie Schüsse, welche unter anderem essentiell in Zweikämpfen sind, wird häufig ein Key-Frame-basierter Ansatz verwendet. Ein Key-Frame beschreibt hierbei entweder die Positionen der einzelnen Extremitäten als Pose, bestehend aus einer Translation und Rotation im 3D-Raum, oder eine Zuweisung von Winkeln zu den Gelenken. Aus einer beliebigen Folge solcher Key-Frames kann dann eine gewollte Bewegung erzeugt werden, indem über die Zeit zwischen diesen interpoliert wird. Die Key-Frames können dabei vollständig statisch sein oder auch dynamisch, um geeigneter auf die gegebene Situation reagieren zu können. Für dynamische Umsetzungen sind Beispiele Yi u. a. [2013] und Wenk u. Röfer [2013], sowie das momentane Schießen aus dem Stand bei B-Human [vgl. Müller u. a., 2011]. Das aktuell verwendete Aufstehen bei B-Human ist eine klassische Umsetzung für statische Key-Frames [vgl. Reichenberg, 2018].

Das Verhalten, welches bestimmt, wann die Roboter wie agieren sollen, ist auch ein großes Thema. Üblicherweise werden Potentialfelder verwendet. Dies sind Richtungsfelder, welche mit einer gewissen Kraft von Punkten abstoßen oder zu diesen anziehen [vgl. Laue u. Röfer [2005], Strobel [2020], Laue [2004]]. Durch die Additionen einer gewichteten Menge solcher Felder können Entscheidungen getroffen werden, für zum Beispiel die Pfadplanung oder wohin der Ball gespielt werden soll. Auch werden Diagramme wie approximierte dominante Regionen Diagramme (Approximate Dominant Region Diagram) oder Voronoi Diagramme eingesetzt [vgl. Nakanishi u. a., 2008]. Komplexere Verhaltenssysteme wie in Dylla u. a. [2008] vorgestellt, greifen auf diese zurück, um größere Strategien umzusetzen.

Andere Ansätze, die sich in dem Bereich Machine Learning wiederfinden, verwenden Reinforcement Learning [vgl. Neri u. a. [2012], Riedmiller u. a. [2009]]. Hier werden Verhaltensabschnitte gelernt und zeigen bessere Ergebnisse gegenüber den vorherigen von Menschen geschriebenen Algorithmen.

Kapitel 4

Grundlagen

4.1 Mathematische Definitionen

Um Berechnungen und Formeln in dieser Arbeit zu vereinfachen, werden vorweg Definitionen und Schreibweisen eingeführt, die sich in Teilen an Röhrig [2018] orientieren. Jene Arbeit stammt ebenfalls aus dem Kontext des RoboCups und wurde beim Team B-Human geschrieben, sodass ähnliches mathematisches Grundwissen erforderlich ist.

In dieser Arbeit werden häufig Positionen und Orientierungen verwendet, begrenzt auf den 2D- und 3D-Raum, welche als Posen zusammen dargestellt werden können. Eine Pose im n-dimensionalen Raum mit der Position $(p_1, p_2, ..., p_n)^T \in \mathbb{R}^n$ und der Rotationsmatrix $R \in \mathbb{R}^n$ als Orientierung sind zusammen eine Matrix der Größe (n+1)×(n+1).

$$M = \begin{pmatrix} R_{1,1} & R_{1,2} & \cdots & R_{1,n} & p_1 \\ R_{2,1} & R_{2,2} & \cdots & R_{2,n} & p_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ R_{n,1} & R_{n,2} & \cdots & R_{n,n} & p_n \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix} \tag{4.1}$$

$R_{i,j}$ ist hierbei der Eintrag in der i-ten Zeile und der j-ten Spalte.

Der Raum aller Posen im n-dimensionalen Raum wird als P^n bezeichnet.

Häufig wird nur ein spezifischer Eintrag wie die Position oder die Rotation einer bestimmten Achse benötigt. Hierfür werden mehrere Notationen definiert.

Für eine Pose M$\in P^n$ mit n $\in \mathbb{R}$ und n ≥ 2 gilt die Notation:

$$M^{(rot)} := \begin{pmatrix} R_{1,1} & \cdots & R_{1,n} \\ \vdots & \ddots & \vdots \\ R_{n,1} & \cdots & R_{n,n} \end{pmatrix} \tag{4.2}$$

$$M^{(trans)} := (p_1, p_2, ..., p_n)^T \tag{4.3}$$

Sei M $\in P^3$ so gelten folgende Notationen:

$$M^{(x)} := M_{1,n+1} \tag{4.4}$$

$$M^{(y)} := M_{2,n+1} \tag{4.5}$$

$$M^{(z)} := M_{3,n+1} \tag{4.6}$$

$$\tag{4.7}$$

Sei M $\in P^2$ so gelten folgende Notationen:

$$M^{(x)} := M_{1,n+1} \tag{4.8}$$

$$M^{(y)} := M_{2,n+1} \tag{4.9}$$

$$M^{(\alpha)} := \arccos(M_{1,1}) \tag{4.10}$$

Zusätzlich wird definiert, dass eine 2D-Pose mit der Rotation α und den Translationen x und y auch als Pose(α,x,y) geschrieben werden kann.

Um die Rotationen einer spezifischen Achse einer 3D-Pose zu bekommen, benötigt es eine komplexere Berechnung. Hierfür werden zuerst die Vorzeichenfunktion *signPos* und *signNeg* definiert:

$$\begin{aligned} \text{signPos(a)} &:= \begin{cases} -1 & , \ a < 0 \\ 1 & , \ a \geq 0 \end{cases} \\ \text{signNeg(a)} &:= \begin{cases} -1 & , \ a \leq 0 \\ 1 & , \ a > 0 \end{cases} \end{aligned} \tag{4.11}$$

Die einzelnen Rotationen für eine Matrix M $\in P^3$ können hiermit folgendermaßen beschrieben werden:

$$M^{(\alpha)} := \begin{cases} \arccos\left(\frac{M_{2,2}}{\sqrt{M_{1,2}^2+M_{2,2}^2}}\right) \cdot -\text{signNeg}(M_{1,2}) & , \text{ falls } \sqrt{M_{1,2}^2 + M_{2,2}^2} \neq 0 \\ 0 & , \text{ sonst} \end{cases} \tag{4.12}$$

$$M^{(\beta)} := \begin{cases} \arccos\left(\frac{M_{0,0}}{\sqrt{M_{0,0}^2+M_{2,0}^2}}\right) \cdot -\text{signNeg}(M_{2,0}) & , \text{ falls } \sqrt{M_{0,0}^2 + M_{2,0}^2} \neq 0 \\ 0 & , \text{ sonst} \end{cases} \tag{4.13}$$

$$M^{(\gamma)} := \begin{cases} \arccos\left(\frac{M_{0,0}}{\sqrt{M_{0,0}^2+M_{1,0}^2}}\right) \cdot \text{signPos}(M_{1,0}) & , \text{ falls } \sqrt{M_{0,0}^2 + M_{1,0}^2} \neq 0 \\ 0 & , \text{ sonst} \end{cases} \tag{4.14}$$

$\mathrm{M}^{(\alpha)}$ entspricht dabei der aktuellen Rotation um die x-Achse, $\mathrm{M}^{(\beta)}$ um die y-Achse und $\mathrm{M}^{(\gamma)}$ um die z-Achse.

Um eine Pose im 2D-Raum zu rotieren, kann die anzuwendende Rotation einfach auf den Winkel addiert werden. Gegeben seien $\mathrm{M} \in P^2$ und eine Rotation $\delta \in \mathbb{R}$, dann gilt für die Rotation der Matrix M:

$$M^{(\alpha)} = M^{(\alpha)} + \delta \tag{4.15}$$

Um eine Pose im 3D-Raum zu rotieren, wird die Rotationsmatrix für die jeweilige Achse mit der Pose multipliziert. Hierfür gelten die folgenden Rotationsmatrizen, gegeben einem Winkel δ:

$$\text{Rot}_x(\delta) := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\delta & -\sin\delta & 0 \\ 0 & \sin\delta & \cos\delta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.16}$$

$$\text{Rot}_y(\delta) := \begin{pmatrix} \cos\delta & 0 & \sin\delta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\delta & 0 & \cos\delta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.17}$$

$$\text{Rot}_z(\delta) := \begin{pmatrix} \cos\delta & -\sin\delta & 0 & 0 \\ \sin\delta & \cos\delta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.18}$$

Die Rotation in einer Achse a einer Pose $\mathrm{M} \in P^3$ erfolgt anschließend durch die linksseitige Multiplikation:

$$M = \text{Rot}_a(\delta) \cdot M \tag{4.19}$$

In dieser Arbeit finden auch 2D-Vektoren Anwendung. Die Notation für den Zugriff auf die x- respektive y-Komponente ist analog wie für Matrizen definiert. Vektoren besitzen unter anderem eine Richtung, welche auch als Winkel zur x-Achse angegeben werden kann. Gegeben einem Vektor v $\in \mathbb{R}^2$ lässt sich dieser Winkel wie folgt berechnen:

$$v^{(\alpha)} := \text{atan2}(v^{(y)}, v^{(x)}) \tag{4.20}$$

4.2 Koordinatensysteme

Wie auch in der Arbeit von Röhrig [2018] existiert im B-Human-Framework kein einheitliches Koordinatensystem, sondern mehrere, zwischen denen je nach Kontext gewechselt wird. Jede Pose liegt daher relativ zu einem spezifischen Koordinatensystem, welche wiederum relativ zum Ursprung des Robotertorsos durch einen Bezugspunkt und einer Orientierung definiert werden kann. Eine Pose kann daher folgendermaßen beschrieben werden:

$$\text{Gegeben sei } n \in \{2, 3\}; A, B \in P^n \tag{4.21}$$

$$\text{A}_\text{B} \text{ ist die Pose A im Koordinatensystem B.} \tag{4.22}$$

Das Wechseln zweier Koordinatensysteme ist durch die linksseitige Multiplikation definiert:

$$\text{Sei } n \in \{2, 3\}; A, B, C \in P^n \tag{4.23}$$

$$A_B = C_B \cdot A_C \tag{4.24}$$

Mit dem Inversen einer Pose kann die Rückrichtung berechnet werden. Diese ist folgendermaßen definiert:

$$A_B{}^{-1} = B_A = \begin{pmatrix} A_{B1,1} & \cdots & A_{Bn,1} & \hat{p}_1 \\ \vdots & \ddots & \vdots & \vdots \\ A_{B1,n} & \cdots & A_{Bn,n} & \hat{p}_n \\ 0 & \cdots & 0 & 1 \end{pmatrix} \tag{4.25}$$

mit

$$A_B{}^{(rot)-1} = B_A{}^{(rot)} \tag{4.26}$$

$$(\hat{p}_1, ..., \hat{p}_n)^T := A_B{}^{(rot)-1} \cdot -A_B{}^{(trans)} \tag{4.27}$$

Die für diese Arbeit wichtigen Koordinatensysteme sind in Abbildung 4.1 dargestellt. Für jedes Koordinatensystem gilt, dass die x-Achse nach vorne zeigt, die y-Achse zur Seite und die z-Achse, wenn vorhanden, nach oben.

Weltkoordinatensystem ω ist 2D und hat seinen Ursprung in der Mitte des Fußballfeldes, wobei die positive x-Richtung zum gegnerischen Tor und die y-Richtung nach links zeigen. In diesem System liegen alle Informationen zum Spielgeschehen vor, wie die eigene Position, Gegner oder der Ball. Ebenfalls werden in der Regel Verhaltensentscheidungen hier getroffen.

Torsokoordinatensystem τ ist 3D und hat seinen Ursprung im Torso des Roboters. Es wird für die Kinematik verwendet und ist unabhängig von der Lage des Roboters im Raum. Die meisten Bewegungsansteuerungen werden in diesem System berechnet, da unabhängig von der Wahrnehmung die Positionen der Gliedmaßen bestimmt und gesetzt werden können.

Roboterkoordinatensystem ρ ist 3D und liegt in der Mitte zwischen beiden Füßen, relativ zu deren Ursprüngen. Die Fußursprünge liegen im Knöchelgelenk projiziert auf die Fußsohle. ρ wird durch die IMU-Messungen mithilfe eines Kalman-Filters geschätzt. Dieses System ist genau dann parallel zum Boden ausgerichtet, wenn der Roboter eine perfekte Zustandsschätzung besitzt. Die Höhe des Ursprunges ist hierbei gleich der des Fußes, der näher am Boden ist. Dadurch ist die z-Achsenposition, welche die Höhenposition angibt, leicht falsch, sofern der Roboter nicht flach auf dem Boden und nicht gekippt steht. In diesem Koordinatensystem werden alle Positionen von Objekten modelliert. Alle Objekte aus dem Weltkoordinatensystem lassen sich somit hier wiederfinden. Dadurch besteht eine von der Lokalisation unabhängige Modellierung über alle Objekte.

Fußkoordinatensystem ϕ ist 3D und liegt im jeweiligen Ursprung der Füße. In Kombination mit dem Roboterkoordinatensystem werden in diesem die Laufschritte und Schüsse geplant, da so die relative Position des nächsten Schwingfußes zum Ball oder anderen Objekten berechnet werden kann.

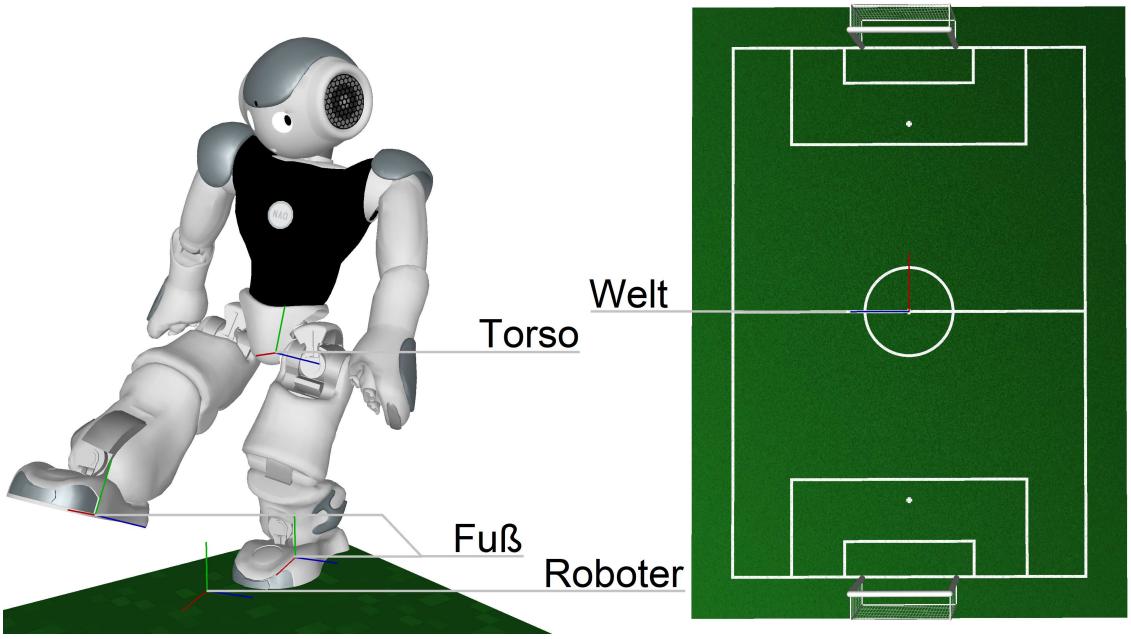

Abbildung 4.1 Die verschiedenen Koordinatensysteme beim Team B-Human.

4.3 Verhalten

Das aktuelle Verhalten der Roboter läuft im Cognition-Thread. Das Verhalten hat die Aufgabe, verschiedene *Repräsentationen* zu setzen, sodass im Motion-Thread auf sie zugegriffen werden kann und alle nötigen Informationen vorhanden sind, um das gewünschte Verhalten umzusetzen. Die Repräsentationen umfassen dabei die Anfragen an den Kopf (*HeadRequest*), die Arme (*ArmRequest*) und die Bewegung des Roboters (*MotionRequest*). Im *MotionRequest* stehen die Information über das angefragte Bewegungsverhalten, die Position und Bewegungsrichtung des Balls, eine Schussanfrage und Schussrichtung sowie eine Pfadvorgabe, um Hindernissen auszuweichen. Das Bewegungsverhalten ist zusätzlich in verschiedene Kategorien gegliedert:

Dribble: Der Roboter soll den Ball in eine vorgegebene Richtung dribbeln. Der Laufschritt gegen den Ball soll dabei möglichst schwach sein.
WalkToPose: Der Roboter soll zu einer vorgegebenen relativen Position laufen. Ob die Ausrichtung sofort oder auf dem Weg zum Ziel eingenommen werden soll, wird ebenfalls vorgegeben.
WalkToBallAndKick: Der Roboter soll an den Ball laufen und einen vorgegebenen Schuss mit einer gesetzten Stärke ausführen, um den Ball in die vorgegebene Richtung zu treten.
Stand: Der Roboter soll stehen bleiben.
GetUp: Der Roboter soll aufstehen. Je nachdem, ob der Roboter bereits steht, sitzt oder auf dem Boden liegt, werden unterschiedliche Bewegungen ausgeführt.
KeyFrameMotion: Der Roboter soll eine statische Bewegung ausführen. Dazu zählen zum Beispiel Bewegungen, um den Ball zu stoppen, indem sich der Roboter auf den Boden wirft.

Für die verschiedenen Kategorien existieren unter anderem jeweils Module, um das angefragte Verhalten umzusetzen. Hierbei sind die Module teilweise voneinander abhängig, wie es in Abbildung 4.2 zu sehen ist.

Das Verhalten im Cognition-Thread selbst ist in zwei Kategorien aufgeteilt. Diese sind im B-Human-Framework als *Cards*, sogenannte Karten, und *Skills*, also Fähigkeiten, bezeichnet. Die Cards haben die Aufgabe zu entscheiden, was der Roboter tun soll und die Skills, wie der Roboter dies umsetzt. Zusätzlich gibt es *Decks*, welche eine Ansammlung von Cards sind, realisiert durch eine Konfigurationsdatei. Je nach Spielsituation wird ein unterschiedliches Deck verwendet. Eine Spielsituation kann hierbei ein Anstoß, Freistoß oder das normale Spielgeschehen sein.

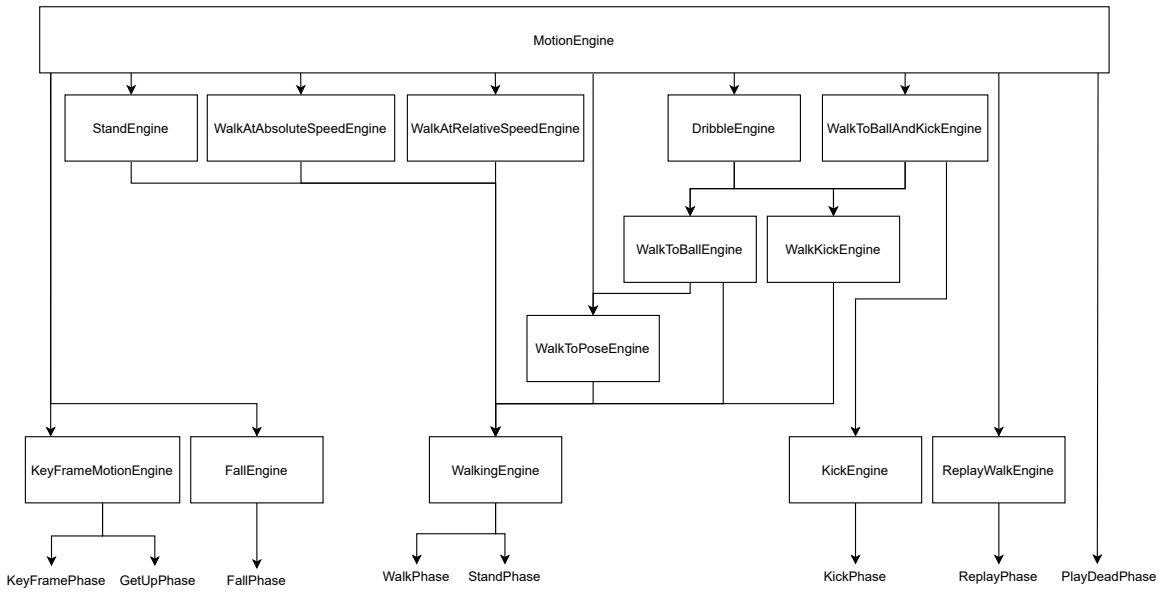

Abbildung 4.2 Die Aufrufhierachie zum Erzeugen einer neuen Phase. Die Engines sind Module, welche die nötigen Schnittstellen in den respektiven Repräsentationen bereitstellen. Die MotionEngine stellt dabei die Repräsentation für die Gelenkanforderung, welche an den NAO gesendet wird.

In jedem Cognition-Frame wird genau eine Card vollständig ausgeführt und muss alle drei Repräsentationen, also HeadRequest, ArmRequest und MotionRequest für den Motion-Thread setzen. Dies geschieht durch die Skills. Entscheidet zum Beispiel die aktuell aktive Card ZweikampfCard, dass der Schuss S in Richtung R ausgeführt werden soll, so ruft sie den Skill *WalkToBallAndKickSkill* auf. Dieser bekommt mehrere Parameter übergeben. Dazu zählen der Schuss, die Richtung, die Schussstärke und zusätzliche optionale Parameter. Diese sind zum Beispiel, wie an den Ball angelaufen werden soll, ob Hindernissen ausgewichen werden muss oder zusätzliche Bedingungen an den Schuss. Eine solche Bedingung wäre, ob der Roboter sich möglichst genau vor dem Ball ausrichten soll, um eine höhere Schussgenauigkeit zu erreichen.

4.4 Potentialfeld

Ein Potentialfeld ist ein spezielles Vektorfeld. Ist v ein Vektorfeld, und es gibt ein differenzierbares Skalarfeld α für das gilt

$$\text{grad } \alpha = v \tag{4.28}$$

dann ist α ein Potential. Wenn es zu einem Vektorfeld v ein Potential α gibt, so ist v ein Potentialfeld.

4.5 Schwerpunkt

Nach Bräunl [2006] kann mit dem Schwerpunkt die Stabilität eines bewegungslosen Roboters gut beschrieben werden. Die einzige Kraft, die den Roboter zum Fall bringen kann, ist die Gravitationskraft. Diese wirkt sich auf den gesamten Körper aus und kann zusammengefasst durch den Schwerpunkt beschrieben werden. Ist der Schwerpunkt innerhalb der Stützfläche, definiert durch die konvexe Hülle über alle Punkte des Roboters, die den Boden berühren, so ist dieser stabil und kann nicht umfallen.

Der Schwerpunkt selber kann durch die Summe der Positionen aller Gliedmaßen, gewichtet durch deren Massen, definiert werden. Gegeben seien $m, p_g \in \mathbb{R}^3; w_g \in \mathbb{R}$, dann gilt für den Schwerpunkt m:

$$m = \frac{\sum_{g \in G} p_g \cdot w_g}{\sum_{g \in G} w_g} \tag{4.29}$$

$$\tag{4.30}$$

G ist hierbei die Menge aller Gliedmaßen, p_g die jeweilige Position und w_g das jeweilige Gewicht dieser. Die Gewichte und Positionen sind aus der Dokumentation des NAOs bekannt und lediglich die Positionen verschieben sich durch die Bewegung der Gelenke zusätzlich, welche durch die Vorwärtskinematik und der Sensordaten berechnet werden.

Bewegt sich nun der Roboter, wirken sich weitere Kräfte auf ihn aus. Je stärker diese werden, desto mehr verliert der Schwerpunkt an Aussagekraft über die Stabilität. So kann es vorkommen, dass der Schwerpunkt zwar noch innerhalb der Stützfläche liegt, durch die einwirkenden Kräfte sich dieser aber unweigerlich aus dieser bewegen und der Roboter somit umfallen wird. Der Schwerpunkt ist daher nur begrenzt geeignet, um über die Stabilität eines bewegenden Roboters auszusagen.

4.6 Zero Moment Point

Neben dem Schwerpunkt kann auch der Zero Moment Point, auch ZMP genannt, verwendet werden, wie dies auch in Böckmann [2015] der Fall ist. So gilt für einen sich bewegenden Roboter, wenn sein ZMP innerhalb der Stützfläche liegt, dass dieser stabil ist. Der ZMP ist daher wie der Schwerpunkt, nur erweitert durch dessen dynamischen Bewegungen [vgl. Dekker, 2009].

Wenn der Schwerpunkt durch das LIPM modelliert wird, gilt nach Kajita u. a. [2003]:

$$m_{\text{zmp}} = x - \frac{h}{g} \cdot \ddot{x} \tag{4.31}$$

$$x, \ddot{x} \in \mathbb{R}^2; h, g \in \mathbb{R} \tag{4.32}$$

Dabei ist x die Position des Schwerpunktes und $\ddot{x}$ dessen Beschleunigung, h die Höhe des Schwerpunktes über dem Boden und g die Erdanziehung von 9.81 m/s^2.

4.7 Motorverzögerung

In der Arbeit von Böckmann [2015] wurden die Motorenverzögerungen der Roboter NAO V5 untersucht. Hier wurden verschiedene Gelenkpositionen angesteuert und darauf folgende Bewegungen gemessen. Dabei kam eine Verzögerung von 30 ms heraus. Da der NAO V5 mit einer Taktrate von 100 Hz läuft, entspricht dies 3 Motion-Frames.

In dieser Arbeit wird primär mit dem Nachfolgemodell NAO V6 gearbeitet, welches mit einer Taktrate von 83.33 Hz arbeitet, um die Gelenke anzusteuern. Es ist wichtig zu wissen, ob die Verzögerung weiterhin gleich bleibt. Dafür wurden exemplarisch sechs Beingelenke aus einer ruhenden Position angesteuert, um sie jeweils zweimal zu bewegen. Die Ansteuerungen sind dabei in Abbildung 4.3 zu sehen. Die Verzögerungen liegen zwischen drei und vier Motion-Frames, was 36 bis 48 ms entspricht. Bei den Verzögerungen von 48 ms herrschten leichte Abweichungen für die Gelenke zu den angeforderten Startpositionen. Dadurch besaßen wahrscheinlich die ersten angesteuerten Positionsänderungen eine zu geringe Differenz zu den gemessenen Positionen, weshalb die Motoren nicht reagiert haben. Somit wird für die NAO V6 Roboter eine Motorverzögerung von 36 ms, also weiterhin drei Durchläufe, angenommen.

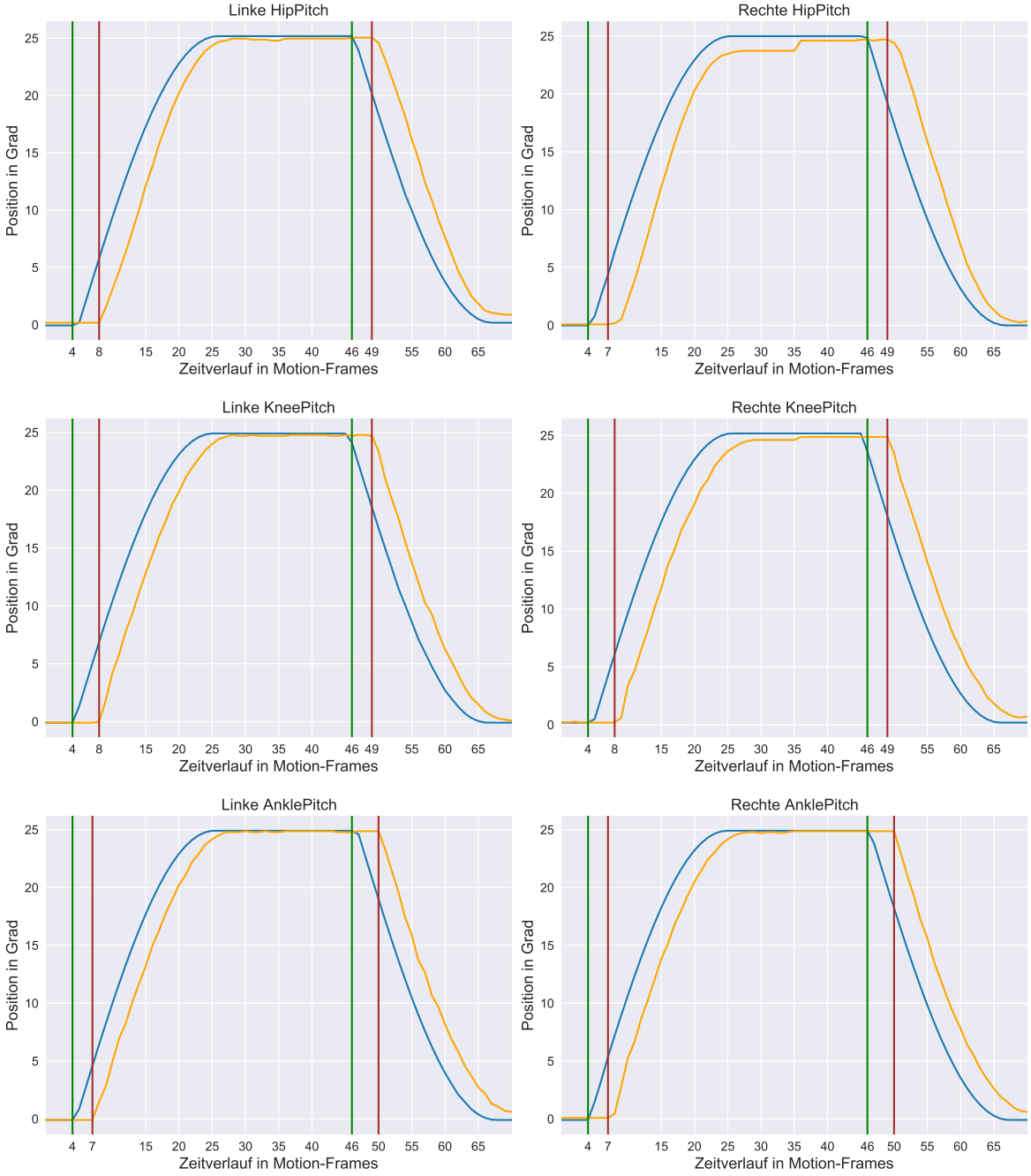

Abbildung 4.3 Dargestellt sind die Motorreaktionen verschiedener Beingelenke des NAO V6. Die sechs Gelenke wurden jeweils einzeln aus einer ruhenden Position über einen kurzen Zeitraum um 25 Grad verschoben, zwei Sekunden ruhen gelassen und wieder auf die ursprüngliche Position angesteuert. Die zwei Sekunden Ruhepause wurde großteils aus den Graphen entfernt, wodurch leichte Sprünge zu sehen sind, welche so nicht geschehen sind. Der NAO wurde während des Experimentes in der Luft gehalten. Zu sehen sind in blau die angesteuerte Position, in orange die gemessene Position, in grün jeweils der Zeitpunkt der ersten Änderung der Ansteuerung, in rot die erste gemessene Änderung.

Kapitel 5

Laufstabilität

Das bisherige Laufen wird seit 2017 verwendet und seitdem weiterentwickelt. Es basiert auf dem vom Team rUNSWift aus dem Jahre 2014 [vgl. Hengst, 2014]. Da im Kontext dieser Arbeit auf das aktuelle Laufen des Teams B-Human aufgebaut wird, welches eine abgewandelte Form des Laufens aus dem Jahre 2019 ist, sollte es weiterhin ähnliche Schwächen haben wie auf den letzten Wettbewerben.

Das Team rUNSWift, ebenso wie das Team Nao-Team HTWK, verwenden weiterhin ein sehr ähnliches Laufen. Gleichzeitig gehören alle drei Teams, B-Human eingeschlossen, zu den drei Top-Teams des letzten RoboCup Wettbewerbes von 2019.

5.1 Das aktuelle Laufen

Das aktuelle Laufen erlaubt omnidirektionale Bewegungsrichtungen. Es kann also sowohl gleichzeitig vorwärts und seitwärts gelaufen werden als auch gedreht, begrenzt durch die physikalische Größe des Roboters.

Ein einzelner Laufschritt ist dabei durch mehrere Variablen definiert. Eine Fußposition wird beschrieben durch drei Variablen für die Translationen im 3D-Raum und einer für die Rotation der z-Achse. Rotationen für die x- und y-Achse werden nicht betrachtet und nehmen somit immer den Wert null ein. Die Füße sind daher parallel zum gedachten Boden. Ein Laufschritt selber besteht ebenfalls aus drei Variablen. Dies sind die angefragten Translationen nach vorne und zur Seite sowie die Rotation. Die Vorwärts- und Rotationsgrößen werden jeweils zur Hälfte auf beide Füße übertragen und die Seitwärtsgrößen vollständig, interpoliert über eine Laufschrittdauer. Dadurch bleibt in der Theorie der Schwerpunkt immer genau zwischen beiden Füßen.

Für die Balancierung werden zwei Messungen einbezogen. Über die Fußdrucksensoren starten neue Laufschritte, jeweils passend zu einem Stand- und Schwingfußwechsel. Denn während eines Laufschrittes dient ein Fuß als Standfuß, der das Gewicht des Roboters trägt und einer als Schwingfuß, welcher keinen Bodenkontakt besitzt. Misst nun der Schwingfuß mehr

Gewicht als der Standfuß, kommt es zum Schrittwechsel und ein neuer Laufschritt beginnt. Zusätzlich wird die Gyrometermessung der y-Achse, also die Neigung nach vorne und hinten, über einen Tiefpass-Filter gefiltert, mit einem Faktor multipliziert und auf das Fußgelenk des Standfußes addiert. Kippt der Roboter nach vorne, drückt sich die Fußsohle somit gegen den Boden, respektive wenn der Roboter nach hinten kippt, zieht sie sich zusammen. Diese beiden Verfahren stabilisieren das Laufen genügend, sodass die Roboter unter normalen Bedingungen, wie über ein Spielfeld zu laufen, nicht umfallen.

5.2 Analyse zwischen German Open und RoboCup 2019

Aktuell laufen die Roboter bei B-Human mit einer Vorwärtsgeschwindigkeit von 250 mm/s und einer Seitwärtsgeschwindigkeit von 200 mm/s, welche bei einer zunehmenden Rotation linear reduziert werden. Die Reduktion basiert auf nachfolgender Formel.

$$\text{Ratio} = \max\left(1 - \frac{\text{AngefragteRotation}}{\text{KeineTranslationBeiRotation}}, 0\right)$$

Die Variable *AngefragteRotation* entspricht der geplanten Rotation des Laufschrittes und *KeineTranslationBeiRotation* der Rotation, bei der keine Translation mehr erlaubt ist. Die angefragte Translation wird mit der Variablen *Ratio* anschließend multipliziert, welche zwischen 0 und 1 liegt. Idealerweise möchte man auf diese Reduzierung verzichten. Ein Verzicht auf diese Translationsreduzierung käme aber einem Laufen gleich, welches zu beliebigen Zeitpunkten die Laufrichtung ändern und dabei beliebig schnell beschleunigen kann.

Es reicht aber in der Theorie bereits aus, wenn die Reduktion nicht bei einer Rotation von 0 anfängt, sondern erst bei einem höheren Wert oder anderweitig weniger Einfluss nimmt. Vergleicht man mit dem Team rUNSWift, so fällt auf, dass deren Roboter ähnliche Vorwärts- und Seitwärtsgeschwindigkeiten besitzen, zwischen 250 mm/s und 300 mm/s für die Vorwärtsgeschwindigkeit respektive 200mm/s für die Seitwärtsgeschwindigkeit. Jedoch verwenden sie eine andere Reduzierung basierend auf der Rotation mit mehreren Schritten. Hier wird zuerst auch ein Wert *Ratio* bestimmt:

$$\text{Ratio} = \sqrt{\left(\frac{\text{Vorwärtsgeschw.}}{\text{MaxVorwärtsgeschw.}}\right)^2 + \left(\frac{\text{Seitwärtsgeschw.}}{\text{MaxSeitwärtsgeschw.}}\right)^2 + \left(\frac{\text{Rotationsgeschw.}}{\text{MaxRotationsgeschw.}}\right)^2}$$

Anschließend wenden sie eine Binärsuche an, bei der alle drei Geschwindigkeiten reduziert werden, bis die Berechnung von *Ratio* nicht mehr einen Wert größer 1 ergibt. Die Binärsuche erfolgt dabei wie folgt: Zuerst werden alle Geschwindigkeiten halbiert und *Ratio* bestimmt. Ist *Ratio* weiterhin größer 1, werden die Geschwindigkeiten erneut um die Hälfte der vorherigen Anpassung halbiert. Diese war im ersten Schritt 50 % und ist hier dann somit 25 %. Ist *Ratio*

kleiner 1, werden die Geschwindigkeiten um die Hälfte der letzten Reduzierung wiederum erhöht. Wurden sie also vorher von 100 % auf 50 % reduziert und sollen wieder erhöht werden, so wird als nächstes 75 % der ursprünglichen Geschwindigkeit geprüft. Dieser Vorgang wird bis zu zehnmal wiederholt, bis *Ratio* maximal groß, aber nicht größer als 1 ist.

Im Vergleich dazu hat B-Human noch bis zum RoboCup 2019 eine leicht komplexere Reduktion, basierend auf der von rUNSWift aus dem Jahr 2016 [vgl. rUNSWift, 2016], verwendet. Zuerst wurde basierend auf den Geschwindigkeiten der Wert *Ratio* wie folgt berechnet:

$$\text{Ratio} = \left(\left(\frac{\text{Vorwärtsgeschw.}}{\text{MaxVorwärtsgeschw.}}\right)^{1.5} + \left(\frac{\text{Seitwärtsgeschw.}}{\text{MaxSeitwärtsgeschw.}}\right)^{1.5}\right)^{1.333} + \left(\frac{\text{Rotationsgeschw.}}{\text{MaxRotationsgeschw.}}\right)^{2}$$

MaxRotationsgeschw. ist hierbei eine konfigurierte Begrenzung, ab welcher Rotationsgeschwindigkeit die Division 1 ergeben soll. Hierfür wurde auf den German Open 2019 die maximale Rotationsgeschwindigkeit verwendet, zum RoboCup 2019 aber auf den Wert 70 Grad reduziert. Anschließend wurde mit derselben Binärsuche die Vorwärts- und Seitwärtsgeschwindigkeit reduziert. Die Rotationsgeschwindigkeit wurde somit immer vollständig umgesetzt. Dadurch haben sich unter anderem die Vorwärts- und Seitwärtsgeschwindigkeit gegenseitig reduziert, bei rUNSWift hingegen alle drei Geschwindigkeiten. Bei der aktuell verwendeten Reduktion von B-Human ist dies nicht mehr der Fall, wodurch größere diagonale Schritte umgesetzt werden können.

Nao-Team HTWK haben ihre Reduktion leider nicht veröffentlicht, wodurch hier kein Vergleich möglich ist. Zusätzlich ist eine alternative Berechnung für die aktuell von B-Human verwendete Reduktion angegeben, welche den Reduktionsfaktor quadriert, und somit die Formel folgendermaßen modifiziert:

$$\text{Ratio} = \max\left(1 - \left(\frac{\text{AngefragteRotation}}{\text{KeineTranslationBeiRotation}}\right)^{2}, 0\right) \tag{5.1}$$

In Abbildung 5.1 ist für alle fünf Reduktionen die prozentuale maximale Vorwärtsgeschwindigkeit dargestellt, abhängig von der angefragten absoluten Rotationsgeschwindigkeit.

Auffällig ist, dass B-Human im Vergleich zu rUNSWift auf dem RoboCup 2019 die Translationen weniger reduzierte, diese auf den German Open aber noch gleich waren. Es muss dazu erwähnt werden, dass unbekannt ist, inwiefern rUNSWift mit dieser Reduktion den Wettbewerb bestritt. Üblicherweise nehmen die Teams über die Dauer des Wettbewerbes mehrere Änderungen vor. Die veröffentlichte Version kann dabei nur die letzte, zu Teilen aufgeräumte Version sein. Dennoch muss festgehalten werden, dass das aktuelle Laufen von B-Human deutlich stärker die Geschwindigkeiten, basierend auf der Rotationsgeschwindigkeit, reduziert als auf dem letzten Wettbewerb.

Abbildung 5.1 Die maximal mögliche Vorwärtsgeschwindigkeit in % relativ zur angesteuerten Rotationsgeschwindigkeit pro Sekunde, für die verschiedenen Ansätze. GO steht für German Open und RC stehend für RoboCup.

Um die Auswirkungen auf die Geschwindigkeiten zu ermitteln, wird die durchschnittliche angesteuerte Rotationsgeschwindigkeit eines Laufschrittes benötigt. Dafür wurde ein Roboter an einer Stelle auf ein Kunstrasenfeld gestellt und geradeaus laufen gelassen. Seine Orientierung sollte dabei beibehalten werden, welche sich durch die Lokalisierung dauerhaft ändern kann. In Abbildung 5.2 ist die angefragte Rotation für jeden Laufschritt auf einer Strecke von drei Metern dargestellt. Anhand dieser Aufzeichnung lässt sich ein signifikantes Problem erkennen. Durch die durchschnittliche Rotationsanpassung von 11.1 Grad/s in jedem Laufschritt ist das aktuelle Laufen von B-Human langsamer als es sein müsste. Nach der Reduktionsformel ergibt sich eine Verlangsamung von $\frac{11.1°/s}{120°/s} \approx 0.0925$, also etwa 9.25 %. Dies ist dadurch bedingt, dass der Roboter dauerhaft seinen Laufweg korrigiert. So dreht er sich abwechselnd nach links und rechts und kann deshalb nicht konstant mit der maximalen Laufgeschwindigkeit laufen. Nach der alten Reduktion von 2019 von B-Human entspräche dies einer Verlangsamung von etwa 1.02 %, für rUNSWift etwa 1.81 %. Die alternative Reduktion würde zu einer Verlangsamung von etwa 1.03 % führen.

Es ist daher ein signifikanter Vorteil zum aktuellen Laufen, wenn die alternative Reduktion verwendet werden würde, um keine größere Reduktion zu besitzen als noch zum letzten RoboCup Wettbewerb. Im Vergleich zu den anderen Teams wäre es auch ein Vorteil, wenn die Roboter mit deutlich höheren Vorwärtsgeschwindigkeiten laufen könnten. Eine Vorwärtsgeschwindigkeit von 300 mm/s würde bereits im Vergleich zu Robotern, die mit nur 250mm/s laufen, eine Beschleunigung von 20 % bedeuten. Im Vergleich zum letzten RoboCup 2019 würde auch die gegenseitige Reduktion der Vorwärts- und Seitwärtsgeschwindigkeit wegfallen, was zu einer weiteren allgemeinen Beschleunigung des Laufes führt. Diese Vorteile können aber nur effektiv genutzt werden, wenn die Roboter mit den Geschwindigkeiten nicht umfallen. Das Laufen muss folglich gegen externe Einflüsse wie Kollisionen mit anderen Robotern oder Unebenheiten im Feld resistenter werden.

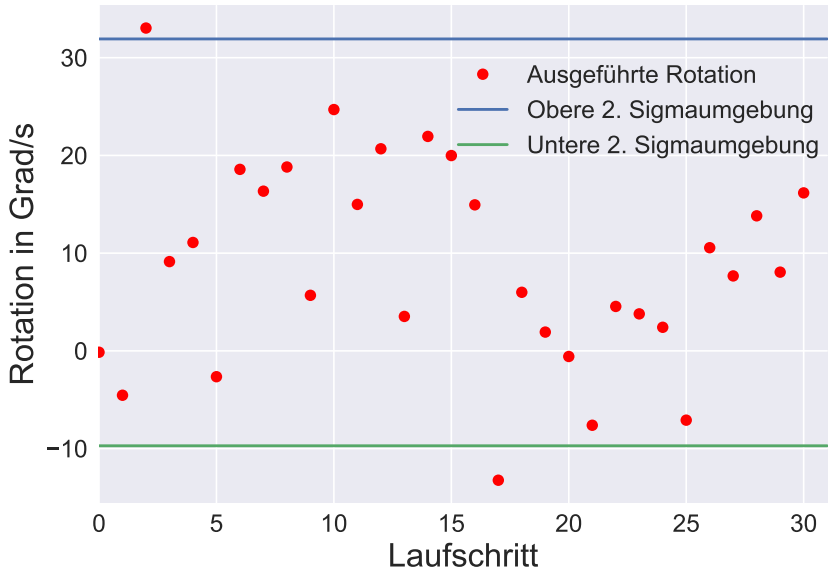

Abbildung 5.2 Die Rotationsanfragen für die Laufschritte (in rot) über drei Meter, angegeben in Grad pro Sekunde. In blau die obere 2. Standardabweichung und in grün die untere. Die zweite Standardabweichung beträgt 20.8 Grad/s. Die durchschnittliche Rotationsanfrage beträgt 11.1 Grad/s.

Wie bereits festgestellt, können die Roboter mit dem aktuellen Laufen und der Laufschrittgrößenregulierung von 2019 in der Regel ohne umzufallen laufen. Angesichts der höheren Umfallrate ohne Fremdeinflüsse auf den German Open 2019 scheint es gleichzeitig bei zu großen Laufschritten zu Problemen zu kommen. Es muss also untersucht werden, welche Effekte bei höheren Laufgeschwindigkeiten das Laufen destabilisieren und somit diese nicht wettbewerbsfähig machen.

5.3 Problemanalyse

Die Roboter fallen häufig in Gegenwart anderer Roboter um. Die einzige bisherige Balancierung, die das verhindern kann, ist eine Fußbalancierung, bei welcher die Gyrometermessungen durch einen Tiefpass-Filter gefiltert werden und mit einem Faktor auf das Standfußgelenk addiert werden, wie schon in Abschnitt 5.1 erwähnt. So drückt der Standfuß gegen den Boden, wenn der Roboter nach vorne kippt, und zieht ihn zusammen, wenn dieser nach hinten kippt. Dies könnte man theoretisch auf die Hüftgelenke übertragen, doch insbesondere beim Umfallen nach vorne ergibt sich ein schwer zu lösendes Problem. So können sich die Hüftmotoren nicht gegen das Robotergewicht bewegen, wenn dieser ein Momentum nach vorne besitzt, wie es in Abbildung 5.3 zu sehen ist. Hier wurde die Balancierung mit dem Standfuß auf das HipPitch-Gelenk übertragen und der Roboter beim auf der Stelle Laufen nach vorne geschubst. Die Abweichungen wurden hierbei zuvor durch die Differenz der gemessenen Gelenkposition und der Ansteuerung von vor drei Motion-Frames berechnet. Die kleinen Abweichungen von bis zu ±2.5 Grad entstehen, weil sich die Gelenke nicht beliebig schnell

bewegen können. Die großen Abweichungen von bis zu ± 10 Grad bei ungefähr dem 175. Motion-Frame kommen daher, dass der Roboter das HipPitch-Gelenk stark entgegen der Kipprichtung bewegen will, das Gelenk aber das gesamte Robotergewicht tragen muss, welches sehr weit nach vorne verschoben ist und sich mit einem Momentum nach vorne bewegt, entgegen der angesteuerten Bewegungsrichtung des HipPitch-Gelenkes. Die Motoren können nicht genügend Gegenkraft aufbringen, wodurch das Gelenk lediglich hängen bleibt und sich nicht bewegen kann.

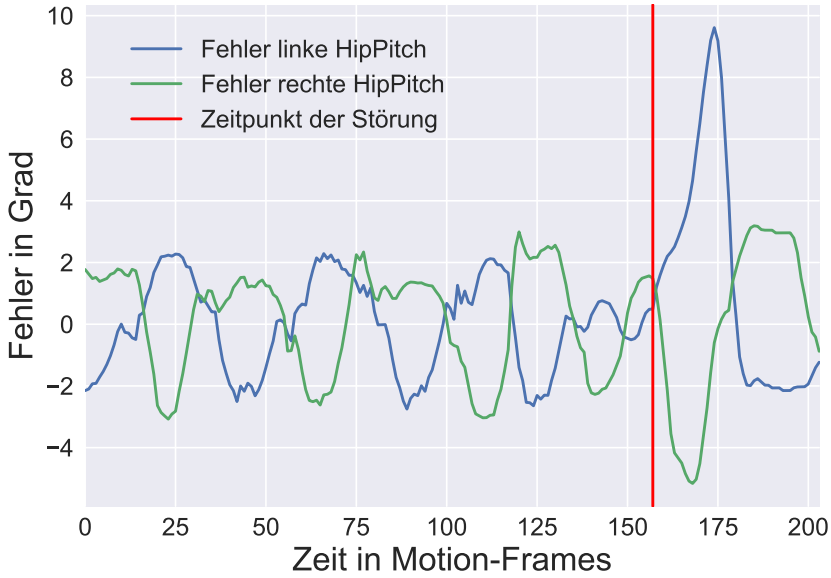

Abbildung 5.3 Ein Roboter lief auf der Stelle und wurde schwach nach vorne geschubst. Die AnklePitch-Balancierung wurde dabei auf das HipPitch-Gelenk übertragen. Dargestellt ist der Ansteuerungsfehler unter der Berücksichtigung der 36-48 ms Verzögerung.

Auch kann in einzelnen Fällen ein anderes Problem bezüglich der Gelenke beobachtet werden. In seltenen Fällen verhalten sich die Gelenke nicht wie erwartet und geben dem Roboter ein zusätzliches Kippmoment. Dies tritt primär bei Laufschritten auf, die zu lange dauern, zum Beispiel weil der Roboter zur Seite kippt, aber dadurch allein nicht umfallen würde. Hier steht der Roboter längere Zeit stabil auf einem Fuß und kippt irgendwann langsam nach vorne. Vom Standfuß bewegen sich hier nun das HipPitch- und KneePitch-Gelenk jeweils so, dass der Roboter zusätzlich nach vorne geneigt wird. Ein solcher Fall ist in Abbildung 5.4 dargestellt. Hier haben sich, seit dem Zeitpunkt, zu dem der Roboter anfing nach vorne zu kippen (schwarze Linie) bis hin zum nächsten Schrittwechsel (gestrichelte schwarze Linie), das HipPitch-Gelenk um 2.5 Grad zusammengezogen und das KneePitch-Gelenk um 3.9 Grad ausgestreckt. Diese 6.4 Grad extra Neigung nach vorne vom Standfuß geben dem Roboter zum einem zusätzlich mehr Drehmoment, gleichzeitig wird der Schwingfuß weniger weit nach vorne platziert, da dieser durch den Standfuß relativ zum Boden nach hinten gezogen wird. Die beiden Gelenke bewegten sich dabei nicht aufgrund reinem Gelenkspiel in diesem Ausmaße, da für diesen Roboter bekannt ist, dass das Gelenkspiel nur ein bis zwei Grad betrug.

Auch würde ein solches Gelenkspiel zu einem Laufen führen, bei dem der Roboter spätestens nach ein paar Laufschritten zwangsweise umfällt. Dieses Umsetzungsproblem der Gelenke ist daher höchstwahrscheinlich ein Regelungsproblem der Motoren. Die Umsetzung der Motoren kann nicht verändert werden und ist vom Hersteller vorgegeben. Zusätzlich ist auffällig, dass der Ansteuerungsfehler des AnklePitch-Gelenkes bis auf 5.2 Grad in dem Zeitraum zunimmt. Die Balancierung über die Gyrometermessungen lässt das AnklePitch-Gelenk gegen den Boden drücken. Jedoch kann das Gelenk nicht dem Robotergewicht entgegenwirken und bleibt dadurch hängen.

