Diplomarbeit

Hierarchische Schrittplanung für humanoide Fußballroboter

Andreas Schmitz

6. Dezember 2010

Gutachter:
Prof. Dr. Sven Behnke
Prof. Dr. Rolf Klein

Betreuer:
Marcell Missura

universitätbonn

Versicherung

Hiermit versichere ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Bonn, den

_____

(Andreas Schmitz)

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 RoboCup

Abbildung 1.1: Team NimbRo vs. Team Darmstadt Dribblers

Die Organisation „RoboCup“ existiert seit dem 04.07.1997 und trägt seitdem jährlich Weltmeisterschaften im Roboterfußball aus. Im Laufe der Jahre kamen weitere Disziplinen, wie „Rescue“ und „At Home“, dazu. Heutzutage treten beim RoboCup Teams verschiedener Universitäten in verschiedenen Disziplinen an. „Soccer“ ist neben „Rescue“ und „At Home“ die wesentlichste Disziplin. Bei „Rescue“ treten Rettungsroboter gegeneinander an, deren Aufgaben aus der Lokalisierung und Bergung von Dummys in schwierigen Gebieten bestehen. Die nächste Disziplin, „At Home“, umfasst Haushaltsroboter. Hierbei sollen Roboter entwickelt werden, die in menschlichen Haushalten Hilfestellungen geben können. Denkbare Aufgaben für diese Art von Robotern können sein, bestimmte Gegenstände zu holen oder wegzubringen oder Menschen wieder zu erkennen. Innerhalb der Disziplin „Soccer“ differenzieren sich die Ligen essenziell hinsichtlich des Abstraktionsgrades, sprich der Komplexität der Welt in der sich die Roboter bewegen. Während in der Simulationsliga ideale Welten simuliert werden, konzentriert sich die „Midsize-League“ auf omnidirektionale

Radfahrzeuge. Dies stellt schon eine deutlich komplexere Welt sowohl in der Wahrnehmung als auch in der Vorhersehbarkeit der Bewegungen dar. In der humanoiden Liga wird die Herausforderung durch das zweibeinige Laufen zusätzlich erschwert. In der „Midsize-League“ sind die Roboter während der Fortbewegung stabil, die humanoiden Roboter schwanken jedoch beim Gehen. Daraus resultiert, dass die Kameraaufnahmen verwackelt sind, was zur Folge hat, dass die wahrgenommene Welt drastisch verrauscht erscheint. Hierzu erschwert das begrenzte Sichtfeld des Roboters die Situation. Zudem ist die Bewegung unsicher, da die Folgepositionen im Vergleich zu den Fahrzeugen nach einer bestimmten Aktion weniger vorhersehbar sind. Dies hat zur Folge, dass die Teilnehmer der verschiedenen Ligen mit unterschiedlichen Problematiken konfrontiert werden. Während in der Simulationsliga große Teams, mit bis zu 11 Spielern pro Team den Wettkampf gegeneinander aufnehmen und somit der Fokus auf gutes Teamspiel gelegt wird, widmet sich die humanoide Liga den substanziellen Problemstellungen der zweibeinigen Roboter. So verkörpern die Erkennung und die Lokalisierung von Objekten in der Computervision enorme Problemstellungen. Die Verhaltensprogrammierung gewährleistet, dass der Roboter zum Ball schreitet und diesen in das Tor schießt oder dribbelt. Zurzeit stellt das Laufen zum Ball eine Herausforderung dar, mit welcher man sich massiv beschäftigt. Die Roboter der meisten Teams gehen hinter den Ball, bleiben stehen und richten sich dann zum Tor aus. Diesen Prozess schnell und flüssig auszuführen, kann somit spielentscheidend sein.

Diese Arbeit befasst sich mit der TeenSize-Klasse aus der Humanoid-Soccer-League. Dabei handelt es sich um Roboter welche eine Größe von 1 Meter bis zu 1,20 Meter messen. Diese duellieren sich in Fußballspielen mit einem Spieler und einem Torwart. Bei dem diesjährigem RoboCup 2010 in Singapur wurde unser Team, die Universität Bonn, Weltmeister in der TeenSize-Klasse.

1.2 Zielsetzung

Bisher wurde der Roboter rein reaktiv mit unterschiedlichem Verhalten gesteuert. Aus den Sensordaten werden deterministische Verhaltensanweisungen generiert, die gar nicht oder nur sehr wenig von früheren Entscheidungen abhängen und kein Modell verwenden, um das Ergebnis der berechneten Aktionen zu verifizieren und verschiedene mögliche Aktionen zu bewerten und miteinander zu vergleichen.

Das Ziel der Diplomarbeit besteht aus dem Entwurf und der Entwicklung einer hierarchischen Planung, die vorausschauende Entscheidungen treffen kann. Dabei sind langfristige Pläne stabiler als kurzfristige. Der Plan wird zyklisch neu berechnet um schnell auf Umgebungsänderungen reagieren zu können. Hierzu wurde ein Verhalten für den Roboter konstruiert, das ihm erlaubt, wahlweise schnell zum Ball zu gehen

und diesen im Lauf zu dribbeln oder zu schießen, ohne vorher langsamer zu werden und sich vor dem Ball ausrichten zu müssen. Der Vorgang, sich auszurichten und damit gezwungenermaßen langsamer zu werden, ist meistens sehr zeitaufwändig. Da es aber spielentscheidend ist, so wenig Zeit wie möglich in den kompletten Anlauf zu investieren, wird diese Arbeit diesen signifikanten Aspekt bearbeiten, um so die besagte Zeit zu mindern. Das Ziel wird erreicht, indem der Roboter bereits während des Gehens seine Schritte so kalkuliert, dass er ausgerichtet am Ball ankommt. Infolgedessen entfällt die zuvor benötigte Zeit des Ausrichtungsprozesses. Aufgrund der Dynamik eines Fußballspieles ist eine oftmalige Neuplanung nötig. Die Planung muss innerhalb eines Schrittes erfolgen. Da Verhalten (auf Ebene $L1$) im 24 Millisekunden Takt berechnet werden und die Computervision den Großteil der Rechenkapazität verwendet, darf ein Teilschritt der Planung nicht mehr als 5-10 ms benötigen.