Eine Balancierung über einzelne Gelenke, um die Roboter vor einem Umfallen zu bewahren, müsste daher dieses Umsetzungsproblem berücksichtigen. Da den Motoren aber keine zusätzliche Kraft gegeben werden kann und eine Hardwaremodifikation ausgeschlossen ist, wäre eine solche Balancierung nicht erstrebenswert.

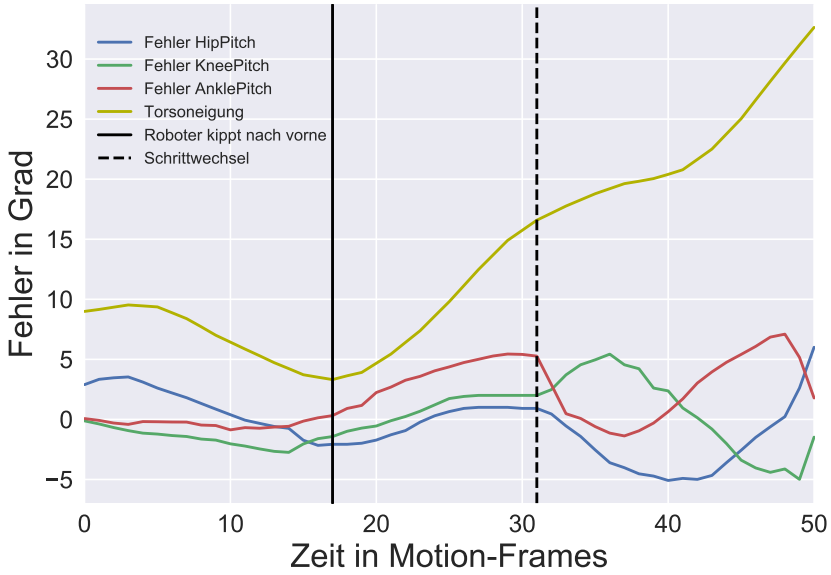

Abbildung 5.4 Ein Roboter lief seitlich und direkt danach geradeaus. Dabei dauerte ein Laufschritt deutlich länger als geplant. In diesem kippte der Roboter langsam nach vorne (schwarze Linie) und bis zum Schrittwechsel (schwarz gestrichelt) bewegten sich die HipPitch- und KneePitch-Gelenke entgegen der angesteuerten Positionen und ließen den Roboter zusätzlich nach vorne kippen. Die Torsoneigung (gelb) auch der Ansteuerungsfehler sind hierbei für die Gelenke HipPitch (blau), KneePitch (grün) und AnklePitch (rot) zu sehen. Am Ende fiel der Roboter nach vorne um.

Es wird daher angenommen, dass die aktuelle Balancierung mit den Fußgelenken ausreichend ist. Es muss daher andere, klar abgrenzbare Probleme geben, deren Behandlung zu einem geringeren Umfallrisiko der Roboter beitragen würde. Um diese Probleme zu identifizieren, werden zwei Experimente durchgeführt.

Im ersten Experiment soll ein Roboter auf eine erhöhte Holzfläche mit einer Höhe von 5 mm laufen. Dieser abrupte Höhenunterschied simuliert Stöße und Kollisionen von vorne, wodurch der Roboter eine Schräglage nach hinten bekommt.

Im zweiten Experiment soll ein Roboter von einer Erhöhung von 5 mm mit erhöhter Geschwindigkeit herunter laufen. Dies soll Stöße von hinten simulieren, wodurch der Roboter eine Schräglage nach vorne bekommt. Die erhöhte Geschwindigkeit soll verhindern, dass der Roboter die Erhöhung zu langsam herunterläuft und dadurch möglicherweise keine Auffälligkeiten aufweist.

Um die Experimente zu bewerten, wird unter anderem der Schwerpunkt relativ zur Stützfläche, bestehend aus beiden Füßen, betrachtet. In Kapitel 4 wurde der Schwerpunkt und der ZMP bereits vorgestellt. In einem sich bewegenden System ist der ZMP zwar die eindeutig bessere Wahl als Stabilitätskriterium. Der Schwerpunkt bietet aber eine sehr einfache Verwendung an, insbesondere für eine erste Untersuchung.

Für das erste Experiment wurden zusätzlich die Fußpositionen betrachtet. Das Resultat ist in Abbildung 5.5 dargestellt, bei dem der Roboter am Ende umfiel. Es tritt dabei ein offensichtliches Problem auf: Obwohl der laufende Roboter wissen könnte, dass der Schwerpunkt sehr weit hinten ist und sogar kurz vor Ende außerhalb der Stützfläche liegt, erkennbar durch die negativen Werte, läuft dieser weiter, als wenn nichts wäre. Dadurch liegt eine Vermutung sehr nahe. Solange der Roboter keinen starken Stößen ausgesetzt wird, durch welche er sehr schnell nach hinten kippt, hat er zum einem genügend Zeit, darauf zu reagieren, zum anderen kann er ein Umfallen sehr leicht verhindern, indem er einfach aufhört weiterzulaufen. Durch das Weiterlaufen verschiebt er seine Stützfläche, welche durch die Füße definiert ist, weiter nach vorne, während sich sein Schwerpunkt, projiziert auf den Boden, langsam nach hinten bewegt. Man kann also sagen: Er zieht sich die Füße vom Boden weg.

Beim zweiten Experiment wurden statt der Fußposition die Fußrotationen in der y-Achse, also nach vorne/hinten, untersucht. Dabei fällt ein Zusammenhang auf (siehe Abbildung 5.6). So gibt es ein Problem mit den Gelenken des Standfußes, ähnlich wie schon im Falle von Abbildung 5.3. Der Standfuß ist um mehrere Grad mehr geneigt als der Schwingfuß, sobald er anfängt nach vorne zu kippen. Dies ist in Abbildung 5.7 deutlich zu erkennen. Unter normalen Umständen ist der Standfuß kaum geneigt und der Schwingfuß bewegt sich fast parallel dazu. Im Experiment wiederum scheinen die Gelenke den Kräften nicht entgegenwirken zu können, wodurch der Standfuß zwar parallel zum Boden bleibt, den Roboter aber nach vorne kippen lässt. Da der Schwingfuß weiterhin parallel zum gedachten Boden bewegt wird jedoch nicht zum echten, bewegt sich dieser in den Boden hinein. Somit entsteht ein verfrühter Schrittwechsel, während der neue Standfuß nur mit der Spitze des Fußes Bodenkontakt besitzt und sich noch mit hoher Geschwindigkeit in einem Winkel in den Boden drückt. Bevor es zu einem weiteren Schrittwechsel kommen kann, fällt der Roboter bereits um.

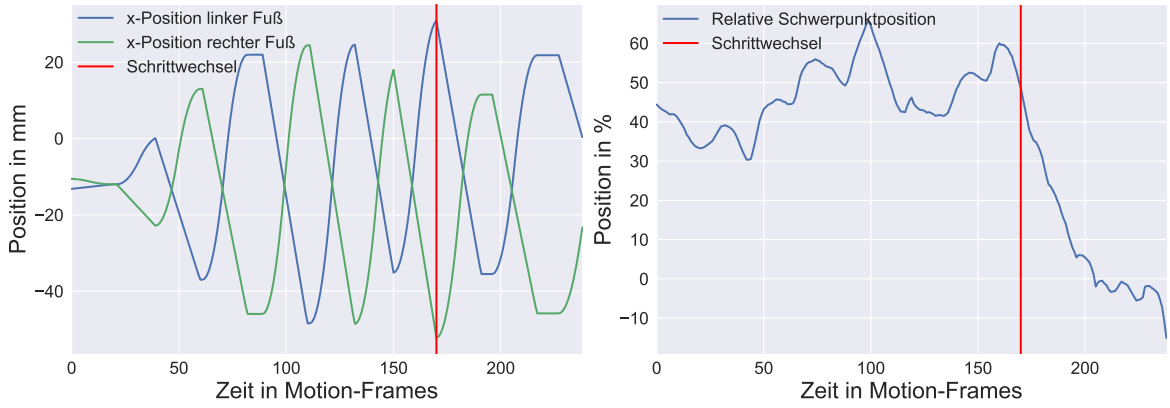

Abbildung 5.5 Die angesteuerten Fußpositionen (links) und die relative Schwerpunktposition in der Stützfläche (rechts), während der Roboter auf eine 5 mm Kante hoch läuft. Die rote Linie markiert den Laufschritt, nachdem ein Fuß auf der Kante abgesetzt wurde.

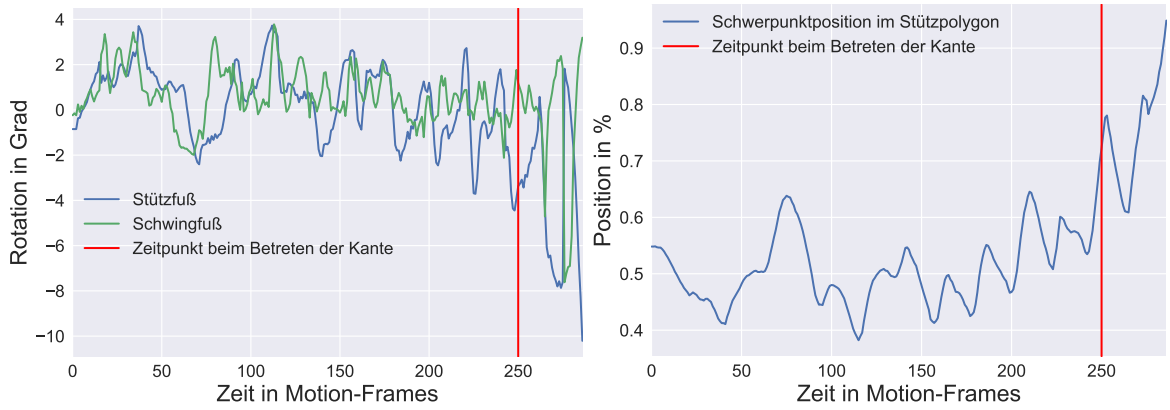

Abbildung 5.6 Die Fußrotationen beider Füße (links) der y-Achse und die Schwerpunktposition im Stützpolygon (rechts), während der Roboter über eine 5mm tiefe Kante herunterläuft. Die rote Linie markiert den Laufschritt, nachdem ein Fuß auf der Kante abgestellt wurde. Die rote Linie markiert den Laufschritt, nachdem ein Fuß auf der Kante abgesetzt wurde

Zusammengefasst kann man also vermuten, dass das Laufen deutlich resistenter gegenüber äußeren Einflüssen wäre, wenn der Roboter seine Laufschritte an die Bewegung seines Schwerpunktes anpassen und den Schwingfuß parallel zum Boden bewegen würde. Hierbei wurden aber nicht alle möglichen Probleme untersucht, die beim Laufen auftreten können. Aus Abschnitt 3.1 ist bereits bekannt, dass die Roboter primär nach hinten oder nach vorne umfallen. Es existieren natürlich auch die Fälle, in denen die Roboter seitlich umfallen. Ein Grund dafür könnte unter anderem ein gestörtes Schwingverhalten sein, wodurch die Laufschritte viel kürzer oder deutlich länger dauern als geplant. Da im Vergleich der anderen Top-Teams, die ein vergleichbares Laufen verwenden, das aktuelle Laufen bereits stabiler zu sein scheint, sollte eine Modifizierung am aktuellen minimal ausfallen. Ebenso laufen derzeit alle verwendeten Roboter mit den gleichen Konfigurationen. Eine Modifizierung sollte dies daher beibehalten.

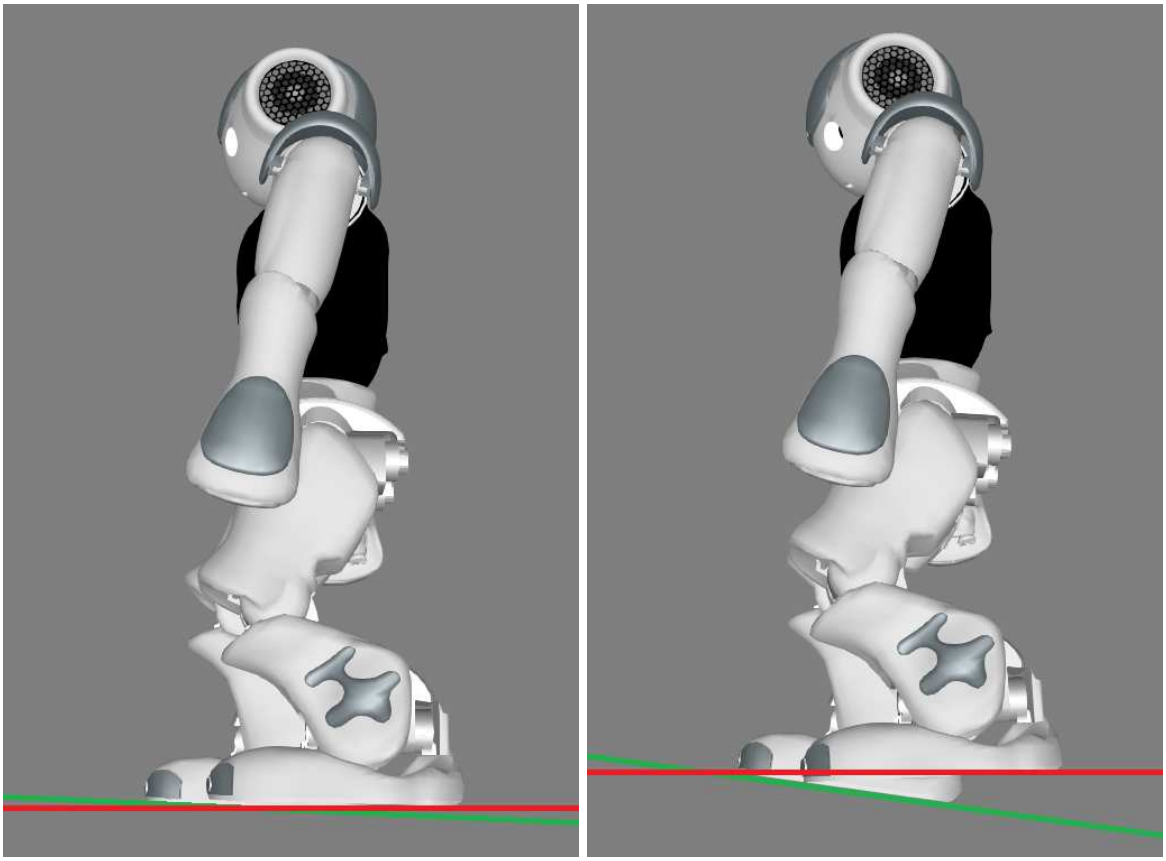

Abbildung 5.7 Die Fußrotationen des Roboters, während dieser läuft. Links sind die Füße unter normalen Umständen zu sehen; rechts, nachdem er über eine Kante gelaufen ist und nach vorne kippt. Der rechte, hintere Fuß ist der Standfuß, der linke, vordere der Schwingfuß. Grün markiert die Bodenebene, rot die Bewegungsrichtung des Schwingfußes.

5.4 Laufschrittanpassung

Damit die Roboter nicht weiterlaufen, während sie potentiell nach hinten umfallen, gibt es in der Theorie drei mögliche Ansätze.

Der erste Ansatz wäre eine Laufschrittgrößenberechnung, bei der der Roboter nur bestimmte Laufschritte ausführen kann, abhängig eines Modells über das Schwingverhalten des Roboters. Hier wird beispielsweise am Anfang eines Laufschrittes eine maximale Geschwindigkeit von 50-100 % festgelegt, da genau diese Laufschritte verhindern würden, dass der Schwerpunkt außerhalb des Stützbereiches gelangt. Eine solche Umsetzung kommt dem *Capture Step*-Framework von Missura u. a. [2019] nahe. Dort werden, basierend auf einem linearisierten invertierten Pendelmodells (LIPM), die Laufschrittgröße und -dauer am Anfang eines Laufschrittes bestimmt, damit der Schwerpunkt innerhalb eines stabilen Bereichs bleibt, gleichzeitig aber die angefragte Laufgeschwindigkeit umgesetzt wird. Dadurch läuft der Roboter stabil in die gewollte Richtung.

Die zweite Möglichkeit ist ein direktes Eingreifen in die Laufschrittumsetzung. Übliche Ansätze verwenden den ZMP, um die Gelenke anzupassen, wodurch die Roboter stabil laufen. Eine solche Umsetzung wird ebenfalls in Missura u. a. [2019] angewandt, sowie vom Team Nao Devils [vgl. Schwarz u. a., 2019]. Im Vergleich dazu verwendet rUNSWift einen PID-Regler, welcher als Regelgröße die Torsoneigung bekommt und die Stellgröße auf die HipPitch-Gelenke addiert [vgl. rUNSWift, 2019]. Für den Ansatz von rUNSWift wurde aber bereits schon festgestellt, dass dieser höchstwahrscheinlich keine Verbesserung bringen wird (siehe Abschnitt 5.3).

Eine Kombination aus beiden Varianten wäre ebenfalls denkbar. Der erste Ansatz soll im Kontext dieser Arbeit nicht verfolgt werden. Wie bereits erwähnt, laufen die Roboter ohne einen solchen Ansatz einwandfrei und versagen lediglich in Grenzfällen. Ebenfalls würde dieser Ansatz teilweise zu spät agieren. Denn eine Destabilisierung könnte direkt nach einem Schrittwechsel auftreten. Bevor es zu einem neuen Schrittwechsel kommt, können 250 ms vergehen, was einer geplanten Laufschrittdauer entspricht. Eine Restriktion der Laufschrittgröße danach könnte in vielen Fällen zu spät sein.

Daher wird der zweite Ansatz verfolgt. Durch ein direktes Eingreifen kann frühestmöglich interveniert werden, wenn es sein muss. Das aktuelle Laufen ist bereits sehr stabil und versagt nur in Grenzsituationen. Es ist daher wichtig, ein unnötiges Eingreifen zu verhindern, um nicht unter normalen Umständen instabiler zu laufen. Daher bedarf es einer Spezifizierung, welche bestimmt, unter welchen Umständen eingegriffen wird. Diese Entscheidung soll basierend auf dem Stützpolygon und dem Schwerpunkt geschehen.

5.4.1 Fußsohlenkalibrierung

Aus Erfahrung mit den Robotern liegt die Vermutung nahe, dass diese unterschiedlich gut vom Hersteller kalibriert sind. Somit könnten Berechnungen mit dem Schwerpunkt und dem Stützpolygon unterschiedliche Ergebnisse liefern, abhängig vom verwendeten Roboter.

Um diese Unterschiede zu verringern, bedarf es einer Kalibrierung. Fehlerquellen sind dabei die Gelenkpositionen, deren Messungen von den Echtwerten abweichen könnten. Ebenso ist unbekannt, inwiefern die IMU im Roboter korrekt verbaut ist. Die Schätzung der Lage des Torsos könnte somit vom Echtwert abweichen, selbst wenn der Roboter ruhig steht. Auch sind die Gewichtsangaben und deren Positionen der Extremitäten potentiell nicht vollständig korrekt.

Die Spezifikation der Füße und das daraus resultierende, vereinfachte Stützpolygon sind in Abbildung 5.8 zu sehen. Die Stützfläche hört dabei vor Ende der Hacke auf, da diese abgerundet ist und keinen direkten Bodenkontakt besitzt. Zwar stützt diese abgerundete Fläche den Roboter, sofern er stark nach hinten geneigt ist und auf den Hacken läuft, der Einfachheit halber wird dies aber nicht weiter betrachtet, um die Modellierung und Algorithmen möglichst simpel zu halten.

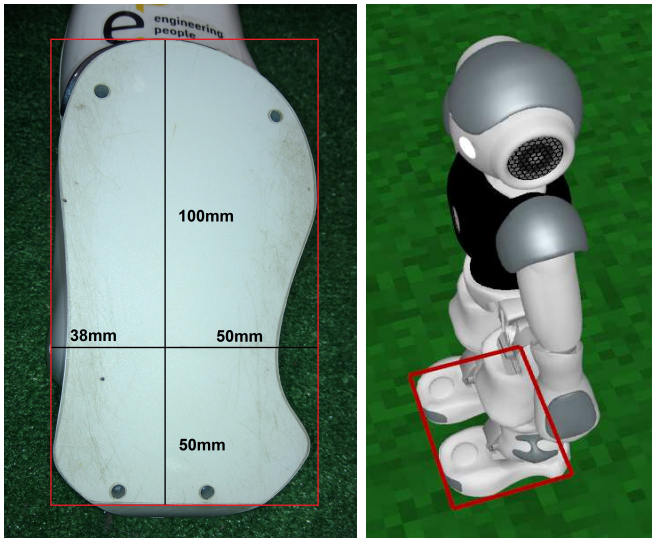

Abbildung 5.8 Die Spezifikation der Fußfläche, dargestellt durch den linken Fuß (links) und das daraus resultierende Stützpolygon (rechts). Die Fußfläche besitzt, ausgehend vom Fußursprung, die Maße 100 mm nach vorne, 50 mm nach hinten, 50 mm nach außen und 38 mm nach innen.

Um nun ein einheitliches Stützpolygon für alle Roboter zu bekommen, kann dieses insofern kalibriert werden, indem gemessen wird, welche Größen es besitzt. Dies kann durch einen einfachen Zusammenhang bestimmt werden. Im ruhenden, stehenden Zustand wird der Roboter nie umfallen, solange der Schwerpunkt innerhalb des Stützpolygons liegt. Sobald sich der Schwerpunkt nach außerhalb verlagert, wird der Roboter aber umfallen. Daraus lässt sich ein einfacher, automatischer Kalibrierungsprozess erstellen.

Der Roboter wird mit parallelen Füßen hingestellt. Um mögliche Messfehler durch die Sensoren zu minimieren, wird eine Stehhöhe von 230 mm verwendet, da die Roboter mit dieser Höhe auch laufen. Die Höhe ist vom Ursprung des Roboters gemessen. Danach wird die Hüfte parallel zum Boden nacheinander nach vorne, hinten, links und rechts verschoben. Die Verschiebung wird dabei mit einer langsamen Geschwindigkeit umgesetzt. Sobald die Gyrometermessungen hohe Ausschläge messen, wird die aktuelle Schwerpunktposition, projiziert auf die Fußebene, gespeichert und die Verschiebung pausiert. Sofern die Gyrometermessungen wieder nahe null zurückfallen, wird die Verschiebung fortgesetzt, ansonsten wird ein Umfallen angenommen.

Durch die Schwerpunktposition können exakte Positionen der Stützpolygonränder bestimmt werden und daraus kalibrierte Werte für die Fußspezifikation. Die inneren Fußkanten werden, da diese nicht durch dieses Verfahren gemessen werden, aus den gemessenen Werten berechnet. Da die Füße eine Breite von 88 mm besitzen, kann aus der Position der äußeren Kante die der inneren ermittelt werden.

Für den linken Fuß ergibt sich dadurch die Berechnung

$$\text{PositionInnereKante}^{(y)} = \text{PositionÄußereKante}^{(y)} - 88$$

respektive für den rechten Fuß

$$\text{PositionInnereKante}^{(y)} = \text{PositionÄußereKante}^{(y)} + 88$$

In Abbildung 5.9 sind nun die kalibrierten Stützpolygone der verfügbaren NAO V6 Roboter zu sehen. Es fällt auf, dass die Fußgrößen für jede Kante zwischen 10 und 20 mm variieren und für jeden Roboter zwischen 5 und 20 mm nach vorne verschoben sind. Was die genauen Ursachen für die Unterschiede sind, wird nicht weiter untersucht. Wie bereits erwähnt, können es Fehlkalibrierungen der IMU oder der Gelenkpositionsmessungen sein. Auch möglich sind Fehlangaben der Gewichte der einzelnen Körperteile.

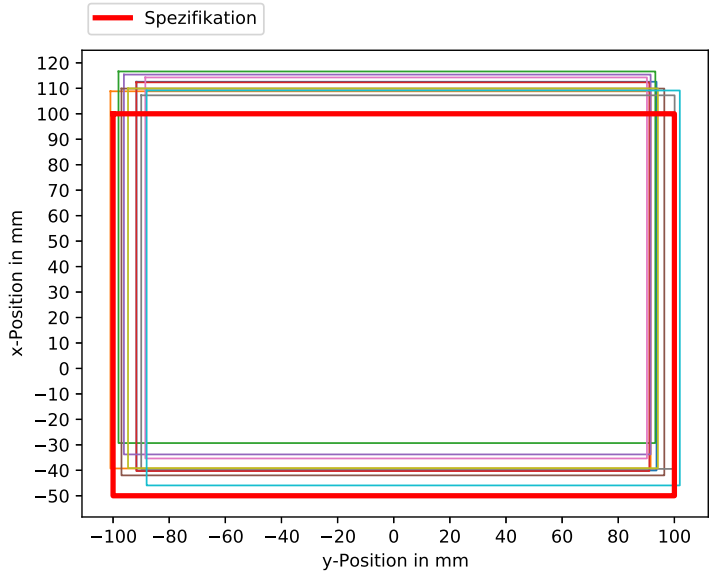

Abbildung 5.9 Die kalibrierten Stützpolygone, relativ zum Ursprung der Füße, von allen zehn NAO V6 Robotern vom Team B-Human, sowie das Polygon nach der Spezifikation (in rot).

5.4.2 Balancekriterium Schwerpunkt gegen ZMP

In dieser Arbeit wird der Schwerpunkt bewusst dem ZMP vorgezogen. Der Schwerpunkt bietet eine einfache Modellierung an, mit welcher das Laufen modifiziert werden kann. Der Schwerpunkt abstrahiert den Zustand des Roboters aber nur teilweise, da die Bewegungsrichtungen nicht durch diesen repräsentiert werden, wie es beim ZMP der Fall ist. Die Verwendung des ZMPs verkompliziert aber eine Laufstabilisierung. Denn bei auftretenden Problemen ist unbekannt, ob die Idee der Stabilisierung unzureichend ist oder andere Gründe das Problem sind. Ebenfalls ist eine korrekte Berechnung des ZMPs sehr aufwendig und würde den Rahmen dieser Arbeit sprengen. Bei der Verwendung des Schwerpunktes ist es hingegen einfacher

abzuschätzen, ob eine Stabilisierung von der Idee her nicht funktioniert oder die Modellierung unzureichend ist. Auch kann angenommen werden, dass der Schwerpunkt beim aktuellen Laufen ein ausreichendes Kriterium darstellt, da dieser zu fast jedem Zeitpunkt zum einem innerhalb des Stützpolygons liegt, zum anderen einen größeren Abstand zu den Rändern dieser besitzt, wie es später in Abbildung 5.11 dargestellt wird.

Betrachtet man den Stützbereich, dargestellt in Abbildung 5.10, so existieren drei Bereiche: der Rote, der Blaue und der Grüne. Durch das Bewegen der Füße verkleinert oder vergrößert sich nur der grüne Bereich. Vereinfacht angenommen fällt der Roboter fast nie um, wenn der Schwerpunkt innerhalb diesem liegt. Daher wird dieser im Folgenden ignoriert. Dadurch wird eine Vergleichbarkeit erreicht, unabhängig davon, wie die Füße positioniert sind, da nur die Schwerpunktposition relativ zu den Fußursprüngen betrachtet werden soll. Möchte man nun die Position des Schwerpunktes bestimmen, kann diese prozentual durch die Position im Stützbereich angegeben werden. Dabei steht für die x-Translation 0 % für den unteren Rand des blauen Bereiches und 100 % für den oberen Rand des roten Bereiches. x-Positionen dazwischen werden durch die Länge des blauen und roten Bereiches interpoliert. Für die y-Translation ist 0 % in der Mitte beider Füße, der linke Rand des linkes Fußes -100 % und der rechte Rand des rechten Fußes 100 %. Liegt die Position im grünen Bereich, wird diese ignoriert.

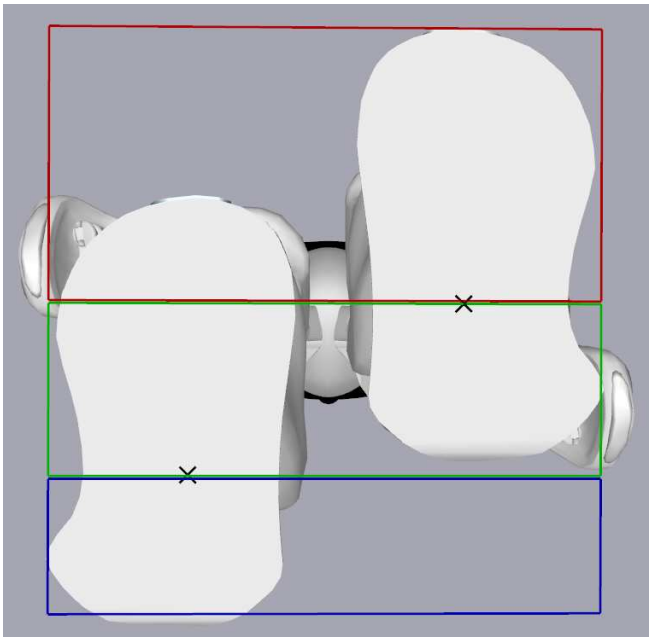

Abbildung 5.10 Die drei Stützbereiche, über die eine vergleichbare relative Schwerpunktposition bestimmt werden kann. Der blaue Bereich liegt hinter dem Ursprung des hinteren Fußes bis hin zu dessen Hacke. Der Bereich umfasst für die x-Translation den Wertebereich von 0 bis 33 %. Der rote Bereich liegt vor dem Ursprung des vorderen Fußes bis zur Fußspitze. Dieser umfasst den Wertebereich von 33 % bis 100 %. Beide Wertebereiche basieren auf den Spezifikationen der Fußflächen. Der grüne Bereich liegt zwischen den beiden anderen Bereichen und ist nicht vorhanden, wenn der Roboter mit parallelen Füßen steht und nimmt keinen Einfluss auf die genannten prozentualen Positionen.

In Abbildung 5.11 sind diese relativen Schwerpunktpositionen eines Roboters aus der ersten Spielhälfte des RoboCup Finales 2019 dargestellt. Es wurden nur Situationen betrachtet, in denen der Roboter normal lief. Die Roboter nehmen unter anderem ihre Hände hinter den Rücken und neigen sich aufgrund des verschobenen Schwerpunktes als Ausgleich nach vorne. Solche Situationen wurden nicht betrachtet, um mehr zu generalisieren. Ebenfalls wurden Schwerpunktpositionen 250 ms vor einem Umfallen ausgelassen. Dieser Zeitraum wurde gewählt, weil ein Laufschritt im Schnitt 250 ms dauert und angenommen wird, dass ein Umfallen von 250 ms zuvor noch nicht final feststeht. Aus der etwas über elf minütigen Spielhälfte entstehen somit die Schwerpunktpositionen von effektiv etwas über fünf Minuten Laufen.

Abbildung 5.11 bestätigt somit die Aussage, dass der Schwerpunkt, welcher keine Aussage über die Dynamik des Roboters bringt, im Normalfall beim aktuellen Laufen immer innerhalb des Stützbereiches liegt und einen größeren Abstand zu den Rändern des Stützbereiches besitzt.

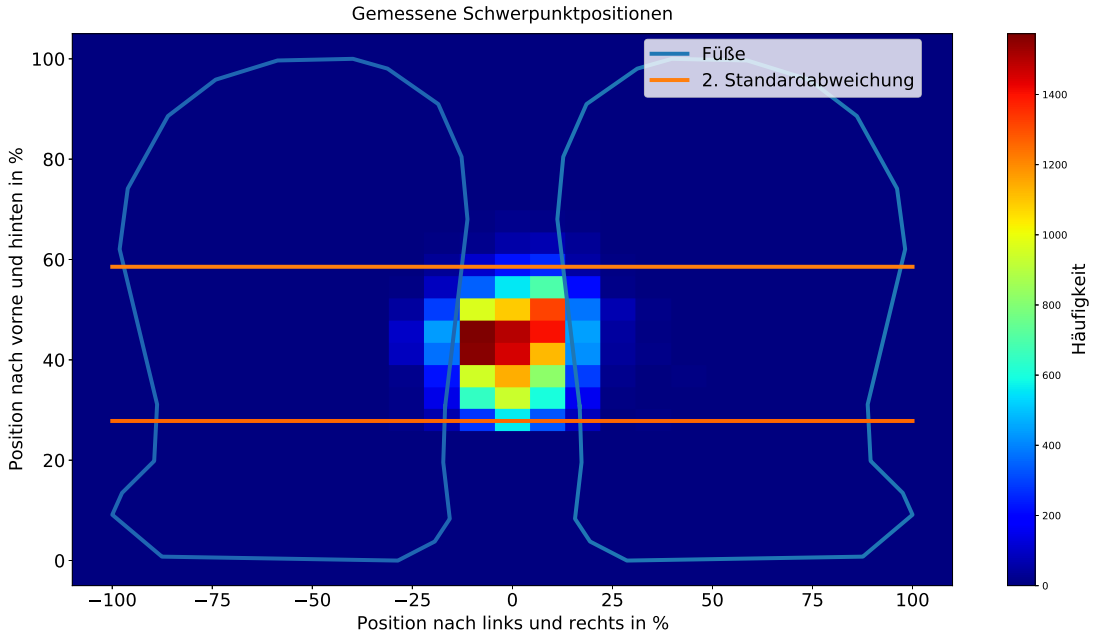

Abbildung 5.11 Die Schwerpunktpositionen relativ zur Stützfläche, definiert nach Abbildung 5.10, aufgenommen von einem Roboter aus der ersten Halbzeit des Finales vom RoboCup 2019. Die x-Positionen -50 % und +50 % liegen im Ursprung des jeweiligen Fußes. Die x-Position 0 % liegt in der Mitte zwischen beiden Füßen. Die x-Position -100 % und +100 % sind die jeweiligen äußeren Ränder beider Füße. Die y-Position 0 % ist die hinterste Fußkante, die y-Position 100 % die vorderste Fußkante. Die orangenen Linien markieren die 2. Standardabweichung der Schwerpunktposition in der x-Translation. Die obere Markierung ist an der Stelle 59 %, die untere bei 28 %.

5.4.3 Funktionsweise

Wie bereits erwähnt, soll eine Anpassung minimal ausfallen. Daher sollte ein Roboter in der Lage sein, ohne einen einzigen Eingriff der Laufschrittanpassung über ein Spielfeld laufen zu können. Genauso sollte, sofern ein Laufschritt modifiziert wurde, der Roboter schnellstmöglich den modifizierten Zustand verlassen können. Inspiriert durch die Vermutung, dass der Roboter lediglich nur kurz warten müsste statt weiterzulaufen, wenn dieser potentiell nach hinten fällt, sowie durch die Beobachtung aus den Videoaufnahmen, in denen die Roboter primär nach vorne oder hinten umfallen, werden nur die Translationen der x-Achse beider Füße beeinflusst und keine Anpassungen in den Rotationen vorgenommen. Die x-Achse betrifft dabei die Vorwärts- und Rückwärtsbewegung. Dadurch ist das entwickelte Verfahren sehr einfach in das aktuelle Laufen eingebunden und greift nur minimal ein.

Die Laufschrittanpassung soll nun, wie in Abbildung 5.12 abgebildet, folgendermaßen funktionieren. Am Anfang eines Laufschrittes wird, basierend auf dem angefragten Verhalten, eine Laufschrittgröße berechnet. Diese Laufschrittgröße wird anschließend über die nächsten Motion-Frames umgesetzt. Die Füße bewegen sich somit über die Laufschrittdauer zu ihren angefragten Endpositionen.

Für die Füße wird zusätzlich ein Toleranzbereich definiert, basierend auf den angesteuerten Fußpositionen, in denen sich der Schwerpunkt, projiziert auf die Fußebene, bewegen darf. Dabei wird der Toleranzbereich von beiden Füßen verwendet. Zwischen Standfuß und Schwingfuß wird somit nicht unterschieden. Der Toleranzbereich besteht hierbei, wie in Abbildung 5.11 dargestellt, aus den Positionen der zweiten Standardabweichung, um den durchschnittlichen Schwerpunkt herum. Ist der Schwerpunkt außerhalb, so wird die Differenz δ zum Toleranzbereich berechnet und zusätzlich auf eine maximale Größe reduziert. Auf die zuletzt angesteuerte Schwingfußposition wird dann δ addiert. Anschließend wird die Differenz $\gamma_{Schwing}$ der neuen angesteuerten Schwingfußposition zur eigentlich geplanten berechnet. Die Hälfte von $\gamma_{Schwing}$ wird zum Schluss von der geplanten Ansteuerungsposition vom Standfuß subtrahiert.

Dadurch ergibt sich in der Theorie ein Laufverhalten, in dem der Roboter nach vorne läuft, um ein Kippen nach vorne zu verhindern respektive sich nach hinten bewegt, um ein Kippen nach hinten zu verhindern.

Der Algorithmus hierfür ist in Algorithmus 1 zu sehen. δ soll dabei basierend auf den Sensordaten berechnet werden, heißt auf der aktuellen Lage im Raum des Roboters, dem Roboterkoordinatensystem ρ, den gemessenen Fußpositionen ϕ und dem durch die Sensordaten berechenbaren Schwerpunkt m. Der Wert δ selber wird anschließend mit den eigentlich geplanten Fußpositionen, berechnet vom Laufmodul, verrechnet, um die neuen Schwing- und Standfußpositionen zu erhalten.

Algorithmus 1: Laufschrittanpassung

Input: Der Schwerpunkt m_ϕ, die geplante, die zuletzt geplante und zuletzt angesteuerte Position vom Stützfuß $\text{stütz}_{\tau\text{geplant}_{t0}}$, $\text{stütz}_{\tau\text{geplant}_{t-1}}$, $\text{stütz}_{\tau\text{aktuell}}$, die geplante, die zuletzt geplante und zuletzt angesteuerte Position vom Schwingfuß $\text{schwing}_{\tau\text{geplant}_{t0}}$, $\text{schwing}_{\tau\text{geplant}_{t-1}}$, $\text{schwing}_{\tau\text{aktuell}}$, die Anpassungsgeschwindigkeit maxVel, der Toleranzbereich $T_{\text{oben}}, T_{\text{unten}}$

Output: Die neuen Positionen beider Füße $\text{stütz}_{\tau\text{neu}}$, $\text{schwing}_{\tau\text{neu}}$

1 $\text{schwing}^{(x)} = \text{schwing}^{(x)}_{\tau\text{aktuell}} + (\text{schwing}^{(x)}_{\tau\text{geplant}_0} - \text{schwing}^{(x)}_{\tau\text{geplant}_{t-1}})$

2 $\gamma_{\text{Schwing}_{t-1}} = \text{schwing}^{(x)}_{\tau\text{aktuell}} - \text{schwing}^{(x)}_{\tau\text{geplant}_{t-1}}$

3 $\text{reset}^{(x)} = \text{clip}(-\text{maxVel}, -\gamma_{\text{Schwing}_{t-1}}, +\text{maxVel})$

4 $$\delta = \begin{cases} m_\phi^{(x)} - T_{\text{unten}} + \text{reset}^{(x)} & , \text{ falls } m_\phi x < T_{\text{unten}} + \text{reset}^{(x)} \\ m_\phi^{(x)} - T_{\text{oben}} + \text{reset}^{(x)} & , \text{ falls } m_\phi^{(x)} > T_{\text{oben}} + \text{reset}^{(x)} \\ 0 & , \text{ sonst} \end{cases}$$

5 $$\text{minValAdjust} = \begin{cases} -\text{reset}^{(x)} & , \text{ falls } \text{reset}^{(x)} > 0 \\ 0, & \text{sonst} \end{cases}$$

6 $$\text{maxValAdjust} = \begin{cases} -\text{reset}^{(x)} & , \text{ falls } \text{reset}^{(x)} < 0 \\ 0 & , \text{ sonst} \end{cases}$$

7 $\text{schwing}^{(x)} = \text{schwing}^{(x)} + \text{reset}^{(x)} +$
$\text{clip}(-\text{maxVal}+\text{minValAdjust}, \delta, \text{maxVal}+\text{maxValAdjust})$

8 $\gamma_{\text{Schwing}_{t0}} = \text{schwing}^{(x)} - \text{schwing}^{(x)}_{\tau\text{geplant}_{t0}}$

9 $\text{stütz}_{\tau\text{neu}} = \text{stütz}_{\tau\text{geplant}_{t0}}$

10 $\text{schwing}_{\tau\text{neu}} = \text{schwing}_{\tau\text{geplant}_{t0}}$

11 $\text{schwing}^{(x)}_{\tau\text{neu}} = \text{schwing}^{(x)}$

12 $\text{stütz}^{(x)}_{\tau\text{neu}} \mathrel{-}= \gamma_{\text{Schwing}_{t0}} \cdot 0.5$

13 return $\text{stütz}_{\tau\text{neu}}$, $\text{schwing}_{\tau\text{neu}}$

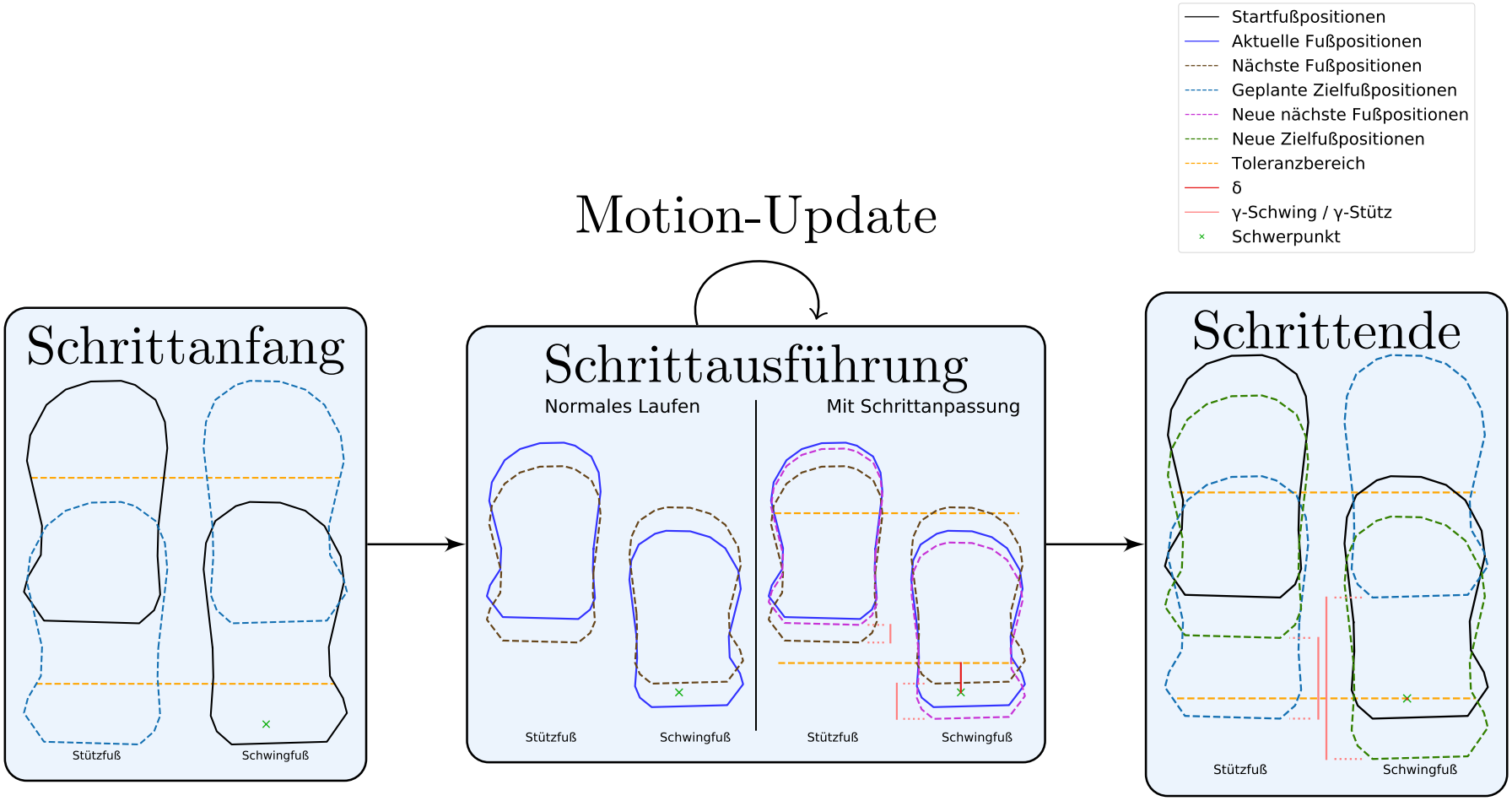

Abbildung 5.12 Das Verhalten der Laufschrittbalancierung. Der Roboter plant, vorwärts zu laufen. In jedem Zyklus propagiert das Laufmodul die Füße weiter Richtung ihrer Zielposition. Die Laufschrittanpassung addiert dabei den Fehlerwert δ auf den Schwingfuß und zieht diesen langsam nach hinten. Die Hälfte der Korrektur $\gamma_{Schwing}$ des Schwingfußes wird vom Standfuß subtrahiert. Am Ende des Laufschrittes lief der Roboter somit in Richtung des Schwerpunktes, also nach hinten. Der Toleranzbereich ist beim Laufschrittanfang relativ zu den Startfußpositionen (schwarz), bei der Laufschrittausführung relativ zu den aktuellen Fußpositionen (dunkelblau) und beim Laufschrittende relativ zu den neuen Zielfußpositionen (dunkelgrün).

In der Entwicklungsphase wurden weitere Varianten ausprobiert, bei denen δ unterschiedlichen Einfluss genommen hat. So wurde getestet, auch auf den Standfuß δ vollständig zu subtrahieren. Ebenso wurde untersucht, abhängig davon, welcher Fuß näher am projizierten Schwerpunkt stand, auf diesen δ zu addieren und vom jeweils anderen δ vollständig oder in Teilen zu subtrahieren. Eine Evaluation dieser Varianten würde den Rahmen der Arbeit sprengen. In der Entwicklungsphase stellte sich aber der vorgestellte Vorgang als signifikant besser heraus, im Vergleich zu den anderen. Verschiedene Gründe lassen sich hierfür vermuten. Eine vollständige Verrechnung von δ mit dem Standfuß scheint zu einem sehr instabilen Laufen zu führen, da der Standfuß das gesamte Gewicht des Roboters trägt, wodurch ein zu schneller Wechsel in der Bewegungsrichtung seinen instabilen Zustand verschlimmert. Ebenso scheint es nicht sinnvoll zu sein, den Standfuß in Richtung des projizierten Schwerpunktes zu bewegen, um ein Umfallen zu verhindern. Dies käme einer Hüftbalancierung gleich, welche, wie bereits in Abschnitt 5.3 untersucht, nur zu weiteren Problemen führen würde, bei denen die Gelenke hängen bleiben und den Roboter nur zusätzlich destabilisieren.

Auch ist, wie bereits in Abschnitt 4.5 beschrieben, der Schwerpunkt nur begrenzt nutzbar für sich bewegende Systeme. Eine genaue Berechnung des ZMPs würde aber ebenfalls den Rahmen dieser Arbeit sprengen, weshalb die Dynamik des Roboters verwendet wird, um eine Vorhersage über den Schwerpunkt zu treffen. Die Annahme bleibt, dass die Roboter beim aktuellen Laufen stabil sind, wodurch der Schwerpunkt ohnehin dauerhaft innerhalb der Stützfläche liegt muss. In Kombination mit der Motorenverzögerung von 36 bis 48 ms reicht es daher aus zu wissen, wo der Schwerpunkt zum Zeitpunkt der Umsetzung der Gelenkansteuerungen sein wird.

Für diese Vorhersage wird ein LIPM modelliert, wie in Abbildung 5.13 veranschaulicht, um die Transformationsmatrix ρ des Roboters vorherzusagen. Das invertierte Pendel besteht aus einer Masse, für uns der Schwerpunkt, an einem starren Faden, der Roboterkörper, welcher mit einem Punkt P (später als k bezeichnet), der Kippkante des Stützfußes, verbunden ist. Durch die Position des Pendelkörpers, seiner aktuellen Winkelgeschwindigkeit und der Gravitationskraft kann approximiert werden, an welcher Stelle sich der Pendelkörper zukünftig befinden wird. Dies kann genutzt werden, um ρ_{tn} für die Zukunft zu bestimmen. Theoretisch könnte auch direkt die Schwerpunktposition im Koordinatensystem ρ_{tn} bestimmt und in die Zukunft approximiert werden. Durch den Zwischenschritt der Berechnung jeder Transformationsmatrix ρ_{tj}, welche zeitlich vor ρ_{tn} kommt, können die Bestandteile des invertierten Pendels mehrmals neu berechnet werden, um eine höhere Genauigkeit zu erreichen. Dies betrifft hauptsächlich den Kipppunkt P, dessen Position sich verändert, wenn ρ in die Zukunft berechnet wird. Dabei wird die Bewegung des Roboters für zukünftige Zeitschritte ignoriert. Diese Modellierung nimmt somit an, dass der Roboter statisch ist. Denn in Extremfällen kann momentan nicht vorhergesagt werden, wie sich die Gelenke verhalten, weshalb eine statische Annahme besser ist als eine Modellierung basierend auf der Ansteuerung.

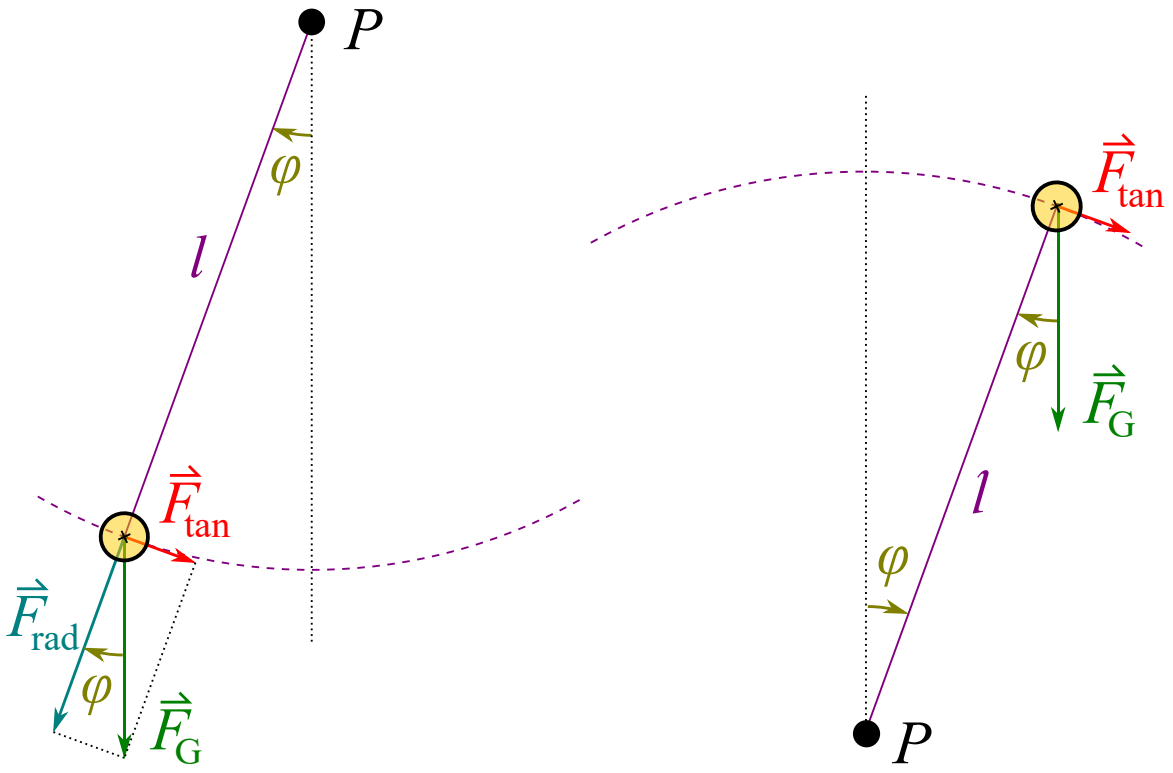

Abbildung 5.13 Links das einfache und rechts das invertierte Pendel. Grafik entnommen aus Krause [2018]

ρ_{tn} selber schließt die Transformation für die Rotation und Translation ein. Für eine korrekte Approximation müsste dafür die Odometrie miteinbezogen werden. Heißt, wenn der Roboter mit 250 mm/s geradeaus läuft, müsste eine Transformation ρ_{tn} mit n gleich einer Sekunde eine zusätzliche Translation von 250 mm nach vorne besitzen. Hierfür soll aber nun folgende Annahme gelten: Ein Roboter, der auf einem Fuß steht, kann seinen Schwingfuß beliebig bewegen, ohne dass sich seine Position auf dem Boden ändert. Erst bei einem Schrittwechsel bewegt sich der Roboter auf dem Boden und ändert somit seine Position. Die Odometrie, welche nur eine x- und y-Translation sowie eine z-Rotation besitzt, ist also während eines Laufschrittes gleich null und springt zum Zeitpunkt des Schrittwechsels. Um diesen Odometriesprung zu umgehen, werden zwei invertierte Pendel modelliert. Eines für den aktuellen Motion-Frame und eines für den letzten. Beide Modelle sind jeweils basierend zum Standfuß, welcher von der Odometrie nicht beeinflusst wird. Ebenfalls kann eine weitere Vereinfachung vorgenommen werden. Die z-Translation der Transformation ρ_{tn} ist für den vorgestellten Anwendungszweck ebenfalls irrelevant. Damit kann die gesamte Translation von ρ_{tn} ignoriert werden. Diese besteht nun nur noch aus einer Rotationsmatrix, welche ihren Koordinatenursprung an derselben Position besitzt wie das Torsokoordinatensystem τ, im Schritt des Roboters. Durch diese Vereinfachung entstehen keine zusätzlichen Ungenauigkeiten, da das invertierte Pendel weiterhin korrekt erstellt werden kann. Gleichzeitig wird der Fehler des Roboterkoordinatensystems umgangen, welcher bereits in Abschnitt 4.2 erwähnt wurde.

Da eine kurzzeitige Vorhersage genügt, werden weitere Einflüsse nicht betrachtet. So bewegen sich bis zum Zeitpunkt der Vorhersage weiterhin die Füße, welche aufgrund ihrer Masse den Schwerpunkt verschieben. Die daraus resultierenden Fehlergrößen werden später evaluiert.

5.5 Bestimmen der Laufschrittanpassung

Für die Vorhersage wird im Robotorkoordinatensystem ρ die Schwerpunktposition m_ρ benötigt, dessen Geschwindigkeit $\dot{m}_\rho$ und der Kipppunkt k_ρ, welcher auf dem Rand der Stützfläche liegt.

Für die Berechnung des Kipppunktes wird, ähnlich zu der aktuell im B-Human-Framework verwendeten Fallerkennung [vgl. Krause, 2018], die Bewegungsrichtung des Schwerpunktes verwendet, um den Schnittpunkt mit der Stützfläche zu bestimmen. Statt einem Kalman-Filter wird hier aber nur der Schwerpunkt und die Transformationsmatrix ρ vom aktuellen und letzten Motion-Frame benötigt. Auch wird nur die Bewegungsrichtung in Roboterkoordinaten berechnet, der Kipppunkt aber in Torsokoordinaten bestimmt, welche nachträglich in das Roboterkoordinatensystem übernommen wird. Dies dient dem Zweck, die Stützfläche nicht zusätzlich in das Roboterkoordinatensystem umrechnen zu müssen.

Für die Berechnung von $\dot{m}_\rho$ gilt:

$$m_{\rho_{t0}} = \rho_{t0} \cdot m_{\tau_{t0}} \tag{5.2}$$

$$m_{\rho_{t-1}} = \rho_{t-1} \cdot m_{\tau_{t-1}} \tag{5.3}$$

$$\dot{m}_\rho = m_{\rho_{t0}} - m_{\rho_{t-1}} \tag{5.4}$$

$$\dot{m}_\tau = {\rho_{t0}}^{-1} \cdot \dot{m}_\rho \tag{5.5}$$

Für den Kipppunkt k wird mithilfe der Fußsohlenkalibrierung eine vereinfachte Stützfläche gebildet, welche aus den vier Eckpunkten $s_j \in \mathbb{R}^3; j \in [1..4]$ besteht und im Torsokoordinatensystem liegen. Um nun den Vektor zu bestimmen, welcher für die Schnittpunktberechnung verwendet werden soll, wird aus den vier Punkten ein konvexes Polygon gebildet sowie ein Richtungsvektor vom Zentrum des Polygons, $SZ \in \mathbb{R}^3$, mit der Geschwindigkeitsrichtung des Schwerpunktes $\dot{m}_\tau$. Mit dem einfachen Test, ob der Punkt s_i nicht auf der gleichen Seite liegt wie der Punkt s_j, relativ zum Vektor $\dot{m}_\tau$, kann der Vektor, bestehend aus den Eckpunkten s_i und s_j, der für die Schnittpunktberechnung verwendet werden soll, bestimmt werden. Gegeben seien zwei Punkte $s_i, s_j \in \mathbb{R}^3; i, j \in [1..4]$, mit $i \neq j$ und $||\dot{m}_\tau|| \neq 0$, dann kann berechnet werden:

$$m_{\text{norm}} = \frac{\dot{m}_\tau}{||\dot{m}_\tau||} \tag{5.6}$$

$$w_i = \begin{pmatrix} m_{\text{norm}}^{(x)} \\ m_{\text{norm}}^{(y)} \end{pmatrix} \times \begin{pmatrix} s_i^{(x)} - SZ^{(x)} \\ s_i^{(y)} - SZ^{(y)} \end{pmatrix} \tag{5.7}$$

$$w_j = \begin{pmatrix} m_{\text{norm}}^{(x)} \\ m_{\text{norm}}^{(y)} \end{pmatrix} \times \begin{pmatrix} s_j^{(x)} - SZ^{(x)} \\ s_j^{(y)} - SZ^{(y)} \end{pmatrix} \tag{5.8}$$

$$w_i \cdot w_j \leq 0 \wedge w_i \neq w_j \tag{5.9}$$

Da vom Zentrum des konvexen Polygons aus ein Vektor verwendet wird, erfüllt genau eine Kombination von Eckpunkten die Gleichung 5.9, welche als Vektor für den Schnittpunkt k verwendet werden. Beide Vektoren $\dot{m}$ und $\overrightarrow{s_i s_j}$ liegen zwar im 3D-Raum vor, die z-Achse kann aber vernachlässigt werden, sodass der Schnittpunkt k im 2D-Raum berechnet und anschließend in den 3D-Raum überführt werden kann. Die Berechnung von k ist in Abbildung 5.14 visualisiert.

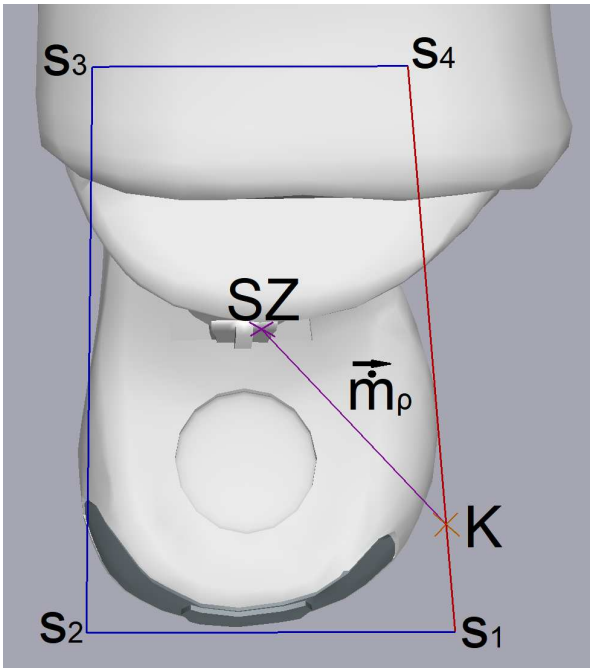

Abbildung 5.14 Die Ränder der Stützfläche des Standfußes (blau), definiert durch die vier Eckpunkte S1 bis S4, sowie der Kipppunkt k, definiert durch den Mittelpunkt SZ der Stützfläche und des Schwerpunktbewegungsvektors $\vec{m}_\rho$.

Mit dem nun bekannten Kipppunkt k und den letzten beiden Schwerpunkten, jeweils im Roboterkoordinatensystem, kann mithilfe des invertierten Pendels der Zustand vorausberechnet werden, um daraus die Transformationsmatrix ρ für einen Zeitpunkt in der Zukunft zu bestimmen. Das LIPM ist dabei wie in Abbildung 5.13 zu sehen. Punkt P ist unser Kipppunkt k_ρ, l ist die Entfernung vom Schwerpunkt m_ρ zu k_ρ, φ ist der aktuelle Winkel, der sich aus

den Positionen von m_ρ und k_ρ ergibt und Fg ist die Gravitationskraft von -9.81m/s^2.

Um nun den Zustand vorherzusagen, wird φ_{t0} und φ_{t-1} berechnet, mithilfe von den aktuellen und letzten Schwer- und Kipppunkten $m_{\rho_{t0}}, m_{\rho_{t-1}}, k_{\rho_{t0}}$. Daraus kann die Winkelgeschwindigkeit $\dot{\varphi}$ ermittelt werden, um den Zustand vorauszuberechnen. Zusätzlich wird die Kraft Fg einbezogen, welche dafür verantwortlich ist, dass der Stützfuß gegen den Boden drückt und den Roboter vor dem Umfallen bewahrt. Mit dieser kann nämlich die Winkelbeschleunigung $\ddot{\varphi}$ bestimmt werden, welche $\dot{\varphi}$ nochmals verändert.

Da das Pendelmodell im 3D-Fall verwendet wird, werden jeweils für die x- und y-Achse die Winkelgeschwindigkeit und -beschleunigung berechnet. Die jeweiligen Achsen werden im Folgenden als a-Achse zusammengefasst, das Koordinatensystem ist weiterhin ρ und wird für die Übersichtlichkeit weggelassen:

$$l_a = \sqrt{(m^{(a)} - k^{(a)})^2 + (m^{(z)} - k^{(z)})^2} \tag{5.10}$$

$$\varphi^{(a)} = \frac{\arccos\left(m^{(z)} - k^{(z)}\right)}{l_a} \tag{5.11}$$

$$\ddot{\varphi}^{(a)} = \frac{-9.81\text{m/s}^2}{l^{(a)}} \cdot \varphi^{(a)} \tag{5.12}$$

Mit Gleichung 5.10 bis Gleichung 5.12 können somit sowohl φ_{t0} und φ_{t-1}, als auch $\ddot{\varphi}$ bestimmt werden. Die einfache Geschwindigkeit $\dot{\varphi}$ erhält man durch die Subtraktion von φ_{t0} und φ_{t-1}. $\dot{\varphi}$ bezieht sich hierbei auf die Geschwindigkeit $\frac{\text{Grad}}{\text{Dauer eines Motion-Frames}}$. Für die Berechnung von $\dot{\varphi}_{t1}$ muss $\ddot{\varphi}$, relativ zu der Dauer eines Motion-Frames von 12 ms, berücksichtigt werden. $\dot{\varphi}$ und $\dot{\varphi}_{t1}$ ergeben sich somit durch:

$$\dot{\varphi} = \varphi_{t0} - \varphi_{t-1} \tag{5.13}$$

$$\dot{\varphi}_{t1} = \dot{\varphi} + \ddot{\varphi} \cdot 0.012\text{s} \tag{5.14}$$

$$\tag{5.15}$$

Die Transformationsmatrix ρ kann nun mithilfe von $\dot{\varphi}_t 1$ einen Motion-Frame in die Zukunft berechnet werden. Dieser Durchlauf, wie in Algorithmus 2 zu sehen, kann beliebig oft durchgeführt werden, um eine gewünschte Anzahl an Motion-Frames vorauszuschauen. Dabei werden k und φ immer neu berechnet, da sich die Position des Kipppunktes jedes Mal leicht verändert. Mithilfe von ρ_{tn} kann dann der Schwerpunkt für den Zeitpunkt von n Motion-Frames in der Zukunft vorhergesagt werden.

Algorithmus 2: Schwerpunkt Prädiktion

Input: Die Schwerpunkte $m_{\tau_{t0}}, m_{\tau_{t-1}}$, die Transformationsmatrizen ρ_{t0}, ρ_{t-1}, der Stützfuß fuß_τ, Iterationsanzahl n

Output: Die Transformationsmatrix ρ_{tn}

```
polygon = PolygonBerechnen(fuß_τ)
ρ_current = ρ_t0
ρ_last = ρ_t−1
m_τcurrent = m_τt0
m_τlast = m_τt−1
for n do
    m_current = ρ_current · m_τcurrent
    m_last = ρ_last · m_τlast
    ṁ_τ = Schwerpunktvektor(m_current, m_last, ρ_current)
    k_τ = KipppunktBerechnen(ṁ_τ, polygon)
    φ̇ = Transformationsänderung(m_current, m_last, k_τ)
    ρ_t+1 = Rotierung(ρ_current, φ̇)
    ρ_last = ρ_current
    ρ_current = ρ_t+1
    m_τlast = m_τcurrent
end
return ρ_current
```

In Abbildung 5.15 sind die Fehlerdistanzen für die Schwerpunktvorhersage mit verschiedenen Iterationsanzahlen dargestellt. Die Fehlerdistanz wurde dabei nicht mit der Schwerpunktposition bestimmt, welche zum vorhergesagten Zeitpunkt gemessen wurde. Stattdessen wurde innerhalb eines Fensters von zwei Motion-Frames um den vorhergesagten Zeitpunkt die gemessene Schwerpunktposition verwendet, welche die geringste absolute Abweichung ergab. Denn solange die vorhergesagte Schwerpunktposition in der nahen Zukunft (einige wenige Motion-Frames) tatsächlich eingenommen wurde, ist für die Nutzbarkeit des Modells egal, ob dies zum vorhergesagten Zeitpunkt geschieht oder um einige Motion-Frames vorher oder später. Dies hätte lediglich zur Auswirkung, dass zum Beispiel die vorhergesagte Schwerpunktposition manchmal der in einem Motion-Frame entspricht, manchmal der in fünf. Eine Vorhersage, welche den Schwerpunkt in die richtige Richtung approximiert, erfüllt ihren Zweck und ist besser, als wenn man nur den aktuell gemessenen Schwerpunkt verwenden würde.

Durch diese Bewertung ergibt sich, dass für alle gezeigten Iterationsanzahlen die vorhergesagten Schwerpunktpositionen zu 95 % eine unter 15 mm Fehlerdistanz besitzen. Wie zu erwarten, nehmen die Fehlerdistanzen für höhere Iterationsanzahlen zu. Auch weisen die Ausreißer von längeren Vorhersagen deutlich größere Fehlerdistanzen von bis zu 5 cm auf. Diese Fehler kommen zum einen durch die fehlerhafte Modellierung zustande. Die Schätzung der eingehenden Matrix ρ durch den aktuell verwendeten Kalman-Filter ist nicht perfekt, sondern teilweise fehlerhaft. Zum anderen ist auch die verwendete Schwerpunktposition nicht korrekt,

da ignoriert wird, dass sich die Füße über die vorhergesagten Zeitschritte bewegen, welche aufgrund ihrer Masse einen Einfluss nehmen.

Als Ziel wird aber nur eine Vorhersage von drei Motion-Frames benötigt. Dessen Ausreißer weisen keine extremen Fehlerdistanzen auf. Somit ist die entwickelte Modellierung ausreichend für den in dieser Arbeit beschriebenen Zweck.

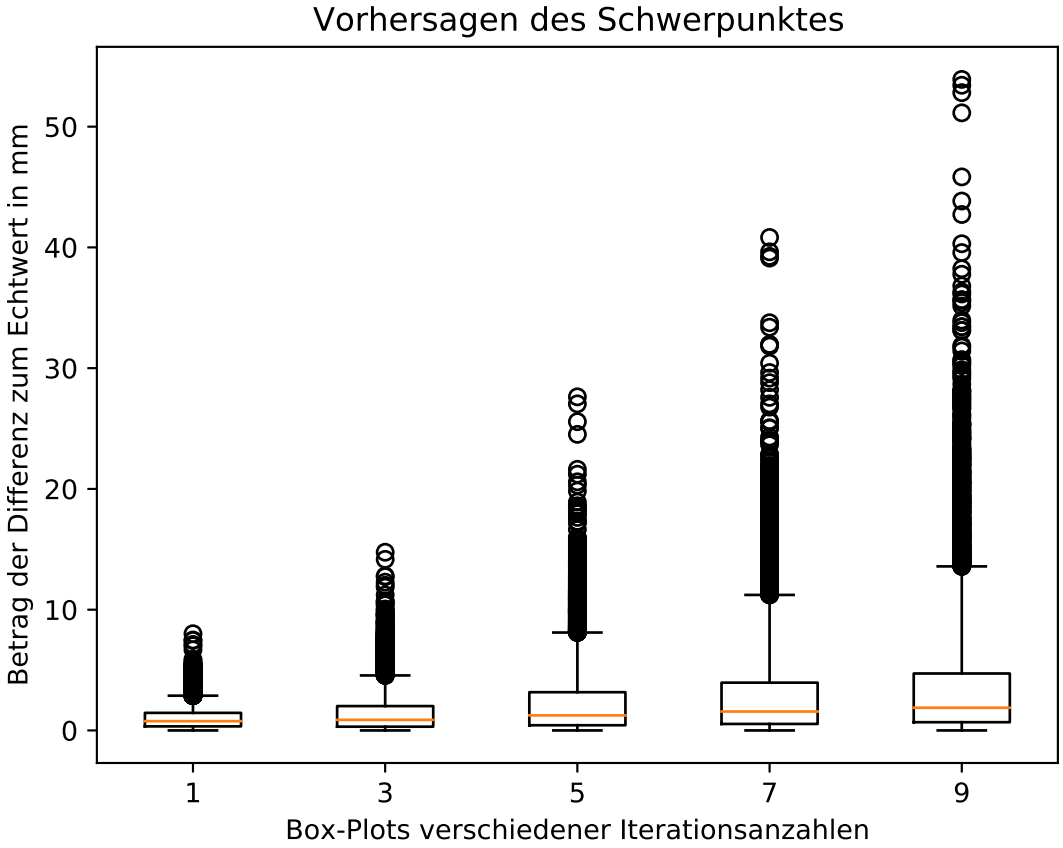

Abbildung 5.15 Box-Plot Diagramme für die absoluten Fehlerdistanzen von verschiedenen vorhergesagten Schwerpunkten. Die Boxen stehen für die unteren bzw. oberen Quartils, der gelbe Strich für den Median, die Whiskers entsprechen 150 % der Quartilsbreite. Der untere Whisker liegt dabei jeweils bei 0, da es keine negativen Fehler gibt. Innerhalb der Quartils liegen 50 % und innerhalb beider Whiskers liegen 95 % der Datenpunkte.

Für das angestrebte Vorgehen der Balancierung, die Füße abhängig vom Schwerpunkt zu verschieben, fehlt nun nur noch der Toleranzbereich, um den Differenzwert δ zu bestimmen, welcher die Differenz zwischen dem Toleranzbereich und der Schwerpunktposition in der x-Achse ist. Da das Verfahren möglichst wenig eingreifen soll, bietet sich an, basierend auf der Auswertung aus Abschnitt 5.4.2, die 2. Sigmaumgebung für die untere und obere Grenze zu verwenden, welche in Abbildung 5.11 dargestellt ist. Dadurch ergäbe sich der Bereich [0.28,0.59]. Da dieser Bereich konfigurierbare Parameter darstellt, welche zwischen den Robotern leicht abweichen können, wird ein etwas größerer Toleranzbereich von [0.25,0.6] verwendet. Dadurch erhält man gerundete Werte, welche einen größeren Toleranzbereich darstellen und gleichzeitig resistenter gegenüber Abweichungen zwischen den Robotern sind.

Um nun den Differenzwert δ zu berechnen, wird der in die Fußebene projiziert Schwerpunkt benötigt. Hierfür kann dieser in das Roboterkoordinatensystem übertragen werden, die z-Position auf die Höhe von dem Standfuß im Roboterkoordinatensystem gesetzt und anschließend zurück in das Torsokoordinatensystem geführt werden.

$$m_{\rho_{tn}} = \rho_{tn} \cdot m_{\tau_{t0}} \tag{5.16}$$

$$\phi_{\rho_{tn}} = \rho_{tn} \cdot \phi_{\tau_{t0}} \tag{5.17}$$

$$\text{mFoot}_{\rho_{tn}} = m_{\rho_{tn}} \cdot (0, 0, \phi^{(z)}_{\tau_{t0}} - m^{(z)}_{\rho_{tn}})^T \tag{5.18}$$

$$\text{mFoot}_{\tau_{t0}} = \rho^{-1}_{tn} \cdot \text{mFoot}_{\rho_{tn}} \tag{5.19}$$

Anschließend kann $mFoot_{\tau_{t0}}$ vom Toleranzbereich (TB) subtrahiert werden, sofern dieser außerhalb liegt.

$$\delta = \begin{cases} \text{mFoot}^{(x)}_{\tau_{t0}} - (\phi_{\text{lowerFoot}} \cdot TB_{\text{lower}})^{(x)} & \text{, falls } \text{mFoot}^{(x)}_{\tau_{t0}} < \phi^{(x)}_{\text{lowerFoot}} \\ \text{mFoot}^{(x)}_{\tau_{t0}} - (\phi_{R\rho_{tn}} \cdot TB_{\text{upper}})^{(x)} & \text{, sonst falls } \text{mFoot}^{(x)}_{\tau_{t0}} > \phi^{(x)}_{\text{upperFoot}} \\ 0 & \text{, sonst} \end{cases} \tag{5.20}$$

mit (5.21)

$$\phi_{\text{lowerFoot}} = \begin{cases} \phi_{\text{left}} & \text{, falls } \phi^{(x)}_{\text{left}} < \phi^{(x)}_{\text{right}} \\ \phi_{\text{right}} & \text{, sonst} \end{cases}$$

$$\phi_{\text{upperFoot}} = \begin{cases} \phi_{\text{left}} & \text{, falls } \phi^{(x)}_{\text{left}} > \phi^{(x)}_{\text{right}} \\ \phi_{\text{right}} & \text{, sonst} \end{cases}$$

$$TB_{\text{lower}} = \left(0, 0, \text{FootLength} \cdot 0.25 - \frac{\text{FK}_{\text{back}}}{\text{FootLength}}\right)^T$$

$$TB_{\text{upper}} = \left(0, 0, \text{FootLength} \cdot 0.6 - \frac{\text{FK}_{\text{back}}}{\text{FootLength}}\right)^T$$

$\phi_{\text{lowerFoot}}$ beschreibt dabei die Position des weiter hinten stehenden Fußes im Torsokoordinatensystem, $\phi_{\text{upperFoot}}$ die Position für den weiter vorne stehenden. *FootLength* ist die Länge des Fußes und FK_{back} die Länge vom Ursprung der Füße zur hinteren Kante an der Hacke, jeweils entnommen von der Fußsohlenkalibrierung aus Abschnitt 5.4.1. Die Parameter 0.25 und 0.6 korrespondieren zu dem definierten Toleranzbereich, in dem sich der Schwerpunkt befinden darf.

δ wird nun auf die angesteuerte Schwingfußposition und $-\frac{\delta}{2}$ auf die angesteuerte Standfußposition mit einer Geschwindigkeitsfunktion addiert. Dadurch bewegen sich die Füße nicht beliebig schnell zu den neuen Positionen, sondern gemäßigt mit einer maximalen Geschwindigkeit.

5.6 Standfußkompensation

Wie schon in Abschnitt 5.3 erwähnt, ist ein anderes großes Problem, dass sich die Füße nicht mehr parallel zum Boden bewegen, sobald die Roboter instabil laufen, zu sehen in Abbildung 5.6 und Abbildung 5.7. Die Bewegung des Schwingfußes in den Boden soll dadurch verhindert werden, indem der Rotationsfehler vom Standfuß auf den Schwingfuß übertragen wird. Jedoch soll dies über eine maximale Rotationsgeschwindigkeit geschehen, um eine flüssige Bewegung zu gewährleisten. Zwar bewegt sich der Schwingfuß so parallel zum Boden, kommt es dann zum Schrittwechsel und der Standfuß besitzt weiterhin einen Rotationsfehler, bleibt dieser auch im nächsten Laufschritt beibehalten. Folglich würde der Roboter unweigerlich nach vorne kippen und häufiger umfallen, als dies bisher geschieht. Um dies zu verhindern, wird die Rotationskompensation vor Ende des Laufschrittes erneut über eine maximale Rotationsgeschwindigkeit wieder entfernt. Dadurch kann sich der Schwingfuß für den Großteil der Laufschrittdauer parallel zum Boden bewegen, riskiert aber nicht ein dauerhaft nach vorne Neigen des Roboters. Die Kompensation erfolgt folgendermaßen:

$$\phi_{\text{SchwingFuß}} = \text{Rot}_y(\phi^{(\beta)}_{\text{StandFuß}} \cdot p) \cdot \phi_{\text{SchwingFuß}} \tag{5.22}$$

$$\text{mit } t, p, c, d \in \mathbb{R}, \text{ und} \tag{5.23}$$

$$p = \begin{cases} 1 - \min\left(\frac{t}{d}, 1\right) & , \frac{t}{d} \geq c \\ 1 & , \text{sonst} \end{cases}$$

Dabei ist t die aktuelle Dauer des Laufschrittes in Sekunden, c die prozentuale Laufschrittdauer, ab wann die Kompensation auf 0 reduziert werden soll, und d die geplante Laufschrittdauer von 0.25 Sekunden. Da ein Schrittwechsel erst nach 50 % der geplanten Laufschrittdauer akzeptiert wird, sollte c zwischen [0.5,1] liegen. Die Begrenzung der Rotationsgeschwindigkeit wird der Übersichtlichkeit halber nicht in den Gleichungen dargestellt. Es werden dazu weitere Restriktionen eingefügt, um die Anforderung, möglichst wenig in das Laufen einzugreifen, zu erfüllen. Da nur verhindert werden soll, dass der Schwingfuß beim Bewegen nach vorne in den Boden bewegt wird, müssen allein negative Rotationen vom Standfuß kompensiert werden. Ebenso führen nur größere Rotationen zu einer Destabilisierung, wie es in Abbildung 5.6 zu sehen ist. Auch soll nur eingegriffen werden, wenn der Roboter tatsächlich zu weit nach vorne geneigt ist. Gleichung 5.22 wird daher um den Interpolationsfaktor I erweitert.

$$\phi_{\text{SchwingFuß}} = \text{Rot}_y(\phi^{(\beta)}_{\text{StandFuß}} \cdot p \cdot I) \cdot \phi_{\text{SchwingFuß}} \tag{5.24}$$

$$\text{mit } I, \text{ MinTorsoY und MinFußrotation } \in \mathbb{R} \text{ und} \tag{5.25}$$

$$I = \text{clip}\left(0, \frac{\rho^{(\beta)}}{-\text{MinTorsoY}}, 1\right) \cdot \text{clip}\left(0, \frac{\phi^{(\beta)}_{\text{Standfuß}}}{-\text{MinFußrotation}}, 1\right)$$

Mit dem Experiment aus Abschnitt 5.4.1, bei dem der Roboter eine Kante herunterläuft, wurden Parameter für c, *MinTorsoY* und *MinFußrotation* experimentell so bestimmt, dass der Roboter nicht umfiel, gleichzeitig auf ebenem Untergrund normal lief. Hier wurde auch eine lineare Interpolation für p von 1 bis 0 getestet. Die sofortige Reduktion auf $p = 1 - c$ erwies sich dabei als signifikant besser. Für die Parameter selber ergab sich für c ein Wert von 0.75, 3 Grad für *MinFußrotation* und 5 Grad für *MinTorsoY*. *MinFußrotation* und *MinTorsoY* wurden nicht optimiert, sondern lediglich passende Parameter gewählt, damit unter normalen Bodenbedingungen die Fußkompensation gleich oder nahe 0 lag. Der Parameter c hingegen wurde über einige wenige Versuche manuell optimiert, damit beim Laufen nach vorne der Schwingfuß möglichst weit nach vorne kommen kann, gleichzeitig aber zum Schrittwechsel eine möglichst geringe Kompensation existiert.

5.7 Evaluation

Die entwickelten Erweiterungen für die Laufstabilisierung werden nun evaluiert. Übliche Ursachen für instabile Laufzustände sind sowohl Kollisionen mit anderen Robotern als auch Unebenheiten im Spielfeldboden wie Linien, Untergrundkanten oder unterschiedliche Abnutzungsgrade des Kunstrasens. Um daher eine sinnvolle Evaluation durchzuführen, die die Roboter in instabile Laufzustände bringt, aber gleichzeitig nicht vollständig unrealistisch und schädlich für die Roboterhardware ist, wird ein speziell designter Parcours verwendet.

Durch lokale Begrenzungen des Aufbaus wird auf einem normalen SPL Spielfeld der Größe 6 m × 9 m eine 9 m gerade Strecke präpariert. Auf dieser sollen die Roboter geradeaus laufen, ohne dabei umzufallen. Die Strecke selber ist in zwei Teile aufgeteilt. Die ersten drei Meter sind nicht präpariert, damit die Roboter unter normalen Idealbedingungen laufen müssen. Auf den restlichen sechs Metern ist die Strecke mit Holzlatten und Holzrampen versehen. Diese Erschwernisse liegen unter dem Kunstrasen, damit die Bodenreibung unverändert bleibt und lediglich eine Bodenunebenheit entsteht. In 5.16 ist diese geplante Streckenpräparierung zu sehen.

Das erste Hindernis ist eine 3 mm hohe Kante, welche zeigen soll, ob das bisherige Laufen bei kleineren, jedoch nicht unwesentlichen Unebenheiten noch stabil sein kann. Das zweite und dritte Hindernis sind jeweils zwei 6 mm hohe Kanten, welche das bisherige Laufen bereits an seine Grenzen bringen soll, die entwickelten Erweiterungen aber damit, zumindest in der Theorie, keine Probleme haben dürften. Das vierte Hindernis ist eine 1 cm erhöhte Plattform. Aus Tests in der Entwicklung ist bereits bekannt, dass eine 1 cm hohe Kante durch die Laufschrittanpassung nicht ausgeglichen werden kann, da die Roboter innerhalb eines Laufschrittes bereits nach hinten umfallen. Deshalb ist der vordere Teil eine zweistufige Plattformerhöhung. Die Roboter sollen so sanfter auf die Erhöhung laufen. Gleichzeitig müssten sie ohne weitere Balanciermaßnahmen umfallen, da der Schwerpunkt durch die Erhöhung zu sehr nach hinten gekippt wird. Die Fläche der beiden vorherigen Erhöhungen ist dabei 16

cm lang, damit ein gesamter Fuß vom NAO darauf abgestellt werden kann. Der hintere Teil der Plattform ist weiterhin eine harte Kante, wodurch die Roboter 1 cm tief nach vorne fallen werden. Dadurch sollen die Roboter möglichst stark nach vorne kippen, damit die Kräfte der Motoren des Standfußes nicht ausreichen und die Roboter ohne weitere Anpassungen im Laufen umfallen müssten. Das fünfte Hindernis ist dasselbe wie das vierte, lediglich statt einer harten hinteren Kante liegt auch hier nun eine Rampe in der Höhe 0.5 cm vor, die die Roboter herunterlaufen sollen. Dies soll sicherstellen, dass die Roboter beim Herunterlaufen weiterhin Bodenkontakt haben und nicht nur die Hacken und Fußspitzen den Boden berühren, was zu Problemen in der Fußwechseldetektion führen könnte. Das sechste Hindernis besteht aus einzelnen kleinen Holzklötzen in der Höhe 3 mm. Hier soll die Laufstabilität getestet werden, für den Fall, dass nur jeweils ein Fuß auf eine Erhöhung trifft, was eine theoretische Verschiebung des Schwerpunktes auf den zu dem Zeitpunkt bestimmten Schwingfuß erzeugen müsste. Es sollen also schlechte Schrittwechsel entstehen und eine allgemeine Störung im Schwingverhalten des Laufens. Das siebte und letzte Hindernis ist dasselbe wie zuvor, nur statt einer Höhe von 3 mm wird eine von 6 mm für die Holzklötze verwendet.

Um einen möglichst guten Überblick über die Laufstabilität zu gewinnen, werden die Roboter in vier verschiedenen Konfigurationen mit der normalen Wettbewerbsgeschwindigkeit von 250 mm/s laufen gelassen. Die vier Konfigurationen sind dabei wie folgt: Das bisherige Laufen ohne die entwickelten Anpassungen (LoA), das bisherige Laufen mit nur der Laufschrittanpassung (LS), das bisherige Laufen mit nur der Rotationskompensation des Schwingfußes (LR), und zu guter Letzt das bisherige Laufen mit allen entwickelten Erweiterungen (LaE).

Da die Roboter unterschiedliche Abnutzungsgrade besitzen und sich potentiell nach jedem Durchgang verschlechtern können, sei es durch weitere Abnutzungen, heiße Gelenke oder andere unbekannte Einflüsse, werden die verschiedenen Konfigurationen wie folgt getestet: Es werden vier Roboter verwendet, die in der Entwicklung nicht benutzt wurde. So sollen Parameter, die auf einen einzelnen Roboter optimiert wurden, das Ergebnis weniger verfälschen. Auf jedem Roboter wird jede Konfiguration mehrmals getestet. Bei wiederholenden Durchgängen wird die Reihenfolge der zu testenden Konfiguration geändert, um den Einfluss von Abnutzungen und co. zu verringern. Für die vier Roboter wird folgendermaßen getestet:

Roboter 1 durchläuft LoA-LS-LR-LaE-LaE-LR-LS-LoA.
Roboter 2 durchläuft LaE-LR-LS-LoA-LoA-LS-LR-LaE.
Roboter 3 durchläuft LS-LoA-LaE-LR-LR-LaE-LoA-LS.
Roboter 4 durchläuft LR-LaE-LoA-LS-LS-LoA-LaE-LR.

Dadurch ergeben sich insgesamt 32 Durchläufe, jeweils acht für jede Konfiguration. Da es passieren kann, dass Roboter unter bestimmten Konfigurationen häufiger umfallen und somit länger aktiv auf der Strecke bleiben, wird nach jedem einzelnen Durchgang der Roboter gewechselt, damit sich der zuvor benutze abkühlen und der Akku aufladen kann. Theoretisch könnte man auch längere Pausen durchführen, damit die Gelenke auf den niedrigst möglichen

Temperaturen bleiben. Dies ist aber aus zeitlichen Gegebenheiten nicht möglich. Zusätzlich, um die Hardware zu schonen, werden die Roboter beim Umfallen per Hand vom Menschen aufgefangen. Ebenfalls wird für jedes Hindernis nur ein Umfallen gezählt, weshalb ein Roboter nach einem Sturz vor das nächste Hindernis gestellt wird. Ausnahme sind die ersten drei Meter der Strecke. Da es hier keine Hindernisse gibt, wird ein Roboter, der in diesem Bereich umfällt, an derselben Stelle wieder aufgestellt. Somit kann ein Roboter theoretisch unendlich oft auf den ersten drei Metern umfallen, aber maximal siebenmal auf den restlichen sechs Metern. Fällt ein Roboter mehrmals auf den ersten drei Metern um, wird dieser nach dem vierten Umfallen an das Ende der Teilstrecke gestellt, um die Hardware nicht zu zerstören.

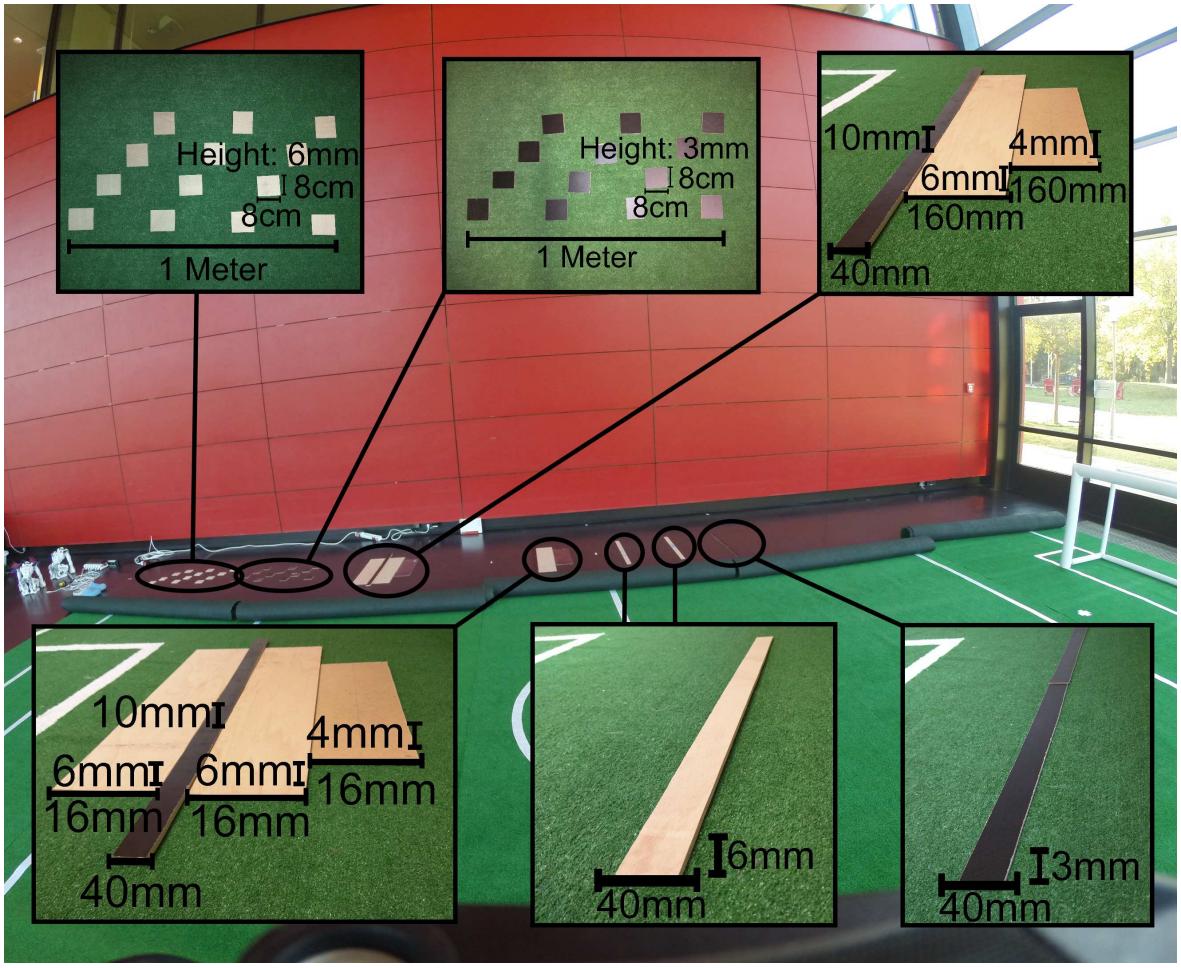

Abbildung 5.16 Der Aufbau des Parcours. Die Roboter laufen von rechts nach links, während die dargestellten Hindernisse unter dem Feldboden liegen.

Zusätzlich zu diesem Durchlauf werden auch noch zwei weitere durchgeführt. Der erste Durchlauf ist basierend auf der bisherigen Wettbewerbsgeschwindigkeit. Es soll untersucht werden, ob die entwickelten Erweiterungen eine signifikant höhere Laufgeschwindigkeit ermöglichen. Daher wird auf derselben Strecke die Versuchsreihe wiederholt, mit den Laufgeschwindigkeiten von 300 mm/s und 350 mm/s, welche 20 % respektive 40 % höher sind als die bisherige. Da aus dem ersten Durchlauf bereits ersichtlich ist, dass lediglich die Kombination aus der

Laufschrittanpassung und der Rotationskompensation des Standfußes eine Verbesserung in der Laufstabilisierung ergibt, wird hier nur noch zwischen LoA und LaE verglichen. Um die Hardware der Roboter zu schonen, werden die Versuchsreihen auf ein Minimum reduziert. So werden auf jeweils zwei Robotern beide Konfigurationen zweimal getestet, jede Konfiguration also insgesamt vier Mal. Um erneut den Einfluss der sich verändernden Hardware zu verringern, werden die Konfigurationen folgendermaßen getestet:

Roboter 1 durchläuft LoA-LaE-LaE-LoA.
Roboter 2 durchläuft LaE-LoA-LoA-LaE.

5.8 Ergebnisse

Die Ergebnisse des ersten Experimentes sind in Tabelle 5.1 zu sehen. Beide entwickelten Erweiterungen zusammen scheinen das Laufen signifikant zu stabilisieren [1]. Die Roboter fielen im Vergleich zum alten Laufen nur noch zu einem Drittel um. Bei beiden einzelnen Verfahren, der Laufschrittanpassung und der Standfußkompensation, fielen die Roboter ähnlich oft wie beim alten Laufen. Die Standfußkompensation besitzt weiterhin das Problem, dass die Roboter beim nach hinten Umfallen weiter geradeaus laufen. Bei der Laufschrittanpassung treten die Roboter weiterhin in den Boden, wenn sie zu weit nach vorne geneigt sind.

Auffällig ist auch, dass das Hindernis mit den 3 mm Holzklötzen den Robotern keine Probleme bereitete. Das Hindernis mit den 6 mm Holzklötzen schien mit der Laufschrittanpassung am besten absolvierbar gewesen zu sein. Aus den Video- und den Logaufnahmen lässt sich aber vermuten, dass die Roboter teilweise neben der Strecke liefen und somit nicht dem Hindernis vollständig ausgesetzt waren. Dies gilt für die meisten Durchläufe, weshalb bei diesem Hindernis eigentlich deutlich mehr Roboter hätten umfallen müssen. Auch ist kein Roboter auf den ersten drei Metern oder beim ersten Hindernis mit der 3 mm Kante umgefallen.

Das Umfallen einiger Roboter kann auch darauf zurückgeführt werden, dass die Hindernisse zu nahe einander lagen, wodurch die Roboter noch leicht instabil durch das vorherige Hindernis liefen. Allgemein hätten alle Hindernisse weiter auseinander positioniert sein müssen, um den Robotern genügend Zeit zu geben, sich wieder zu stabilisieren. So haben die Hindernisse gegenseitig Einfluss genommen, was nicht beabsichtigt war.

Die Ergebnisse des zweiten Experimentes, mit den erhöhten Laufgeschwindigkeiten, sind in Tabelle 5.2 zu sehen. Mit den neuen Entwicklungen sind die Roboter zwar weiterhin weniger umgefallen, aber hochgerechnet trotzdem mehr als bei der Laufgeschwindigkeit von 250 mm/s. Auch fielen beim alten Laufen die Roboter bei den beiden 6 mm Kanten mit der

[1] Ein Video für den Vergleich des alten und neuen Laufens findet sich unter https://youtu.be/N_Q7qLDYqyY

Konfiguration	LoA	LS	LR	LaE	Gesamt
Ersten drei Meter	0	0	0	0	0
3 mm Kante	0	0	0	0	0
Erste 6 mm Kante	6	4	4	2	16
Zweite 6 mm Kante	5	4	4	1	14
Erste 1 cm Rampe	3	4	5	0	12
Zweite 1 cm Rampe	3	5	3	0	11
3 mm Holzklötze	0	0	0	0	0
6 mm Holzklötze	4	1	4	4	13
Gesamt	21	18	20	7	66

Tabelle 5.1 Die Anzahl umfallender Roboter für die unterschiedlichen Laufkonfigurationen, mit einer Laufgeschwindigkeit von 250 mm/s. Jede Konfiguration wurde achtmal ausgeführt.

Geschwindigkeit von 300 mm/s weiterhin kein einziges Mal. Aus den Videoaufnahmen lässt sich nur vermuten, dass die Füße zufällig ideal platziert wurden, da während der Entwicklung ein solches Hindernis einige Male zu Destabilisierungen geführt hatte. Die Roboter kippten zwar erst nach hinten, aber der Folgelaufschritt setzte den nächsten Fuß bereits hinter das Hindernis, wodurch der Roboter, aufgrund der höheren Laufgeschwindigkeit, sofort wieder nach vorne kippen konnte.

Auch sind hier Roboter bei den 3 mm Holzklötzen umgefallen. Dies kann aber auf den Aufbau der Strecke zurückgeführt werden, da die Hindernisse, wie bereits erwähnt, weiter auseinander hätten liegen müssen.

Konfiguration	LoA 300 $\frac{mm}{s}$	LoE 300 $\frac{mm}{s}$	Loa 350 $\frac{mm}{s}$	LoE 350 $\frac{mm}{s}$	Gesamt
Ersten drei Meter	0	0	0	0	0
3 mm Kante	0	0	0	0	0
Erste 6 mm Kante	0	1	2	2	5
Zweite 6 mm Kante	0	1	3	0	4
Erste 1 cm Rampe	3	0	4	0	7
Zweite 1 cm Rampe	1	0	2	1	4
3 mm Holzklötze	2	0	0	1	3
6 mm Holzklötze	3	3	2	3	11
Gesamt	9	5	13	7	34

Tabelle 5.2 Die Anzahl umfallender Roboter mit und ohne die Anpassungen am Laufen, mit einer Laufgeschwindigkeit von 300 mm/s und 350 mm/s. Beide Konfigurationen wurden für beide Geschwindigkeiten jeweils viermal ausgeführt.

Dass sich die Umfallrate mit den entwickelten Maßnahmen mit zunehmender Laufgeschwindigkeit weiterhin erhöht, scheint auf zwei Probleme zurückzuführen zu sein. Zum einem gibt es keine Regulierung für das Schwingverhalten des Roboters. Dadurch können einzelne Laufschritte zu lange dauern, wodurch die Roboter häufig diagonal umfielen oder sogar vollständig

seitlich. Zum anderen versagen die Motoren des Standfußes auch, wenn die Roboter zu sehr nach hinten geneigt sind, während der Laufschritt noch ausgeführt wird. Solche Zustände kamen aber hauptsächlich zustande, weil die Füße zu spät angepasst wurden. Ein solcher Fall ist in Abbildung 5.17 zu sehen, bei dem ein Roboter beim Durchgang mit einer Laufgeschwindigkeit von 350 mm/s umfiel. Der vorhergesagte Schwerpunkt war innerhalb des definierten Toleranzbereiches, während die reine Oberkörpermasse, der Schwerpunkt ohne die Beinmasse, bereits außerhalb war. In dem dargestellten Fall war ein Fuß sehr weit vorne und die Füße allgemein sehr weit auseinander, resultierend durch die hohe Laufgeschwindigkeit. Dadurch verlagert sich der berechnete Schwerpunkt nach vorne, gleichzeitig wird die Oberkörperrobotermasse nicht einmal mehr durch die Hacke des hinteren Fußes gestützt, sondern die Masse der Füße allein verhindern ein Umfallen. Die Laufschrittanpassung berechnet keine Anpassung und der Roboter läuft weiter geradeaus. Kurze Zeit später ist der Roboter genügend nach hinten gekippt, wodurch die Laufschrittanpassung aktiv eingriff, jedoch viel zu spät. Auch eine Erhöhung der Vorhersage verhindert dieses Problem nicht vollständig, da eine Vorhersage von neun statt drei Motion-Frames nur drei Motion-Frames früher eingegriffen hätte, auf Kosten von mehr Ungenauigkeit (siehe Abbildung 5.15). Zwischen den Zeitpunkten, zu denen die Oberkörpermasse und der vorhergesagte Schwerpunkt den Toleranzbereich verlassen, vergehen in diesem Fall sechs Motion-Frames. In dieser Zeit haben sich beide Füße bereits um jeweils 6 cm bewegt, was 40 % der Stützfläche eines Fußes beträgt.

Da die Umfallrate bei höheren Laufgeschwindigkeiten zunimmt, scheint das zu späte Eingreifen der Laufschrittanpassung noch ein großes Problem zu sein. Inwiefern wiederum ein früheres Eingreifen die Umfallrate beeinflussen würde, ist fraglich, da nicht klar ist, wie sich das Laufverhalten der Roboter verändern würde. Dennoch scheint das angepasste Laufen mit einer 40% höheren Laufgeschwindigkeit genauso stabil zu sein wie das bisherige Laufen unter normaler Laufgeschwindigkeit.

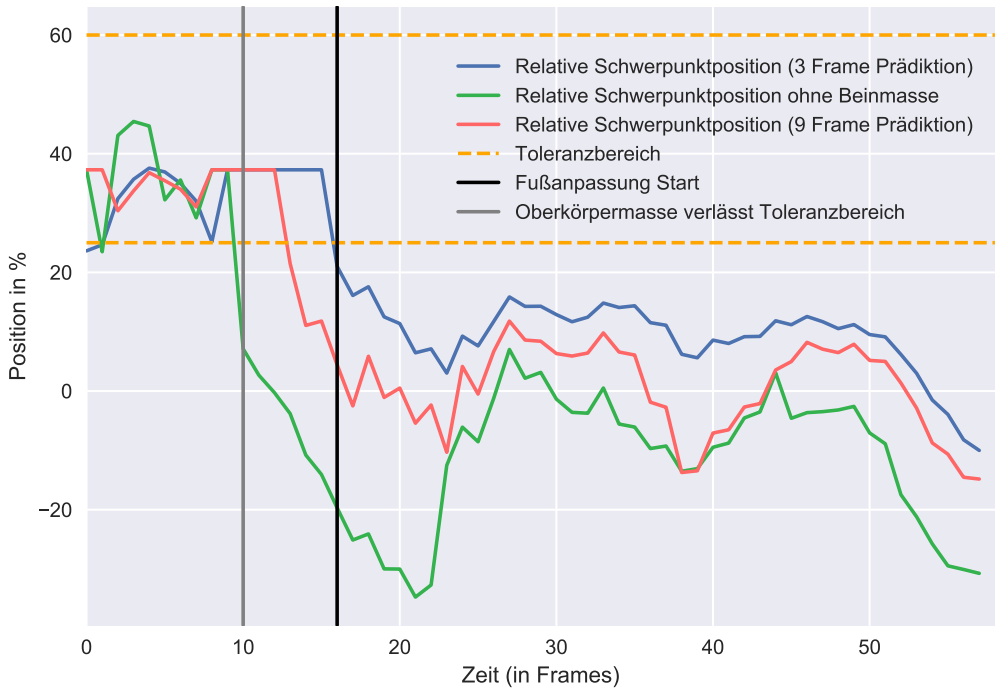

Abbildung 5.17 Verschiedene Schwerpunktmassen im Vergleich. Der Zeitpunkt, zu dem die Laufschrittanpassung eingriff, ist durch die schwarze Markierung dargestellt. Die Oberkörpermasse verließ hier aber sechs Motion-Frames zuvor (graue Markierung) den definierten Toleranzbereich.

5.9 Fazit

Die entwickelten Ansätze, die Laufschrittanpassung und Standfußkompensation, stabilisieren das aktuelle Laufen und erlauben eine höhere Laufgeschwindigkeit, erfüllen gleichzeitig aber auch die Anforderung, nur minimal einzugreifen. Dennoch ist noch viel Luft nach oben. So ist die entwickelte Laufschrittanpassung ausreichend, um die Roboter bei kleineren Stößen zu stabilisieren. Bei größeren braucht es aber einen anderen Ansatz, da die Roboter zum einem zu schnell umfallen, wodurch die Laufschrittanpassung sowohl zu langsam als auch zu wenig agiert, zum anderen muss die Fehlpositionierung des Standfußes bedacht werden.

Die Entwicklung ermöglicht somit nur ein allgemein schnelleres Laufen, welches resistenter gegenüber leichten Einflüssen ist. Größere Einwirkungen, wie auf andere Roboterfüße zu laufen, führen weiterhin zu umfallenden Robotern.

In Abschnitt 3.1 wurde festgestellt, dass Kollisionen die Hauptursache für das Umfallen der Roboter bei B-Human sind, worunter das auf die Füße Laufen fällt. Für das zukünftige Spielverhalten ist also absehbar, dass die B-Human Roboter zwar schneller laufen können, aber weiterhin ähnlich oft umfallen werden.

Aus der Evaluation zeigt sich für das entwickelte Verfahren auch, dass die Laufschrittanpassung teilweise zu langsam agiert, weil die Füße zu viel Einfluss auf den Schwerpunkt nehmen, wodurch insbesondere bei hohen Geschwindigkeiten zu spät eingegriffen wird. Dieses Problem könnte minimiert werden, indem ein alternativer Schwerpunkt verwendet wird, welcher nicht die Füße miteinbezieht. Auf die Schwerpunktpositionen nimmt dies, wie es in Abbildung 5.18 zu sehen ist, fast keinen Einfluss. Ob sich dadurch eine Verbesserung bei insbesondere höheren Laufgeschwindigkeiten ergibt, könnte noch zusätzlich untersucht werden.

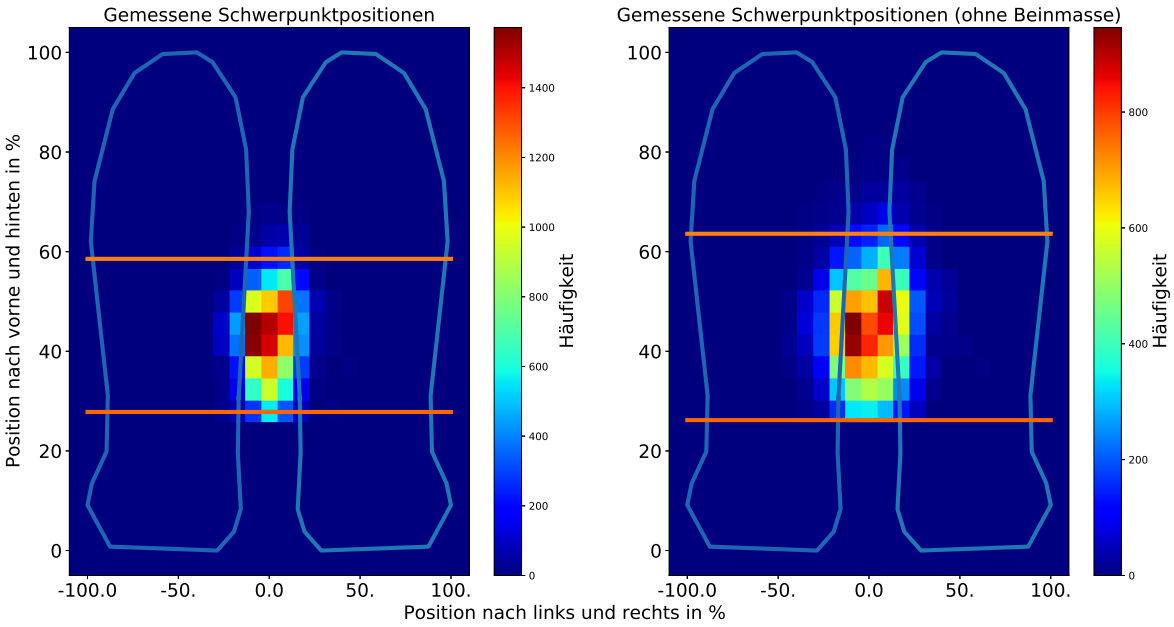

Abbildung 5.18 Die Schwerpunktpositionen relativ zur Stützfläche mit dem Schwerpunkt ohne die Beinmasse (rechts), verglichen mit dem normalen Schwerpunkt (links) aus Abbildung 5.11. Die neuen 2. Sigmaumgebungen (orange Linie) liegen bei 26 % und 64 %.

Kapitel 6

Schüsse

6.1 Die aktuellen Schüsse

Die aktuellen Schüsse aus dem Laufen heraus durchlaufen bisher drei Abschnitte. Zuerst wird vom Verhaltensteil entschieden, welcher Schuss in welche Richtung ausgeführt werden soll. Dies ist je nach Kontext unterschiedlich. So kann auf das Tor geschossen, ein Pass ausgeführt oder einfach der Ball im Zweikampf mit einem Gegner getreten werden. Dieser Kontext ist den Schüssen selber nicht bekannt.

Im zweiten Abschnitt wird im Motion-Thread nach einem Schrittwechsel, aber vor dem nächsten Laufschritt, geprüft, ob der aktuell angefragte Schuss ausgeführt werden kann. Dies geschieht durch eine einfache Abfrage über Schwellwerte. Hier muss die aktuelle ausführbare Schussrichtung und die Positionierung in der x- und y-Translation zum Ball lediglich in einen vordefinierten Wertebereich fallen. Die Wertebereiche sind dabei jeweils definiert durch eine Ober- und Untergrenze um die Referenzwerte herum, festgelegt durch eine statische Konfiguration. Dadurch dürfen jeweils die Schussrichtung als auch die Translationen zum Ball nur um einen Wert von ± Z zum Referenzwert abweichen. Sofern Richtung und Positionierung innerhalb der Schwellwerte fallen, wird der Schuss ausgeführt. Ansonsten wird ein Laufschritt berechnet, um den Roboter näher an die Referenzpositionierung zum Ball zu bringen.

Im dritten und letzten Teil kann nun der Schuss ausgeführt werden. Dafür wird aus der Konfiguration, die dem Schuss zugeordnet ist, eine Laufschrittgröße vorgegeben. Hier kann ein Schuss aus bis zu zwei Laufschritten bestehen. Der erste besteht dabei aus nur einer Laufschrittgröße oder wird komplett ausgelassen. Dies wird für jeden Schuss einzeln konfiguriert. Der zweite Laufschritt besitzt immer eine Laufschrittgröße sowie eine kubische Spline-Beschreibung für den Schussfuß. Diese kubische Spline-Beschreibung dient als Versatz in der Translation für den Schwingfuß und wird auf die berechnete Ansteuerung des Schwingfußes des Laufmoduls addiert. Hier können beliebig viele Punkte im kartesischen Koordinatensystem definiert werden. Die zeitlich zugeordnete Position wird dabei durch eine prozentuale Angabe in der Laufschrittdauer angegeben. Die Spline dient dabei als Versatz, welcher bei

0 anfängt und 0 endet. Dadurch kann der Schussfuß zum Beispiel nach vorne beschleunigt werden, um den Ball mit möglichst viel Kraft zu treten. Gleichzeitig wird sichergestellt, dass am Ende des Laufschrittes die Füße so positioniert sind, als wäre ein normaler Laufschritt ausgeführt worden. Die Laufschrittgröße wird von dem normalen Laufmodul umgesetzt, während die Spline-Position von dem Schussmodul berechnet und auf die Fußpositionen addiert wird. Die Spline-Positionen sind dabei statische Positionen, welche beim gleichen Schusstyp immer identisch sind.

6.2 Die Anforderungen an ein neues Schießen

Eine große Schwäche dieses Verfahrens sind die statischen Laufschritte. Liegt der Ball nicht ideal, kann ein Fehltritt nur verhindert werden, indem die Schwellwerte für die Startbedingung niedrig genug gesetzt sind. Für Schüsse, die aus zwei Laufschritten bestehen, verhindern aber selbst die Schwellwerte keine Fehltritte. Falls sich der Ball im ersten Laufschritt noch bewegt oder sich der Roboter währenddessen dreht oder verschiebt, was auf dem Kunstrasen mit den glatten Sohlen üblich ist, wird der Ball unweigerlich schlecht getroffen. Dadurch entsteht eine große Abweichung in der Schussrichtung, welche vermeidbar sein müsste.

Eine stetige Anpassung an die Ballposition während eines Schusses ist aber nicht denkbar. Die aktuellen Laufschritte dauern geplant 250 ms. Dazu kommt die Motorenverzögerung von 36 bis 48 ms und die Trägheit der Gelenke, welche sich nicht sofort in eine andere Richtung bewegen können. Das heißt, die letzten angefragten Gelenkwinkel eines Laufschrittes, welche noch vor einem Schrittwechsel ausgeführt werden, liegen bei 202 bis 214 ms nach Laufschrittanfang. Ebenfalls ist der Ball eine Kugel und der zum Roboter am nächsten liegende Punkt befindet sich auf der Höhe des Mittelpunktes, welcher beim aktuellen Ball 5 mm beträgt. Dadurch muss der Ball mit mindestens einem leichten, zeitlichen Abstand zum Laufschrittende berührt werden, damit der Fuß noch eine Höhe von 5 mm zum Boden besitzt und genug Zeit hat, diese Höhe auf null vor dem Laufschrittende zu bringen. Auch sollte der Ball immer mit etwas zeitlichem Abstand vor dem Laufschrittende berührt werden, da die Laufschritte auch etwas vor den geplanten 250 ms enden können. Falls also eine Laufschrittkorrektur während des Laufschrittes nötig wäre, könnte diese nur am Anfang umgesetzt werden.

Auch ist aus den vergangen Wettbewerben zu beobachten, dass die Schussreichweiten stark streuen. So rollt der Ball bei einem Drehschuss das eine Mal die geplanten 100 cm, das andere Mal über das halbe Feld für eine Strecke von fast 300 cm.

Somit ergeben sich drei Aspekte, die für ein neues Schuss-Framework zu berücksichtigen sind. Die geplanten Laufschritte, basierend auf einer Konfigurationsdatei, sollten relativ zum Ball berechnet werden. Dadurch wird der Einfluss minimiert, falls der Roboter nicht perfekt zum Ball ausgerichtet ist, sowie wenn er bei einem Schuss, der aus zwei Laufschritten besteht, während des ersten Laufschrittes rutscht, dreht oder sich der Ball verschiebt. Der zweite Aspekt bezieht sich auf die Entscheidung, ob der Schuss ausgeführt oder abgebrochen werden kann.

Das Wegfallen von statischen Laufschritten erlaubt größere Abweichungen in der Positionierung zum Ball. Bei mehrschrittigen Schüssen kann nach jedem Laufschritt erneut geprüft werden, ob der Ball erreichbar ist. Der dritte Aspekt bezieht sich auf die Reichweitenstreuung. Durch das Berechnen von Laufschrittgrößen basierend auf der relativen Ballposition soll der Ball konstanter während des Laufschrittes berührt werden. Ob das für eine geringere Streuung ausreicht, wird am Ende in der Evaluation (siehe Abschnitt 6.4) untersucht.

6.3 Das neue Schussmodul

Das neue Schussmodul soll im Gegensatz zum aktuellen nicht mehr nur einen Laufschritt anfragen und auf diesen Offsets in den Translationen addieren, sondern mehrere Laufschritte und zwischen diesen interpolieren. Die Bewegung der Füße wird somit vollständig vom Laufmodul übernommen. Mit den entwickelten Anpassungen aus Abschnitt 5.1 sollen dadurch mehr Freiheiten ermöglicht werden, wie sich die Füße bewegen, gleichzeitig verhindern, dass der Roboter instabil wird und umfällt. Die Menge der Laufschritte kann dabei als eine Menge von Key-Frames gesehen werden. Auch sollen die Konfigurationen der Schüsse keine festen Laufschrittgrößen mehr beinhalten, sondern relative Positionen zum Ball, aus denen sich dann die Laufschrittgrößen dynamisch bestimmt lassen. Dadurch soll die korrekte Berührung des Balles mit den Füßen gewährleistet werden, was unter anderem größere Abweichungen im Anlaufen an den Ball erlaubt. Auf diese Weise müssen die Roboter in der Theorie nicht mehr sehr genau vor dem Ball stehen, sondern können auch mehrere Zentimeter falsch positioniert sein, den Ball aber weiterhin korrekt treffen. Inwiefern Einflüsse wie Rutschen auf dem Kunstrasen zu Problemen führen können, wird dabei später untersucht. Zusätzlich erlaubt dieser Ansatz eine dynamische Schussstärke. Diese kann erreicht werden, indem basierend auf einem Stärkewert, welcher zwischen 0 und 1 liegen muss, zwischen jeweils zwei relativen Ballpositionen interpoliert wird. Für den Stärkewert 0 sowie für 1 ist dabei jeweils eine relative Position vorgegeben. Da der bisherige Schussansatz unter anderem sehr starke Schüsse erzeugt, indem Offsets auf den Schwingfuß addiert werden, wird dies auch hier unterstützt. Die bisher verwendeten Offsets erzeugen Laufschritte, bei denen Stand- und Schwingfuß unterschiedliche Laufschrittgrößen umsetzen. Für starke Schüsse führt der Schwingfuß dadurch einen sehr großen Laufschritt aus, während der Standfuß einen normalen kleinen ausführt. Für das neue Schussverfahren werden diese unterschiedlichen Laufschrittgrößen erzeugt, indem mithilfe der relativen Positionen zwei verschiedene Laufschrittgrößen berechnet werden. Mit den normalen relativen Positionen werden die Laufschrittgrößen für den Schwingfuß bestimmt. Zusätzlich werden auf die relativen Positionen Offsets addiert und aus den verschobenen Positionen die Laufschrittgrößen des Standfußes berechnet. So führt der Schwingfuß einen sehr großen Laufschritt aus, der Standfuß nur einen kleinen. Abschnitt 6.3.7 erläutert genauer, wie dadurch schwächere oder stärkere Schüsse erzeugt werden. Die Genauigkeit der Schüsse bleibt dabei erhalten, aufgrund der Schnittstellendefinition des Laufmoduls, welche in Abschnitt 6.3.1 genauer erklärt wird.

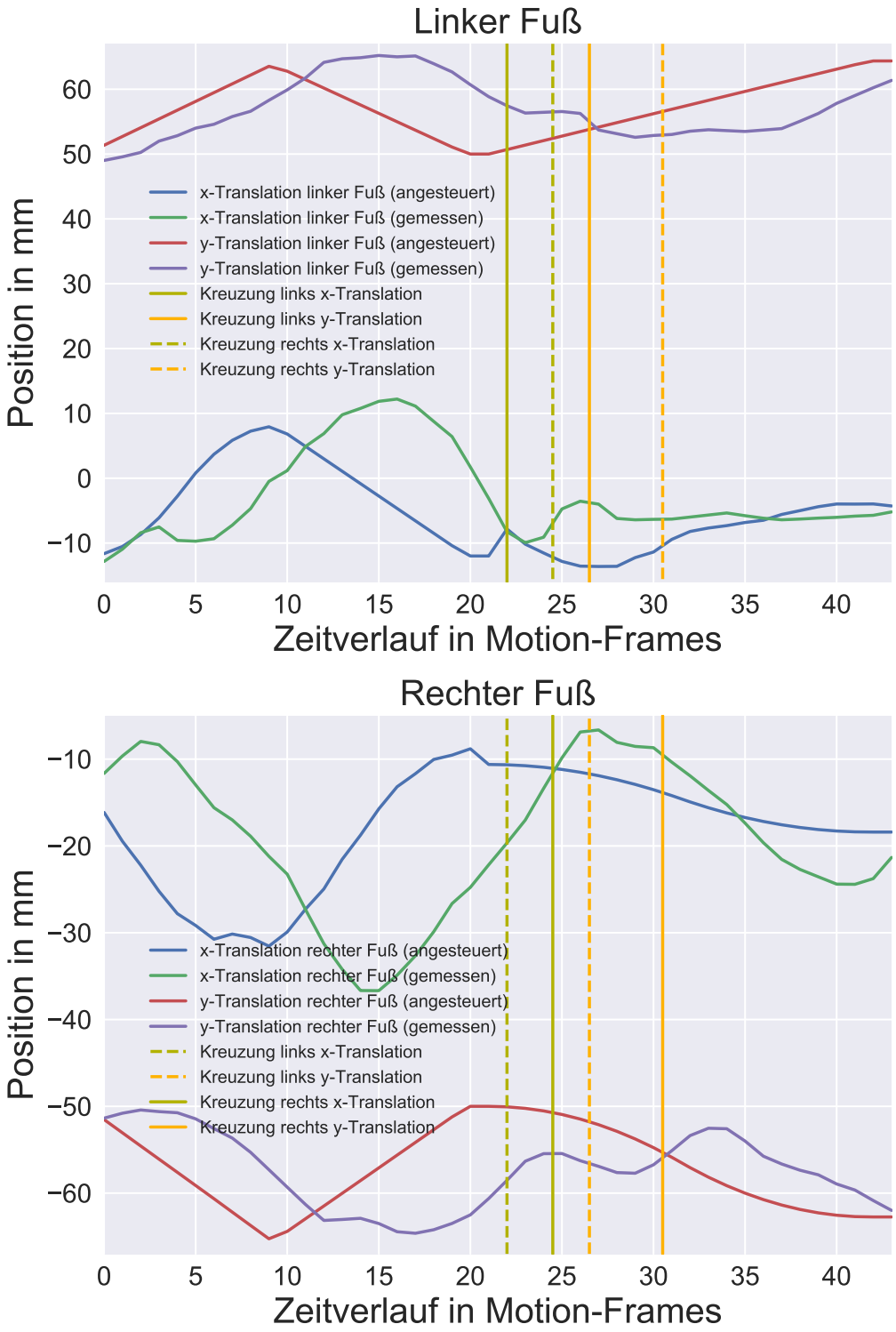

Abbildung 6.1 Vergleich der Ansteuerung und Umsetzung beider Füße. Bis zum Motion-Frame 22 ist der linke Fuß der Schwingfuß. Am Anfang eines Laufschrittes wurde ein diagonaler Laufschritt angefordert, der nach der Hälfte der Laufschrittdauer wieder auf null zurück gesetzt wurde.

Für die Interpolation zwischen den relativen Positionen innerhalb eines Laufschrittes könnten, wie bei den bisherigen Schüssen, Splines verwendet werden. Diese haben aber das Problem, dass die Roboter die Anfragen gar nicht so genau umsetzen können, wie es in Abbildung 6.1 zu sehen ist. Wie schon erwähnt, dauert ein Laufschritt geplant 250 ms. Richtungswechsel innerhalb eines Laufschrittes brauchen in x-Translation beider Füße, egal ob Schwing- oder Standfuß, bis zu 72 ms (inklusive der Verzögerung in der Ansteuerung). Da sich der Schwingfuß aber ohne Bodenwiderstand bewegen kann, erreicht er seine Zielposition 24-36 ms vor dem Standfuß. In der y-Translation dauert der Richtungswechsel ebenfalls 72 ms für den Schwingfuß, aber 96 ms für den Standfuß. Auch hier erreicht der Schwingfuß die Zielposition deutlich vor dem Standfuß, mit einem Abstand von 60 ms. Auch werden die Zielpositionen jeweils erst im Folgelaufschritt erreicht.

Auf eine flüssige Interpolation sowie auf Richtungswechsel innerhalb eines Laufschrittes kann und muss daher verzichtet werden. Stattdessen wird eine lineare Interpolation verwendet. Richtungswechsel werden durch die Konfiguration der Schüsse vermieden.

Die Einbindung des Schussmoduls ist letztendlich wie in Abbildung 6.2 dargestellt. Am Anfang eines Motion-Frames wird auf einen Schrittwechsel geprüft. Ist dies der Fall, so wird entweder ein aktuell ausgeführter Schuss fortgeführt oder geprüft, ob einer ausgeführt werden soll. Trifft eines von beiden zu, aktiviert sich das Schussmodul. Falls noch kein Schuss ausgeführt wird, werden mehrere Vorbedingungen geprüft, ob der angefragte Schuss sinnvoll ausgeführt werden kann. Anschließend werden die Laufschrittgrößen berechnet. Sofern diese ausführbar sind, werden alle nötigen Informationen an das Laufmodul übergeben. In jedem Motion-Frame wird dann der Schuss basierend auf den Laufschrittgrößen ausgeführt und die entsprechenden Gelenkwinkel bestimmt. Kommt es zu einem Schrittwechsel, beginnt der Zyklus erneut. Falls kein Schuss ausgeführt werden soll oder kann, wird an den Ball angelaufen, wofür von einem anderen Modul eine Laufschrittgröße berechnet und an das Laufmodul übergeben wird.

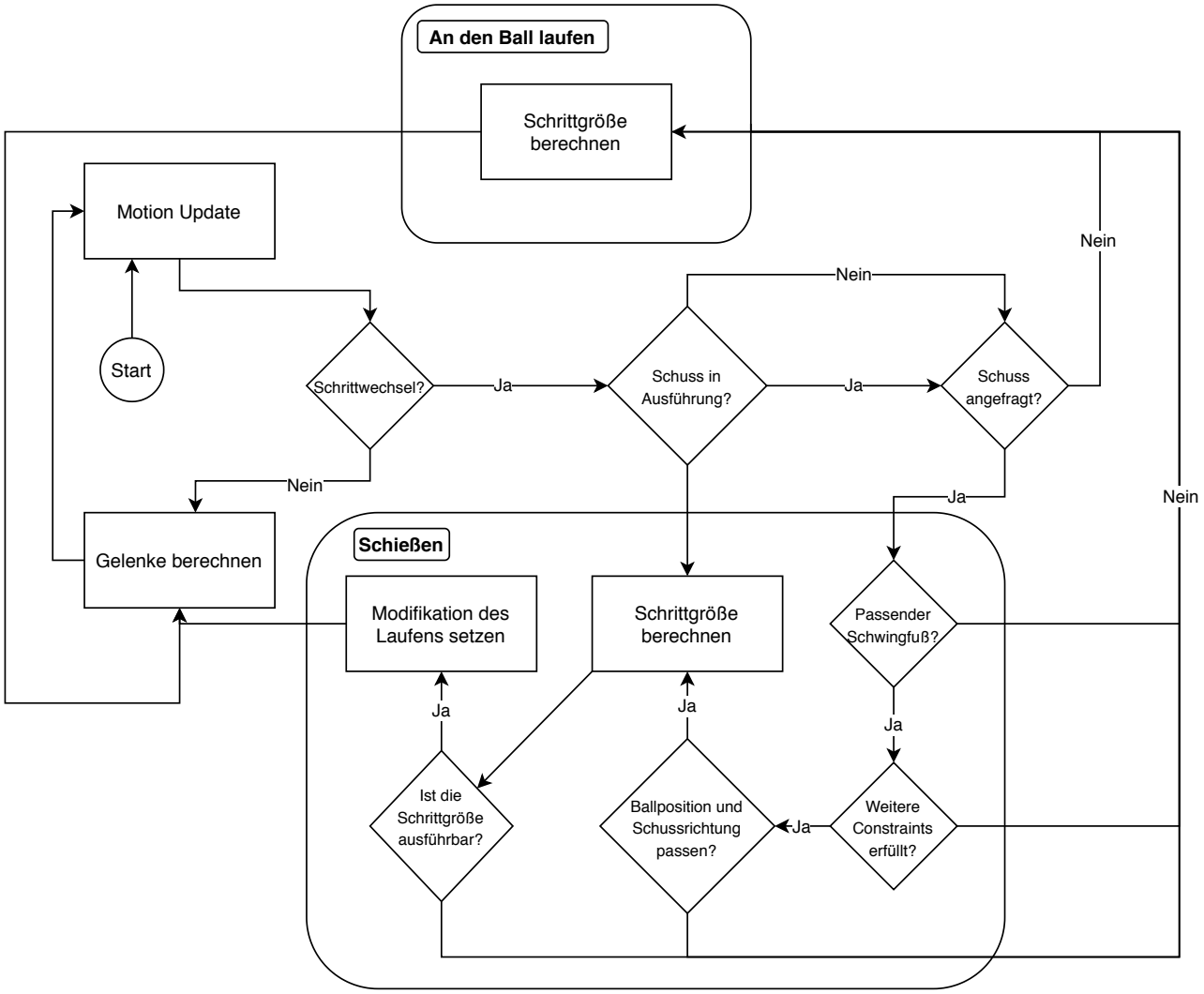

Abbildung 6.2 Der Ablauf für die Bestimmung und Berechnung der Schüsse. Bei einem Schrittwechsel wird auf die Fortführung oder Anfrage eines Schusses geprüft. Ist dies der Fall, so wird eine mögliche Ausführung berechnet, die Menge der Laufschrittgrößen bestimmt und diese an das Laufmodul übergeben. Falls die Ausführung nicht möglich ist, wird normal weiter an den Ball herangelaufen.

6.3.1 Laufschrittgrößenberechnung

Für die Laufschrittgrößenberechnung werden für jeden Schusstyp eine Menge von Laufschritten definiert, welche jeweils eine Menge von definierten Positionen relativ zum Ball besitzen. Ein Schuss wird daher vereinfacht definiert durch:

$$\begin{aligned} \text{Schuss} &= \{\text{Schritt}\} \\ \text{Schritt} &= \{\text{Pose}\} \\ \text{Pose} &= (\text{Rotation, x-Translation, y-Translation}) \end{aligned}$$

Um aus diesen relativen Positionen zum Ball nun Laufschrittgrößen zu berechnen, kann die Schnittstellendefinition des Laufmoduls ausgenutzt werden. Die hierfür übergebene Laufschrittgröße besagt im übertragenen Sinne lediglich, wo sich am Ende des Laufschrittes der Schwingfuß relativ zu dessen Ursprungsposition zum Boden befinden soll. Die Ursprungsposition ist hierbei die Fußposition, welche eingenommen werden würde, wenn der Roboter stehen bleibt. Diese kann jederzeit mithilfe der Laufschrittgröße (0,0,0) angefragt werden. Das Laufmodul bringt dann selbstständig beide Füße zurück auf ihre Ursprungspositionen.

Mithilfe dieses Zusammenhanges reicht es also, die Ballposition in ein Koordinatensystem zu bringen, welches im Ursprung der hypothetischen Ursprungsposition des Schwingfußes liegt. Von hier aus können die relativen Ballpositionen auf die Ballposition addiert werden und die resultierenden Positionen korrespondieren zu den benötigten Laufschrittgrößen. Dieser Ablauf ist in Abbildung 6.3 dargestellt. Zum Zeitpunkt der Laufschrittgrößenberechnung (*Ausgangslage*) wurde ein Schrittwechsel gemessen, wodurch Schwing- und Standfuß eindeutig sind. Das Koordinatensystem, dargestellt durch die beiden Pfeile, welches zwischen beiden Füßen liegt, wird zuerst in die Ursprungsposition des Schwingfußes verschoben. Da beide Füße parallel zueinander stehen müssen, mit einem Abstand von 10 cm, definiert durch die Spezifikation des Roboters, kann einfach von der Pose des Standfußes um 10 cm in y-Richtung transliert werden. Der Ball befindet sich nun im Koordinatensystem der Ursprungsposition des Schwingfußes (*Nullpunkt*). Von hier aus kann die relative Position zum Ball auf den Ball addiert werden (*Relative Position*). In der Abbildung wurde hierfür als Beispiel eine relative Position von (-100. , 0.) verwendet. Auf die Ballposition addiert erhält man dann die orangene Position. Angenommen, der Schuss benötigt zusätzlich noch eine Schussrichtung von 30 Grad, dann müsste die relative Position (-100. , 0.) nur um 30 Grad rotiert werden, wodurch man eine von (-86.603 , 50.) erhält. Addiert auf die Ballposition ergibt sich dann die hellgrüne Position. Schlussendlich kann für die Bestimmung der Laufschrittgröße (*Schrittgröße*) einfach die berechnete Position (orange bzw. hellgrün) übernommen werden. Die Rotation für die Laufschrittgröße entspricht der Schussrichtung, also 0 Grad respektive 30 Grad.

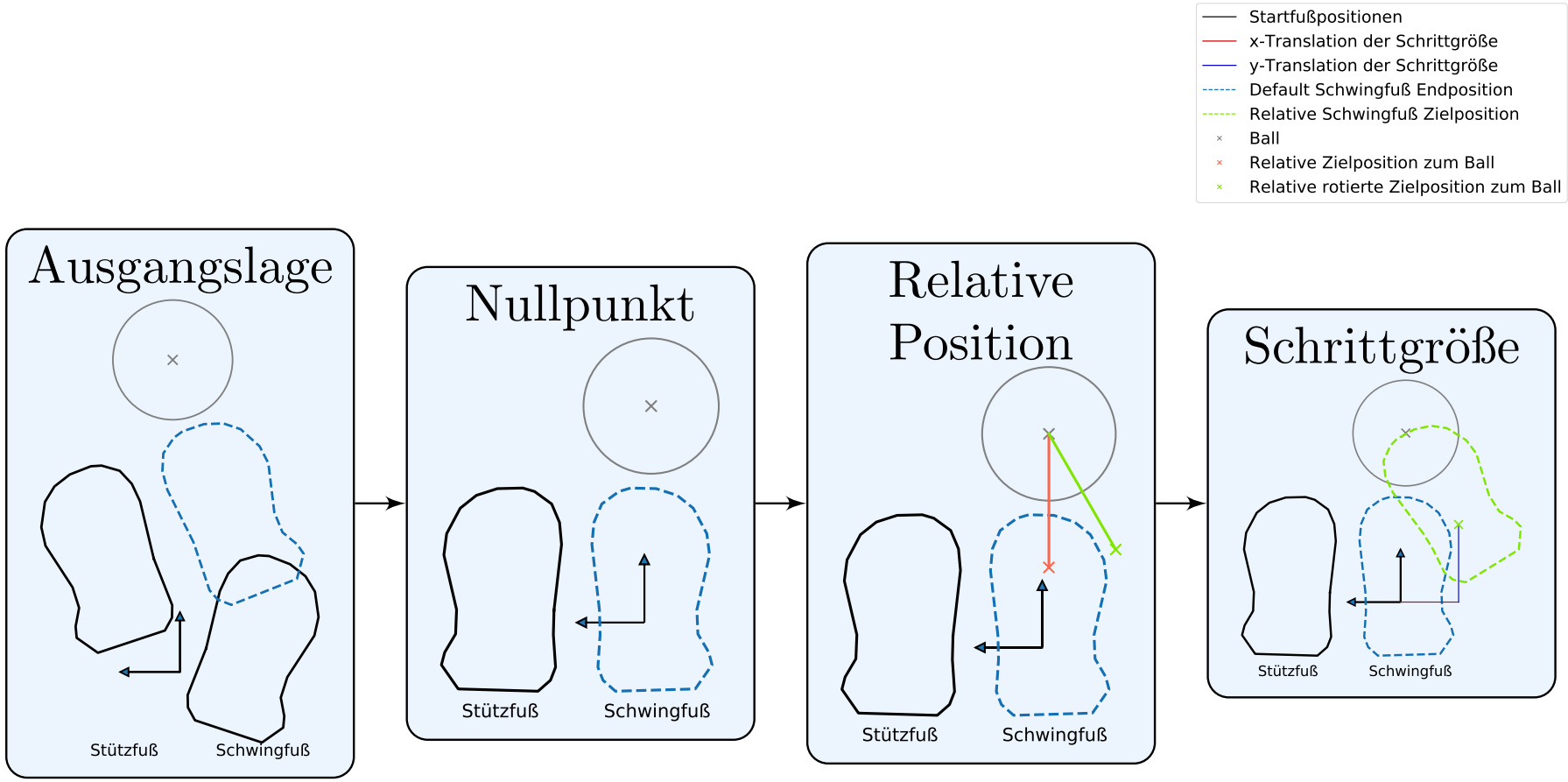

Abbildung 6.3 Berechnungsablauf, um aus den aktuellen Fuß- und Ballpositionen eine Laufschrittgröße zu bestimmen, basierend auf der konfigurierten relativen Ballposition. Die beiden Pfeile zeigen jeweils den Ursprung des aktuellen Koordinatensystems.

Mathematisch gesprochen kann dieser Ablauf durch folgende Gleichung erreicht werden. Für die Ursprungsposition gilt:

$$\text{ursprungSchwing}_{\rho 2D} = \text{stütz}_{\rho 2D}^{-1} \cdot \text{hüftVerschiebung} \tag{6.1}$$

Mit

$$\text{hüftVerschiebung} = \begin{cases} (0, 100.) & \text{, falls linker Fuß der Stützfuß ist} \\ (0, -100) & \text{, sonst} \end{cases} \tag{6.2}$$

$$\text{stütz}_{\rho 2D} = \text{Pose}(\text{stütz}_{\rho}^{(\gamma)}, \text{stütz}_{\rho}^{(x)}, \text{stütz}_{\rho}^{(y)}) \tag{6.3}$$

Sei α die Schussrichtung und $ball \in P^2$ im Roboterkoordinatensystem auf dem Feld, dann gilt für die Laufschrittgröße:

$$\text{ballOffset} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \text{relativePosition} \tag{6.4}$$

$$\text{Laufschrittgröße} = \text{ursprungSchwing}_{\rho 2D} \cdot \text{ball} \cdot \text{ballOffset} \tag{6.5}$$

Die Berechnung der Laufschrittgröße muss nun zusätzlich noch durch die Odometrie erweitert werden. Durch das verwendete Framework vom Team B-Human kommen die Informationen der Ballposition aus dem Cognition-Thread, während die Laufschrittgrößenberechnung im Motion-Thread abläuft. Um die übergebene Ballposition korrekt zu verwenden, wird zum Zeitpunkt der berechneten Ballposition zusätzlich die Odometrie angegeben. Durch Subtraktion der Odometrie zum Zeitpunkt der Laufschrittgrößenberechnung entsteht eine korrigierte Ballposition. Die Odometrie zur Ballposition wird im Folgenden als $Odometrie_{\text{Cognition}}$ und die zum Motion-Thread als $Odometrie_{\text{Motion}}$ angegeben. Die Odometrie selber ist eine 2D-Pose. Gleichung 6.5 wird daher wie folgt erweitert:

$$\begin{aligned} \text{Laufschrittgröße} = \text{ursprungSchwing}_{\rho 2D} \cdot \text{ball} \cdot \text{Odometrie}_{\text{Motion}}^{-1} \\ \cdot \text{Odometrie}_{\text{Cognition}} \cdot \text{ballOffset} \end{aligned}$$

Ebenfalls muss auch die Schussrichtung mit der Odometrie korrigiert werden, welche zum selben Zeitpunkt wie die der Ballposition angegeben ist. Jedoch kann ein Schuss aus mehreren Laufschritten bestehen, weshalb hier die Odometrie verwendet werden muss, zum Zeitpunkt der ersten Ausführung des Schusses. Diese Odometrie wird im Folgenden als $Odometrie_{\text{CognitionStart}}$ bezeichnet. Für die Schussrichtung gilt also:

$$\alpha_{\text{korrigiert}} = \alpha - (\text{ursprungSchwing}_{\rho 2D}^{-1} \cdot (\text{Odometrie}_{\text{Motion}}^{-1} \cdot \text{Odometrie}_{\text{CognitionStart}}))^{(\alpha)} \tag{6.6}$$

Der Einfluss der Schussstärken und deren Interpolation wird in Abschnitt 6.3.7 behandelt.

6.3.2 Startbedingung

Um zu entscheiden, ob ein Schuss ausgeführt werden kann, wird die Überprüfung in zwei Segmente aufgeteilt. Zum einem in die Startbedingung und zum anderen in die Ausführungsbedingung (siehe Abschnitt 6.3.4). Die Startbedingung beinhaltet jede Prüfung, die erfolgen kann, bevor die Laufschrittgröße berechnet wird. Diese Prüfungen sind im Folgenden aufgeführt.

Um instabile Laufschrittfolgen zu verhindern, kann nur ein Schuss ausgeführt werden, wenn die vorherige Bewegung ein Laufschritt war. Dies dient dazu, damit auf einem Schuss nicht sofort wieder ein Schuss folgt und das Laufmodul mindestens einen Laufschritt Zeit hat, den Roboter zu stabilisieren. Ebenfalls soll der Roboter nicht direkt aus dem Stand oder nach dem Aufstehen schießen können, da hier der erste Laufschritt im Normalfall deutlich früher einen Schrittwechsel verursacht als geplant.

Die zweite Bedingung umfasst die Kontrolle, ob der nächste Schwingfuß ausreichend ist, um den angefragten Schuss auszuführen. Beispielsweise gibt es bei einem Seitwärtsschuss nur einen Laufschritt, weshalb der nächste Schwingfuß der Schussfuß sein muss. Hingegen soll beim Vorwärtsschuss der erste Schwingfuß der nicht Schussfuß sein, auch bezeichnet als Vorschritt. Als Erweiterung des neuen Schussansatzes dürfen bestimmte Schüsse auch diesen Vorschritt überspringen, unter der Voraussetzung, dass die nachfolgenden Bedingungen übereinstimmen. Dadurch soll öfter ein Schuss sofort ausgeführt werden, da ansonsten eine unnötige Wartezeit von über einer halben Sekunde entsteht. Denn wenn der Schussfuß der Schwingfuß ist, dieser aber ohne einen Vorschritt nicht gegen den Ball treten darf, muss ein Laufschritt gewartet und der Vorschritt ausgeführt werden.

Zu guter Letzt wird die Ballentfernung zum Schussfuß und die Abweichung zur Schussrichtung geprüft. Für die Ballentfernung wird, wie schon in Abschnitt 6.3.1 erwähnt, das Koordinatensystem verschoben, wodurch die Ballposition relativ zum Schussfuß vorliegt. Die Schussrichtung wird ebenfalls mit der Rotation des Standfußes und der Odometrie verrechnet. Für jeden Schuss liegen zusätzlich für die Rotationen und Translation Wertebereiche vor, in denen die Ballentfernung und Schussrichtung liegen müssen.

Waren alle Prüfungen erfolgreich, werden die Laufschrittgrößen berechnet und die Abbruchbedingung aus Abschnitt 6.3.4 kontrolliert. Ist diese negativ, kann der Schuss ausgeführt werden.

6.3.3 Schrittmodifikationen

Nachdem die Laufschrittgröße aus Abschnitt 6.3.1 bestimmt wurde, werden mehrere Parameter, definiert durch die Konfiguration für die jeweilige Laufschrittgröße, angewendet. Dabei gibt es zwei Kategorien. In der einen wird die Laufschrittgröße normal modifiziert. Bei der zweiten Kategorie wird eine neue Laufschrittgröße berechnet und die alte gespeichert. Die alte wird für die Abbruchbedingung (siehe Abschnitt 6.3.4) später als Vergleich benötigt. Die

erste Kategorie beinhaltet die Modifikation über eine minimale Translation der Laufschrittgröße und ob die Translation des vorherigen Laufschrittes übernommen werden soll, damit sich die Füße in den Translationen nicht bewegen. Die zweite Kategorie beinhaltet die Modifikation über eine maximale Translation, definiert für alle Schüsse, oder ob eine Rotation oder Translation von 0 übernommen werden muss, definiert für jeden einzelnen Laufschritt.

Als Beispiel für die minimale und maximale Translation nehmen wir an, eine der berechneten Laufschrittgrößen besitzt als x-Translation den Wert 70 und als y-Translation -5. Die konfigurierte minimale Translation ist jeweils 0, während die maximale Translation basierend auf einer maximal erlaubten Laufgeschwindigkeit berechnet wird. Für das aktuelle Laufen, wie schon in Abschnitt 5.1 erwähnt, wäre dies in x-Richtung 250 mm/s respektive in y-Richtung 200 mm/s. Als Laufschrittgröße erhalten wir dann den Wert 62.5 in x-Richtung respektive 50 in y-Richtung. Dadurch wäre die modifizierte Laufschrittgröße 62.5 respektive 0, da für die x-Translation keine Werte größer 62.5 respektive für die y-Translation keine Werte kleiner 0 erlaubt wären.

Beim Übernehmen der Laufschrittgröße des vorherigen Laufschrittes wird sich auf den vorherigen Laufschritt vor dem Schuss bezogen oder, falls ein Schuss aus mehreren Laufschrittgrößen besteht, auf den vorherigen Laufschritt innerhalb des Schusses. Für sowohl die Rotation als auch für die Translation wird dann einzeln, basierend auf der Konfiguration, entschieden, ob eine der vorherige Laufschrittgrößen übernommen werden soll. Da das Ziel hierbei ist, dass sich die Füße in der jeweiligen Bewegungsrichtung nicht weiter bewegen, muss das Gegenteil der letzten Laufschrittgröße übernommen werden. Hierfür wird diese lediglich mit -1 multipliziert. Das Inverse ist an dieser Stelle falsch, weil genau die negative Laufschrittgröße zu keiner Änderung in der Ansteuerung des Laufmoduls führen würde. War beispielsweise die vorherigen Laufschrittgröße gleich (20 Grad, 50, 20) und der Roboter soll den Laufschritt (10 Grad, 0, 0) ausführen und die x-Translation der Fußposition unverändert lassen, so wird der Laufschritt (10 Grad, -50, 0) ausgeführt, damit sich die Füße in der x-Translation während des Laufschrittes nicht bewegen.

6.3.4 Abbruchbedingung

Für die Abbruchbedingung werden die berechneten Laufschrittgrößen aus Abschnitt 6.3.1 mit den modifizierten aus Abschnitt 6.3.3 verglichen. Wurden die Laufschrittgrößen zu sehr verringert, bewertet durch Wertebereiche jeweils für jeden Schusstyp, so wird der Schuss nicht ausgeführt. Hier reicht es schon aus, wenn von der Menge der Laufschrittgrößen nur ein einzelner Wert, also eine einzige Rotation oder Translation in x- oder y-Richtung, zu sehr verringert wurde.

6.3.5 Interpolationsberechnung der Laufschrittgrößen

Wie bereits in Abschnitt 6.3 erwähnt, soll für einen Schuss zwischen den Laufschrittgrößen interpoliert werden. Hierfür wird basierend auf der erlaubten maximalen Laufgeschwindigkeit berechnet, wie lange die Ausführung der einzelnen Laufschrittgrößen dauert. Anschließend kann die Dauer von allen Laufschrittgrößen summiert werden und damit ein prozentualer Anteil am Laufschritt berechnet werden. Dieser prozentuale Anteil ist dann der zeitliche Interpolationswert.

Die Ausführungsdauer wird wie folgt berechnet: Gegeben der vorherigen Laufschrittgröße *preStep*, der aktuellen *currentStep* und der maximalen Laufgeschwindigkeit *maxVel*, jeweils als Pose(Rotation, x-Translation, y-Translation), so gilt

$$\text{diff} = \text{abs}(\text{currentStep - preStep}) \tag{6.7}$$

$$\text{time} = \max\left(\frac{\text{diff}^{(\alpha)}}{\text{maxVel}^{(\alpha)}}, \frac{\text{diff}^{(x)}}{\text{maxVel}^{(x)}}, \frac{\text{diff}^{(y)}}{\text{maxVel}^{(y)}}\right) \tag{6.8}$$

Für eine beliebige Laufschrittgröße des Schusses kann deren Interpolationsdauer $schrittDauer_s$ wie folgt bestimmt werden:

$$\text{timeSumme} = \sum_{i=0}^{n} \text{time}_i \tag{6.9}$$

$$\text{schrittDauer}_s = \frac{\text{time}_s}{\text{timeSumme}} \tag{6.10}$$

Da einige Schüsse durch manuell eingestellte prozentuale Anteile besser funktionieren, existiert auch hier die Option, diese über die Konfiguration vorzugeben. Die Variable *time* aus Gleichung 6.8 wird dann durch diese ersetzt.

Schlussendlich erhält man eine Menge von Laufschrittgrößen, welche folgende Parameter besitzen:

$$\begin{aligned}&\text{laufSchussSchritte} = \{\text{laufSchritt}\}\\ &\text{laufSchritt} = [\text{schwing: (float, float), stütz: (float, float), rotation: float,}\\ &\qquad\qquad\text{laufSchrittDauer: float, ausgeführt: bool}]\end{aligned}$$

Für die Ausführung des Schusses gilt dann folgender Pseudocode:

Algorithmus 3: Ausführung des Schusses

Input: Ein Object vom Typ laufSchussSchritte, die aktuelle Laufschrittdauer aktuelleSchrittDauer, die geplante Laufschrittdauer maximaleDauer, der Vorzeichenwert swingSign und die aktuellen Fußpositionsvariablen forwardSwing0, forwardSupport0, forwardSwing, forwardSupport, sideSwing0, sideSupport0, sideSwing, sideSupport, footHeightSwing0, footHeightSupport0, footHeightSwing, footHeightSupport, turnVal0

Output: Die Translationen beider Füße $\text{trans}_{\text{schwing}}$, $\text{trans}_{\text{stütz}}$ und die Rotation r

```
1  index = 0; vorherigeDauer = 0; dauerOffset = 0;
2  while index < laufSchussSchritte.size do
3      if laufSchussSchritte[index].ausgeführt then
4          dauerOffset += laufSchussSchritte[index].schrittDauer;
5          index += 1;
6      end
7  end
8  benutzteDauer = maximaleDauer · laufSchussSchritte[index].schrittDauer;
9  benutzteAktuelleDauer = aktuelleSchrittDauer - dauerOffset;
10 if benutzteAktuelleDauer / benutzteDauer ≥ 1 ∧ index + 1 < laufSchussSchritte.size
     then
11     laufSchussSchritte[keyframeIndex].ausgeführt = true;
12     dauerOffset += maximaleDauer · laufSchussSchritte[index].schrittDauer;
13     index += 1;
14     benutzteDauer = maximaleDauer · laufSchussSchritte[index].schrittDauer;
15     benutzteAktuelleDauer = aktuelleSchrittDauer - dauerOffset;
16     forwardSupport0 = forwardSupport;
17     forwardSwing0 = forwardSwing;
18     sideSwing0 = sideSwing;
19     sideSupport0 = sideSupport;
20     turnVal0 = turnVal;
21 end
22 schwingSchritt = laufSchussSchritte[index].schwing;
23 stützSchritt = laufSchussSchritte[index].stütz;
24 interpolation = min(1,benutzteAktuelleDauer / benutzteDauer);
25 sideSupport = sideSupport0 + (-stützSchritt^(y) - sideSupport0) · interpolation;
26 sideSwing = sideSwing0 + (schwingSchritt^(y) - sideSwing0) · interpolation;
27 forwardSupport = forwardSupport0 + (-stützSchritt^(x) · 0.5 - forwardSupport0) ·
     interpolation;
28 forwardSwing = forwardSwing0 + (schwingSchritt^(x) · 0.5 - forwardSwing0) ·
     interpolation;
29 turnVal = turnRL0 + (swingSign · laufSchussSchritte[index].rotation · 0.5 -
     turnVal0) · interpolation;
30 fußHöhe(footHeightSwing, footHeightSwing0, footHeightSupport0,
     footHeightSupport);
31 return (forwardSwing,sideSwing,footHeightSwing),
     (forwardSupport,sideSupport,footHeightSupport), turnVal;
```

6.3.6 Vorwärts- und Drehschuss

Mit dem neuen Schussansatz soll die Möglichkeit geschaffen werden, in beliebigen Schussrichtungen zwischen -45 und +45 Grad Vorwärtsschüsse ausführen zu können statt nur in genau die Richtung -45/+45 Grad (Drehschuss) oder 0 Grad (Vorwärtsschuss), wie es bisher der Fall war. Das Framework ermöglicht dieses bereits, da hierfür einfach der Vorwärtsschuss mit einer beliebig angefragten Schussrichtung verwendet werden kann. Aus den relativen Ballpositionen erhält man dann automatisch valide Laufschrittgrößen.

Die Konfiguration sowohl des Vorwärtsschusses als auch des Drehschusses wurde relativ zu deren geplanten Schussrichtungen erstellt. Zwischen beiden bestehen leichte Abweichungen in den relativen Ballpositionen sowie in anderen Parametern, die für die Schussmodifikation wichtig sind (siehe Abschnitt 6.3.3). Deshalb wird stattdessen ein anderer Ansatz verwendet. Die angefragte Schussrichtung wird auf den Wertebereich [-45,45] begrenzt. Mit dieser Schussrichtung werden, basierend auf der Konfiguration des Vorwärts- und des Drehschusses, die jeweiligen Laufschrittgrößen bestimmt. Zwischen den Laufschrittgrößen beider Schüsse wird anschließend basierend auf der Schussrichtung interpoliert.

$$f = \text{Range}\left(0, \frac{\text{abs}(\text{Schussrichtung})}{45}, 1\right) \tag{6.11}$$

$$\text{Laufschritt}_i = f \cdot \text{Laufschritt}_{i_{\text{Drehschuss}}} + (1 - f) \cdot \text{Laufschritt}_{i_{\text{Vorwärtsschritt}}} \tag{6.12}$$

Dieselbe Interpolation wird auch auf die Interpolationsberechnung angewendet (siehe Abschnitt 6.3.5).

6.3.7 Schussstärkenmodifikation

Das vorgestellte Verfahren bewegt die Füße im Normalfall nur mit der maximal erlaubten Bewegungsgeschwindigkeit. Um sowohl stärkere als auch schwächere Schüsse zu erzeugen, wird zum einem ähnlich zum alten Schussverfahren auf Offsets zurückgegriffen, zum anderen wird eine zweite relative Position zum Ball angegeben.

Wie schon in Abschnitt 6.3 aufgezeigt, wird vom Verhalten eine Schussstärke übergeben, welche eine Zahl zwischen 0 und 1 ist. Bei 0 soll die schwächste Variante des Schusses ausgeführt werden, um eine geringe Reichweite zu erzeugen, während bei 1 die stärkste Variante des Schusses für die größte Reichweite ausgeführt wird. Dies wird umgesetzt, indem mithilfe der Schussstärke zwischen den relativen Ballpositionen für die minimal und maximale Stärke interpoliert wird. Es gilt also für eine Schussstärke s:

$$\text{pose}_{\text{interpoliert}} = \text{pose}_{\text{min}} \cdot (1 - s) + \text{pose}_{\text{max}} \cdot s$$

Um neben dieser dynamischen Schussstärke auch noch eine allgemein höhere Schussstärke zu erreichen, was insbesondere für Schüsse, die eine sehr hohe Reichweite haben sollen, wichtig ist, gibt es auch noch Offsets für den Schwingfuß, die auf die relative Ballposition addiert werden. Dadurch soll der Schwingfuß mit höherer Geschwindigkeit gegen den Ball treten können. Da die vorgegebene Schussstärke auch hier Einfluss nehmen soll, werden erneut zwei Offsets für jeweils die minimale und maximale Schussstärke angegeben, zwischen denen genauso interpoliert wird wie bei der relativen Ballposition. Die Verwendung dieser Offsets geschieht nun wie folgt: Gegeben der Schussstärke $pose_{\text{interpoliert}}$ wird zuerst die relative Ballposition bestimmt. Mit dieser Position wird die Laufschrittgröße erstellt, mit welcher der Schwingfuß angesteuert wird. Parallel dazu wird eine zweite relative Ballposition bestimmt, berechnet durch:

$$\text{pose}_{\text{standfuß}} = \text{pose}_{\text{interpoliert}} - \text{offset}$$

$pose_{\text{standfuß}}$ wird ebenfalls zu einer Laufschrittgröße umgerechnet und für die Ansteuerung des Standfußes verwendet. Die Offsets sind dabei nur Translationen. Unterschiedliche Rotationen in der z-Achse für beide Füße sind bei dem verwendeten NAO V6 aufgrund seines Aufbaues unmöglich (siehe Abschnitt 2.2). Trotz dieser unterschiedlichen Laufschrittgrößen, welche indirekt von einander abhängig sind, wird der Schwingfuß weiterhin korrekt basierend der konfigurierten relativen Ballposition platziert. Denn effektiv resultieren diese nur in einer Hüftverschiebung. Gleichzeitig kann dadurch die Bewegungsgeschwindigkeit von Schwing- und Standfuß beeinflusst werden, damit sich der Schwingfuß deutlich schneller bewegt und somit den Ball stärker trifft, welcher als Resultat weiter rollt.

6.3.8 Gelenkoffsets

Mit den Modifikationen aus Abschnitt 6.3.7 lassen sich zwar in der Theorie bereits starke Schüsse nach vorne erzeugen, wie sie auch schon im alten Schussverfahren umgesetzt werden. Jedoch sind die Schussreichweiten nicht konstant. Insbesondere für Torschüsse möchte man eigentlich, solange das Verhalten dies mithilfe der Perzeption unterstützt, bereits aus großer Entfernung schießen. Das aktuelle Schussmodul, welches für sehr weite Schüsse benutzt wird, muss den Roboter zuvor zum Stehen bringen, erreicht dafür aber konstant Schussreichweiten von über fünf Meter. Dies sollte in der Theorie auch aus dem Laufen mit dem neuen Verfahren möglich sein. Dabei existiert aber ein Problem bei der Ausführung der Schüsse. Vereinfacht kann man sagen, dass ein Schuss weit gehen wird, wenn sich die Gelenke des Schwingfußes in Schussrichtung schnell und sich die Gelenke des Standfußes gleichzeitig in die Gegenrichtung bewegen. Bei einem Vorwärtsschuss ginge daher der Schwingfuß nach vorne, der Standfuß nach hinten. Betrachtet man aber die Umsetzung eines Roboters, der einen starken Vorwärtsschuss ausführen soll, dargestellt in Abbildung 6.4, so fällt auf, dass sich die

KneePitch- und HipPitch-Gelenke des Schwingfußes nicht gleichzeitig mit hoher Geschwindigkeit zum Zeitpunkt der Ballberührung bewegen, wodurch dieser unter einen Meter rollte. Diese Asynchronität existiert, weil der Schwingfuß am Anfang des Laufschrittes nach oben und am Ende nach unten geht, damit sich dieser über den Boden bewegen kann. Da sich der Fuß nach der Hälfte des Laufschrittes nach unten Richtung Boden bewegt, wird auch das HipPitch-Gelenk Richtung Boden geführt, also entgegen der Schussrichtung. Gleichzeitig hat der Standfuß seine Zielposition bereits erreicht. Der Ball wird also fast nur durch die Bewegung des KneePitch-Gelenkes getroffen und rollt dadurch nur für eine kurze Reichweite. Das alte Schussverfahren behandelt dieses Problem, indem zum einem ein sehr großer Laufschritt angesteuert wird, der einer Laufgeschwindigkeit von 400 mm/s entspricht. Gleichzeitig wird auch die z-Translation (die Höhe) des Schwingfußes mit den Offsets verschoben. Dadurch bewegt sich zum Zeitpunkt der Ballberührung der Standfuß weiterhin nach hinten und die KneePitch- und HipPitch-Gelenke des Schwingfußes nach vorne. Bei einem so großen Laufschritt könnte der Roboter aber einerseits kurzzeitig instabil laufen, zum anderen kann die Laufhöhe mit dem neuen Ansatz nicht modifiziert werden.

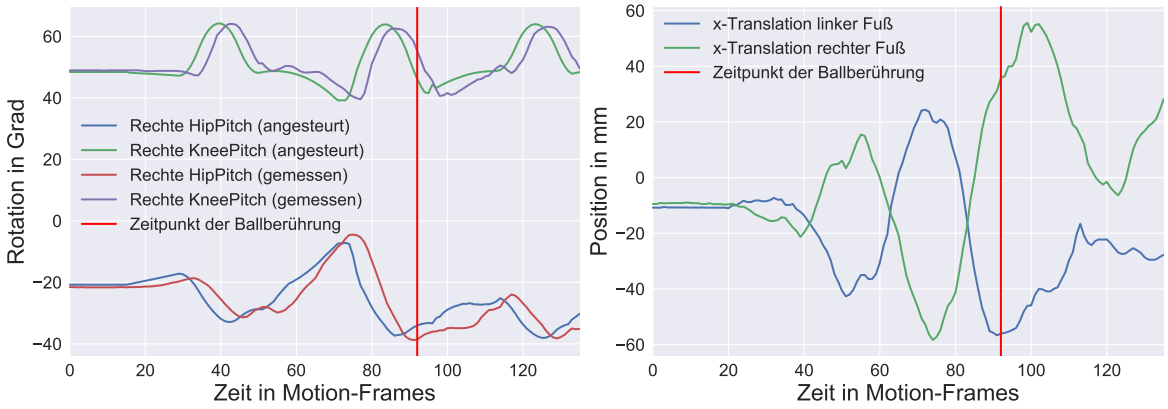

Abbildung 6.4 Die angesteuerten und umgesetzten Fußpositionen (rechts) und Schwingfußgelenke (links) bei einem versuchten starken Schuss. Der Roboter lief aus dem Stand zum Ball und berührte ihn zum Zeitpunkt der roten Markierung.

Stattdessen gibt es hier zwei weitere Möglichkeiten, mehr Kontrolle über die Bewegung der Gelenke zu erlangen. Zum einem könnte ein Gelenk-Controller eingeführt werden, welcher basierend auf der Umsetzung der Gelenke die Ansteuerung modifiziert, um die Gelenke des Schwingfußes in der Bewegung zu synchronisieren. Da wir aber keinen direkten Zugriff auf die Motoren der Gelenke haben und keine Informationen bekannt sind, wie der Hersteller SoftBank Robotics die Motoren ansteuert, könnte ein solcher Ansatz zu unbekannten Nebeneffekten führen. Stattdessen wird eine einfachere Möglichkeit verwendet.

Es werden sowohl Offsetwerte als auch Interpolationsparameter spezifiziert, damit während eines Laufschrittes mithilfe von Sinuskurven Offsets auf Gelenke addiert werden. Ein Beispiel hierfür ist in Abbildung 6.5 zu sehen. Für ein Gelenk wird ein Offsetwert definiert, hier -15 Grad. Ebenso werden für die Interpolation jeweils drei Zeitpunkte definiert, jeweils als

prozentuale Angabe in der Laufschrittdauer. Der Anfang der Interpolation (hier 25 %), der Zeitpunkt wenn der Offsetwert maximal sein soll (hier 40 %) und der Zeitpunkt wenn die Interpolation wieder auf 0 zurückkehren soll (hier 85 %). Dadurch soll um den Zeitpunkt herum eine flüssige Interpolation gewährleistet werden, wenn der Offsetwert maximal ist. Gleichzeitig wird möglichst schnell der Offsetwert zum Ende des Laufschrittes wieder entfernt. Die resultierenden ruckartigen Bewegungen am Anfang und Ende der Interpolation sollten keine Probleme darstellen, da die Positionsänderungen, hier verrechnet mit der geplanten Ansteuerung der Gelenke, zu keinen großen Bewegungsänderungen führen dürften. Ebenfalls ist es für das Laufschrittende deutlich wichtiger, dass dann keine Offsets mehr existieren, statt sie langsam zu interpoliert, um eine Destabilisierung des Laufens zu verhindern.

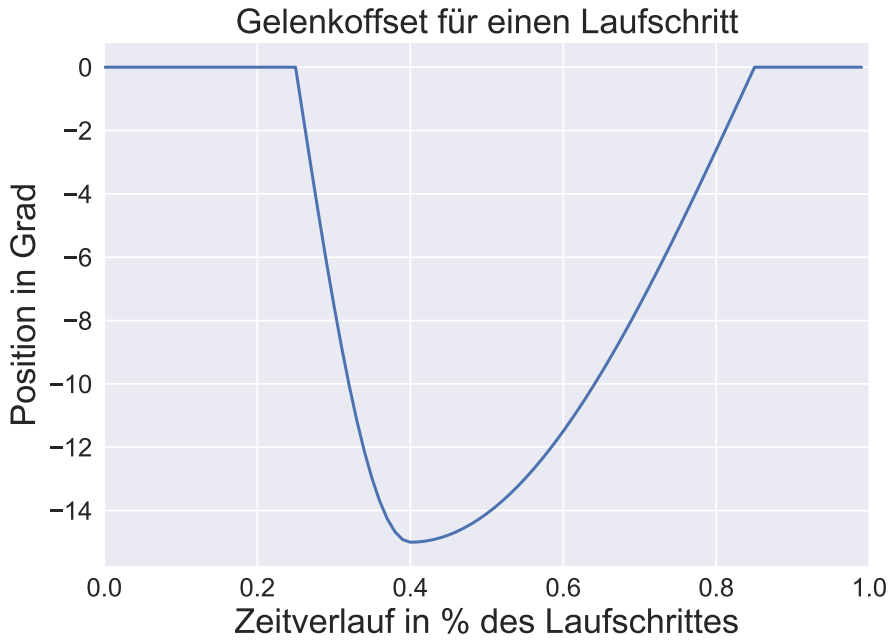

Abbildung 6.5 Ein Beispiel für die Umsetzung der Offsets für einzelne Gelenke. Hier wird ab 25 % des Laufschrittes ein Offset berechnet, welcher bis 40 % des Laufschrittes maximal wird (-15 Grad) und anschließend bis 85 % des Laufschrittes auf 0 zurückkehrt.

6.3.9 Erstellung der einzelnen Schüsse

Für die Erstellung der einzelnen Schüsse orientierte man sich an dem alten Schussverfahren. Das bestehende Verfahren besitzt jeweils eine Konfiguration für verschiedene Schüsse, definiert nach dem rechten Fuß als Schussfuß. Für Schüsse mit dem linken Fuß werden Rotationen und y-Translationen einfach gespiegelt. Die vorhandenen Schüsse sind in Tabelle 6.1 zu sehen.

Alle drei Vorwärtsschüsse könnten mit dem neuen Verfahren zusammengefasst werden, wodurch eine Schussstärke von 0 eine Reichweite von 45 cm erzeugt, respektive eine von 1 eine Reichweite von 300 cm erzeugt. Die Reichweite von 160 cm könnte durch Interpolation über die Schussstärke erreicht werden. Aus Tests ist aber bekannt, dass eine Reichweite von 300 cm ohne die Verwendung von Gelenkoffsets (siehe Abschnitt 6.3.8) nur sehr inkonsistent erzeugt

werden kann. Stattdessen werden nur der kurze und mittlere Vorwärtsschuss zusammengefasst. Gleichzeitig wird eine dynamische Reichweite für den Drehschuss hinzugefügt. Diese wird benötigt, da beliebige Schusswinkel von bis zu 45 Grad ermöglicht werden sollen, wofür der Vorwärtsschuss und der Drehschuss jeweils eine minimale und eine maximale Reichweite erfordern.

Schuss Typ	Schusswinkel (in Grad)	Reichweite (in cm)
Kurzer Vorwärtsschuss	0	45
Mittlerer Vorwärtsschuss	0	160
Langer Vorwärtsschuss	0	300
Seitwärtsschuss	-90	180
Drehschuss	45	90

Tabelle 6.1 Die Schusstypen des alten Schussverfahrens und deren Reichweiten. Die Reichweiten wurden den Konfigurationsdaten entnommen.

Die Reichweiten wurden für den Vorwärts- und Drehschuss nicht an die des alten Schussverfahrens angelehnt. Stattdessen wurde versucht, sowohl einen möglichst schwachen als auch starken Schuss zu erzeugen. Anschließend wurden die Offsets der relativen Ballpositionen (siehe Abschnitt 6.3.3) so per Hand optimiert, dass die maximale Reichweite konstant hoch war. Bei der minimalen Reichweite wurde lediglich die Berührung des Balles sichergestellt. Die restlichen Parameter wurden basierend auf auftretenden Fehlern bei der Erstellung der Schüsse so eingestellt, dass die Schüsse den Roboter nicht destabilisieren und den Ball nicht versehentlich zu früh berühren, aufgrund von Vorschritten, bei denen der Roboter nicht ideal vor dem Ball steht.

Für den Seitwärtsschuss wurde lediglich versucht, den Schuss so stark wie möglich zu bekommen. Hierfür wurde eine relative Ballposition gewählt, die in einem großen Seitwärtsschritt resultiert.

Für den langen Vorwärtsschuss wurden relative Ballpositionen und Offsets gesetzt, die dem Schwingfuß eine möglichst hohe Geschwindigkeit geben. Außerdem wurden die Gelenkoffsets (siehe Abschnitt 6.3.8) verwendet, damit sich das Hüftgelenk des Schussfußes zur Ballberührung nach vorne bewegt.

Die resultierenden maximalen Reichweiten aller Schüsse sind durch die beschriebene Herangehensweise nicht mehr identisch gegenüber den alten Reichweiten. Stattdessen sollten diese aber konstanter sein, wodurch das Verhalten der Roboter genauere Pässe und Torschüsse planen kann. Die tatsächlichen neuen Reichweiten werden unter anderem mithilfe der Evaluation vermessen.

Um das gesetzte Ziel der höheren Genauigkeit zu erreichen, besteht jeder Gesamtlaufschritt aus mindestens zwei einzelnen Laufschritten. Für die Vorschritte besitzt der erste eine relative Ballposition, die bei der Bewegung des Schwingfußes zu keiner versehentlichen Ballberührung

führt, während der zweite den Fuß an eine geeignete Position neben den Ball positioniert. Für die Schussschritte besitzt der erste Laufschritt eine relative Ballposition, die den Fuß hinter den Ball perfekt ausgerichtet platziert, während der zweite den Fuß gegen den Ball treten lässt. Bei dem Vorwärts- und Drehschuss wurde, für die Ausrichtung hinter den Ball, eine Entfernung vom Ballmittelpunkt zum Fußursprung in der x-Achse von zwischen 17 und 18 cm gewählt. Diese Entfernung wurde experimentell ermittelt, indem der Ball vor die Füße mehrerer Roboter gelegt und die geschätzten Entfernungen aus Sicht der Roboter gemittelt wurden. Für den Seitwärtsschuss wurde für die y-Achse eine Entfernung von 15 cm gewählt. Aufgrund des Ballradius von 5 cm und der Fußbreite vom Fußursprung Richtung Ball von 5 cm existiert somit eine Strecke von 5 cm zum Beschleunigen des Fußes. Für den langen Vorwärtsschuss wurde eine Entfernung von 23 cm gewählt, da höhere Reichweiten erzeugt werden konnten, wenn der Schussfuß eine größere Strecke zum Beschleunigen besaß.

6.3.10 Probleme des Drehschusses

Eine große Herausforderung bei den Schüssen aus dem Laufen ist die korrekte Umsetzung der Ansteuerungen. Während der Entwicklung fiel dabei primär bei größeren Drehschüssen auf, dass der Ball oftmals nicht getroffen wurde. Hier haben die Gelenke teilweise nicht die angesteuerten Werte umgesetzt. Um dieses Problem zu minimieren, wurde verglichen, ob es eine Auswirkung hat, wenn der Roboter vor dem Schuss kurz stehen bleibt, falls er vorher einen größeren Laufschritt ausgeführt hat. Dieser Vergleich ist in Abbildung 6.6 zu sehen. Hier weist der erste Laufschritt, welcher den nicht schießenden Fuß neben den Ball platziert, bereits Probleme für den Standfuß (linker Fuß) auf, wenn vorher ein größerer Laufschritt ausgeführt wurde. Die Fußposition hängt der Ansteuerung stark hinterher, erreicht aber die angesteuerte Position vor dem Schrittwechsel. Wenn vorher kein größerer Laufschritt ausgeführt wurde, wichen die erreichten Positionen nur leicht von den angesteuerten ab.

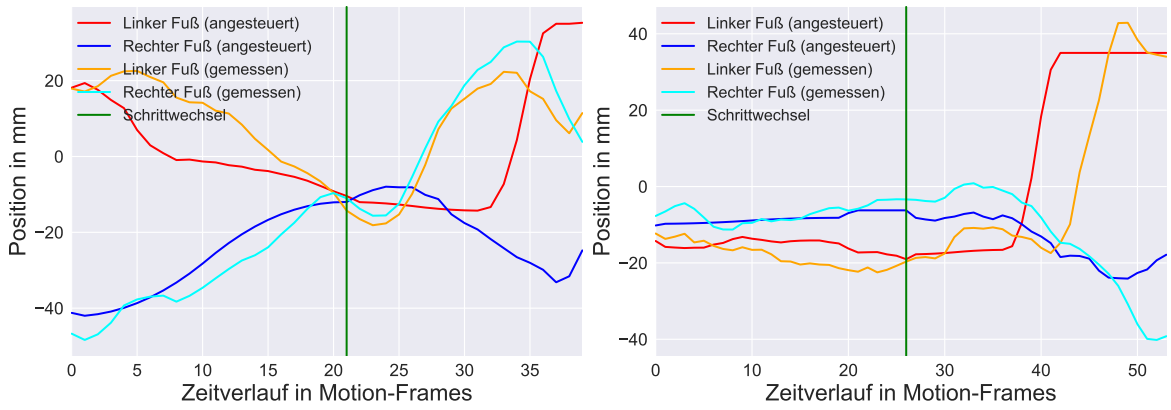

Abbildung 6.6 Die Fußansteuerungen der x-Translation während eines Drehschusses, mit dem linken Fuß als Schussfuß. Links wurde vorher ein größerer Vorwärtsschritt ausgeführt, rechts wurde vorher angehalten. Der Schrittwechsel (grün) markiert den Übergang beider Laufschritte des Drehschusses, wobei der zweite Laufschritt den Ball tritt.

Wenn vorher ein größerer Laufschritt ausgeführt wurde, bewegt sich der Schussfuß nach dem Schrittwechsel scheinbar zu früh nach vorne (orange verglichen mit rot), während der Standfuß sich ebenfalls nach vorne bewegt, obwohl die Ansteuerung den Fuß nach hinten bewegen sollte (cyan verglichen mit blau). Betrachtet man die Gelenkwinkel, fällt auf, dass das HipYawPitch-Gelenk, welches die Rotationen der Füße ermöglicht, stark hinterherhängt. Dies ist in Abbildung 6.7 zu sehen. Wenn vorher kein größerer Laufschritt ausgeführt wurde, bewegen sich die beiden Füße in die angesteuerten Richtungen, da sich das HipYawPitch-Gelenk korrekt bewegt. Es liegt daher die Vermutung nahe, dass durch den späteren Schrittwechsel das Gelenk genügend Zeit hat, zum Zeitpunkt des Schrittwechsels die Zielposition zu erreichen.

Statt also vor einem Drehschuss zu warten, falls der vorherige Laufschritt zu groß war, wäre es daher besser, einen zu frühen Schrittwechsel zu verhindern. Da das aktuelle Laufen aber keine aktive Kontrolle über die Laufschrittdauer hat und in dieser Arbeit keine entwickelt wird, muss somit in solchen Fällen leider gewartet werden, um den Drehschuss funktionsfähig zu bekommen.

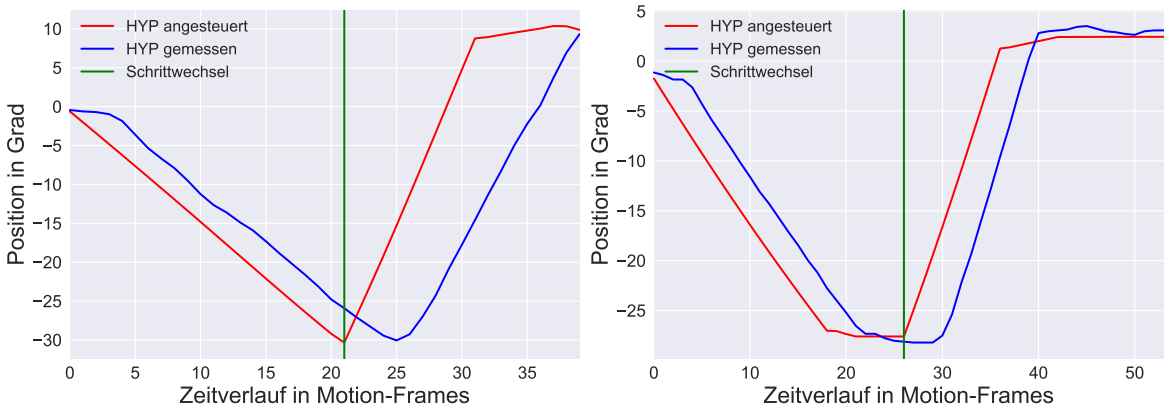

Abbildung 6.7 Die Ansteuerung des HipYawPitch-Gelenkes während eines Drehschusses, mit dem linken Fuß als Schussfuß. Links wurde vorher ein größerer Vorwärtsschritt ausgeführt, rechts wurde vorher angehalten. Der Schrittwechsel (grün) markiert beide Laufschritte des Drehschusses, wobei der zweite den Ball tritt.

6.4 Evaluation

6.4.1 Aufbau

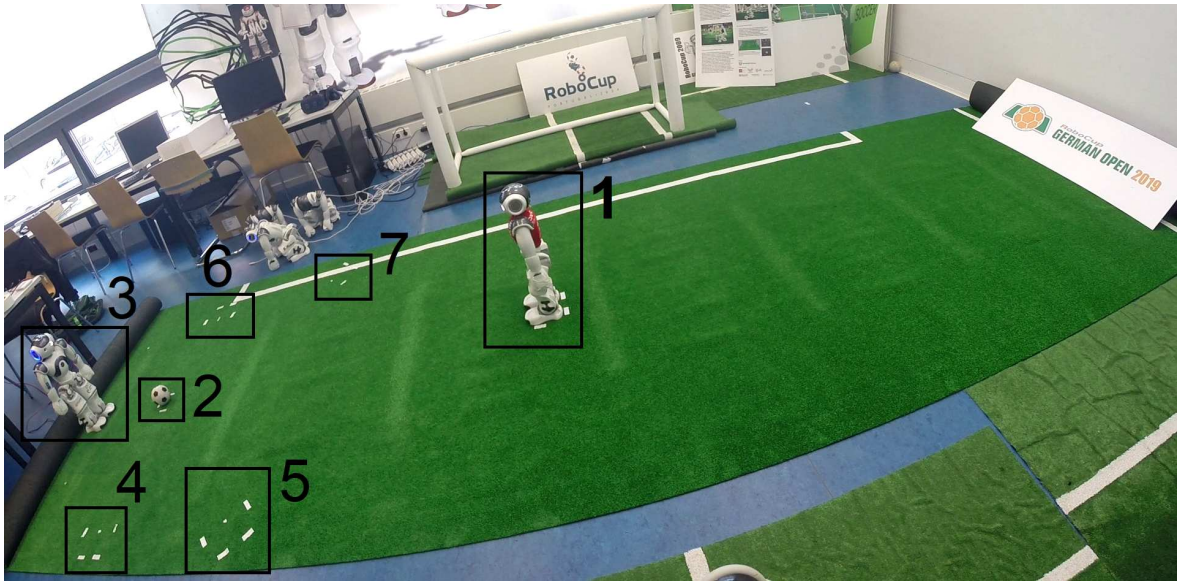

Abbildung 6.8 Feldaufbau für die Schussevaluation. Positionen 1 und 4-7 sind für den Referenzroboter, über den die Schussrichtung bestimmt wird. Position 2 ist die feste Position für den Ball. Position 3, 4 und 6 sind die Startpositionen für den zu schießenden Roboter. Die Feldlänge von Ball zur Wand beträgt 5 m, die Breite zu beiden Seiten jeweils 1 m.

Um nun die gesetzten Ziele, die erhöhte Genauigkeit und geringere Reichweitenstreuung der entwickelten Schüsse im Vergleich zu den alten, zu evaluieren, wird ein spezieller Aufbau verwendet (Abbildung 6.8), damit der Einfluss anderer Fehlerquellen minimal gehalten wird.

Es wird ein 2 m × 6 m großes Kunstrasenfeldstück verwendet, gemäß der SPL Regeln. Der zu schießende Ball liegt an einem Ende. An einer vorgegebenen Position steht ein NAO V6 Roboter, welcher als Landmarke dient. Die Aufgabe für den zu schießenden Roboter ist es, den Ball in einem Winkel relativ zu dem Landmarken Roboter zu schießen. Dadurch wird jeglicher Fehler, welcher aus der nicht perfekten Lokalisierung resultieren kann, eliminiert. Fehler in der Bildverarbeitung der Robotererkennung, welche den Roboter, der als Landmarke dient, nicht perfekt erkennen, können nachträglich durch Logaufnahmen nachvollzogen werden. Dadurch bleiben effektiv nur vier Fehlerquellen übrig. Fehler in der Torsolagenschätzung können die relative Position des Balles zu den Füßen verschieben, wodurch der Ball nicht ideal getroffen wird. Selbiges gilt für die Robotererkennung. Die zweite Fehlerquelle sind die Gelenke des Roboters, welche nicht zwangsläufig die geplanten Positionen rechtzeitig erreichen oder sogar übersteuern. Als Folge wird der Ball ebenfalls nicht ideal getroffen. Die dritte Fehlerquelle sind Kräftewirkungen des Roboters selber, da dieser sich während des Schusses unerwartet verschieben könnte, sei es aufgrund dessen, dass er auf dem Kunstrasen rutscht oder sich sogar dreht. Die vierte Fehlerquelle ist der Kunstrasen, welcher trotz kleinerer Unebenheiten

die Trajektorie des Balles beeinflussen kann. Da diese Fehlerquellen auf beide Schussverfahren gleichermaßen Einfluss nehmen, können diese weiterhin zueinander verglichen werden.

Für die Auswertung wird die Schussreichweite und -richtung gemessen. Die Reichweite ist definiert als die Entfernung zwischen der Endposition, an welcher der Ball liegen bleibt, und der Startposition des Balles.

Für die Auswertung der Start- und Endposition wird neben einer manuellen Auswertung auch eine basierend auf dem Robotermodell erstellt. So sollte sich dies in beiden Auswertungsfällen widerspiegeln, sofern ein Schussverfahren besser ist als das andere.

Das Experiment läuft nun wie folgt ab: Es werden die vier Schüsse beider Verfahren auf zwei Roboter abwechselt ausgeführt. Die Schüsse sind der normale Vorwärtsschuss (VS), der lange Vorwärtsschuss (VSL), der Drehschuss um 45 Grad (DS) und der Seitwärtsschuss (SS). Die Roboter starten dabei immer zwischen 50 cm und 100 cm entfernt zum Ball, mit ungefähr der Ausrichtung, die sie am Ball haben werden. Der Landmarkenroboter steht dabei so, dass der Winkel des auszuführenden Schusses den Ball genau mittig über die sechs Meter Strecke bewegen müsste. Der Aufbau ist in Abbildung 6.8 zu sehen.

Für den Ablauf soll ein Roboter einen Schuss ausführen und danach stehen bleiben. Lediglich beim Seitwärtsschritt dreht er sich anschließend noch, damit er sehen kann, wo der Ball liegen bleibt. Dadurch kann er selbstständig die Ballendposition ermitteln und basierend auf seiner Odometrie berechnen, wie weit der Ball gerollt ist und in welchem Winkel. Falls der Landmarken Roboter auf der Rollbahn des Balles steht, wird dieser nach Ausführung des Schusses per Hand zur Seite bewegt, um eine Kollision zu vermeiden.

Jeder Schuss wird zehnmal ausgeführt, jeweils fünf für beide Roboter. Da die Schüsse symmetrisch sind, wird mit beiden Füßen geschossen. Dabei wird eine Aufteilung von drei Schüssen mit dem linken und zwei Schüssen mit dem rechten Fuß für Roboter 1 verwendet. Für Roboter 2 sind es zwei Schüsse mit dem linken und drei mit dem rechten Fuß.

Zusätzlich werden auch die zwei neuen Erweiterungen der Schüsse, die dynamischen Reichweiten und die beliebige Schussrichtung des Vorwärtsschusses, evaluiert. Dies wird nur für das neue Verfahren durchgeführt, da die alten Schüsse keine dynamischen Reichweiten und Schussrichtungen besitzen.

Der Ablauf des Experiments ist dasselbe. Statt vier verschiedenen Schüssen wird hier nun, begrenzt durch den zeitlichen Aufwand, nur ein Vorwärtsschuss mit einer Richtung von 22.5 Grad ausgeführt, mit den Stärken 0 %, 50 % und 100 %. Dabei steht der Landmarken Roboter auf der Mitte des Feldes. Es wurde sich explizit gegen weitere Varianten entschieden, da die Feldunebenheiten zu viele zusätzliche Unsicherheiten in das Experiment einbringen. Ebenfalls wird hier nur ein Roboter verwendet.

6.4.2 Ergebnisse

Die Ergebnisse sind in Tabelle 6.2 und Tabelle 6.3 zu sehen. Die Unterschiede der per Handmessung und denen aus der Sicht der Roboter kommen womöglich von der nicht perfekten Odometrie, wodurch leichte Richtungsunterschiede entstehen. Auch wurde der Ball bei den längeren Schüssen teilweise nicht mehr erkannt, da die Ballerkennung nur bis zu einer Entfernung von 3 m konstant funktioniert.

Die starken Abweichungen der Richtungen können zudem durch die Feldunebenheiten erklärt werden. Aufgrund der aktuellen COVID-19 Situation konnte kein geeigneteres Feld für die Evaluation beschafft werden. Da das Ziel der neuen Schüsse nicht war, die gleichen Reichweiten zu besitzen, kann dazu die durchschnittliche Reichweite vorerst ignoriert werden.

Der Vorwärtsschuss ist vergleichbar gut wie der vorherige. Da dieser hauptsächlich für das Dribbeln und Gewinnen von Zweikämpfen verwendet werden soll, ist es verkraftbar, dass der Ball zwischen 50 cm und 150 cm liegen bleibt.

Messung	VS		VSL		DS		SS	
	Alt	Neu	Alt	Neu	Alt	Neu	Alt	Neu
Reichw. Ø (in cm)	190.74	102.31	368.05	351.87	139.80	216.92	283.12	217.59
Reichw. σ (in cm)	54.84	47.28	49.46	91.26	54.67	40.69	26.81	49.7
Richtung Ø (in Grad)	14.82	14.82	10.11	18.93	19.31	5.25	11.18	15.67
Richtung σ (in Grad)	10.61	19.22	8.25	16.69	14.99	7.39	9.55	13.28

Tabelle 6.2 Die Ergebnisse der per Hand gemessenen Schussreichweiten und -richtungen. Zu sehen ist der Durchschnitt der Reichweiten (Reichw. Ø), die Standardabweichung der Reichweiten (Reichw. σ), die durchschnittliche Richtungsabweichung (Richtung Ø) und die Standardabweichungen der Richtungsabweichungen (Richtung σ)

Messung	VS		VSL		DS		SS	
	Alt	Neu	Alt	Neu	Alt	Neu	Alt	Neu
Reichw. Ø (in cm)	179.11	94.72	366.87	327.9	131.40	217.96	275.90	218.28
Reichw. σ (in cm)	45.51	44.24	59.6	76.7	59.29	41.52	34.03	52.07
Richtung Ø (in Grad)	13.53	13.99	9.6	14.70	17.78	6.88	9.39	13.18
Richtung σ (in Grad)	9.43	11.19	6.58	10.16	9.93	6.67	6.84	10.36

Tabelle 6.3 Die Ergebnisse der von den Robotern gemessenen Schussreichweiten und -richtungen. Zu sehen ist der Durchschnitt der Reichweiten (Reichw. Ø), die Standardabweichung der Reichweiten (Reichw. σ), die durchschnittliche Richtungsabweichung (Richtung Ø) und die Standardabweichungen der Richtungsabweichungen (Richtung σ)

Der lange Vorwärtsschuss ist hingegen signifikant schlechter. So streut dieser im Vergleich zum alten Schuss von der Reichweite und Richtung jeweils fast doppelt so stark. Den Videoaufnahmen nach zu urteilen drehten und verschoben sich die Roboter während des Schusses,

wodurch der Ball häufig sehr schlecht getroffen wird. Dieses schlechtere Treffen ist auch aus Sicht der Roboter zu erkennen, wie in Abbildung 6.9 dargestellt.

Der Drehschuss um 45 Grad funktioniert deutlich besser. Dieser Schuss profitiert sehr davon, basierend auf dem Ball und der gewollten Schussrichtung, den Fuß nach dem Drehschritt auszurichten. Ebenfalls scheinen die Roboter während des Schusses erheblich weniger zu rutschen, wodurch die Richtungsstreuung im Vergleich zu allen anderen Schüssen mit 5.25 Grad mit Abstand am geringsten ausfällt. Es muss dazu angemerkt werden, dass der alte Drehschuss viermal den Ball nicht getroffen hat. Dies liegt an dem in Abschnitt 6.3.10 bereits beschriebenen Problem. Für die Statistik wurden diese Versuche nicht mit einberechnet, damit es einen Vergleich gibt zwischen den Schüssen, die auch tatsächlich den Ball berührt haben.

Der Seitwärtsschuss weißt eine größere Standardabweichung in sowohl der Reichweite als auch in der Richtung auf. Nach den Videoaufnahmen zu urteilen liegt dies an Ausreißern, bei denen eine unzureichende Startbedingung für diesen Schuss vorliegt. So kommt es manchmal vor, dass der Schussfuß zwar korrekt neben dem Ball steht, der Standfuß aber weiter vorne, weil der Roboter vorher geradeaus gelaufen ist. Dadurch wird für den Schwingfuß angesteuert, sich zuerst nur seitlich zu bewegen und am Ende des Laufschrittes aber parallel zum Standfuß zu platzieren. Somit entsteht eine Ansteuerung, in der ein schneller Seitwärtsschritt ausgeführt wird, welcher aufgrund der begrenzten Gelenkgeschwindigkeiten in einem schnellen diagonalen Laufschritt resultiert. Mit einer geeigneteren Startbedingung müsste der Seitwärtsschuss daher besser sein.

Abbildung 6.9 Verglichen der Moment, in dem der Ball beim langen Vorwärtsschuss getroffen wird, wenn der Roboter während des Schusses nicht rutscht (links) und wenn der Roboter rutscht (rechts)

Für den langen Vorwärtsschuss wurde zusätzlich, aufgrund der deutlich höheren Ungenauigkeit, untersucht, ob ein langsameres Schießen eine Verbesserung hervorrufen kann. Alternativ könnte man natürlich eine direkte Kompensation versuchen, welche aus der gemessenen Rotation des Roboters während des Schusses die angesteuerten Laufschrittgrößen modifiziert, damit der Ball weiterhin korrekt getroffen wird.

Schaut man sich aber die Rotationsänderungen in den Odometriedaten an, welche in Abbildung 6.10 abgebildet sind, lässt sich eine weitere Vermutung äußern. Dadurch, dass der Ball beim alten Schuss später getroffen wird, dreht sich der Roboter mehrmals, wobei die zweite Drehung die erste aufhebt. Diese Rotation kann aber aus den Videoaufnahmen nicht bestätigt werden. Interessanterweise gibt es keinen offensichtlichen Unterschied in den Messungen, wenn der Roboter den Ball gut trifft verglichen mit Situationen, in denen er ihn schlecht trifft. Eine Kompensation, die entgegen möglichen Drehungen oder Verschiebungen während des Schusses arbeitet, wäre daher nicht denkbar.

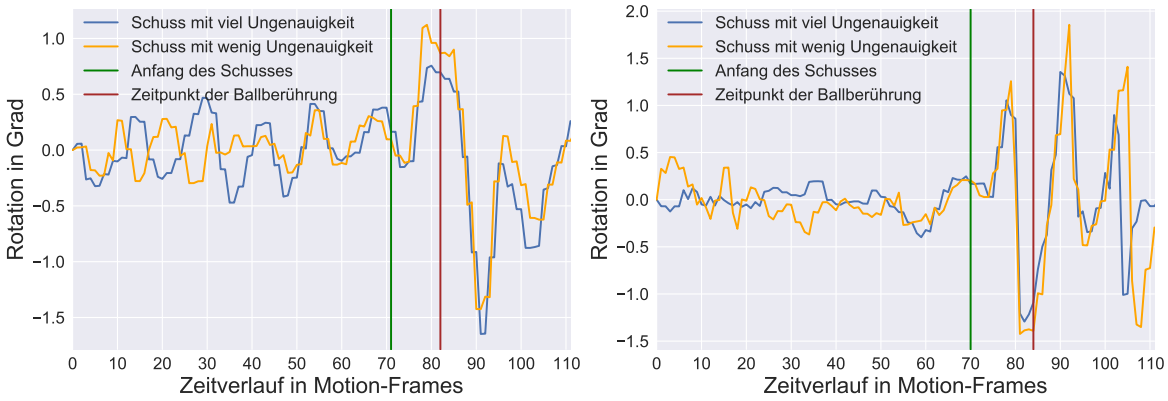

Abbildung 6.10 Die Rotationsänderung der Odometrie, während der langen Vorwärtsschüsse. Links mit dem neuen Verfahren, rechts mit dem alten. In blau jeweils ein Schuss, mit viel Abweichung in der Schussrichtung und in orange ein sehr genauer Schuss. Grün markiert den Zeitpunkt, zu dem der Laufschritt, der den Ball trifft, beginnt und dunkelrot markiert den Zeitpunkt, zu dem der Ball berührt wird.

Für die Verlangsamung des Schusses wird die Geschwindigkeit des Standfußes auf 160 mm/s beschränkt. Dieser durfte sich zuvor mit einer Geschwindigkeit von 300 mm/s bewegen. Auch durfte sich der Schussfuß am Anfang des Schusses nach hinten bewegen, um auszuholen. Dies führte aber, beurteilend durch die Videoaufnahmen, zu einer ruckartigen Bewegung, die den Roboter zusätzlich drehte. Die Rückwärtsbewegung wurde daher ebenfalls reduziert.

Um diese Änderungen zu evaluieren, wurde mit einem Roboter fünfmal der Schuss sowohl mit dem linken Fuß als auch mit dem rechten Fuß ausgeführt. Das Ergebnis ist in Tabelle 6.4 zu sehen. Die Richtungsungenauigkeit ist nun signifikant geringer und ähnlich dem alten Schuss. Die Schussreichweite streut weiterhin deutlich mehr. Dies liegt an einem fehlenden Anlaufplan. Beim alten Schuss stand der Roboter sehr weit weg vom Ball, wodurch er mit einem Vorschritt eine konstante Entfernung und Geschwindigkeit zum Ball besaß. Beim neuen Schuss darf die Entfernung mehr abweichen, wodurch der Roboter den Ball öfter nicht ideal trifft. Betrachtet man die Rotationsänderung der Odometrie, zu sehen in Abbildung 6.11, wird der Ball nun ähnlich zum alten Schuss getroffen, sobald der Roboter sich scheinbar wieder zurückdreht.

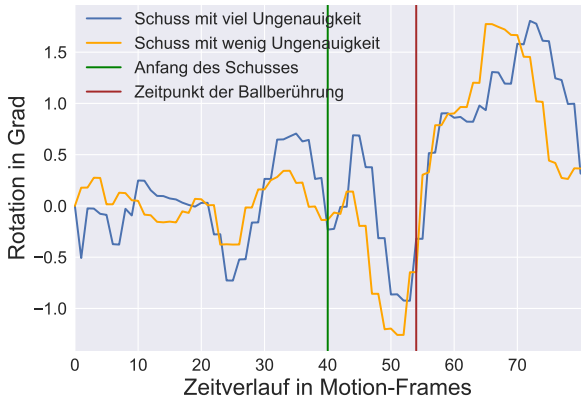

Abbildung 6.11 Die Rotationsänderung der Odometrie, während der neuen langen Vorwärtsschüsse. Der Schuss wurde angepasst, damit sich die Füße sanfter bewegen. In blau jeweils ein Schuss, mit viel Abweichung in der Schussrichtung und in orange ein sehr genauer Schuss. Grün markiert den Zeitpunkt, zu dem der Laufschritt, der den Ball trifft, beginnt und dunkelrot markiert den Zeitpunkt, zu dem der Ball berührt wird.

Messung	VSL mit Anpassungen
Reichw. Ø (in cm)	317.75
Reichw. σ (in cm)	101.7
Richtung Ø (in Grad)	8.76
Richtung σ (in Grad)	6.39

Tabelle 6.4 Die Ergebnisse des angepassten langen Vorwärtsschusses. Zu sehen ist der Durchschnitt der Reichweiten (Reichw. Ø), die Standardabweichung der Reichweiten (Reichw. σ), die durchschnittliche Richtungsabweichung (Richtung Ø) und die Standardabweichungen der Richtungsabweichungen (Richtung σ)

Das Ergebnis des zweiten geplanten Experimentes mit den Vorwärtsschüssen von 22.5 Grad ist in Tabelle 6.5 zu sehen. Die Reichweiten reduzieren sich zwar, dennoch müsste sich bei einer Stärke von 50 % eine Reichweite von ungefähr 90 cm ergeben statt nur 72.36 cm. Die Durchschnittsreichweite bei 100 % liegt bei 157.58 cm. Da der Schuss eine Interpolation zwischen dem Vorwärts- und Drehschuss ist, sollte sich eine Durchschnittsreichweite bestimmen lassen, berechnet nach der Formel

$$\text{Reichweite} = \text{ReichweiteVS} \cdot \left(1 - \frac{\text{Richtung}}{45\text{Grad}}\right) + \text{ReichweiteDS} \cdot \frac{\text{Richtung}}{45\text{Grad}}$$

Aus dem ersten Experiment sind die Reichweiten bekannt und für die Richtung können die 22.5 Grad gesetzt werden. Somit ergibt sich

$$\begin{aligned}\text{Reichweite} &= 94.72\text{cm} \cdot \left(1 - \frac{22.5\text{Grad}}{45\text{Grad}}\right) + 217.96\text{cm} \cdot \frac{22.5\text{Grad}}{45\text{Grad}} \\ &= 94.72\text{cm} \cdot 0.5 + 217.96\text{cm} \cdot 0.5 \\ &= 156.34\text{cm}\end{aligned}$$

Messung	100 %	50 %	0 %
Reichw. Ø (in cm)	157.58	72.36	42.12
Reichw. σ (in cm)	45.11	22.26	11.18
Richtung Ø (in Grad)	9.77	22.58	6.09
Richtung σ (in Grad)	25.53	37.76	19.08

Tabelle 6.5 Die Ergebnisse der Interpolation eines Drehschusses von 22.5 Grad. Zu sehen ist der Durchschnitt der Reichweiten (Reichw. Ø), die Standardabweichung der Reichweiten (Reichw. σ), die durchschnittliche Richtungsabweichung (Richtung Ø) und die Standardabweichungen der Richtungsabweichungen (Richtung σ) relativ zur Schussstärke von 100 %, 50 % und 0 %

Die tatsächliche durchschnittliche Reichweite weicht also nur um 1.24 cm ab.

Die Richtungen haben in allen Fällen eine sehr hohe Standardabweichung, welche auf die Feldunebenheiten zurückzuführen sind. Diese sind entlang des Feldes in Schussrichtung. Dadurch, dass der Ball nicht mittig sondern in einem Winkel über das Feld rollt, haben die Feldunebenheiten deutlich mehr Einfluss auf die Richtung des Balles eingenommen, da diese ihn in seiner Rollbahn ablenken.

6.5 Fazit

Insgesamt kann festgehalten werden, dass das Prinzip von dynamischen Reichweiten und die Interpolation des Vorwärtsschusses mit dem Drehschuss funktioniert. Die neuen Schüsse sind ähnlich gut und je mehr Rotation in einem Schuss auftritt, desto besser gelingen sie im Vergleich zu den alten Schüssen. Dennoch konnte keine signifikante Erhöhung der Genauigkeit erreicht werden. Das alte statische Verfahren, bei dem sich der Roboter vorher vor dem Ball ausrichtet, ist daher qualitativ gleich gut wie die dynamischen Fußplatzierungen. Das neue Verfahren verspricht aber eine schnellere Ausführbarkeit, da sich die Roboter weniger lange für die Schüsse ausrichten müssen, sowie die Möglichkeit, dynamischere Richtungen auszuführen, wie es am 22.5 Grad Schuss zu sehen ist.

Eine große Schwäche beider Schussverfahren ist die Verschiebung der Roboter während der Schüsse, wodurch der Ball nicht ideal getroffen wird. Folglich bleiben die hohen Reichweiten- und Richtungsungenauigkeiten erhalten. Die exakten Ursachen und Lösungen dieser müssen für ein perfekt genaues Schussverfahren gefunden werden. Insbesondere für weite Schüsse müssen sich beide Füße schnell bewegen können, was aber aktuell zu unpräzisen Schüssen führt.

Kapitel 7

Zweikampf

Wie schon in Abschnitt 3.1 erwähnt, fallen im Spielgeschehen die Roboter hauptsächlich in der Gegenwart anderer Roboter um. Bezogen auf den letzten RoboCup 2019 waren von den 78 Stürzen 74 eine Folge durch Kollisionen, dabei 31 das Resultat von Zweikämpfen. Zweikämpfe sind dabei solche Situationen, in denen ein eigener und ein gegnerischer Roboter um den Ballbesitz kämpfen und dabei den Ball bewegen. Da der aktuelle Zweikampf auch noch weitere Schwächen besitzt, werden diese zuerst klargestellt und anschließend ein neues Zweikampfverhalten vorgestellt, welches sowohl diese Schwächen beheben als auch die Anzahl von Stürzen minimieren soll.

7.1 Der aktuelle Zweikampf

In Abschnitt 4.3 wurde bereits der Aufbau des strategischen Verhaltens, welches im Cognition-Thread läuft und durch Decks von Cards umgesetzt ist, vorgestellt. Der bisherige Zweikampf ist hierbei eine Card. Die Start- und Endbedingung prüfen jeweils, ob der Roboter den Ball spielt. Ebenfalls wird für die Startbedingung geprüft, ob eine Menge von Abfragen für den Ball und die Weltmodellierung ausreichend erfüllt werden. So muss sich der am nächsten stehende erkannte Roboter (erkannt durch die Robotererkennung) innerhalb eines Winkelbereiches befinden und darf nur eine maximale Distanz entfernt stehen. Auch darf der Ball nicht zu weit weg oder in einem zu großen Winkel liegen. Aus historischen Gründen wird zusätzlich das Zweikampfverhalten in beiden Elfmeterboxen verboten. Für die Endbedingung muss lediglich der Ball oder der am nächsten stehende Roboter weit genug entfernt platziert sein. Für das Zweikampfverhalten selber wird basierend auf der Entfernung zum Ball ein erlaubter Winkelbereich interpoliert. Je näher der Ball liegt, desto weniger darf sich an die Schussrichtung angepasst werden respektive je weiter der Ball entfernt liegt, desto mehr darf noch rotiert werden. Dies dient dazu, dass man sich noch relativ frei entscheiden kann, wo hingeschossen wird, wenn der Ball noch weit weg liegt. Ist dieser aber fast schon vor den Füßen, soll lieber möglichst schnell ein Schuss ausgeführt werden.

Basierend auf dieser erlaubten Schussrichtungsanpassung wird für jeden Schuss eine Menge von Schussrichtungen geprüft und durch eine Bewertungsfunktion bewertet. Dabei wird für jeden Schuss und jede Schussrichtung berechnet, wo der Ball voraussichtlich liegen bleiben wird. Diese Feldposition wird dann durch ein Potentialfeld bewertet. Zusätzlich bekommen vordefinierte Schusstypen einen kleinen positiven Bonus, da diese sonst zu selten ausgeführt werden. Die gesamte Bewertungsfunktion kann als Potentialfeld gesehen werden, mit einigen harten Kanten, dessen Feldrichtung nicht berechnet wird. Zuerst werden bestimmte Gegebenheiten geprüft. Positionen außerhalb des Spielfeldes, mit Ausnahme von hinter der gegnerischen Grundlinie und Richtungen zur eigenen, werden als etwas schlechter als die maximal beste Bewertung gesetzt. Torschüsse werden hingegen gleich der besten, während Schüsse, die am Tor vorbei gehen, als sehr schlecht gesetzt. Liegt die Ballendposition sonst normal innerhalb des Spielfeldes, wird ein übliches Potentialfeld verwendet. Dabei ist das eigene Tor ein abstoßendes und das generische Tor ein anziehendes Feld. Zusätzlich sind jeweils die Spielfeldmitte für die y-Achse sowie die Torpfosten für die x-Achse anziehende Felder. Auch wird sowohl für jeden verbündeten Roboter ein anziehendes Feld gebildet als auch für jeden gegnerischen Roboter ein abstoßendes Feld. Für die gegnerischen Roboter wird dabei, statt dem lokalen eigenen Modell über gegnerische Roboter, das Teammodell verwendet, welches Erkennungen von allen Teammitgliedern über einen Kalman-Filter zusammenführt.

Anschließend werden die Bewertungen noch leicht angepasst. Schüsse desselben Typs sowie einer ähnlichen Schussrichtung, die dem besten Schuss aus dem vorherigen Cognition-Frame entsprechen, bekommen einen kleinen Bonus. Der Schussrichtungsbonus ist dabei linear interpoliert. Je weiter die Schusspose entfernt ist, also je weiter der Roboter laufen müsste, um den Schuss auszuführen, desto größer wird ein Malus auf die Bewertung addiert. Der best bewertete Schuss, definiert durch Schusstyp und Schussrichtung, wird dann mithilfe eines Skills ausgeführt, indem dieser dem *MotionRequest* übergeben wird.

Ein Beispiel für die Bewertungsfunktion ist in Abbildung 7.1 zu sehen. Das eigene Tor ist links, das gegnerische rechts.

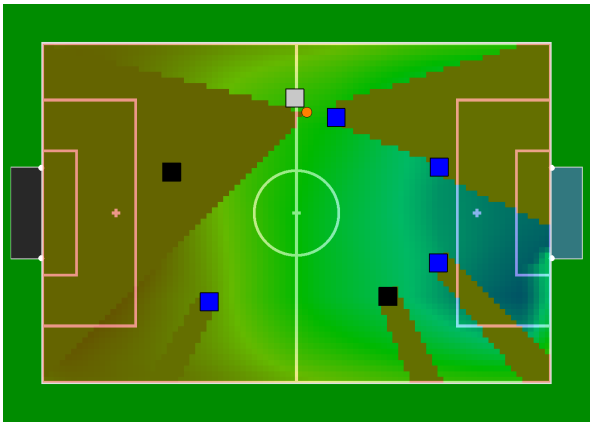

Abbildung 7.1 Die Bewertungsfunktion des aktuellen Zweikampfes. Blaue Bereiche sind sehr gut, braune Bereiche sehr schlecht. Die weiße Box ist der Ball spielende Roboter, blaue Boxen Gegner und schwarze Teammitglieder.

7.2 Problemanalyse

Für den RoboCup 2019 wurden die Schwächen des aktuellen Zweikampfes untersucht. Hierfür wurden die Videoaufnahmen verwendet. So gab es insgesamt 203 Zweikämpfe vom Team B-Human, davon 56 (27.6 %) allein im Finale. Die Heatmaps der Feldpositionen von den Zweikämpfen sind in Abbildung 7.2 und Abbildung 7.3 zu sehen. Dabei ist jeweils links das eigene und rechts das gegnerische Tor. Der Großteil der Zweikämpfe für den gesamten Wettbewerb findet dabei auffällig in der eigenen Hälfte in der Nähe des Mittelkreises statt. Diese Ortsansammlung kommt daher, dass B-Human bis zum Halbfinale im Schnitt 8.2 Tore schoss und als Folge dessen die gegnerischen Teams 9.2 Anstöße pro Spiel hatten. B-Human selber bekam nur jeweils einen Anstoß pro Spiel, da bis dahin keine Gegentore fielen. Zwangsläufig konnte der Ball, als Resultat von den vielen Gegenanstößen, sehr oft leicht in die eigene Feldhälfte gebracht werden, bevor dieser durch den Zweikampf in die gegnerische Hälfte gespielt werden konnte.

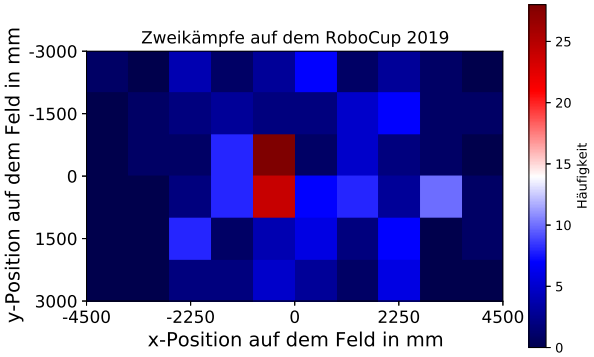

Abbildung 7.2 Feldposition aller B-Human Zweikämpfe auf dem RoboCup 2019. Insgesamt gab es 203 Zweikämpfe, im Schnitt 29 pro Spiel.

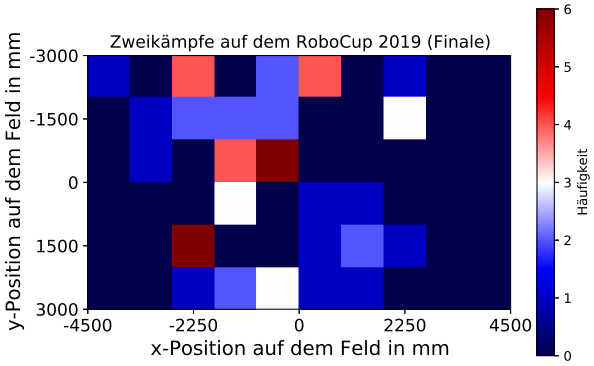

Abbildung 7.3 Die Feldposition der B-Human Zweikämpfe im Finale auf dem RoboCup 2019. Es gab dabei 56 Zweikämpfe, 27.8 %.

Zum Finale hingegen konnte das gegnerische Team Nao-Team HTWK deutlich besser in die eigene Feldhälfte gelangen, obwohl beide Teams fast die gleiche Anzahl an Anstößen hatten. Dabei bekam B-Human zwei und Nao-Team HTWK drei Anstöße. Hierfür gibt es mehrere

Gründe. Zum einem kamen von den 31 Stürzen in Zweikämpfen 10 davon im Finale vor. Dies reicht aber nicht für eine vollständige Erklärung für einen solchen großen Unterschied zwischen dem Finale und dem restlichen Wettbewerb aus. Es ist also wahrscheinlicher, dass der Zweikampf vom Team Nao-Team HTWK besser ist und häufiger gewinnt, zumindest im Vergleich zum Team B-Human. Deshalb wurde zusätzlich untersucht, wie oft welches beider Teams einen Zweikampf im Finale gewann. Gewonnen bezeichnet hierbei, dass der Ball nach dem Zweikampf hinter dem Gegner liegen und der Ballbesitz erhalten bleibt. Falls der Ball nur leicht hinter den Gegner rollt und der Ball spielende Roboter umfällt oder den Ball aus der Sicht verliert (und somit nicht direkt zum Ball läuft), wodurch der Gegner den Ballbesitz gewinnt, zählt das als ein gewonnener Zweikampf für den Gegner. Hierfür wurden zusätzlich nur die Situationen betrachtet, in denen von beiden Teams jeweils ein Roboter auf der gegenüberliegenden Seite unmittelbar nahe am Ball stand. In jeder anderen Situation hätte jeweils ein Roboter einen großen Vorteil, da dann der Gegner teilweise zu spät am Ball wäre oder der Ball einfach geradeaus hätte gespielt werden können. Diese Zweikämpfe werden im Folgenden als Nahzweikämpfe bezeichnet. B-Human gewann davon nur knapp 50 %. Auf den gesamten RoboCup betrachtet, gewann B-Human 26 von 51 Nahzweikämpfe. Allein im Finale wurden 12 Nahzweikämpfe gewonnen, aber 16 verloren. In den restlichen Spielen wurden im Schnitt 2.3 gewonnen und 1.5 verloren. Die Spielweise des Nao-Team HWTK führt also zu mehr Nah- und Zweikämpfen, zudem werden letztere auch etwas häufiger gewonnen. Gründe hierfür sind, dass das aktuelle Zweikampfverhalten gegen den Ball tritt, selbst wenn dahinter ein Gegner steht. Dadurch prallt der Ball ab und rollt häufig hinter den eigenen Roboter. Auch wird der Ball im Nahbereich, wenn dieser direkt vor den Füßen liegt, oftmals nicht erkannt. Wenn der Ball in solchen Situationen nicht zeitnah zur Seite rollt, verliert der Roboter den Ball, läuft rückwärts und aktiviert sein Verhalten für die Ballsuche. Nao-Team HTWK hingegen, wie es schon in Abschnitt 3.1 erwähnt wurde, bleibt seit dem RoboCup 2019 hinter dem Ball stehen und wartet. Gleichzeitig werden Bälle in solchen Situationen häufiger erkannt, wodurch deren Roboter seltener den Ball verlieren und den Ballbesitz sichern können. Zusätzlich ist Nao-Team HTWK sehr gut im Dribbeln. Gewinnen deren Roboter daher einen Zweikampf, können sie eine deutlich größere Strecke zurücklegen, bevor es zu einem erneuten Zweikampf kommt.

Weitere Schwächen, deren Auswirkungen schwer durch Videoaufnahmen und Logdaten auswertbar sind, sind Probleme in Situationen, in denen die als die beste bewertete Schussentscheidung entweder offensichtlich schlecht ist oder die bisherige Modellierung und Perzeption, bestehend zum Teil aus der Roboter- und Ballerkennung, die Entscheidungsfindung sehr erschweren. Offensichtlich schlechte Entscheidungen betreffen das Laufen in andere Roboter oder das Schießen des Balles außerhalb des Feldes. Der bisherige Zweikampf führt immer einen Schuss aus, egal wie schlecht die Bewertung ist. Würde also jeder Schuss, der ausgeführt werden darf, den Ball außerhalb des Feldes schießen, wäre das Verhalten gezwungen, einen solchen Schuss auszuführen. Auf die Perzeption bezogen ist das Verhalten sehr davon abhängig, wie gut diese ist. False-Positives oder auch Verschiebungen in der Robotererken-

nung führen zwangsläufig zu Entscheidungen, die für einen menschlichen Außenbetrachter als unsinnig bewertet werden. Jedoch muss das Verhalten stabil genug funktionieren, damit es auch bei Fehlerkennung noch sinnvolle Entscheidungen trifft. Insbesondere bei Verschiebungen in der Erkennung, wie in Abbildung 7.4 zu sehen, ist aus der Sicht des Roboters der erkannte Roboter vor statt hinter dem Ball. Denn die Mitte der unteren Kante der weißen Box ist die geglaubte Position des Gegners. Dabei ist der Punkt zwischen den Ursprüngen beider Füße als Position zu verstehen. Die Fußlängen zählen nicht dazu, weshalb jedes erkannte Hindernis eigentlich noch einige Zentimeter verschoben werden müsste. Dies führt aktuell zwangsläufig dazu, dass der Ball geradeaus gespielt werden kann, da ja scheinbar hinter dem Ball kein Hindernis steht.

Neben diesen Beobachtungen zur Effektivität des Zweikampfes existiert aber auch noch das Problem mit den Stürzen in Zweikämpfen. Kategorisiert man die 31 Stürze, um zu verallgemeinern, wie die betroffenen Roboter relativ zueinander standen, so sind einige Auffälligkeiten zu bemerken. In Abbildung 7.5 sind alle wichtigen Konfigurationen dargestellt, in denen ein eigener Roboter (in schwarz) und ein gegnerischer Roboter (in hellblau) nahe am Ball stehen könnten. Dabei gab es auf dem RoboCup 2019 folgende Verteilung: 3-mal fielen Roboter im Fall a) um, 11-mal in Fall b) und 17-mal in Fall c). Stehen beide Roboter gegenüber, scheint ein eigener Roboter eher selten umzufallen. Betrachtet man aber den Kontext, so rollt der Ball in dieser Konfiguration häufig zur Seite. Anschließend dreht sich der eigene Roboter, um an den Ball zu laufen, und geht in die Konfiguration b) oder c) über. Spätestens hier läuft der eigene Roboter entweder selber auf die Füße des Gegners oder der Gegner läuft in unseren Roboter, wodurch sich dessen Füße unter die eigenen schieben. Folglich fällt der eigene Roboter um. Da dies weiterhin in solchen Situationen geschieht (siehe Abschnitt 5.9), muss eine Lösung im Verhalten gefunden werden.

Eine weitere Beobachtung ist, dass häufig im Spielverlauf der Ball durch einen Zweikampf weit weg geschossen wird. Der Ball kommt dabei zwar in den meisten Fällen dem gegnerischen Tor näher und wird häufig in Freiräume gespielt. Jedoch wird dadurch der Ballbesitz vorerst aufgegeben, wodurch der Gegner einige Sekunden mit dem Ball frei laufen kann, wodurch kein eigener Roboter einzugreifen vermag. Solche Geschehnisse waren zum Teil bedingt, weil die Schüsse nur eine Schussreichweite besaßen. Durch die entwickelten Schüsse aus Kapitel 6 besitzt der Zweikampf mehr Freiheiten zu entscheiden, wie weit die Schüsse gehen sollen, um den Ballbesitz nach dem Zweikampf zu sichern.

Zusammenfassend lässt sich sagen, dass der neue Zweikampf trotz der Probleme in der Robotererkennung stabile Entscheidungen treffen können muss. Gleichzeitig müssen Kollisionen mit anderen Robotern minimiert werden, ohne dabei den Ballbesitz zu verlieren. Auch sollen die neuen Schussreichweiten ausgenutzt werden. Die Art und Weise, wie die Roboter an den Ball laufen, die erst im Motion-Thread vollständig berechnet wird, bleibt unverändert, um den neuen Zweikampf für sich alleinstehend am Ende bewerten zu können. Insgesamt darf aber das Spielgeschehen durch ein passiveres Vorgehen auch nicht verlangsamt werden.

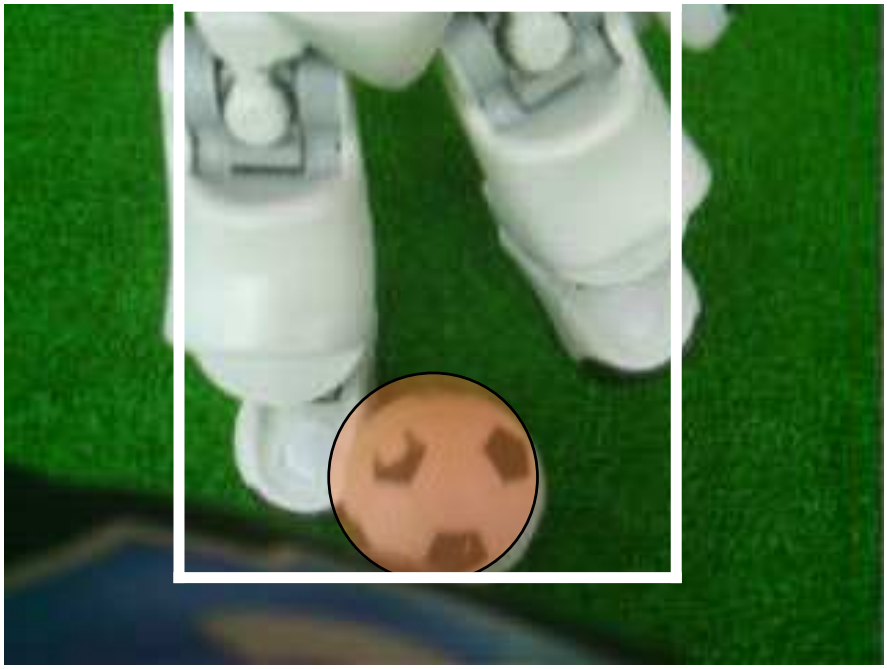

Abbildung 7.4 Ein Bild der unteren Kamera. Zu sehen ist ein erkannter Roboter (weiße Box) und der Ball (orange gefüllter Kreis).

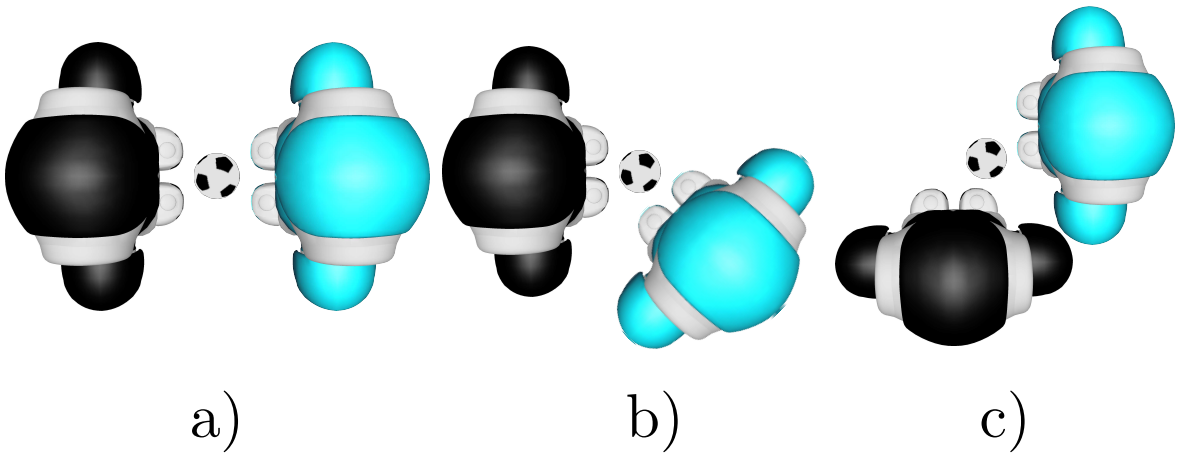

Abbildung 7.5 Dargestellt die Zweikampfsituationen, in denen die Roboter umfallen. Die Situationen sind dabei stark vereinfacht. Der schwarze NAO gehört zum eigenen Team, der blaue NAO zum gegnerischen Team. a) beide Roboter stehen gegenüber. b) ein Roboter ist leicht rotiert und steht diagonal zum Ball. c) ein Roboter ist 90 Grad rotiert zum Ball.

7.3 Der neue Zweikampf

Der neue Zweikampf soll Kollisionen mit anderen Robotern vermeiden und insgesamt sinnvollere Entscheidungen treffen. Wie schon in Abschnitt 3.2 erwähnt, ist eine übliche Herangehensweise hierfür die Verwendung einer Laufschrittplanung im Nahbereich. Mithilfe der

Perzeption und den erkannten Hindernissen kann entschieden werden, wo die Füße platziert werden können und wo nicht. Im aktuellen B-Human System existiert ein solcher Nahbereich nicht und aus den aktuellen fehlerbehafteten Robotererkennungen, welche nur aus vereinfachten Rechtecken bestehen, lässt sich schwer ein Mehrwert schaffen. Daher wird im Zuge dieser Arbeit nur ein vereinfachter Ansatz umgesetzt, um mithilfe der Auswahl der Schusstypen die Kollision mit anderen Robotern zu minimieren.

Damit nicht immer ein Schuss ausgeführt werden muss, kommen zwei Ansätze zum Einsatz. Zum einem soll es einen neuen Standard-Schuss geben, welcher vom Design das Kollidieren mit dem anderem Zweikampfroboter minimal halten soll, dennoch das Gewinnen des Zweikampfes sicherstellen kann. Zum anderen sollen Schüsse, die außerhalb des Feldes gehen oder durch andere Roboter blockiert werden, nicht ausführbar sein. Falls dadurch kein Schuss ausgeführt werden kann, soll sich der Roboter zwischen Tor und Ball stellen und eine kurze Zeit warten. Sofern nach einer Wartezeit kein geeigneter Schuss berechnet werden konnte, deaktiviert sich das Zweikampfverhalten. Aufgrund des aktuellen Verhalten-Frameworks würde dadurch in den meisten Fällen das Dribbelverhalten aktiv werden, welches den Ball in das Tor befördert. Diese beiden Ansätze zusammen sollen die Chance, den Zweikampf zu gewinnen, erhöhen sowie die Umsetzung von offensichtlich schlechten Schüssen minimieren.

Um mit den Problemen der Perzeption umzugehen, wird eine Korrektur für die erkannten Roboter eingefügt. Hierfür wird angenommen, dass die geglaubte Position des erkannten Roboters weder im noch vor dem Ball liegen kann, wenn die geglaubte Position sehr nahe am Ball liegt und der Ball erkannt wird. Stattdessen müsste dieser mit einem gewissen Abstand neben oder hinter dem Ball stehen. Die Korrektur soll solche Roboter somit nachträglich vom Ball weg verschieben. Roboter, die weiter weg vom Ball stehen, müssen nicht korrigiert werden. Einige Zentimeter Fehlerkennung auf mehr als einem Meter Entfernung sollte nämlich durch die Bestimmung der Schussrichtung zu keinen Problemen führen. Inwiefern diese Korrektur ausreichend ist, wird später in der Evaluation untersucht.

Zu guter Letzt müssen noch Schüsse, bei denen Kollisionen unvermeidbar sind, verhindert werden. Hierfür wird ein vereinfachter Ansatz einer Laufschrittplanung im Nahbereich umgesetzt. Die echten Positionen der Füße und Beine anderer Roboter sind unbekannt und es existiert, wie schon erwähnt, lediglich eine fehlerbehaftete Modellierung von erkannten Robotern. Daher soll für jeden möglichen Schuss die Schusspose berechnet werden. Diese beschreibt, wo sich der Roboter relativ zum Ball befinden wird, wenn er den Schuss ausführen kann. Mit dieser Schusspose soll geprüft werden, ob es zu einer möglichen Kollision kommen könnte. Für Vorwärtsschüsse können dann die Entfernungen von den eigenen Füßen zu denen der anderen Roboter bestimmt werden. Falls die Entfernung zu gering ausfällt, ist eine Kollision sehr wahrscheinlich. Für alle anderen Schüsse kann zwischen Schusspose und aktueller Feldposition eine Gerade gebildet werden. Um den Zweikampfroboter herum wird dann ein Kreis erzeugt und auf einen Schnittpunkt von Gerade und Kreis geprüft. Wird in einem der beiden Fälle eine Kollision erkannt und der Schussfuß ist weiter weg vom Hindernis als der Standfuß, wird die Bewertung des Schusses verschlechtert. Der Schuss könnte auch

komplett verboten werden, dafür müsste aber die Robotererkennung weniger False-Positives haben und geringere fehlerhafte Positionen ausgeben. Stattdessen wird der Schussfuß präferiert verwendet, der unwahrscheinlicher in eine Kollision resultiert. Gleichzeitig können so noch Tore geschossen werden, auch wenn eine Kollision sehr wahrscheinlich wäre.

Um die erwähnten Anforderungen in ein Verhalten für den Zweikampf zu bringen, soll ein Potentialfeld als Bewertungsfunktion verwendet werden. Dies erleichtert zum einem die Entscheidung über die Schussreichweiten, zum anderen wird das strategische Verhalten vorgegeben, wohin der Ball generell gespielt werden soll. Das Potentialfeld soll als Bewertungsfunktion dienen und sehr ähnlich zum bisherigen des alten Zweikampfes aufgebaut sein, dabei aber kurze Schüsse und mögliche Pässe mit berücksichtigen. Die Feldrichtung ist für den Zweikampf weiterhin irrelevant, jedoch wird diese später für Abschnitt 7.4.1 benötigt. Auch soll ein Kompromiss geschaffen werden, der Robotererkennung zu vertrauen, um mögliche zukünftige Zweikämpfe zu vermeiden. Dadurch soll weniger in Freiräume gespielt werden, welche nicht ausgenutzt werden können, gleichzeitig das Umspielen von gegnerischen Robotern ermöglichen. Auf approximierte dominante Regionen Diagramme oder Voronoi Diagramme wird bewusst verzichtet, da hierfür die aktuelle Robotererkennung nicht ausreichend verwendbar und zu fehlerbehaftet ist. Jedoch wird eine Umsetzung von Distanz- und Zeitberechnungen für den Einfluss von gegnerischen Robotern miteinbezogen.

7.3.1 Der neue Standard-Schuss

Der neue Standard-Schuss wird mithilfe dem in dieser Arbeit entwickelten Schussverfahren (siehe Kapitel 6) erstellt. Der Roboter soll hier mittig hinter dem Ball stehen, eine V-Form mit seinen Füßen bilden und danach seitlich weglaufen, was in einer 90 Grad Schussrichtung resultiert. Die V-Form löst sich dabei innerhalb der ersten zwei seitlichen Laufschritte auf. Die V-Form ist in Abbildung 7.6 dargestellt.

Ein übliches Problem im Zweikampf, bei dem sich beide Roboter gegenüber stehen, ist, dass der Ball von beiden Robotern mit den Füßen berührt wird, dadurch teilweise zwischen diesen kurzzeitig hängen bleibt und zusammengedrückt wird. Sobald ein Roboter leicht zurück oder zur Seite läuft, kann sich der Ball frei und zu den Seiten wegrollen. Die V-Form soll als Behälter für den Ball dienen, in dem er sich aufhalten kann, ohne aus diesem herausrollen zu können. Durch die Seitwärtsschritte wird der Ball anschließend mit der inneren Fußkante mitgezogen. Der Ball kann dabei nicht sofort hinter den eigenen Roboter rollen, da der Fuß hierfür im Weg ist. Wenn der Ball geradeaus oder diagonal rollt, prallt er am gegenüber stehenden Roboter ab und bewegt sich erneut in die V-Form hinein.

Ein großes Problem bei diesem Schuss ist aber, dass der Ball, insbesondere bei nicht optimalen Lichtbedingungen, selten bis gar nicht erkannt wird, wenn dieser in der V-Form liegt. Auch sind die Laufschritte sehr ungenau, wenn der Roboter mit parallel ausgerichteten Füßen innerhalb von zwei Laufschritten diese in eine V-Form genau hinter den Ball bringen soll. Daher soll das Einnehmen der V-Form nicht Teil des Standard-Schusses sein. Stattdessen

wird das im Motion-Thread vorhandene Verhalten für an den Ball laufen für den Standard-Schuss erweitert. Falls sowohl die Ballentfernung geringer als ein Schwellwert ist, jeweils für die x- und y-Translation, als auch die noch benötigte Rotation für die Schusspose unter einem Schwellwert liegt, sollen die Füße eine V-Form einnehmen. Die V-Form selber ist lediglich eine Rotationsanfrage an das Laufen. Jedoch muss diese so übergeben werden, dass sich der Roboter anschließend nicht über zwei Laufschritte dreht, sondern mit einem Laufschritt den einen Fuß dreht und mit dem zweiten den anderen. Dies ist bei dem Roboter NAO V6 umständlich, da beide Füße über das HipYawPitch-Gelenk verbunden sind. Dreht sich also ein Fuß in der z-Achse, so dreht sich der andere Fuß entgegengesetzt. Lediglich durch den Bodenkontakt bleibt der Standfuß an Ort und Stelle. Falls eine V-Form mit einem gesamten Winkel von 40 Grad eingenommen werden soll, kann also folgender Trick angewendet werden: Wie bei den neuen Schüssen, wird der Standfuß als Ursprung des Koordinatensystems verwendet. Ist der linke Fuß der Standfuß, so wird das Koordinatensystem um -20 Grad respektive 20 Grad gedreht, wenn der rechte Fuß der Standfuß ist. Die Rotation des Koordinatensystems verrechnet mit der Schussrichtung und der Rotation der Schusspose aus der Konfiguration ist dann die anzusteuernde Rotation. Die Translation ist der Translationsunterschied zu der Schusspose, welche durch eine Konfiguration relativ zum Ball definiert wird. Stehen beide Füße anfangs also parallel zueinander, so ist ein Fuß nach dem ersten Laufschritt bereits korrekt für die V-Form rotiert. Im zweiten Laufschritt rotiert der andere Fuß nach. Wurden die Laufschritte perfekt umgesetzt, so ist die V-Form eingenommen, ansonsten passen sich die Fußposition über die nächsten Laufschritte noch leicht an.

Abbildung 7.6 Die V-Form der Füße beim neuen Standard-Schuss

Für die Ausführung des Schusses wird normalerweise die Ballposition und Schussrichtung im Koordinatensystem des Schussfußes geprüft. Dieses Koordinatensystem wird zusätzlich um die Hälfte des Winkels der V-Form entgegen rotiert und um 50 mm in Richtung des Standfußes in der y-Achse verschoben, um die Ausrichtung mittig zwischen beiden Füßen zu bekommen. Eine perfekte Ausführung des Schusses ist nicht nötig, wodurch die erlaubte Abweichung der Schussrichtung sehr hoch sein darf. Auch kann der Schuss frühzeitig abgebrochen werden, wenn der Schuss mehr als zwei Key-Frames besitzt und der aktuelle Laufschritt dem dritten oder späteren Key-Frame entspricht. Wurde der Ball länger nicht gesehen, wird der Schuss abgebrochen. Der Roboter hat in einem solchen Fall bereits einen Laufschritt zur Seite gemacht und die Füße zusammengezogen. Hier ist es besser, nach dem Ball zu suchen, falls dieser doch am Gegner hängen blieb und der eigene Roboter für diesen effektiv nur aus dem Weg lief.

7.3.2 Das Potentialfeld

Das gesamte Potentialfeld für den Zweikampf soll aus insgesamt sieben einzelnen Feldern bestehen, unterteilt in unterschiedliche Aufgabenbereiche, die in Weltkoordinaten berechnet werden. Für die spätere Evaluation (siehe Abschnitt 7.4.1) wird noch ein achtes Feld hinzugefügt, welches beim Zweikampf keine Verwendung findet. Da die Felder nur als Bewertungsfunktion für den Zweikampf dienen, wird auf die Eigenschaft der Differenzierbarkeit verzichtet. Um trotzdem Feldrichtungen zu bestimmen, werden diese für Abschnitt 7.4.1 als Vektoren von Spielfeldpunkt zum Potentialfeldmittelpunkt definiert. Für die gesamte Umsetzung gilt: Das Tor ist gut, Bereiche außerhalb des Feldes sind schlecht, Teammitglieder sind gut, während gegnerische und unbekannte Roboter vermieden und der Ballbesitz beibehalten werden soll. Aus Erfahrung durch Simulationstests und echten Spielen dienen zwei weitere Felder dem Umspielen von Gegnern und einem Art Zeitspiel.

Zur Berechnung der Feldstärke wird ein lineares Modell verwendet. Die Variable *Radius* bezeichnet dabei die Einflussreichweite des Feldes. Distanzen größer diesem *Radius* erzeugen eine Feldstärke von 0. *Distanz* beschreibt bei abstoßenden Feldern die Entfernung des Feldpunktes zum nächstliegenden Maximum, respektive Minimum bei anziehenden Feldern. *Stärke* wird aus der maximalen Stärke des Feldes und dessen maximalem Radius *MaxRadius* berechnet. So soll bei einer Distanz von 0 die maximale Feldstärke entstehen.

$$\text{Feldstärke} = \max((\text{Radius - Distanz}) \cdot \text{Stärke}, 0) \tag{7.1}$$

$$\text{Stärke} = \frac{\text{Feldwert}}{\text{MaxRadius}} \tag{7.2}$$

Die Berechnung der Feldrichtung wird für den Zweikampf nicht benötigt, da das Potentialfeld nur als Bewertungsfunktion für einzelne Feldpositionen dienen soll. Da sie aber in Abschnitt 7.4.1 als Art Pfadplanung benötigt wird, um den Gradientenabstieg von Potentialfeldern zu nutzen, wird diese Feldrichtung dafür wie folgt berechnet:

$$\text{Feldvektor} = (\text{Potentialfeldmittelpunkt - Feldpunkt}) \tag{7.3}$$

$$\text{Feldrichtung} = -\frac{\text{Feldvektor}}{||\text{Feldvektor}||} \tag{7.4}$$

Durch die Gewichtung über die Feldstärke wird anschließend die Feldrichtung korrigiert. Anziehende Felder haben negative Feldstärken, wodurch die Feldrichtung in Richtung des anziehenden Potentialfeldes zeigt. Hingegen haben abstoßende Felder positive Feldstärken, damit die Feldrichtung entgegengesetzt zeigt.

Für die Abbildungen Abbildung 7.7 bis Abbildung 7.15 gilt dabei folgende Legende: Braun steht für sehr stark abstoßende Feldbereiche, grün für neutrale (Feldstärke gleich 0) und blau für stark anziehende Feldbereiche. Die Pfeile zeigen die Feldrichtungen. Das weiße Viereck ist der Ball spielende Roboter, aus dessen Sicht die Potentialfelder berechnet werden. Blaue Boxen sind Gegner und schwarze Teammitglieder. Der orangene Kreis ist der Ball. Das eigene Tor ist links, das gegnerische rechts.

Feldrand: Für den Feldrand soll ein abstoßendes Feld verwendet werden, welches kein einzelnes Maximum besitzt, sondern jeder Punkt außerhalb des Feldes dem Maximum entspricht. Das Minimum hingegen wird als Entfernung zum Feldrand definiert. Jeder Punkt, der weit genug vom Feldrand entfernt ist, aber innerhalb des Spielfeldes, ist gleich dem Minimum. Das Minimum ist hierbei gleich 0, damit dieses neutral ist. In Abbildung 7.7 ist ein bespielhaftes Feld dargestellt, mit einem Radius relativ zum Feldrand von 500 mm.

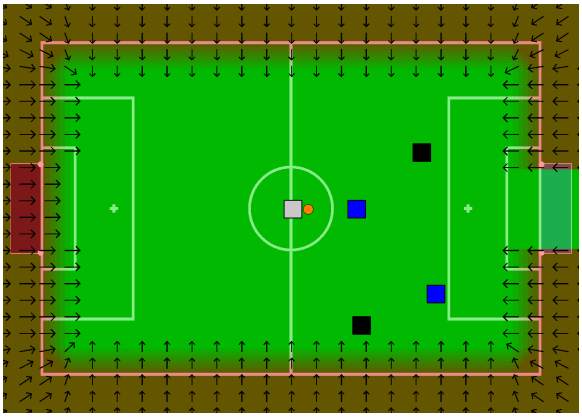

Abbildung 7.7 Das abstoßende Potentialfeld des Feldrandes. Außerhalb des Feldes ist das Maximum, das Minimum mit leichten Abstand zur Feldlinie innerhalb des Feldes.

Hindernisse: Als Hindernisse werden alle Robotererkennungen gesehen, egal ob gegnerische oder Roboter aus dem eigenen Team. Diese sollen abstoßende Felder repräsentieren. Hierbei soll das Maximum um die Position des Hindernisses herum liegen. Das Maximum ist somit kein einzelner Punkt, sondern durch einen inneren Radius um das Hindernis herum beschrieben, welcher sich dynamisch verändert. Jeder Punkt innerhalb dieses inneren Radius gehört zum Maximum. Ebenso soll das Minimum kein fester äußerer Radius sein, sondern durch die Distanz zwischen dem Hindernis und dem nächst stehenden Teammitglied beschrieben werden. Dieser äußere Radius berechnet sich durch

$$\text{Radius}_{\text{außen}} = \frac{\text{Distanz}_{\text{Hindernis}} + \text{Distanz}_{\text{Teammitglied}}}{2} \tag{7.5}$$

Das Ziel ist hierbei, Regionen zu erstellen, in denen das eigene Team die Kontrolle hat, wie es auch in Nakanishi u. a. [2008] der Fall ist. Jedoch statt durch die Verwendung von approximierten dominanten Regionen Diagrammen soll die Berechnung der Feldstärke des Potentialfeldes diese Bewertung übernehmen. Mit der Berechnung des äußeren Radius $Radius_{\text{außen}}$ kann somit zugewiesen werden, dass jeder Feldpunkt, der näher an einem Teammitglied liegt, nicht von einem Hindernis beeinflusst wird. Alle anderen Feldpunkte werden zunehmend schlechter, je näher sie an einem Hindernis liegen. Der innere Radius $Radius_{\text{innen}}$ soll sich basierend auf der Torentfernung verändern und durch einen Abstandsbereich, definiert durch die Parameter $Innen_{\text{min}}$ und $Innen_{\text{max}}$, relativ zu $Radius_{\text{außen}}$ berechnet werden. Gegeben einem minimalen und maximalen Abstand $Innen_{\text{min}}$, $Innen_{\text{max}}$, der Entfernung zur Tormitte vom Feldpunkt *Torentfernung* und einer Interpolationsreichweite IR gilt:

$$\text{Factor}_{\text{Radius}_{\text{innen}}} = \min\left(0, \frac{\text{Torentfernung}}{\text{IR}}\right) \tag{7.6}$$

$$\text{Radius}_{\text{interpoliert}} = \text{Factor}_{\text{Radius}_{\text{innen}}} \cdot \text{Innen}_{\text{min}} + (1 - \text{Factor}_{\text{Radius}_{\text{innen}}}) \cdot \text{Innen}_{\text{max}} \tag{7.7}$$

$$\text{Radius}_{\text{innen}} = \text{Radius}_{\text{außen}} - \text{Radius}_{\text{interpoliert}} \tag{7.8}$$

IR ist dabei ein Parameter und beschreibt, ab welcher Entfernung zum Tor $Factor_{\text{Radius}_{\text{innen}}}$ kleiner wird. Für die Berechnung der Feldstärke muss nun noch die Distanz vom Feldpunkt zum nächstliegenden Randpunkt des Maximums berechnet werden.

$$\text{Distanz} = \max(0, \text{Distanz}_{\text{Hindernis}} - \text{Radius}_{\text{innen}}) \tag{7.9}$$

Zwar existiert keine Modellierung darüber, wie erkannte Roboter rotiert sind, dennoch kann angenommen werden, dass Positionen hinter einem Gegner schwerer erreichbar sind als Position vor diesem, aus Sicht des Gegners. Falls der Gegner maximal schlecht ausgerichtet ist,

müsste sich dieser zuerst um 180 Grad drehen, um den Ball laufen und währenddessen erneut drehen. Wenn dieser optimal ausgerichtet ist, müsste er nur noch an und um den Ball laufen. Der eigene Roboter könnte in beiden Fällen im Normalfall einfach geradeaus laufen, was deutlich schneller ist. Falls der eigene Roboter falsch ausgerichtet ist, kann angenommen werden, dass spätestens beim oder nach dem Ausführen des Schusses dieser in Schussrichtung ausgerichtet und somit ideal rotiert ist. Daher sollen Positionen, die hinter dem Hindernis sind, besser sein als vor diesem. Hierfür wird ein Interpolationsfaktor W eingeführt. Dieser ist gleich 0, wenn die Feldposition in einem kleinen Winkelbereich vor dem Hindernis liegt und gleich 1, wenn der Winkel zwischen Feldposition und Hindernis zu groß ist. Dabei ist eine gerade Richtung zum eigenen Tor ein Winkel von 0 Grad und in Richtung des gegnerischen Tores ein Winkel von 180 Grad. Zwischen den Bereichen wird interpoliert. Da den Robotererkennungen nicht vollständig vertraut wird, soll es einen Winkelbereich geben, in dem W immer 0 ist (W_{min}), und erst außerhalb davon zu 1 interpoliert werden (W_{max}). Ist der Winkel kleiner W_{min}, so ist W gleich 0, ist der Winkel zwischen W_{min} und W_{max}, so liegt auch W zwischen 0 und 1. Gegeben den Parametern W_{min} und W_{max}, gilt:

$$W = \min\left(1, \frac{\max\left(0, \operatorname{abs}\left(\text{vectorToObRot}^{(x)}\right) - W_{\text{min}}\right)}{W_{\text{max}}}\right) \tag{7.10}$$

Gleichung 7.5 wird nun mit W erweitert, um $Radius_{\text{außen}}$ dynamisch zu verringern:

$$\text{Radius}_{\text{außen}} = \frac{\text{Distanz}_{\text{Hindernis}} + \text{Distanz}_{\text{Teammitglied}}}{2} \cdot W \tag{7.11}$$

Durch die Interpolation von $Radius_{\text{innen}}$ sollte der Einfluss von Gegnern minimal gehalten werden, falls mögliche Ballendpositionen hinter diesen nahe am Tor liegen. Denn nahe am Tor soll, wenn entsprechend parametrisiert, $Radius_{\text{innen}}$ an Feldpositionen um ein Hindernis herum negativ sein. Das eigentliche Maximum wird somit nie erreicht und die Feldstärke ist folglich auch im Hindernismittelpunkt schwächer. Auch wird ausgenutzt, dass Feldbereiche hinter einem gegnerischen Roboter besser sind als seitlich neben diesem. Zusätzlich wird die Feldstärke für jede Position im Tor gleich 0 gesetzt, da Hindernisse im Tor keinen Einfluss haben sollen.

Zwischen den Hindernissen wird hier nicht unterschieden, obwohl B-Human eine Repräsentation zur Teamzuordnung besitzt. Diese basiert auf der Erkennung der Trikotfarbe und wird aufgrund ihrer schlechten Performanz im gesamten System bisher nicht verwendet. Durch die Berechnung von $Radius_{\text{außen}}$, basierend auf der Entfernung zum nächst stehenden Teammitglied, sollte die Feldstärke von Hindernissen, die eigentlich Teammitglieder sind, nur einen geringen Einfluss nehmen. Nur wenn die geglaubte Position auf dem Feld vom Teammitglied stark von der des Hindernisses abweicht, kann das Potentialfeld einen negativen Einfluss ha-

ben. Jedoch kann man hier unmöglich wissen, ob nicht doch ein echter gegnerischer Roboter dort steht. Die Feldrichtungen hinter den Teammitgliedern resultieren aus der Tatsache, dass Gegner und Teammitglieder nicht unterschieden werden.

Beispiele für dieses Potentialfeld sind in Abbildung 7.8 zu sehen.

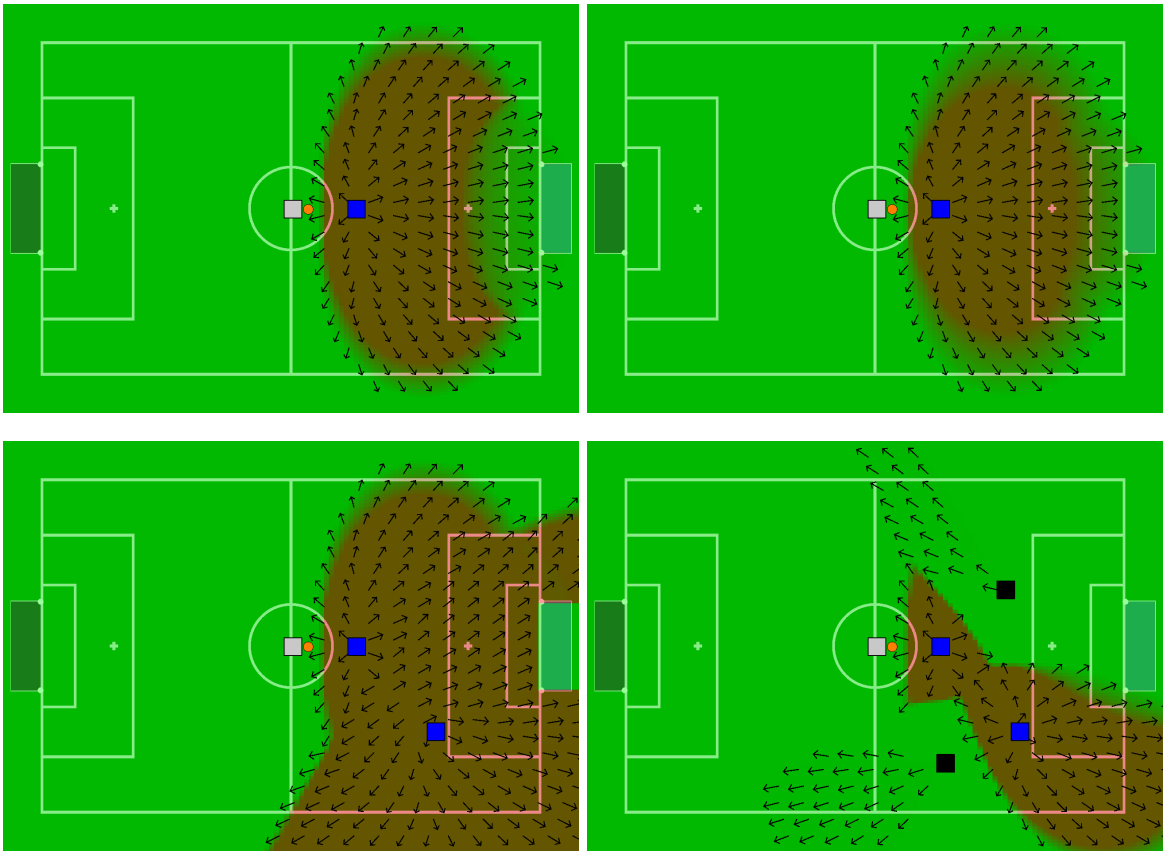

Abbildung 7.8 Das abstoßende Potentialfeld der Hindernisse. Oben links mit Default Werten $Innen_{min} = 125$ und $Innen_{max} = 1500$. Oben rechts mit $Innen_{min} = 250$ und $Innen_{max} = 500$. Unten links mit Default Werten und zwei Hindernissen. Unten rechts mit Default Werten, zwei Hindernissen und zwei Teammitgliedern.

Tor: Das Tor soll, ähnlich wie zuvor, ein sehr einfaches anziehendes Feld sein. Das Minimum liegt in den Ecken in der eigenen Feldhälfte, das Maximum ist der gesamte Torbereich. Für die Feldstärke ist der Radius daher die Entfernung von der gegnerischen Tormitte zu den eigenen Ecken. Das Feld ist in Abbildung 7.9 dargestellt.

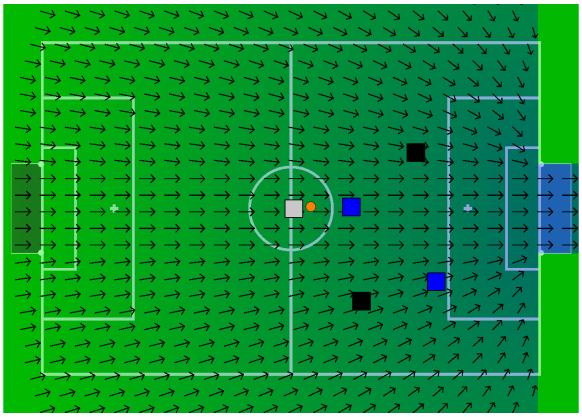

Abbildung 7.9 Das anziehende Potentialfeld für das gegnerische Tor.

Torwinkel: Die Ecken in der gegnerischen Feldhälfte liegen strategisch ungünstig, denn von dort aus sind aufgrund des Toröffnungswinkels Torschüsse fast unmöglich. Daher soll hier ein schwaches abstoßendes Feld verwendet werden. Dieses soll einer Feldrichtung entgegen der gegnerischen Grundlinie entsprechen und das Minimum auf Höhe der Strafraumlinie haben. So sollen die Bälle wieder etwas tiefer in das Spielfeld gespielt werden, um eine bessere Torschussmöglichkeit zu erhalten. Ob diese strategische Entscheidung tatsächlich von Vorteil ist, ist schwer zu sagen. Bezogen auf das menschliche Fußballspiel wird dort ähnlich vorgegangen, um einen besseren Toröffnungswinkel zu bekommen sowie die gegnerischen Spieler zu umspielen, unter anderem durch Pässe nahe vor dem Tor. Beim Roboterfußball könnte ein Verzicht hingegen bei schlechteren Teams sowie durch Ausnutzen der Gegebenheiten dafür sorgen, dass es zu häufigen Eckstößen für das eigene Team kommt. Auch könnten unter Umständen die gegnerischen Roboter leicht umspielt werden, da aus Erfahrung die Teams häufig Probleme mit diesem Feldbereich nahe am Tor und der Grundlinie haben. In Abbildung 7.10 ist das Feld zu sehen.

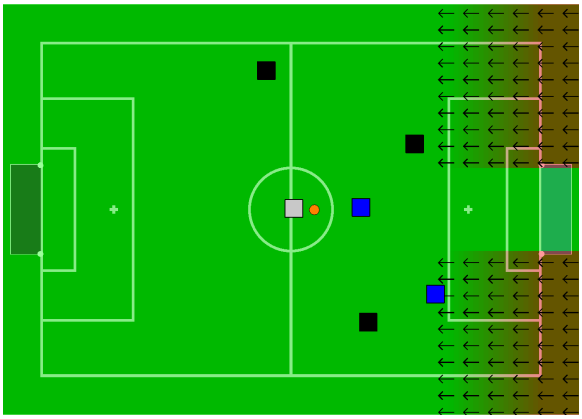

Abbildung 7.10 Das abstoßende Potentialfeld der gegnerischen Grundlinie.

Ballentfernung: Um den Ballbesitz zu gewährleisten, wird sich an dem Team Nao-Team HTWK orientiert. Deren Roboter dribbeln primär den Ball in Richtung des Tores. Selbst in Zweikämpfen wird der Ball nur um die Gegner herum gedribbelt, wodurch sie lange im Ballbesitz bleiben. Dies soll durch ein Potentialfeld erreicht werden, welches basierend auf der Entfernung zum Ball einen größeren Bereich für das Minimum hat und bis zu einer größeren Entfernung zum Maximum wird. Dadurch sollen kurze Schüsse präferiert werden, welche zwar eine kurze Distanz aufweisen aber eine bessere Spielqualität ermöglichen. Weite Schüsse hingegen werden vermieden und können nur durch die Addition anderer Potentialfelder ermöglicht werden. Da das Feld kein reines anziehendes sein soll, sondern sowohl ein anziehendes als auch abstoßendes, damit es einen Einfluss über das gesamte Spielfeld hat, wird die Feldstärke folgendermaßen modifiziert:

$$\text{Feldstärke}_{\text{ball}} = \text{Feldstärke} - \frac{\text{Feldwert}_{\text{ball}}}{2} \tag{7.12}$$

Die Feldstärke wird also um die Hälfte der Feldstärke verschoben. Das Minimum und das Maximum sind somit die gleichen Stärken, nur mit unterschiedlichen Vorzeichen.

Damit hinterher Rückpässe keine Seltenheit werden, nimmt dieses Potentialfeld keinen Einfluss auf Feldpositionen hinter dem Ball, bezogen auf den Vektor zwischen Ball und gegnerischem Tor. Dafür wird der Winkel zwischen Feldposition und Ballposition berechnet. Ist der absolute Winkel davon größer einem Schwellwert, so wird das Feld gleich 0 gesetzt. Das Feld ist in Abbildung 7.11 zu sehen.

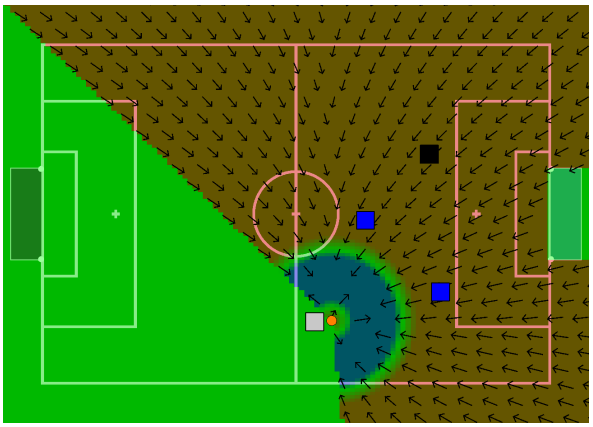

Abbildung 7.11 Das anziehende und abstoßende Potentialfeld des Balles.

Pässe: Das Potentialfeld der Hindernisse erlaubt bereits in der Theorie indirekte Pässe. Hierfür müssen die Stärken der Felder aber sehr gut aufeinander abgestimmt sein. Gleichzeitig wird nicht beachtet, dass Ballendpositionen nahe an einem Teammitglied (oder in dessen Laufweg) besser sind als irgendwo in einem Freiraum. Dies ist sehr gut in Abbildung 7.8 zu erkennen, da jede Position ohne eine Feldrichtung eine Feldstärke gleich 0 hat und somit, bezogen auf das Potentialfeld der Hindernisse, gleich gut ist. Daher wird ein anziehendes Po-

tentialfeld für Pässe hinzugefügt. Der Punkt für das Maximum ist definiert als feste Distanz vom Teammitglied in Richtung Tor. So soll ein Pass mit einem gewissen Abstand vor dem Teammitglied gespielt werden. Die Größe des Potentialfeldes ist ein statischer Radius. Das Minimum liegt in der Mitte des Feldes, das Maximum von 0 am Rande und außerhalb diesem. Damit aber nun sowohl keine Pässe entstehen, wo der Ball neben einem Gegner liegen bleibt, als auch sehr kurze Pässe, bei denen der Ball nur einen halben Meter rollt, wird dieses Potentialfeld noch mit der Distanz des Ball spielenden Roboters und allen Hindernissen skaliert. Ist ein Hindernis eine feste Entfernung nahe am Feldpunkt, so wird das Potentialfeld an dieser Stelle auf 0 gesetzt. Hierfür wird die Entfernung von einem Meter verwendet. Der aktuell weitreichendste Schuss aus dem Laufen beträgt, nach Abschnitt 6.4 zu urteilen, 3.2 m. Bei dem aktuellen Reibungswiderstand für den Ball von 0.35 $\frac{m}{s^2}$ ergibt sich mit der Formel

$$\begin{aligned}\text{Geschwindigkeit} &= \sqrt{\text{Reibungswiderstand} \cdot \text{Distanz} \cdot 2} \\ &\Rightarrow \sqrt{0.35\frac{\text{m}}{\text{s}^2} \cdot \text{Distanz m} \cdot 2} = 1.496\frac{\text{m}}{\text{s}}\end{aligned}$$

eine Ballgeschwindigkeit von 1.496 $\frac{m}{s}$. Verrechnet mit der Ballreibung erhält man die Dauer, bis der Ball nach dem Schuss liegen bleibt: $\frac{1.496\frac{m}{s}}{0.35\frac{m}{s^2}} \approx 4.276s$. Bei der aktuellen Laufgeschwindigkeit von 0.25 $\frac{m}{s}$ kann ein Roboter in der Zeit $0.25\frac{m}{s} \cdot 4.276s \approx 1.07m$ laufen. Unter Berücksichtigung, dass der Passroboter zum Zeitpunkt des Schusses noch steht oder nicht optimal in Richtung der Ballendposition rotiert ist, wird die Distanz, die der Roboter in der Zeit zurücklegen kann, unter einem Meter liegen. Ein Meter Entfernung als Mittelpunkt des Potentialfeldes ist daher ein guter Kompromiss. Ob das betroffene Hindernis ein Teammitglied oder ein Gegner ist, wird hierbei nicht unterschieden, da die Erkennung wie bereits erwähnt, nicht ausreichend gut ist, um Fehlentscheidungen zu vermeiden.

Für die Skalierung, basierend auf der Entfernung zum Ball spielenden Roboter, muss der Feldpunkt für den Pass eine Mindestentfernung weit weg liegen und wird über eine maximale Entfernung verstärkt. Dies soll aber nur für Passpositionen in der gegnerischen Feldhälfte gelten. Denn jeder Pass erzeugt ein Risiko, welches zum Ballverlust führen kann. Ein solcher Ballverlust in der eigenen Hälfte kann sehr schnell zu einem Gegentor führen. Die Feldstärke wird also folgendermaßen modifiziert:

$$\text{Feldstärke}_{\text{pass}} = \text{Feldstärke} + \max\left(0, \frac{\text{Distanz}_{\text{Passroboter}}}{\text{ReichtweiteInterpolierung}}\right) \cdot \text{Stärke}_{\text{pass}}$$

ReichtweiteInterpolierung ist dabei der Skalierungsparameter. Stehen beide Roboter nebeneinander, so gibt es keinen Bonus. Stehen sie weit auseinander, so gibt es den vollen Bonus. Zusätzlich wird, sofern das Potentialfeld der Ballentfernung berechnet wurde, dieses für jede Feldposition wieder entfernt, für die das Potentialfeld der Pässe eine Feldstärke ungleich 0 berechnet hat. Dadurch soll das Feld der Ballentfernung keinen negativen Einfluss auf Pässe haben.

Das Feld für drei Teammitglieder ist in Abbildung 7.12 zu sehen.

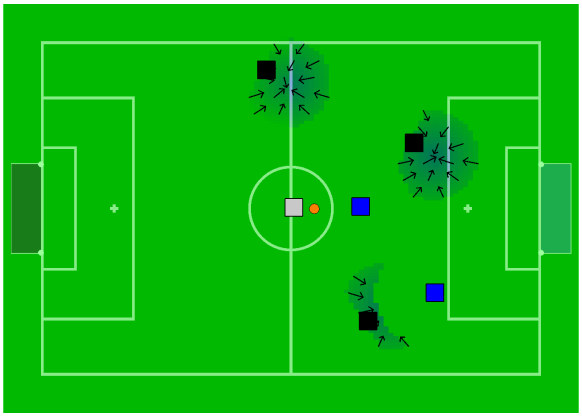

Abbildung 7.12 Das anziehende Feld für Pässe. Pässe in der eigenen Hälfte sind weniger gut, Pässe in der Nähe von Hindernissen werden nach Möglichkeit vermieden.

Zeitspiel: Aus Erfahrung durch Spiele gegen das Nao-Team HTWK ist bekannt, dass B-Human häufig Probleme hat, den Ball in die gegnerische Hälfte zu bekommen. Als Folge läuft ein Roboter seitlich dem Ball hinterher, während der gegnerische Roboter geradeaus mit dem Ball dribbelt. Da ein B-Human Roboter dem Ball hinterherläuft, aber diesen nicht spielen kann, dürfen keine anderen Roboter des Teams um den Ballbesitz kämpfen und der Ball bewegt sich zwangsläufig über die Zeit immer näher an das eigene Tor. Um diesem Verhalten entgegen zu wirken, wird ein Potentialfeld hinzugefügt, welches einen langen Schuss quer über das Feld bekräftigen soll. Dadurch müssen sich beide Teams neu auf dem Feld aufstellen. Dies ist vorteilhaft für die B-Human Roboter. Denn häufig ist die Aufstellung durch den Roboter, der dem Ball nur hinterherläuft, sehr lückenhaft. Diese nachteilige Aufstellung kann auf diese Weise zeitweise behoben werden. Um ein nutzloses Zeitspiel zu verhindern, sind nur Feldpositionen in der eigenen Hälfte vom Potentialfeld eingeschlossen. Der Mittelpunkt ist dabei der y-Achsenpunkt auf der jeweils gegenüberliegenden Seite zur aktuellen Ballposition, in der Mitte von Feldmitte und Außenlinie, die mit den aktuellen Feldmaßen 1.5 m beträgt. Die Feldstärke für das Minimum wird dabei mit der Ballposition skaliert. Je näher der Ball an der Außenlinie liegt, desto größer soll das Minimum werden. Die Skalierung ist dabei die y-Position des Balles über die Strecke von Mittelpunkt des Feldes zur Außenlinie, also 3 m. Es gilt somit:

$$\text{Feldstärke}_{\text{Zeitspiel}} = \text{Feldstärke} \cdot \min\left(1, \frac{\text{abs}\left(\text{Ballposition}^{(y)}\right)}{3000}\right) \tag{7.13}$$

Dieses Potentialfeld ist in Abbildung 7.13 zu sehen. Eine alternative Lösung wäre eine Anpassung an die Rollenverteilung der Roboter, damit kein Roboter nur dem Ball hinterherläuft und eine mögliche Ballbesitzübernahme verhindert.

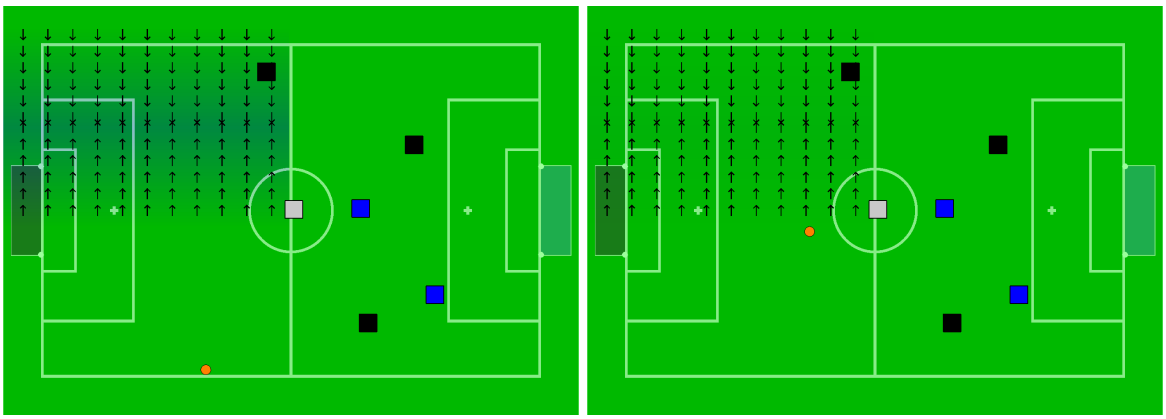

Abbildung 7.13 Das anziehende Potentialfeld für einen Schuss quer über das Feld.

Roboterausrichtung: Da für die Evaluation (siehe Abschnitt 7.4.1) später noch von dem Gradientenabstieg von Potentialfeldern Gebrauch gemacht werden soll, wird ein achtes Feld vorgestellt, welches für den Zweikampf keine Verwendung findet. Für die Evaluation soll unter anderem ein alternatives Dribbelverhalten angewendet werden. Hierfür ist zu berücksichtigen, welche Ausrichtung der Roboter auf dem Spielfeld hat. Ist er nach links rotiert, so soll er präferiert nach links dribbeln, respektive ist er nach rechts rotiert, so soll er präferiert nach rechts dribbeln.

Dafür wird, basierend auf der Ausrichtung des Roboters auf dem Feld, ein Kegel definiert, welcher als Potentialfeld dienen soll. Je näher ein Feldpunkt an den äußeren Rändern des Kegels liegt, desto schwächer ist das Feld. Der Kegel wird durch einen Radius und einen Winkel definiert. Die Feldstärke wird somit durch die Distanz zum Roboter und dem Winkel zwischen Roboter und Feldpunkt bestimmt. Feldpunkte, die eine größere Distanz als der Radius haben oder in einem größeren Winkel zum Roboter liegen als vom Kegel definiert, werden einer Feldstärke gleich 0 gesetzt. Für alle anderen Punkte wird die Feldstärke nicht durch Gleichung 7.1 berechnet, sondern mit der folgenden Formel:

$$\text{Feldstärke} = \max\left(0, \frac{\text{Feldwert}}{2} \cdot \left(\left(1 - \frac{\text{Winkel}}{\text{Kegel}_{\text{maxWinkel}}}\right) + \cdot \left(1 - \frac{\text{Distanz}}{\text{Kegel}_{\text{maxRadius}}}\right)\right)\right) \tag{7.14}$$

Die Variable *Winkel* wird dabei aus der relativen Ballposition aus dem Roboterkoordinatensystem berechnet.

Dieser Ansatz zeigt in der Praxis aber zwei Probleme. Dadurch, dass das Potentialfeld in die Richtung des Roboters zeigt, würde er den Ball automatisch in die eigene Feldhälfte spielen, wenn er um 90 Grad oder mehr rotiert ist. Die Ausrichtung des Roboters wird deshalb auf den Bereich -45 bis 45 Grad begrenzt, damit dieses Feld immer in die Richtung der gegnerischen Grundlinie zeigt. Das zweite Problem ist die allgemeine Verwendung der Ausrichtung des Ro-

boters. Da dieses Potentialfeld nicht für sich alleine für das Dribbeln verwendet wird, sondern in Kombination mit einer Teilmenge der anderen, hat dieses Feld keine echte Auswirkung. Nehmen wir an, der Roboter ist um 90 Grad gedreht und läuft um den Ball. Die Ausrichtung wird auf 45 Grad begrenzt und das Zentrum des Kegels liegt somit auch bei 45 Grad. Durch die Addition der anderen Felder wird aber durch den Gradientenabstieg eine Feldposition ermittelt, dessen Winkel zum Ball nur sehr unwahrscheinlich in eine Richtung von 45 Grad resultieren, sondern deutlich niedriger oder unter Umständen höher liegen wird. Der Roboter wird sich also über die Zeit immer weiter um den Ball drehen und hat somit irgendwann eine geringere Ausrichtung als 45 Grad. Dadurch wird auch die Ausrichtung des Kegels geringer, wodurch der Gradientenabstieg ebenfalls einen noch geringeren Winkel erzeugt. Der Roboter dreht sich somit noch mehr, bis der Kegel die gleiche Ausrichtung hat, als würde das Potentialfeld der Roboterausrichtung nicht benutzt werden. Um dies zu verhindern, wird die Ausrichtung des Roboters über die Zeit gefiltert. Dadurch soll sich die Ausrichtung des Kegels bei schnellen Änderungen der Roboterausrichtung nur langsam ändern. Hierfür könnte der Mittelwert der letzten x-vielen Ausrichtungen des Roboters berechnet werden. Aufgrund der deutlich einfacheren Implementierung und des geringeren Speicherverbrauches wird stattdessen ein Tiefpass-Filter angewendet. Dieser lässt sich wie folgt umsetzen:

$$\text{Ausrichtung}_{\text{neu}} = \text{Ausrichtung}_{\text{alt}} \cdot \text{Frequenz} + \text{Ausrichtung}_{\text{aktuell}} \cdot (1 - \text{Frequenz}) \tag{7.15}$$

Für die *Frequenz* wird der Wert 0.99 verwendet. Unter der Berücksichtigung, dass diese gefilterte Ausrichtung in jedem Cognition-Frame berechnet wird, erfolgt 60-mal die Sekunde eine Aktualisierung. Abbildung 7.14 zeigt dieses Potentialfeld.

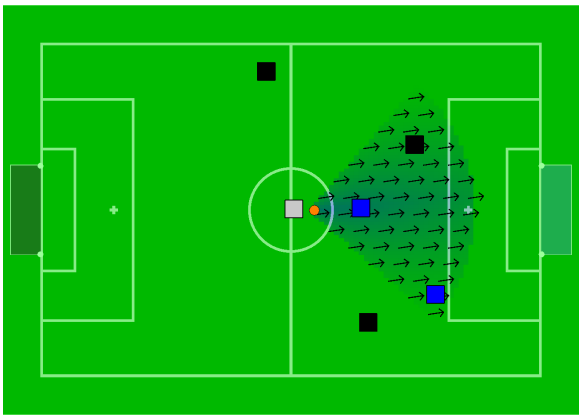

Abbildung 7.14 Das anziehende Potentialfeld für die Ausrichtung des Roboters.

In Abbildung 7.15 sind einige Beispiele für die Kombination aller Potentialfelder, außer der Roboterausrichtung, zu sehen. Die Feldstärken wurden dabei wie folgt parametrisiert: Der Feldrand hat eine Feldstärke von 3, Hindernisse eine von 1 und der Torwinkel eine von 0.5. Bei den anziehenden Feldern hat das Tor eine von -1.5, Pässe eine von -0.6, die Ballentfernung, wegen der Verschiebung, eine von -0.25 bis 0.25 und das Zeitspiel eine von -0.5. Die Feldstärken wurden per Hand zugewiesen und nicht optimiert. Stattdessen wurden die Werte nach Gefühl

eingestellt, damit die Kombination der Felder ein sinnvolles gesamtes Potentialfeld ergeben. Der Feldrand hat eine deutlich höhere Feldstärke als alle anderen Felder, damit auch bei der Kombination der anziehenden Felder jeder Feldpunkt außerhalb des Spielfeldes schlechter bewertet wird als jeder Punkt innerhalb. Das Tor ist etwas größer als die anderen und auch in der Kombination mit dem Feld des Torwinkels nahe -1, damit dieses deutlich anziehender ist als jedes andere einzelne Feld. Hindernisse liegen bei 1, damit diese nach Möglichkeit vermieden werden, gleichzeitig aber niedriger als das Tor, damit selbst nahe dem Tor dieses noch anziehend ist. Alle anderen Feldstärken liegen nahe -0.5, um einen gleichmäßigen Einfluss zu haben, aber die Richtung zum Tor unter Vermeidung von Hindernissen und dem Feldrand weiterhin gegeben ist.

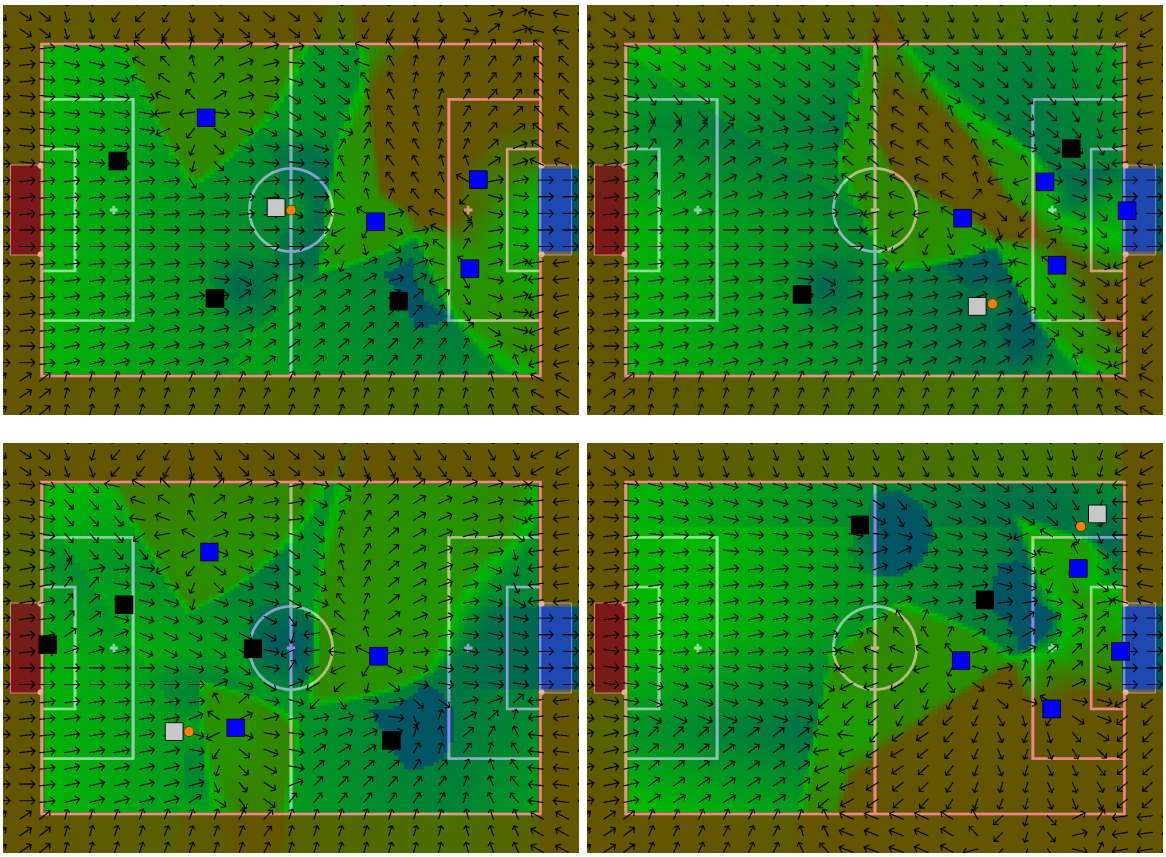

Abbildung 7.15 Alle Potentialfelder zusammen.

7.3.3 Das Zweikampfverhalten

Das Verhalten für den Zweikampf soll im Groben ähnlich zum aktuellen aufgebaut sein. Für jede mögliche Kombination aus Schussrichtung und Schusstyp soll das Potentialfeld aus Abschnitt 7.3.2 als Bewertungsfunktion aufgerufen und anschließend der beste Schuss ausgeführt werden. Da der Zweikampf in dem Verhaltens-Framework von B-Human eine Card darstellt, benötigt es auch eine Start- und Abbruchbedingung. Um die Vergleichbarkeit zum alten Zweikampf zu gewährleisten, werden die gleichen Bedingungen übernommen. Lediglich

die Abfrage über den Elfmeterraum wird entfernt. Offensichtlich strategisch ungünstige Entscheidungen werden durch das Potentialfeld und weiteren Abfragen, die später vorgestellt werden, vermieden.

Dieser Ablauf soll aber mit den neuen Anforderungen erweitert werden. Gleichzeitig müssen auch Einschränkungen getroffen werden, da die Rechenzeit durch die aufwändigere Bewertungsfunktion und der größeren Menge an Schussreichweiten nur begrenzt viele Prüfungen von Schussrichtungen erlaubt. In Abbildung 7.16 ist der Programmablauf des Zweikampfverhaltens vereinfacht abgebildet.

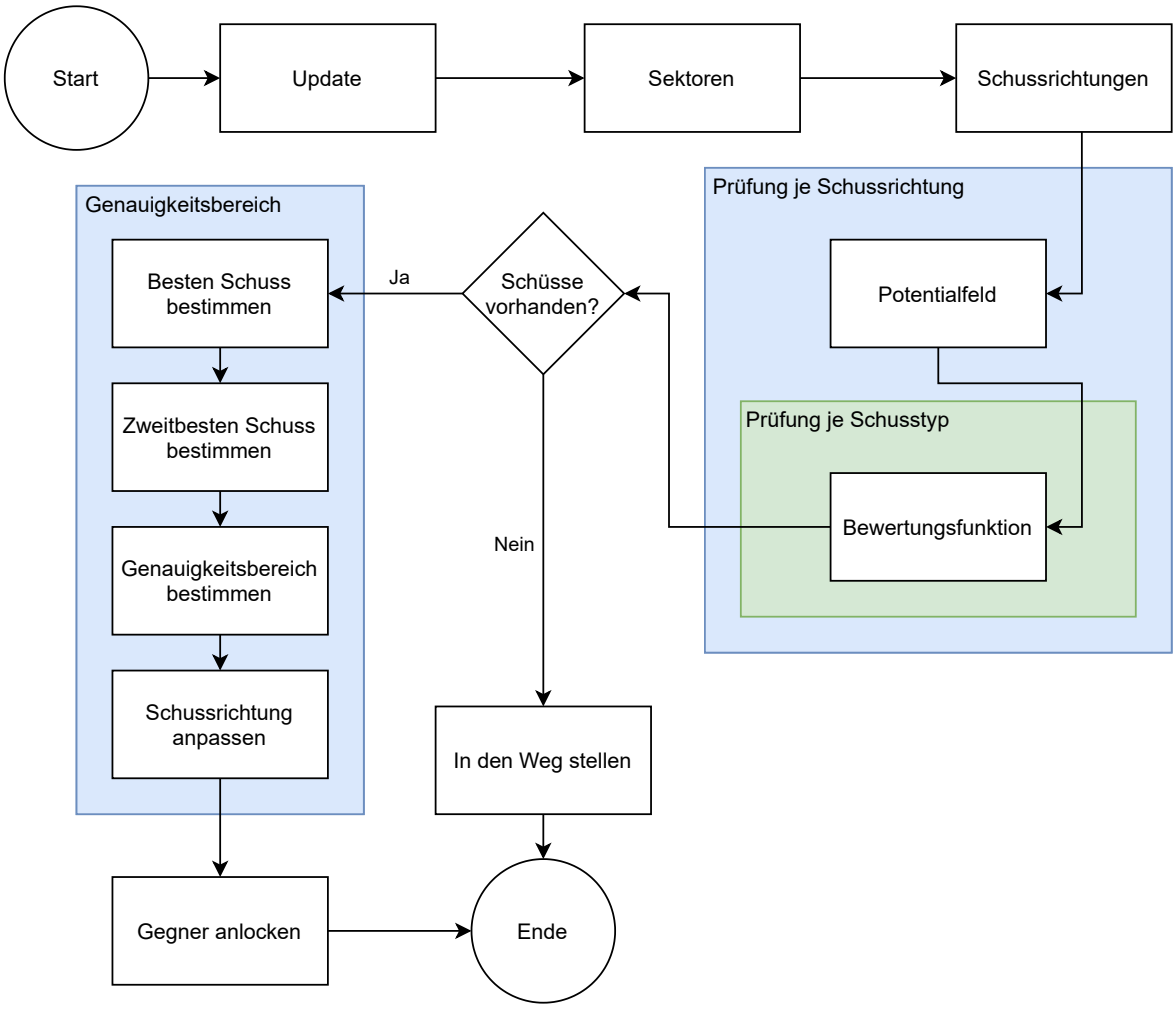

Abbildung 7.16 Das Verhalten des neuen Zweikampfes als Flussdiagramm. Blaue und grüne Kästchen stellen Schleifenblöcke dar. Das Ende zeigt das Ende der Berechnung vom Zweikampf, welcher im nächsten Cognition-Frame erneut aufgerufen wird und bei Start anfängt.

Für alle Abbildungen von Abbildung 7.18 bis Abbildung 7.23 gilt die Legende: Blaue Kästchen sind gegnerische Roboter, schwarze Kästchen Teammitglieder, der weiße Kasten der Ball spielende Roboter, der orange Kreis der Ball, das linke Tor ist das eigene und rechts das gegnerische.

Update: Sofern der Zweikampf bereits einen Schuss im letzten Cognition-Frame berechnet hat, wird dessen Schussrichtung mit der Odometrie aktualisiert. Ebenfalls wird das aktuelle Hindernis, welches am nächsten am Ball steht, neu berechnet. Dieses wird, um Folgen von Fehlern in der Perzeption zu minimieren, anschließend in seiner geglaubten Position korrigiert. Hierfür wird der Vektor zwischen Hindernis und Ball berechnet. Die Länge des Vektors ist die Distanz des Hindernisses zum Ball. Liegt die Distanz innerhalb eines Schwellwertes, so steht das Hindernis zu nahe am Ball. Unabhängig davon, ob das Hindernis nun vor oder hinter dem Ball steht, wird dieses nun mit einem festen Abstand hinter den Ball verschoben. Hierfür wird angenommen, dass die x-Koordinate des Hindernisses, also die Positionen nach vorne und hinten, fehlerhaft ist, die y-Koordinate, also die Position nach links und rechts, zwar auch fehlerhaft aber korrekter ist. Die Verschiebung erfolgt wie in Abbildung 7.17 dargestellt. Von der relativen Ballposition, im relativen Koordinatensystem des Roboters, der das Zweikampfverhalten ausführt, wird ein Punkt berechnet, dessen x-Position um den Ballradius hinter den Ball und um die y-Position des Hindernisses zur Seite verschoben ist. Mithilfe dieses Punktes wird der Vektor vom Ball zu diesem Punkt berechnet und um den Ballradius oder die y-Koordinatendifferenz verschoben, abhängig davon, welcher der beiden Werte größer ist. Gegeben der relativen Ballposition *BallRelativ* und der relativen Hindernisposition *Hindernis* als 2D-Posen im Roboterkoordinatensystem, kann die gesamte Berechnung mit der folgenden Formel berechnet werden:

$$\begin{aligned}\text{Hindernis}_{\text{Neu}} = \text{Pose}((\text{Ballradius}, \text{Hindernis}^{(y)} - \text{BallRelativ}^{(y)})^{T(\alpha)}, 0, 0)\cdot \\ \text{Pose}(0, \max(\text{abs}(\text{Hindernis}^{(y)} - \text{BallRelativ}^{(y)}), \text{Ballradius}), 0) + \text{BallRelativ}\end{aligned} \tag{7.16}$$

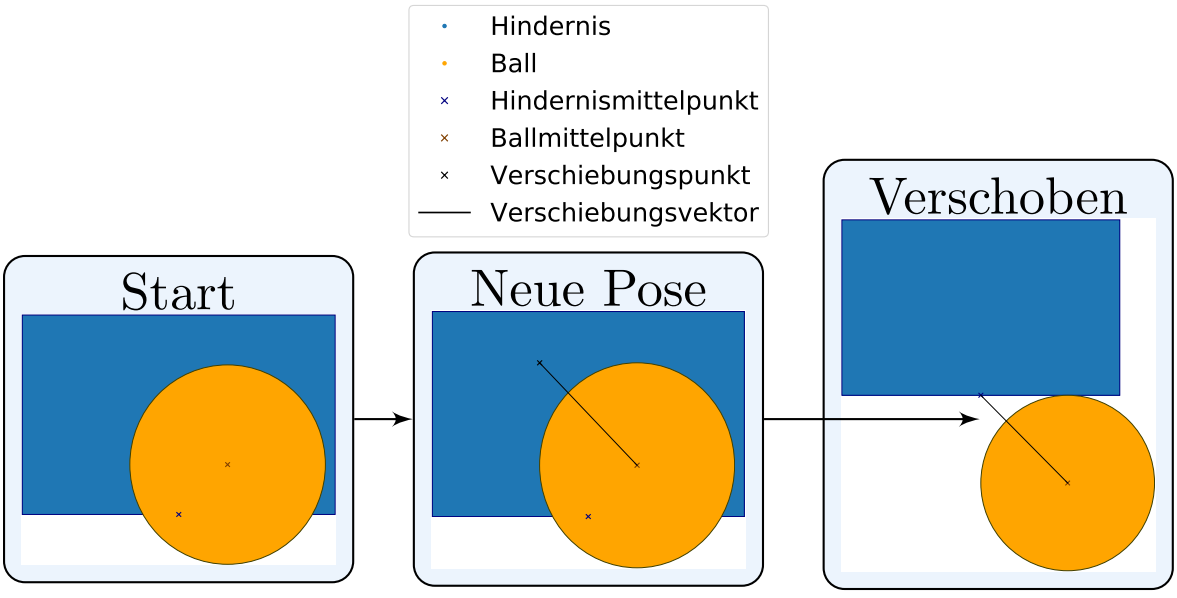

Abbildung 7.17 Die Verschiebung des Hindernisses hinter den Ball.

Für spätere Verhaltensentscheidungen wird zusätzlich noch ein Seitenbereich berechnet, welcher für das Erlangen des Ballbesitzes wichtig ist. Hierbei handelt es sich um Situationen, in denen der eigene Roboter seitlich zum gegnerischen steht und nicht zwischen Ball und Tor laufen kann, bevor der Gegner den Ball bewegt hat. Taktisch bleibt hier nur übrig, den Ball in Richtung der Außenlinie zu schießen, um anschließend den Ballbesitz zurückzugewinnen. Dafür wird mithilfe von Schwellwerten die Ausrichtung des Roboters geprüft, ob dieser seitlich genug rotiert ist, ob der Winkel zwischen Ball und dem eigenen Roboter in Feldkoordinaten groß genug ist und ob der Gegner, der am nächsten am Ball steht, nahe genug an diesem steht. Trifft alles ein, so wird ein statischer Winkelbereich von 88 bis 110 Grad respektive -110 bis -88 Grad in Weltkoordinaten erstellt. Dieser Winkelbereich wird des weiteren als der Seitenbereich bezeichnet.

Sektoren: Viele Schussrichtungen sind von der aktuellen Situation auf dem Feld abhängig. So ist es wichtig zu wissen, sowohl welche Richtung mit einem Hindernis kollidieren würde als auch in welcher Distanz sowie welche Richtungen in einen möglichen Torschuss resultieren könnten. Daher werden Sektoren berechnet. Hierfür besitzt das B-Human-Framework ein sogenanntes *SectorWheel*. Dieses beinhaltet eine Menge von Sektoren, welche jeweils vier Werte besitzen: einen minimalen und maximalen Winkel, eine Distanz und einen Typen. Das Winkelpaar gibt den Richtungsbereich an, der Typ die Art des Sektors und die Distanz die Entfernung, für die der Richtungsbereich gilt. Die hierbei wichtigen Typen sind freie Sektoren, Hindernisse und Tore. Die Distanz soll bei Hindernissektoren die Entfernung vom Ball zum Hindernis sein, für die anderen Typen wiederum unbegrenzt.

Für die Erstellung des SectorWheels wird für jedes Hindernis aus der eigenen Robotererkennung ein Winkelbereich bestimmt. Die Robotererkennung berechnet bereits eine Breite für jedes Hindernis, definiert als zwei Positionen links und rechts vom Hindernis. Diese beiden Positionen werden um das dreifache vom Ballradius zusätzlich verschoben, um zu gewährleisten, dass der Ball später knapp an Hindernissen vorbeigeschossen werden kann. Der Faktor drei wurde aus mehreren Gründen gewählt. Der Ball selber besitzt eine Breite, die Hindernisbreiten können fehlerhaft sein, die Hindernisse können sich bis zum Schuss noch bewegen und die Schüsse selber haben eine gewisse Schussungenauigkeit (siehe Abschnitt 6.4). Anschließend wird sowohl der Winkel zwischen Ball und der linken und rechten Hindernisposition als auch die Distanz zum Hindernis bestimmt. Dem SectorWheel wird danach noch der Torwinkelbereich übergeben. Das SectorWheel behandelt selbständig überlappende Winkelbereiche und besitzt anschließend eine Menge von Sektoren, welche Tor, Hindernis- oder freie Sektoren sind. Ein Beispiel für das SektorWheel ist in Abbildung 7.18 zu sehen.

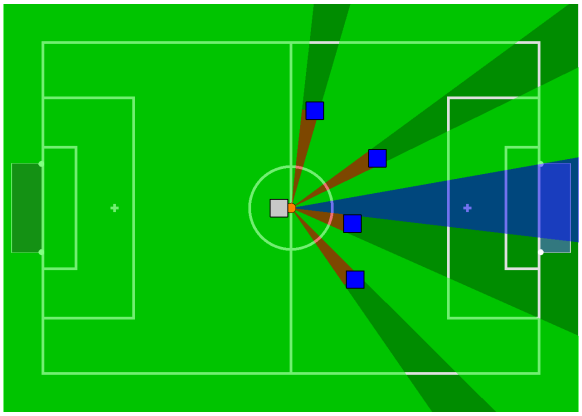

Abbildung 7.18 Die berechneten Sektoren des SectorWheels. Hellgrün sind alle freien Sektoren, blaue Sektoren sind Torschusswinkel, rote Sektoren beinhalten Hindernisse und habe eine begrenzte Reichweite.

Schussrichtungen: Um eine sinnvolle Auswahl an Schussrichtungen zu bekommen, dessen Menge klein ist aber gleichzeitig nach Möglichkeit immer die beste Richtung mit beinhaltet, werden mehrere Generierungen mit unterschiedlichen Schwerpunkten verwendet. Als erstes wird um die zuletzt geplante Schussrichtung herum eine Menge von Richtungen erstellt. Gab es im letzten Cognition-Frame keinen Schuss, so wird die Schussrichtung 0 verwendet, welche der aktuellen Ausrichtung des Roboters entspricht. Für die Menge der Schussrichtungen werden zwei Bereiche definiert. Im nahen Bereich werden mit hoher Auflösung Schussrichtungen erstellt, im entfernten Bereich solche mit geringer Auflösung. Dadurch soll die beste Schussrichtung um die vorherigen herum gefunden, gleichzeitig aber eine komplett andere Schussrichtung auch in Betracht gezogen werden können. Ein Beispiel hierfür ist in Abbildung 7.19 zu sehen. Als zweites sollen, falls der Seitenbereich berechnet wurde, von diesem die minimale und maximale Richtung zu der Menge der Schussrichtungen hinzugefügt werden, wie in Abbildung 7.20 dargestellt.

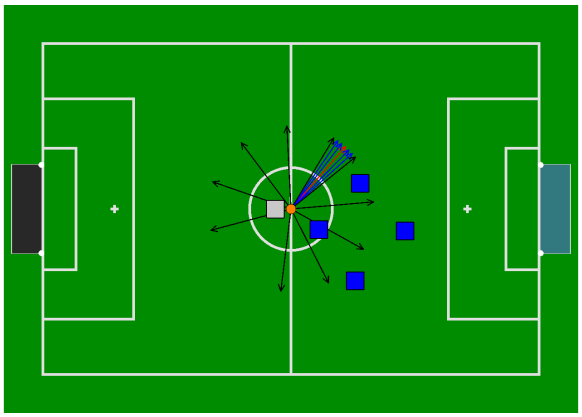

Abbildung 7.19 Die generierten Schussrichtungen um die vorherige geplante Schussrichtung (rot) herum. Blau ist der Nahbereich, schwarz sind komplett andere Schussrichtungen.

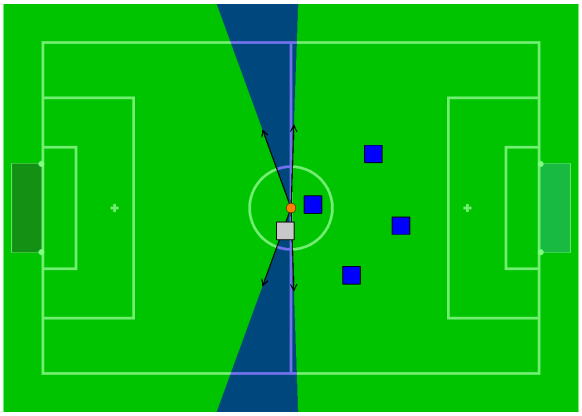

Abbildung 7.20 Die vier Schussrichtungen (Pfeile) für den Seitenbereich (blaue Sektoren).

Als nächstes werden die Schussrichtungen für den neuen Standard-Schuss hinzugefügt. Der neue Standard-Schuss wurde im Detail in Abschnitt 7.3.1 erklärt. Dessen Schussrichtung für die Planung ist im 90 Grad Winkel nach links oder rechts. Dieser Schuss soll aber nur ausgeführt werden, wenn die Schussrichtung den Roboter zwischen Ball und Tor positionieren kann. Für die zu prüfende Schussrichtung wird um die Richtung 90 und -90 Grad in Weltkoordinaten herum ein kleiner Winkelbereich erstellt und dessen Minimum und Maximum den Schussrichtungen hinzugefügt. Die Referenzrichtungen 90 und -90 Grad werden dabei in der Nähe des eigenen Tores verschoben, damit sie dem Winkel der Orthogonalen des Vektors zwischen Ball und dem eigenen Tor gleichen. Hierfür wird zuerst der Winkel zwischen Tormitte und Ball in Weltkoordinaten berechnet.

$$\text{Winkel}_{\text{Tor}} = (\text{Tor} - \text{Ball})^{(\alpha)}$$

Anschließend wird zwischen dem statischen 90 Grad Winkel und dem Winkel der Orthogonalen zum Tor interpoliert, basierend auf dem Unterschied der x-Translation von Ball und Torraum.

$$\text{Interpolationsfaktor} = \min\left(1, \frac{\max(0, \text{Ball}^{(x)} - \text{Torraum}^{(x)})}{\frac{\text{Grundlinie}^{(x)}}{2} - \text{Torraum}^{(x)}}\right)$$

$$\text{WinkelLinks}_{\text{DefaultSchuss}} = \text{Interpolationsfaktor} \cdot 90\text{Grad} + (1 - \text{Interpolationsfaktor}) \cdot (90\text{Grad} + \text{Winkel}_{\text{Tor}})$$

$$\text{WinkelRechts}_{\text{DefaultSchuss}} = \text{WinkelLinks}_{\text{DefaultSchuss}} - 180\text{Grad}$$

Diese Interpolation der Richtung ist in Abbildung 7.21 in den unteren beiden Bildern zu sehen, mit den normalen Fällen in den oberen. Der rote Kegel in den Abbildungen wird an einer späteren Stelle erläutert. Die rote Linie ist die Position der x-Translation vom Torraum, die blaue Linie die Position der x-Translation des Parameters $\frac{\text{Grundlinie}^{(x)}}{2}$.

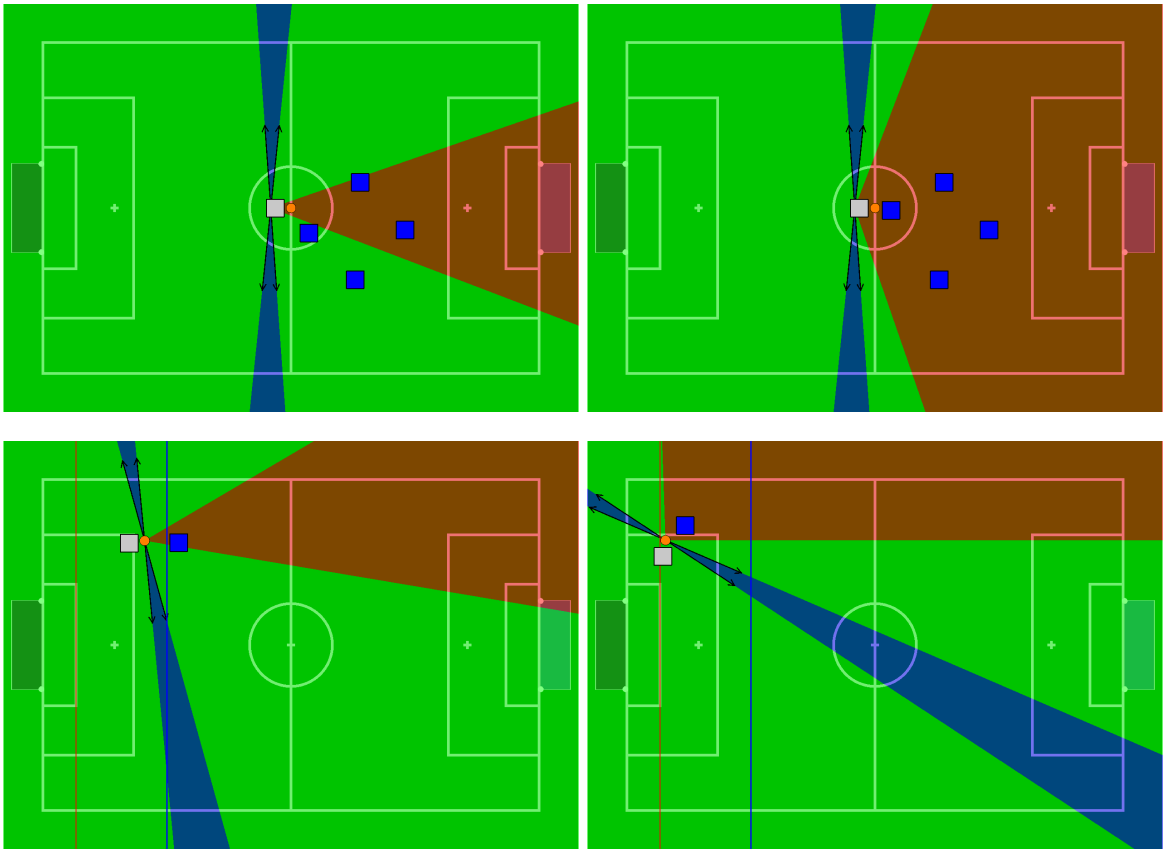

Abbildung 7.21 Die Schussrichtungen (Pfeile) für den Standard-Schuss in den verschiedenen Situationen. Der rote Kegel ist der Winkelbereich, in dem sich das nächste Hindernis befinden muss. Im oberen linken Bild befindet sich das Hindernis nicht im Kegel und ist relativ weiter weg, wodurch der rote Kegel klein ist. Oben rechts ist das Hindernis direkt am Ball, wodurch der rote Kegel sehr groß ist. Unten links ist der Ball etwas in der Nähe vom Tor, wodurch die Schussrichtungen leicht rotiert sind, unten rechts ist der Ball sehr nahe am Tor, wodurch die Schussrichtungen stark rotiert sind. In beiden unteren Bildern zeigt die blaue Linie die x-Position, ab wann die Schussrichtungen rotiert werden, und die rote Linie die x-Position, ab wann diese gleich der Orthogonalen des Torvektors sind.

Zu guter Letzt wird mithilfe von dem SectorWheel für jeden Torsektor die Mitte des Sektors als Schussrichtung hinzugefügt. Falls sich am Ende des Zweikampfes ein Torschuss als bester herausstellt, werden automatisch im nächsten Cognition-Frame um diesen herum mehrere Schussrichtungen erstellt, damit ein besserer Torschusswinkel innerhalb des Torsektors gefunden werden kann.

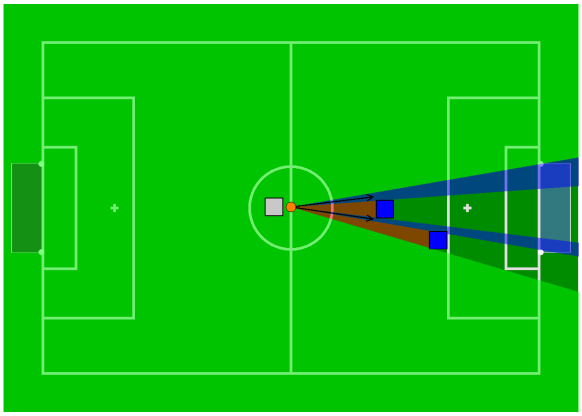

Abbildung 7.22 Die Schussrichtungen (Pfeile) in den Torsektoren (blau).

Prüfen der Schussrichtungen: Für die zu prüfende Schussrichtung werden zuerst mithilfe von dem Potentialfeld für eine Menge von Schussreichweiten mögliche Ballendpositionen bewertet. Da die Schüsse unterschiedliche Schussstärken haben können, wird die Menge an Reichweiten benötigt, die geprüft werden muss. Dafür wird für jeden Schuss die minimale und maximale Reichweite genommen und Reichweiten zwischen diesen mit einer festen Abtastrate berechnet. Ausnahme ist der Vorwärts- und Drehschuss. Da diese zusammen einen dynamischen Schuss darstellen, wird das Minimum der minimalen Reichweite und das Maximum der maximalen Reichweite beider Schüsse genommen, um zwischen diesem Bereich die Reichweiten zu generieren. Hat der Vorwärtsschuss also eine Reichweite von 0.5 bis 1 m und der Drehschuss 1 m bis 2 m, so werden zwischen 0.5 und 2 m mithilfe der Abtastung die Reichweiten erstellt. Als Beispiel für eine Abtastrate von 5 würde daher gelten:

$$\text{Abtastschritt} = \frac{\text{ReichweiteMax - ReichweiteMin}}{\text{Abtastrate}} \tag{7.17}$$

$$\Rightarrow$$

$$\text{Abtastschritt} = \frac{2\text{m} - 0.5\text{m}}{5} = 0.25\text{m} \tag{7.18}$$

Die Reichweiten, die somit hinzugefügt werden, sind (in m) 0.5, 0.8, 1.1, 1.4, 1.7, 2.

Anschließend werden von jedem Schuss noch die minimale und maximale Reichweite hinzugefügt, die Menge an Reichweiten aufsteigend sortiert und gefiltert, wodurch jede Reichweite nur einmalig vorkommt. Auf den Vorwärts- und Drehschuss bezogen werden daher noch die Reichweiten (in m) 0.5, 1. und 2. hinzugefügt, wobei durch die Filterung später nur noch einmalig 0.5 und 2. vorkommen wird.

Für das Potentialfeld kann nun mithilfe der Schussrichtung für jede Reichweite der Endpunkt auf dem Spielfeld berechnet werden. Falls die Schussrichtung in einem Torsektor liegt, wird die Reichweite zusätzlich auf die Entfernung zum Tor begrenzt. Dies ist nötig, damit für Torschüsse immer ein Feldpunkt geprüft wird, der im Tor liegt und nicht aufgrund seiner

Reichweite außerhalb des Tores landet. Anschließend werden zwei Kombinationen aus den Potentialfeldern erstellt. Zum einem wird die Kombination Feldrand, Tor, Torwinkel, Pässe und Zeitspiel berechnet. Dadurch sollen der Standard-Schuss sowie Schüsse, die den Ballbesitz erobern, sinnvoller bewertet werden können. Der Ball kann in Situationen erobert werden, in denen der eigene Roboter ihm seitlich hinterherläuft, während der Gegner den Ball geradeaus spielt. Denn das Ziel besteht darin, dem Gegner den Ball abzunehmen, statt den Ball in eine optimale Position zu bringen. Für die zweite Kombination werden zusätzlich die Potentialfelder für die Hindernisse und die Ballentfernung hinzugefügt. Diese Bewertung wird für jede andere Situation verwendet. In beiden Bewertungsfällen wird gespeichert, ob das Potentialfeld für die Pässe ein Ergebnis ungleich null hatte. Dadurch soll das Zweikampfverhalten später wissen können, ob ein Schuss mit seiner Reichweite ein Pass ist.

Bewertungsfunktion: Für die aktuelle Schussrichtung existiert nun eine Liste an Reichweiten, die jeweils zwei Bewertungen zugewiesen haben. Für jeden Schusstypen wird nun eine weitere Bewertung durchgeführt, um situationsbedingt ungünstige Schüsse zu vermeiden sowie einen mehrmaligen Wechsel, welcher Schuss ausgeführt werden soll im Vergleich zum vorherigen Cognition-Frame.

Dafür werden zuerst einige einfache Bedingungen geprüft, um die meisten Berechnungen von Schüssen zu überspringen. Für Seitwärtsschüsse konnte in den vergangenen Wettbewerbsspielen immer wieder beobachtet werden, dass diese in Richtung der gegnerischen Grundlinie ausgeführt wurden, da sie scheinbar vom alten Zweikampf als schneller ausführbar bewertet wurden. In diesen Situationen stand aber fast immer ein Gegner sehr nahe am Ball und meistens so im Weg, dass der Ball nur an die Beine des Gegners gedrückt wurde. Auch muss der eigene Roboter in einem solchen Moment über seine Schulter gucken, wodurch er weder den Ball sehen noch Gegner konstant erkennen kann. Dadurch existiert immer das Risiko, dass sich bis zum Ausführen des Schusses die Situation auf dem Spielfeld signifikant verändert hat. Daher soll der Seitwärtsschuss nur noch ausführbar sein, wenn der Roboter aktuell nicht zwischen Ball und Tor steht. Wenn er zwischen beiden steht, muss seine Schussrichtung im Winkelbereich der beiden Torpfosten zum Ball liegen. Der erlaubte Winkelbereich ist in Abbildung 7.23 zu sehen.

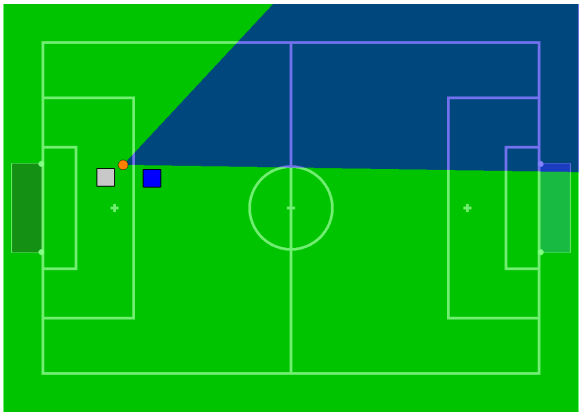

Abbildung 7.23 Der Sektor (blau), in dem die Schussrichtung für den Seitwärtsschuss liegen muss.

Neben dem Seitwärtsschuss wird auch die Ausführbarkeit des Standard-Schusses geprüft. So darf der Seitenbereich nicht berechnet worden sein. In einem solchen Fall impliziert der Seitenbereich nämlich, dass der eigene Roboter seitlich am Ball steht und somit eine zu lange Zeit benötigt, um sich zwischen Ball und Tor zu positionieren. Auch muss der am nächsten stehende gegnerische Roboter nahe genug am Ball sein. Denn der Standard-Schuss zieht den Ball nur seitlich mit sich. Steht der Gegner noch zu weit weg, kann dieser damit nicht umspielt werden. Zuletzt sind nur Schüsse in dem Winkelbereich erlaubt, welche für die Schussrichtungen explizit für den Standard-Schuss hinzugefügt wurden (siehe Abbildung 7.21, blauer Kegel). Auch gilt für den Gegner, dass dieser in einem gewissen Winkelbereich hinter dem Ball steht (roter Kegel). Dieser Winkelbereich ist eine Interpolation aus jeweils 2 Winkelbereichen. Steht der Gegner deutlich weiter weg vom Ball als der eigene Roboter, so wird ein kleinerer Winkelbereich gewählt (oben links). Ist er mindestens gleich nahe, so wird der größere Winkelbereich verwendet. Zusätzlich wird der Bereich mit der Schussrichtung (blauer Kegel) in der Nähe des eigenen Tors rotiert. Als Stabilitätskriterium, da die Hinderniserkennung sehr fehlerbehaftet ist, werden der kleine und der große Winkelbereich durch jeweils zwei deutlich größere Bereiche ausgetauscht, falls der letzte geplante Schusstyp der Standard-Schuss war. Ohne dieses Stabilitätskriterium könnte ansonsten, aufgrund der schwankenden Robotererkennung, der Standard-Schuss von Cognition-Frame zu Cognition-Frame abwechselnd ausführbar oder nicht ausführbar sein.

Für jeden Schusstypen wird auch geprüft, um wie viel Grad sich der Roboter für den Schuss drehen müsste, um diesen auszuführen. Ähnlich wie beim alten Zweikampf darf sich der Roboter, abhängig von seiner Entfernung zum Ball, nur um eine maximale Rotation drehen. Wird diese überschritten, so wird der Schuss mit der entsprechenden Schussrichtung übersprungen. Es gelten aber zwei Ausnahmen: Der Standard-Schuss darf sich immer beliebig drehen, da dieser bereits nur unter bestimmten Gegebenheiten ausgeführt wird. Zum anderen dürfen sich auch Schüsse, die in die Richtung des Seitenbereiches gehen (siehe Abbildung 7.20), beliebig viel drehen. Hier wird nämlich angenommen, dass ein Schuss von der Seite trotz hoher benötigter Rotation schneller ausführbar ist als wenn um den Ball herumgelaufen werden würde.

Ebenfalls sollte sich der Roboter in einem gewissen Rahmen drehen, damit er beim Schuss später nicht in den Gegner läuft.

Anschließend wird eine weitere Menge von Bedingungen geprüft, die vom Schusstyp und der Schussreichweite abhängig sind. Auch folgen weitere Bewertungen, um Schüsse zu priorisieren.

Zuerst werden die Schusspose und die Schussreichweiten bestimmt. Wie schon in Abschnitt 6.1 erwähnt, existiert für jeden Schuss eine Konfiguration, in der sowohl eine relative Position zum Ball definiert ist als auch eine Angabe über die minimale und maximale Schussreichweite. Für jeden Schuss können die Schussreichweiten übernommen werden, mit Ausnahme von dem Vorwärts- und Drehschuss. Hier soll, basierend auf der Schussrichtung, die relative Ballposition und die Reichweiten interpoliert werden. Für die Interpolation wird der Interpolationsfaktor *IF* wie folgt berechnet:

$$\mathrm{IF} = \mathrm{range}\left(0, \frac{\text{Schussrichtung}}{\text{Schussrotation}}, 1\right)$$

Die Schussrotation entspricht hierbei der konfigurierten Schussrichtung des Drehschusses. Wird der Drehschuss nach links geprüft, entspricht dies 45 Grad respektive für rechts -45 Grad. Mit dem Interpolationsfaktor werden dann die Reichweiten und die relative Ballposition wie folgt berechnet:

$$\mathrm{Parameter} = \mathrm{IF} \cdot \mathrm{Parameter}_{\mathrm{Drehschuss}} + (1 - \mathrm{IF}) \cdot \mathrm{Parameter}_{\mathrm{Vorwärtsschuss}}$$

Für die Schusspose wird vom Ball aus die relative Ballposition addiert, ähnlich wie es in Abschnitt 6.3.1 für die Laufschrittgrößenberechnung umgesetzt wurde (siehe Abbildung 6.3, Relative Position).

Nun kann auf die Liste zurückgegriffen werden, die zu jeder Schussreichweite jeweils zwei Bewertungen vorliegen hat. Es wird jeder Eintrag geprüft, um den mit der besten Bewertung zu finden, wobei die Reichweite innerhalb der Schussreichweite des Schusses liegen und in einer Ballendpositionen innerhalb des Feldes resultieren muss. Falls die Schussrichtung im Seitenbereich liegt, wird die erste Kombination der Potentialfelder verwendet, bei welcher Hindernisse und die Ballentfernung ignoriert wird. Ansonsten wird die vollständige Kombination benutzt. Findet beim Listendurchgang eine neue bessere Bewertung statt, so wird die vorherige zwar überschrieben, aber die neue Bewertung wird um den Wert 0.01 verschlechtert. Dadurch sollen stärkere Schüsse priorisiert werden, sofern diese nur als minimal schlechter bewertet wurden. Auch wird sich für spätere Zwecke gemerkt, ob der Schuss ein Pass ist. Falls der Schuss keine Bewertung erhält, weil die minimale Schussreichweite nicht ausführbar ist, da der Ball außerhalb des Feldes gelangen würde, wird dieser übersprungen.

Die nun vorhandene Bewertung wird nachträglich noch erweitert. Torschüsse, die das Tor erreichen, bekommen einen Bonus. Liegt der Ball im eigenen Elfmeterbereich, der Schuss würde den Ball aber aus diesem bekommen, so erhält dieser ebenfalls einen Bonus. Schüsse, die den Ball in die eigene Spielfeldhälfte schießen, bekommen einen Malus. Auch für die benötigten Rotationen, die der Roboter für die zu erreichende Schusspose umsetzen muss, bekommt der Schuss einen Malus. Ist der aktuell geprüfte Schusstyp derselbe wie der zuletzt geplante und kein Torschuss, so bekommt dieser einen festen Bonus sowie einen skalierenden Bonus, je geringer die benötigte Rotation für die Schusspose ist. Bei Torschüssen wird somit kein Bonus vergeben, wenn der Schusstyp gleich bleibt, damit der am schnellsten auszuführende Schuss umgesetzt wird, um die Torchance nicht zu verlieren.

Zu guter Letzt wird für jeden Schuss auf die Kollision mit Hindernissen geprüft, um ein mögliches Umfallen zu minimieren. Eine Kollision wird nur akzeptiert, wenn der Schussfuß weiter weg von einem Hindernis ist als der Standfuß. Hierfür werden zwei verschiedene Ansätze verwendet. Falls der zu prüfende Schuss ein normaler oder der lange Vorwärtsschuss mit keinem zu großem Schusswinkel ist, so wird an der Schusspose geprüft, ob der Fuß, der näher am Hindernis steht, zu nahe an diesem ist. Hierfür werden zwei Positionen definiert. Zum einem wird eine äußere obere Ecke von dem Fuß als Kontrollpunkt genommen. Ist diese zu nahe, dann würde der Roboter schon vor dem Schuss mit dem Hindernis kollidieren. Zum anderen wird ein ähnlicher Kontrollpunkt verwendet, der weiter nach vorne geschoben ist. Dieser stellt die Eckposition des Fußes nach einem Schuss dar. Ist diese zu nahe, würde der Roboter nach dem Schuss mit dem Hindernis kollidieren. Für beide Eckpositionen wird die Distanz zum Hindernis berechnet und mit einem Schwellwert verglichen. Die Prüfung wird für alle bekannten Hindernisse aus der Robotererkennung durchgeführt. Ist nur eine Eckposition zu nahe an einem Hindernis, so wird die Bewertung des Schusses um einen kleinen Wert verschlechtert.

Wird ein anderer Schusstyp oder ein Schuss mit einem großen Schusswinkel ausgeführt, so wird der zweite Ansatz geprüft. Nur für das aktuelle Hindernis, welches am nächsten steht, wird für eine Kollision geprüft. Hierbei wird um das Hindernis herum ein Kreis definiert, dessen Radius dieselbe Distanz besitzt wie im ersten Ansatz. Statt aber auf die Distanz zu prüfen, wird von der aktuellen Position des Roboters zur Schusspose ein Vektor gebildet und auf einen Schnittpunkt mit dem Kreis des Hindernisses geprüft. Ist ein Schnittpunkt vorhanden, kommt es voraussichtlich zu einer Kollision. Die Bewertung des Schusses wird um einen großen Wert verschlechtert.

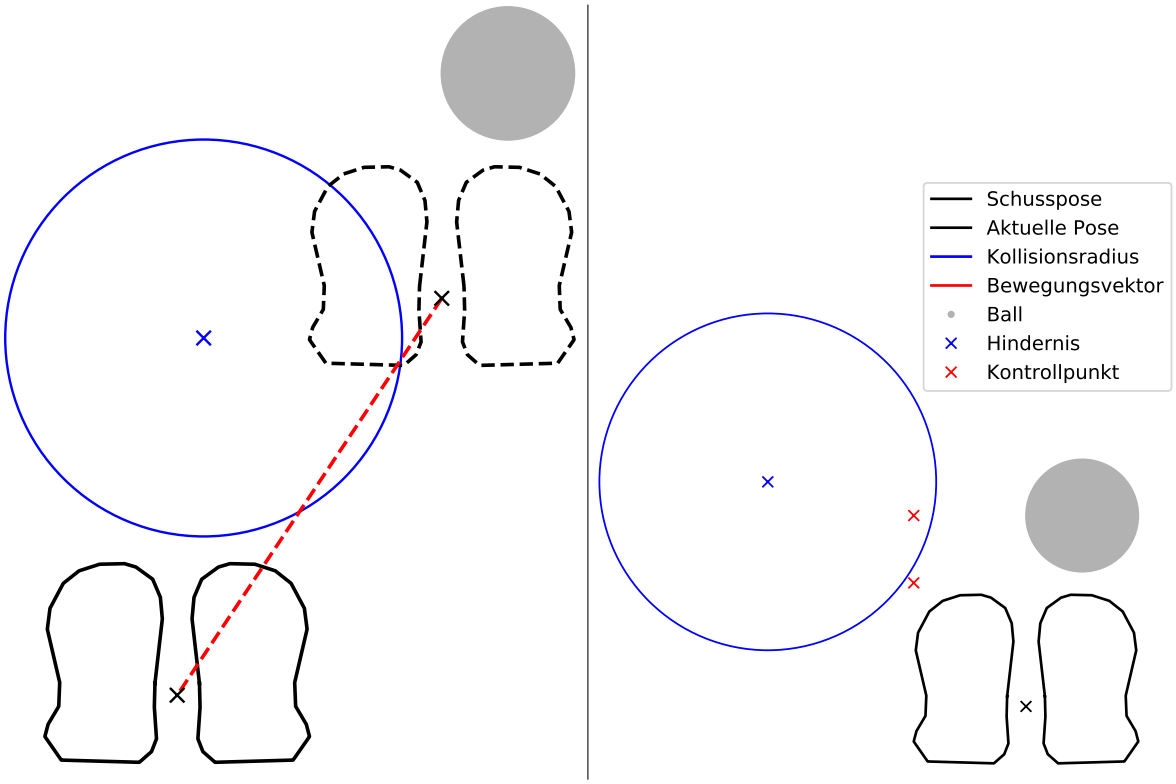

Abbildung 7.24 Die beiden Verfahren für die Kollisionsprüfung mit den Hindernissen.

Schuss bestimmen: Falls mindestens ein Schuss existiert, der eine Bewertung bekommen hat, wird der bestbewertete aus der Liste aller geprüften Schüsse genommen. Hierfür wird einmalig die gesamte Liste abgearbeitet, der momentan beste Schuss mit dem aktuell betrachteten Schuss in der Liste verglichen und der bessere gespeichert. Es werden aber Standard-Schüsse priorisiert, auch wenn deren Bewertung deutlich schlechter sind. Wenn mehrere Standard-Schüsse geprüft wurden, wird der mit der besten Bewertung verwendet. Falls der auszuführende Schuss von Typ Vorwärtsschuss ist und einen Torschuss mit geringen Schusswinkel darstellt, wird dieser anschließend durch den langen Vorwärtsschuss ausgetauscht. Denn aufgrund der Reichweitenstreuung wird das Tor nicht immer erreicht, wenn der Vorwärtsschuss nahe seiner maximalen Schussreichweite ausgeführt wird. Falls der Schusswinkel in den Seitenbereich fällt und vom Schusstyp ein Vorwärtsschuss und kein Pass ist, wird zusätzlich für den späteren Skill-Aufruf gesetzt, dass für diesen kein Vorschritt erlaubt ist. Der erste Laufschritt des Schusses muss also bereits den Ball berühren. Dadurch werden unter anderem Drehschüsse verhindert, da ein solcher den Roboter meist für die Schusspose in den Gegner platziert.

In speziellen Situationen wird aber auch kein einziger Schuss geprüft, wodurch keiner ausgeführt werden kann. Dies geschieht häufig an der gegnerischen Grundlinie, da hier häufig jeder Schuss den Ball außerhalb des Feldes schießen würde. Falls für einen kurzen Zeitraum, wie 250 ms, kein Schuss ausgeführt werden konnte, wird eine Pose 180 mm hinter dem Ball,

zwischen Ball und Tor, berechnet, zu welcher der Roboter laufen soll. Kommt es für längere Zeit, wie 500 ms, zu keinem ausführbaren Schuss, so wird das Zweikampfverhalten verlassen, die Abbruchbedingung der Card wird also erfüllt und kann erst zwei Sekunden später wieder aktiv werden. Durch den aktuellen Aufbau des Verhalten-Frameworks mit den Cards aktiviert sich in den meisten Fällen in der Zeit das Verhalten für das Dribbeln des Balles in Richtung Tor.

Genauigkeitsbereich: Sofern ein Schuss berechnet wurde, soll anschließend noch die Ausführung des Schusses vereinfacht werden. Statt nur einer Schussrichtung kann nämlich auch ein Schussrichtungsbereich übergeben werden, welcher als eine minimale und maximale Richtung definiert ist. Dafür wird zuerst der zweitbeste Schuss bestimmt, wofür erneut die Liste alle geprüften Schüsse verglichen wird. In diesem Fall gelten aber andere Bedingungen. Der aktuell betrachtete Schuss in der Liste wird nur als zweitbester Schuss gespeichert, wenn dessen Bewertung schlechter als die des besten Schusses, jedoch besser als die des bisher zweitbesten Schusses ist. Auch muss der Schusstyp des zweitbesten Schusses ein anderer sein als die des besten Schusses oder die Schussrichtung muss um einen größeren Wert abweichen. Anschließend wird die Bewertung des zweitbesten Schusses nochmals um einen kleinen Wert verschlechtert. Dadurch wird ein kleiner Bewertungsunterschied beider Schüsse sichergestellt.

Für den Genauigkeitsbereich werden nun erneut das Potentialfeld und die Bewertungsfunktion für den Schusstypen des besten Schusses mit dessen Schussreichweite aufgerufen, aber mit Abweichungen in der Schussrichtung. So soll um die geplante Richtung herum in kleinen Grad-Abständen der Schuss bewertet und so ein Genauigkeitsbereich gefunden werden, in dem die Bewertung des Schusses im Vergleich zum zweitbesten Schuss weiterhin besser ist. Falls hier eine Schussrichtung eine bessere Bewertung erhält als der momentan beste Schuss hat, wird diese als neue angesteuerte Schussrichtung gesetzt.

Gegner anlocken: Um die Erfolgsquote des Zweikampfes zu erhöhen, insbesondere für den Standard-Schuss, bei dem ein Gegner sehr nahe am Ball sein muss, damit dieser gut funktioniert, soll der Roboter mit der Ausführung des Schusses unter Umständen kurzzeitig warten. Dieses aktive Warten gilt aber nur für den Vorwärts- und Standard-Schuss und auch nur, wenn der Schuss kein Pass oder Torschuss ist. Auch muss sich das Hindernis, welches am nächsten zum Ball steht, noch einen gewissen Abstand weit weg befinden. Ebenfalls wird nur für maximal zwei Sekunden gewartet und anschließend kann erst nach einigen Sekunden eine neue Wartezeit starten. Speziell für den Vorwärtsschuss wird zusätzlich geprüft, ob der Abstand von der geplanten Ballendposition zum Hindernis einen Schwellwert überschreitet. Dies ist für die Drehschüsse wichtig, da diese unter anderem eng am Gegner vorbeigeschossen werden könnten.

7.4 Evaluation

Der neue Zweikampf wird in zwei Stufen evaluiert. In der Simulation werden 5 gegen 5 Spiele durchgeführt. Ein Team verwendet dabei das neue Zweikampfverhalten und das andere Team das bestehende. Um den Einfluss von Zufällen möglichst gering zu halten, werden vier Konfigurationen durchlaufen. Die Simulation bietet an, Ground Truth (GT) Daten zu verwenden. Den Robotern können so die echten Positionen aller Roboter und die des Balles übergeben werden, ohne die Notwendigkeit der Bildverarbeitung. Ohne die Bildverarbeitung kann die Simulation ungefähr doppelt so schnell durchlaufen werden. Daher werden zwei Spiele mit GT für vier Stunden und zwei Spiele ohne GT für zwei Stunden laufen gelassen. Eine Stunde entspricht bis zu drei ganzen Spielen, wobei hier die Zeit nie angehalten wird. Insgesamt werden so also bis zu 36 Spiele evaluiert, wobei es innerhalb der vier Durchgänge keine Pause gibt.

Zusätzlich wird jeweils das Verhalten für das Dribbeln und an den Ball Laufen (kurz D&L) mit einem alternativen Verhalten ausgetauscht. Das Dribbeln bietet die besten Möglichkeiten, Zweikämpfe schon im Voraus zu verhindern, indem Gegner vermieden werden. Daher soll mithilfe des entwickelten Potentialfeldes der Gradientenabstieg auf diesem angewendet werden, um ein hindernisvermeidendes Dribbeln zu erhalten (siehe Abschnitt 7.4.1). Das an den Ball Laufen wird hingegen insofern verändert, dass sich die Roboter nicht permanent geradeaus ausrichten, sondern auch seitlich an den Ball laufen dürfen. Dadurch drehen sich die Roboter deutlich weniger und sollen insbesondere in Zweikämpfen schneller dem Ball folgen können. Die Anpassungen sollen hierbei nur eine Veränderung hervorrufen und keine allgemeine Verbesserung darstellen.

Es existieren somit vier verschiedene Konfigurationen: GT mit dem normalen restlichen Verhalten (GT Default) für vier Stunden, GT mit dem alternativen Dribbeln und Laufen (GT D&L) für vier Stunden, ohne GT mit dem normalen Verhalten (N Default) für zwei Stunden und ohne GT mit der Alternative (GT D&L) für zwei Stunden. Um den Zweikampf zu bewerten, werden mehrere Metriken aus dem menschlichen Fußball verwendet. So werden die durchschnittliche Balldistanz, die Tore, Torschüsse und Standardsituationen pro Halbzeit betrachtet. Üblich wäre auch der Ballbesitz, welcher aber hier durch die Balldistanz ersetzt wird.

Als zweite Stufe wird auf den echten NAOs verglichen. Aufgrund der aktuellen COVID-19 Pandemie sind richtige Spiele nicht möglich. Daher wird auf vordefinierte eins gegen eins Situationen zurückgegriffen. Hier soll primär sowohl die Effektivität des neuen Standard-Schusses evaluiert werden als auch die Maßnahmen, Kollisionen mit anderen Robotern zu vermeiden. Hierfür sollen an den Außenlinien und im Mittelkreis Zweikämpfe getestet werden. Für den Standard-Schuss wird zum einem der Ball vor einem Dummy-Roboter mittig gelegt und der andere Roboter führt jeweils das alte und neue Zweikampfverhalten aus. Zum anderen wird der Ball zwischen zwei spielenden Robotern platziert, wobei hier alt gegen neu spielt. Dies wird 5-mal im Mittelkreis und 10-mal an der Außenlinie durchgeführt. Die Roboter

werden nach der Hälfte getauscht, damit diese selber möglichst wenig Einfluss nehmen.

Für die Vermeidung von Kollisionen werden sowohl die Situationen aus Abbildung 7.5 b) und c) mit Dummy-Robotern 5-mal durchgeführt als auch die Situation, wo der Ball seitlich neben dem Dummy-Roboter liegt und der Ball spielende Roboter geradeaus spielt.

7.4.1 Alternatives Dribbeln

Für das alternative Dribbeln wird, wie bereits erwähnt, das entwickelte Potentialfeld verwendet, exklusive den Feldern für die Pässe und für den Ballbesitz, aber inklusive dem Potentialfeld für die Roboterausrichtung. Für diese Feldkombination wird der Gradientenabstieg ausgehend von der Ballposition bestimmt. Mit der Feldrichtung wird anschließend die aktuelle Feldposition für 1 cm vorgerechnet. Dies wird für eine Strecke von 1 m wiederholt. Mit dieser Endposition wird von der aktuellen Ballposition der benötige Schusswinkel bestimmt. Um nicht in Hindernisse zu schießen, wird zusätzlich erneut das SectorWheel verwendet. Befindet sich die Schussrichtung in einem zu nahen Hindernissektor, so wird der nächstbeste Sektor verwendet, der die geringste Schussrichtungsanpassung benötigt, den Ball aber nicht außerhalb des Feldes schießt.

7.4.2 Simulationsdurchläufe

Die Ergebnisse der Simulation sind in Tabelle 7.1 zu sehen. Der neue Zweikampf scheint im Vergleich zum alten fast immer besser zu sein. Es werden sowohl mehr Tore als auch mehr Torschüsse erzielt. Ebenso erhält der neue Zweikampf mehr Freistöße, jedoch gibt es Probleme bei Torfreistößen. Torfreistöße werden dann vergeben, wenn das gegnerische Team über die Grundlinie am Tor vorbei außerhalb des Feldes schießt. Zwar kann es sein, dass durch die höhere Torschussrate auch mehr Schüsse am Tor vorbei gingen. Ebenso könnte aber der alte Zweikampf häufig außerhalb des Feldes geschossen und dabei einen Gegner getroffen haben, wodurch ein Freistoß zu eigenen Gunsten entstand. Der neue Zweikampf versucht hingegen, den Ballbesitz zu bewahren, zu sehen in Abbildung 7.25. Mit wenigen Ausnahmen ist der neue Zweikampf darin auch erfolgreich, denn erst in der Nähe des gegnerischen Tores geht der Ballbesitz verloren. Bezogen auf den Durchschnitt über alle Konfiguration liegt der Ball im Schnitt 60 mm näher an einem verbündeten Roboter als an einem Gegner. Die allgemein hohen Balldistanzen von 600 bis 800 mm kommen dadurch zustande, dass der Ball immer geschossen wird, wobei die geringste Entfernung bei ungefähr 500 mm und die längeren im Zweikampf bei bis zu 3000 mm liegen. Da der neue Zweikampf kürzere Schüsse präferiert, liegt der Ball auch als Resultat näher an einem eigenen Roboter.

Auch befindet sich der Ball, mit einer leichten Tendenz, häufiger in der gegnerischen Hälfte, wobei dieser weiterhin in der Mitte des Feldes zentriert liegt, wie es in Abbildung 7.26 zu sehen ist. Dies scheint ein gutes Indiz dafür zu sein, dass tatsächlich mehr Zweikämpfe gewonnen werden. Denn wie auch in den RoboCup Spielen von 2019 hatte das Team mit dem neuen

Zweikampf mehr Tore geschossen, wodurch der Gegner den Ball häufiger beim Anstoß in die eigene Hälfte bringen konnte. Trotz dieses Nachteils wird der Ball aber konstant genug in die gegnerische Hälfte gebracht.

Auch ist kein offensichtlicher Unterschied zwischen GT und der normalen Perzeption zu erkennen. Beispielsweise zeigen die Konfigurationen GT Default und N Default große Unterschiede in den Toren und Kick-Ins pro 10 min, die Konfigurationen GT D&L und N D&L hingegen nur noch minimale Unterschiede. Dazu ist die Simulation weit von der Realität entfernt. Neben der Perzeption verhalten sich die Gelenke der simulierten NAOs nicht wie die der echten. Sie laufen signifikant stabiler und fallen nur um, wenn zwei NAOs direkt ineinander laufen, und selbst dann nur in den geringsten Fällen.

Konfiguration		Balldistanz in mm	Tore pro 10min	Torschüsse pro 10min	Eckstöße pro 10min	Kick-Ins pro 10min	Torfreistöße pro 10min
GT	Default	**627.47** : 689.54	**0.46** : 0.33	**2.58** : 2.54	**0.92** : 0.75	2.54 : **2.67**	**0.29** : 0.27
	D&L	**641.47** : 681.79	**0.67** : 0.5	**4** : 1.96	**1** : 0.46	**2.33** : 1.5	**0.29** : 0.21
N	Default	**696.11** : 761.03	**0.75** : 0.17	**3.00** : 2.25	0.42 : **0.58**	**3.08** : 1.67	**0.58** : 0.42
	D&L	**634.18** : 721.03	0.5 : 0.5	**3** :2.17	**0.75** : 0.5	**1.92** : 1.58	0.25 : **0.42**
Ø	Durchschnitt	**649.81** : 713.35	**0.59** : 0.38	**3.15** : 2.23	**0.77** : 0.57	**2.47** : 1.85	0.32 : **0.35**

Tabelle 7.1 Die Metriken der Simulationsdurchläufe für den Zweikampf. Verglichen ist jeweils Neu gegen Alt. Die jeweils besseren Werte sind dick hervorgehoben. Der neue Zweikampf ist in fast jeder Metrik besser. GT steht für Ground Truth, N für auf der Perzeption basierend.

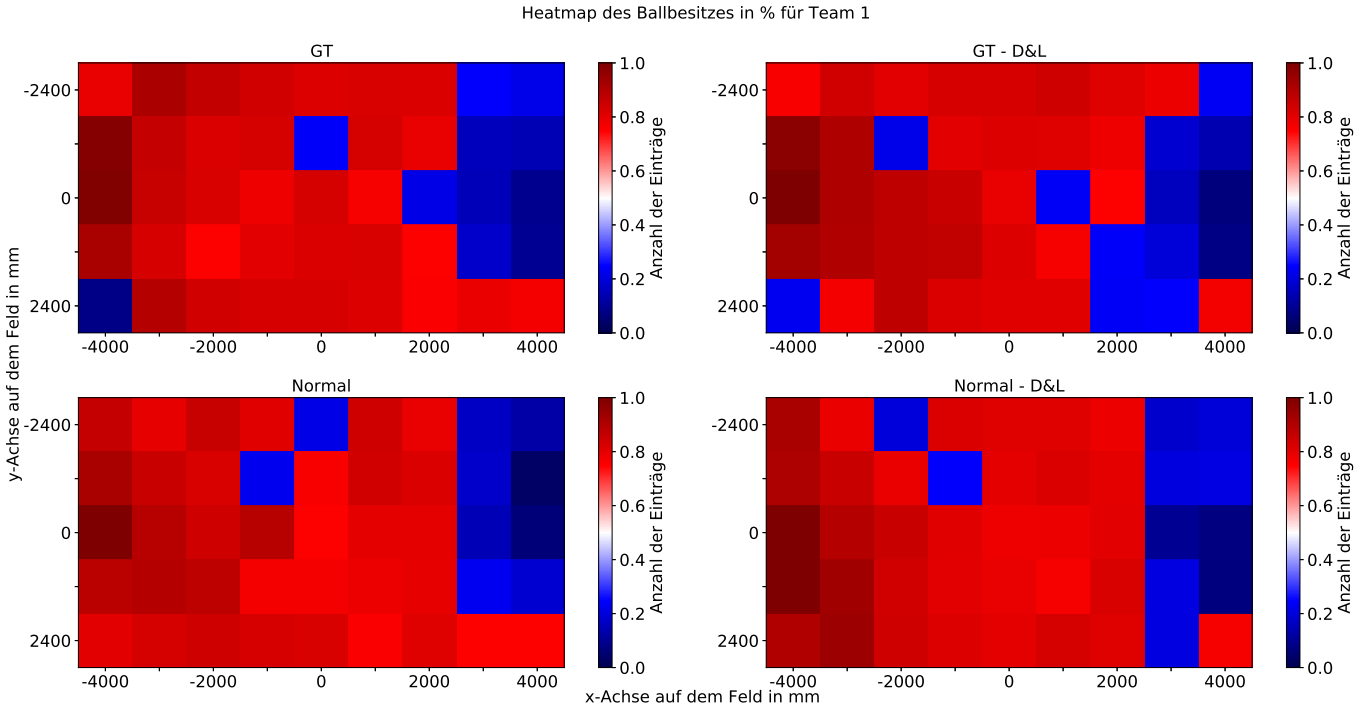

Abbildung 7.25 Die Heatmap des Ballbesitzes der vier Simulationsdurchläufe. Rot stellt Feldbereiche dar, in denen der neue Zweikampf häufiger den Ballbesitz hat, blau hingegen der alte Zweikampf.

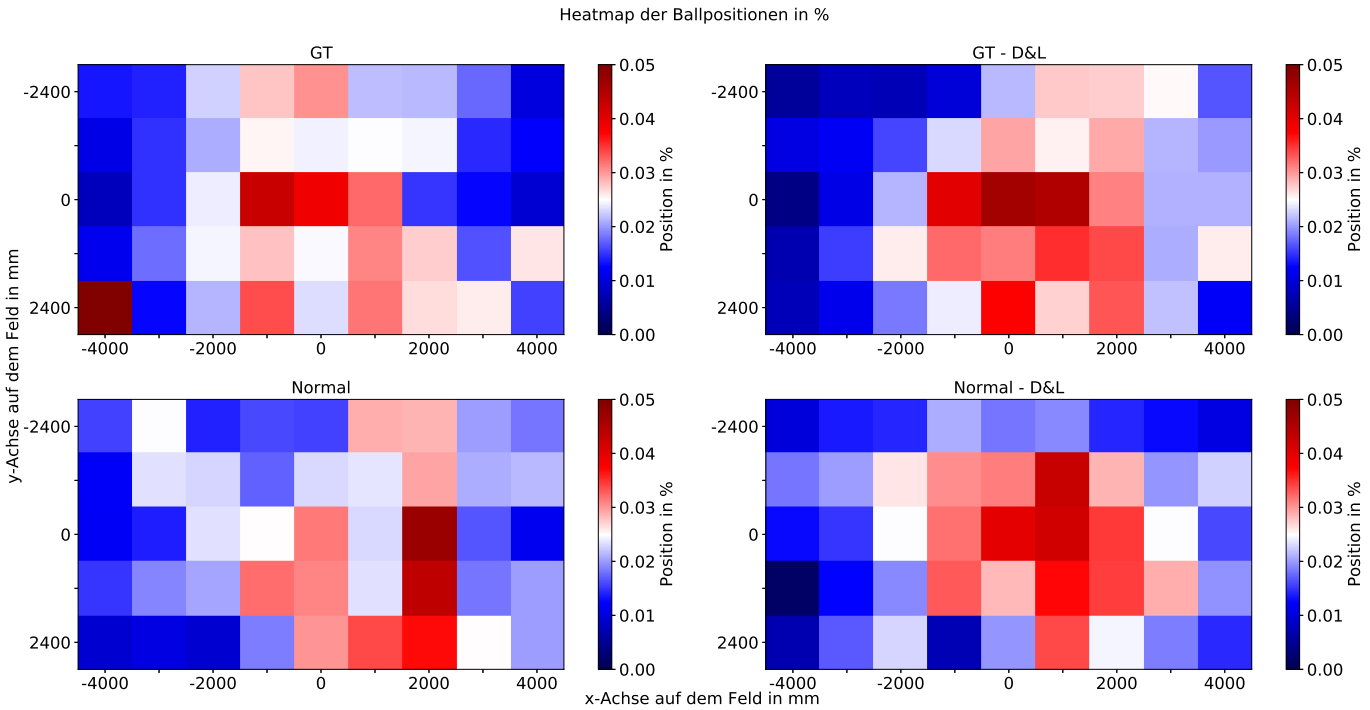

Abbildung 7.26 Die Heatmap der Ballpositionen der vier Simulationsdurchläufe. Rot stellt die häufigsten Ballpositionen dar, blau die seltensten.

7.4.3 Durchläufe auf den echten NAOs

Die Ergebnisse der Evaluation auf den echten Robotern sind in Tabelle 7.2 zu sehen. Gegen den Dummy (linke Tabelle) hat der neue Zweikampf häufiger gewonnen, 8 von 14 (57.14 %), während der alte nur 5 von 14 (35.7 %) gewann. Dabei rollte der Ball beim alten Zweikampf mit 6 statt 3 Malen doppelt so häufig außerhalb des Feldes. Auch blieb der Ball beim neuen Zweikampf anschließend im Schnitt näher am Roboter liegen.

Im direkten Vergleich (rechte Tabelle) gewann der neue Zweikampf in 11 von 12 Fällen. In einem Fall hat keiner der beiden Roboter gewonnen, da beide den Ball mehrmals nicht erkannt haben und vom Ball wegliefen, aber 2-mal den Ball zwischenzeitlich berührt haben. Interessanterweise fiel der Roboter mit dem neuen Zweikampf 3-mal um, während der alte nur 2-mal fiel. Der alte Zweikampf lief hierbei einmal aggressiv in den anderen Roboter hinein, um den Ball vorwärts zu spielen. In allen anderen Fällen scheint das an den Ball Laufen die Ursache zu sein. Die Roboter drehen sich etwas auf der Stelle, um anschließend dem Ball zu folgen. Dadurch wird der Gegner zum einem nicht mehr gesehen, zum anderen wird geradeaus an den Ball gelaufen, ohne dem Gegner auszuweichen.

Im Durchlauf für das Umfallen schnitt der neue Zweikampf aber schlechter ab. Hier fielen die Roboter in 7 von 16 Fällen. Mit dem bisherigen Zweikampf fielen sie nur in 5 von 16 Fällen. Der Standard-Schuss scheint den Roboter zu nahe an den Gegner zu bringen, wenn dieser leicht seitlich zum Ball steht. Dadurch tritt dieser häufig auf die Füße des anderen. Dies ließe sich nur verhindern, wenn der Robotererkennung mehr vertraut werden könnte, was

bei Erkennungen wie denen in Abbildung 7.27 schwer umsetzbar ist. Es könnte aber auch getestet werden, ob die Ausführung des Standard-Schusses reduziert werden kann, indem das Hindernis deutlich mittiger hinter dem Ball stehen muss. Ebenso scheint die Kollisionsprüfung für den Schnittpunkt des Laufweges mit dem Hindernis (siehe Abbildung 7.24, links) nicht auszureichen. Die linke Prüfung erkennt keine Kollisionen, wenn der eigene Roboter schon nahe am Ball steht. Die rechte Prüfung zwar hingegen schon, jedoch wird der Schuss nicht verschlechtert und die Ausführung nicht verhindert. Grund hierfür: Nur wenn der Schussfuß weiter weg vom Hindernis ist als der Standfuß, wird die Bewertung des Schusses verschlechtert. So soll immer der Schussfuß gewählt werden, der den Roboter am weitesten vom Hindernis entfernt positioniert. Dadurch kann aber nicht beurteilt werden, welche Schussrichtungen ein höheres Risiko haben als andere.

Für die Situation, in der der Roboter seitlich zum Ball stand und anlief, fiel dieser beim neuen Zweikampf in 3 von 4 Malen um. Durch das Anlaufen an den Ball dreht sich der Roboter und die Schussrichtung geht an das Maximum des Seitenbereiches. Dadurch ist die Schusspose näher am Hindernis. Hier wird zwar die andere Kollisionsabfrage verwendet, diese verschlechtert aber ebenfalls den Schuss nicht. Besser wäre es, sich zu drehen, um parallel zum Hindernis zu schießen, was aber mit diesem Verfahren nicht umsetzbar ist. Mit der aktuellen Hinderniserkennung ist es unmöglich zu wissen, welche Schussrichtung parallel zum Hindernis wäre. Dafür müsste die Ausrichtung von Robotern erkannt werden.

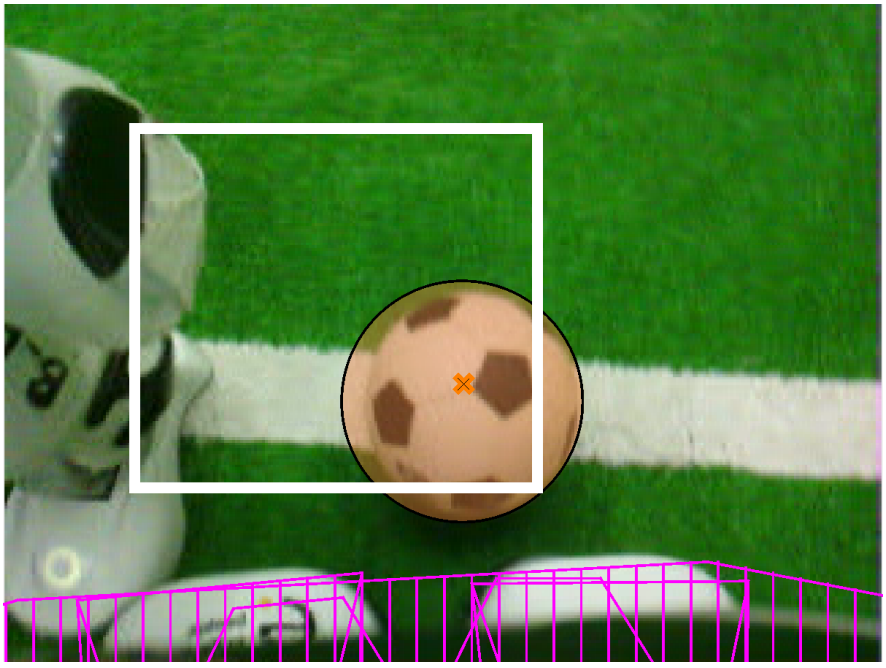

Abbildung 7.27 Ein Bild von der unteren Kamera. Die weiße Box ist ein erkanntes Hindernis. Der Ball ist dadurch nicht frei, da hinter diesem ein geglaubter Roboter steht.

Kontext / Version	Alt	Neu
Aus dem Feld	6	3
Zweikampf gewonnen	5	8
Reichw. Ø (in mm)	134	86
Reichw. σ (in mm)	78	28

Kontext / Feldbereich	Mittelkreis	Außenlinie
Alt Gewonnen	0	0
Neu Gewonnen	3	8
Unentschieden	1	0
Kick In für Alt	0	0
Kick In für Neu	0	1
Umgefallen Alt	1	1
Umgefallen Neu	0	3

Tabelle 7.2 Ergebnisse der Zweikämpfe auf den echten NAOs. Links die Zweikämpfe gegen Dummy-Roboter für den Standard-Schuss. Rechts die Versus Zweikämpfe sowie die Durchgänge für das Umfallen.

7.5 Fazit

Der neue Zweikampf zeigt vielversprechende Ergebnisse, da dieser im Direktvergleich besser als der alte abschneidet. Trotz des passiveren Ansatzes werden Zweikämpfe häufiger gewonnen und der Ballbesitz bewahrt. Allgemein scheinen auch sinnvollere Schüsse ausgeführt zu werden, da in der Simulation der neue Zweikampf in fast allen Metriken besser ausfällt. Es lässt sich dadurch vermuten, dass das Spielgeschehen nicht langsamer wird, sondern im Gegenteil aufgrund der höheren Torrate und Torschüsse sogar schneller.

Jedoch wurde das Ziel, weniger umzufallen, nicht erreicht. Zum einen funktioniert die Bewertung über die Hinderniskollision nicht, wenn der Schusswinkel das Problem ist statt der Schussfuß. Zum anderen verwendet der Standard-Schuss keine Kollisionsprüfung, wodurch dieser unter Umständen eher in einen anderen Roboter läuft, als wenn dieser, wie im alten Zweikampf, einfach einen Schuss ausführt.

Es bräuchte also eine konkrete Hindernisvermeidung und eine bessere Hinderniserkennung. Mit diesen könnte eine explizite Laufschrittplanung erstellt werden, welche die Kollision mit anderen Robotern vermeidet. Als Zwischenlösung könnte die Ausführung der Schüsse erweitert werden. Aktuell darf die Startkonfiguration, wie der Roboter relativ zum Ball stehen darf, bereits stark abweichen. Trotzdem wird versucht, den Schuss genau auszuführen. Hier könnte das Verhalten oder die Hindernismodellierung eine Verschiebung vorgeben, damit der Ball nicht mehr geplant mittig getroffen, sondern mit der Fußseite angeschnitten wird. Denn in solchen Zweikämpfen ist das Ziel nicht immer, den Ball perfekt zu treffen, sondern nur vom Gegner zu entfernen. Den Ball also allein mit der Fußkante zu treffen würde daher ausreichen, um die Schussrichtung zu bewahren und ebenso eine Kollision mit dem Gegner zu vermeiden.

Kapitel 8

Gesamtfazit und Ausblick

In dieser Arbeit wurden Ursachen untersucht, die für umfallende Roboter während eines Fußballspieles verantwortlich sind. Hierfür wurden Videoaufnahmen ausgewertet, um grundlegende Problembereiche zu identifizieren. So wurde festgestellt, dass die meisten Stürze in Gegenwart anderer Roboter geschehen und generell über mehrere Teams hinweg beobachtbar ist, dass schneller laufende Roboter häufiger umfallen. Aufgrund dieser Tatsachen wurde das aktuell verwendete Laufen untersucht und es wurden dabei mehrere Probleme festgestellt. Zum einem sind die Motoren nicht für die einwirkenden Kräfte auf die Gelenke ausreichend, wodurch Gelenke häufig Fehlpositionen aufweisen. Auch resultiert eine Disharmonie von Roboterzustand und der angefragten Laufschrittgröße in solchen Bewegungen, die den Roboter zwangsläufig zum Umfallen bringen. Die entwickelten Ansätze, sowohl mithilfe eines linearisierten invertierten Pendelmodells den Schwerpunkt in die Zukunft zu projizieren und damit eine Laufschrittanpassung online zur Ausführung durchzuführen als auch die Gelenkfehlposition des Standfußes mit dem Schwingfuß zu kompensieren, haben in der Evaluation sehr gute Ergebnisse gezeigt. Die Disharmonie wurde auf diese Weise durch die Laufschrittanpassung und die Probleme der Fehlpositionen durch die Kompensation erfolgreich reduziert. Dennoch ermöglichen die Entwicklungen nur ein etwas stabileres, wenngleich deutlich schnelleres Laufen, welches weiterhin bei Kollisionen mit anderen Robotern in Stürzen resultiert.

Darauf aufbauend wurde ein Zweikampfverhalten entwickelt, welches den Kontakt mit anderen Robotern minimieren soll. Hierfür wurden die Schüsse aus dem Laufen neu umgesetzt, um mehr Kontrolle über den Ball zu erlangen. Diese Schüsse nutzen das Laufen im vollen Umfang, indem die Fußbewegungen als Laufschritte definiert werden, deren Größen durch relative Ballpositionen beschrieben sind. Eine höhere Ballkontrolle wurde dabei aber nicht erreicht, da die Reichweiten und Richtungen ähnlich streuen. Jedoch ermöglichen sie dynamischere Reichweiten und eine schnellere Ausführung.

Mit diesen Eigenschaften wurde ein neuer Zweikampf vorgestellt, welcher mithilfe eines Potentialfeldes Schüsse bewertet und mit einem passiveren Verhalten, welches den Ballbesitz bewahren soll, die Möglichkeiten der neuen Schüsse ausnutzt und Kollisionen mit anderen

Robotern minimieren soll. Im direkten Vergleich sowohl in der Simulation als auch zwischen echten NAOs scheint der neue Zweikampf besser zu sein. Ebenfalls führen die neuen dynamischen Schussreichweiten und die allgemein höheren Entscheidungsmöglichkeiten zu keiner Verlangsamung sondern sogar zu einer Beschleunigung des Spielgeschehens. Jedoch existiert fortwährend das Problem, dass auf die Füße anderer Roboter getreten wird, wodurch die Sturzrate weiterhin hoch bleibt.

Trotz der gleichbleibenden Umfallrate können die Roboter nun dafür schneller laufen, ohne dabei häufiger zu stürzen als zuvor. Der neue Zweikampf ist dazu ein guter Ersatz, weist aber im Bezug auf Kollisionen dieselben Probleme auf. Hierfür bräuchte es eine Hindernismodellierung im Nahbereich des Roboters, damit eine Laufschrittplanung umgesetzt werden kann, welche andere Roboter vermeidet. Eine Umgebungsmodellierung ist folglich zwingend notwendig, um die Umfallrate signifikant zu minimieren. Diese ist aber aufgrund der aktuellen Robotererkennung, welche sehr fehlerbehaftet ist, nicht umsetzbar. Ebenso sind die Erweiterungen des Laufens verbesserungswürdig. Die Roboter besitzen eigentlich genügend Reaktionszeit und Bewegungsgeschwindigkeit, um in Extremfällen, wie dem Treten auf die Füße anderer Roboter, nicht umzufallen. Hierfür sollte entweder der ZMP miteingebracht werden, um eine bessere Laufschrittanpassung zu erzielen, oder der Ansatz von Missura u. a. [2019] vollständig umgesetzt werden. Deren Laufumsetzung korrigiert die Fußpositionen basierend auf der Bewegung des Schwerpunktes, was im Kern der Laufschrittanpassung dieser Arbeit gleicht. Abgesehen davon bleibt offen, warum genau die Schüsse weiterhin hohe Ungenauigkeiten aufweisen. Ob die Roboter beim Schießen tatsächlich rutschen, ist nur eine Vermutung und es bedarf weiterer Untersuchungen, ob die Fehlerquellen nicht auch in der Ballerkennung oder der Fußansteuerung liegen.

Literaturverzeichnis

[Böckmann 2015] Böckmann, Arne: *Entwicklung einer dynamischen Schussbewegung mit dem humanoiden Roboter NAO*, Diplomarbeit, 2015

[Belter 2019] Belter, Dominik: Efficient modeling and evaluation of constraints in path planning for multi-legged walking robots. In: *IEEE Access* 7 (2019), S. 107845–107862

[Bräunl 2006] Bräunl, Thomas: *Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2006. – ISBN 3540343180

[Dekker 2009] Dekker, MHP: *Zero-Moment Point Method for Stable Biped Walking*, Eindhoven University of Technology, Internship report, 2009

[Dortmund 2021] Dortmund, Nao D.: *YouTube Kanal Nao Devils*. 2021. – https://www.youtube.com/channel/UCp_3raHDiDfTqcIygMrg-Ow

[Dylla u. a. 2008] Dylla, Frank ; Ferrein, Alexander ; Lakemeyer, Gerhard ; Murray, Jan ; Obst, Oliver ; Röfer, Thomas ; Schiffer, Stefan ; Stolzenburg, Frieder ; Visser, Ubbo ; Wagner, Thomas: Approaching a Formal Soccer Theory from the Behavior Specification in Robotic Soccer, 2008. – ISBN 9781845640644, S. 161–186

[Hengst 2014] Hengst, Bernhard: rUNSWift Walk2014 Report / School of Computer Science & Engineering University of New South Wales. Sydney 2052, Australia, 2014. – Forschungsbericht. – http://cgi.cse.unsw.edu.au/~robocup/2014ChampionTeamPaperReports/20140930-Bernhard.Hengst-Walk2014Report.pdf

[HTWK 2019] HTWK, Nao-Team: *HTWKMotion*. 2019. – https://github.com/NaoHTWK/HTWKMotion

[Kajita u. a. 2003] Kajita, S. ; Kanehiro, F. ; Kaneko, K. ; Fujiwara, K. ; Harada, K. ; Yokoi, K. ; Hirukawa, H.: Biped walking pattern generation by using preview control of zero-moment point. In: *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on* Bd. 2, 2003. – ISSN 1050–4729, S. 1620–1626 vol.2

[Kajita u. a. 2001] Kajita, S. ; Kanehiro, F. ; Kaneko, K. ; Yokoi, K. ; Hirukawa, H.: The 3D linear inverted pendulum mode: a simple modeling for a biped walking pattern

generation. In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)* Bd. 1, 2001, S. 239–246 vol.1

[Kajita u. a. 2010] Kajita, S. ; Morisawa, M. ; Miura, K. ; Nakaoka, S. ; Harada, K. ; Kaneko, K. ; Kanehiro, F. ; Yokoi, K.: Biped walking stabilization based on linear inverted pendulum tracking. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, S. 4489–4496

[Katayama u. a. 1985] Katayama, Tohru ; Ohki, Takahira ; Inoue, Toshio ; Kato, Tomoyuki: Design of an optimal controller for a discrete-time system subject to previewable demand. In: *International Journal of Control* 41 (1985), Nr. 3, S. 677–699

[Krause 2018] Krause, Daniel: *Fallerkennung durch Vorhersage der Schwerpunkttrajektorie für den humanoiden Roboter NAO*, Bachelorarbeit, 2018

[Laue 2004] Laue, Tim: *Eine Verhaltenssteuerung für autonome mobile Roboter auf der Basis von Potentialfeldern.* Bremen, Germany, Universität Bremen, Diploma Thesis (Diplomarbeit), 2004

[Laue u. Röfer 2005] Laue, Tim ; Röfer, Thomas: A Behavior Architecture for Autonomous Mobile Robots Based on Potential Fields. In: Nardi, Daniele (Hrsg.) ; Riedmiller, Martin (Hrsg.) ; Sammut, Claude (Hrsg.) ; Santos-Victor, José (Hrsg.): *RoboCup 2004: Robot Soccer World Cup VIII.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2005. – ISBN 978–3–540–32256–6, S. 122–133

[Missura u. a. 2019] Missura, Marcell ; Bennewitz, Maren ; Behnke, Sven: Capture steps: Robust walking for humanoid robots. In: *International Journal of Humanoid Robotics* 16 (2019), Nr. 06, S. 1950032

[Müller u. a. 2011] Müller, Judith ; Laue, Tim ; Röfer, Thomas: Kicking a Ball – Modeling Complex Dynamic Motions for Humanoid Robots. In: Solar, Javier R. (Hrsg.) ; Chown, Eric (Hrsg.) ; Ploeger, Paul G. (Hrsg.): *RoboCup 2010: Robot Soccer World Cup XIV* Bd. 6556, Springer, 2011 (Lecture Notes in Artificial Intelligence), S. 109–120

[Nakanishi u. a. 2008] Nakanishi, Ryota ; Murakami, Kazuhito ; Naruse, Tadashi: Dynamic Positioning Method Based on Dominant Region Diagram to Realize Successful Cooperative Play. In: Visser, Ubbo (Hrsg.) ; Ribeiro, Fernando (Hrsg.) ; Ohashi, Takeshi (Hrsg.) ; Dellaert, Frank (Hrsg.): *RoboCup 2007: Robot Soccer World Cup XI.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. – ISBN 978–3–540–68847–1, S. 488–495

[NaoTH 2021] NaoTH, Berlin U.: *YouTube Kanal Berlin United - NaoTH.* 2021. – https://www.youtube.com/user/teamhumboldt

[Neri u. a. 2012] Neri, J. R. F. ; Zatelli, M. R. ; Santos, C. H. F. ; Fabro, J. A.: A Proposal of QLearning to Control the Attack of a 2D Robot Soccer Simulation Team.

In: *2012 Brazilian Robotics Symposium and Latin American Robotics Symposium*, 2012, S. 174–178

[Reichenberg 2018] Reichenberg, Philip: *Entwicklung eines dynamischen Aufstehens unter Vermeidung inkorrekter Bewegungszustände*, Bachelorarbeit, 2018

[Röhrig 2018] Röhrig, Enno: *Entwicklung eines dynamischen Schussverhaltens während des Laufens für den humanoiden Roboter Nao*, Diplomarbeit, 2018

[Riedmiller u. a. 2009] Riedmiller, Martin ; Gabel, Thomas ; Hafner, Roland ; Lange, Sascha: Reinforcement learning for robot soccer. In: *Autonomous Robots* 27 (2009), Nr. 1, S. 55–73

[Röfer u. a. 2019] Röfer, Thomas ; Laue, Tim ; Baude, Andreas ; Blumenkamp, Jan ; Felsch, Gerrit ; Fiedler, Jan ; Hasselbring, Arne ; Hass, Tim ; Oppermann, Jan ; Reichenberg, Philip ; Schrader, Nicole ; Weiss, Dennis: *B-Human Team Report and Code Release 2019*. 2019. – https://github.com/bhuman/BHumanCodeRelease/raw/master/CodeRelease2019.pdf

[rUNSWift 2016] rUNSWift: *rUNSWift-2016-release*. 2016. – https://github.com/UNSWComputing/rUNSWift-2016-release

[rUNSWift 2019] rUNSWift: *rUNSWift-2019-release*. 2019. – https://github.com/UNSWComputing/rUNSWift-2019-release

[Sabe u. a. 2004] Sabe, Kohtaro ; Fukuchi, Masaki ; Gutmann, J-S ; Ohashi, Takeshi ; Kawamoto, Kenta ; Yoshigahara, Takayuki: Obstacle avoidance and path planning for humanoid robots using stereo vision. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004* Bd. 1 IEEE, 2004, S. 592–597

[Schwarz u. a. 2019] Schwarz, Ingmar ; Urbann, Oliver ; Larisch, Aaron ; Brämer, Dominik: *TeamReport2019*. 2019. – https://github.com/NaoDevils/CodeRelease/blob/master/TeamReport2019.pdf

[Softbank Robotics 2021] https://www.softbankrobotics.com/emea/en/company Abgerufen am: 08.02.2021

[Strobel 2020] Strobel, Philipp: *Teamstrategie und -koordination im Kontext des Roboterfußballs*, Humboldt-Universität zu Berlin, Diplomarbeit, 2020

[Tsogias 2016] Tsogias, Alexis: *Laufen für den NAO-Roboter mittels Zero-Moment Point Preview Control*, Diplomarbeit, 2016

[Vukobratović u. Borovac 2004] Vukobratović, Miomir ; Borovac, Branislav: Zero-moment point—thirty five years of its life. In: *International journal of humanoid robotics* 1 (2004), Nr. 01, S. 157–173

[Wenk u. Röfer 2013] Wenk, Felix ; Röfer, Thomas: Online generated kick motions for the NAO balanced using inverse dynamics. In: *Robot Soccer World Cup* Springer, 2013, S. 25–36

[Yi u. a. 2013] Yi, Seung-Joon ; McGill, Stephen ; Lee, Daniel D.: Improved Online Kick Generation Method for Humanoid Soccer Robots. In: *8th workshop of humanoid soccer robots, Humanoids Conference*, 2013