1.3 Gliederung der Arbeit

Das zweite Kapitel beschreibt und erklärt die benötigten Grundlagen. Angefangen mit dem verwendetem Roboter namens „Dynaped“ (2.1), welcher mit einer stetig weiterentwickelten Anwendung unter Windows XP als Software (2.2) betrieben wird. In der hier entwickelten Planung kommt der A*-Algorithmus häufig zum Einsatz und wird im Abschnitt 2.3 erläutert, sowie das mehrlagige Perzeptron, welches im Anschluss daran in (2.4) erklärt wird.

Das dritte Kapitel dieser Arbeit stellt die Grundkonzepte der hierarchischen Planung vor. Nachdem das Prinzip der hierarchischen Planung (3.2) erörtert wurde, wird eine Übersicht über die einzelnen Ebenen (3.3) gegeben.

Das in der Arbeit untersuchte hierarchische Planungssystem beinhaltet fünf Ebenen. Die erste Ebene, die Aktionsplanung, ist nicht Teil dieser Arbeit und wird deswegen in 4 nur knapp erläutert. Die zweite Ebene, die Ebene der Pfadplanung (5), plant zunächst einen groben 2D-Pfad. Als Erstes wird die Pfadplanung unter Verwendung des A*-Algorithmus (5.2) vorgestellt, danach unter Verwendung von Viapunkten (5.3). Als Nächstes tritt die dritte Ebene, die Trajektorienplanung (6), in Kraft, welche einen dynamischen 6D-Pfad unter Berücksichtigung der Roboterdynamik erstellt. Hierbei wird der vorberechnete grobe Pfad bis zu einem Zwischenziel verfeinert. Die Trajektorienplanung wurde unter Verwendung des A*-Algorithmus (6.1)und drei verschiedenen geometrischen Kurven getestet. Zum einen mit Splines, Bezierkurven (6.1) und zum Schluss mit Klothoiden (6.2). Das Kapitel der Trajektorienplanung wird mit einer Zusammenfassung beendet. Die letzte Ebene dient der Schrittplanung (7). Hierzu wird zunächst ein Schrittmodell (7.2) benötigt, damit eine Schrittplanung mit A*-Algorithmus (7.3) ermöglicht werden kann. Alternativ kann die Schrittplanung auch mit einem Funktionsapproximator bewältigt

werden, für diese Herangehensweise können verschiedene Funktionsapproximatoren (7.5) verwendet werden. Im letzten Kapitel dieser Arbeit werden die Ergebnisse (8) präsentiert.

1.4 Verwandte Arbeiten

Zweibeinige Fortbewegung ist ein herausfordernder, aber sehr beliebter Forschungsbereich. Zahlreiche, teilweise grundlegend unterschiedliche Konzepte wurden in den letzten Jahrzehnten vorgestellt. Diese Arbeit konzentriert sich nur auf zwei verschiedene Gangarten. Auf zentrale Muster basierende Methoden werden in [13, 14] beschrieben. Die Anwendbarkeit dieser Methoden wurde in den letzten Jahren wiederholt im RoboCup bei der humanoiden Fußballliga demonstriert. Ebenso wurden für inverse Kinematik basierte Lösungen in [15, 16] und [19] sehr erfolgreiche Implementierungen präsentiert, die sich das Zero Moment Point Kriterium zu Nutze machen [26].

Es wurden verschiedenste Typen geometrischer Kurven für das Generieren von Robotertrajektorien angepasst. In [2] wurden Bézierkurven verwendet, um Trajektorien für omnidirektionionale Radroboter zu berechnen. Die Parameter der Trajektorien wurden iterativ optimiert um Geschwindigkeitsbeschränkungen einzuhalten. In [1] wurden zwei funktionale Splines zu einem zweidimensionalen Spline zusammengesetzt und für die Trajektorienplanung verwendet. Da es sich bei beiden Verfahren um iterative Methoden handelt, sind die Laufzeitbeschränkungen nur schwer einzuhalten. Die Anfangskrümmung ist nicht kontrollierbar, was allerdings für häufiges Neuplanen unerlässlich ist. Außerdem lassen sich auf den beschriebenen geometrischen Kurven nur schwer Schritte verteilen.

Eine andere interessante Möglichkeit wurde mit Klothoiden gefunden [6]. J. Laumond und G. Arechavaleta wiesen in [3, 4, 5] nach, dass die menschlichen Fortbewegungsbahnen gut mit Trajektorien nicht-holonomer Fahrzeuge approximiert werden können. Solche Bahnen sind stückweise aus elementaren Klothoidenbögen zusammengesetzt. Dazu wurden mit einer Motion Capture Anlage Bewegungen von Personen aufgenommen, die in einem hindernisfreien Raum durch eine Tür gingen. Die Bahnen konnten mit Klothoidenketten approximiert werden. D. S. Meek und D. J. Walton beschrieben in [9] wie zwei Punkte mit gegebener Ausrichtung mit Klothoidenketten verbunden werden können. Allerdings musste die Krümmung in den Startpunkten Null sein. In [7, 8] wurden beliebige Krümmungen, inklusive der Null, erlaubt. Allerdings gaben Meek und Walton strenge Bedingungen an, die für die Funktion des Algorithmus benötigt wurden. Weitere interessante Veröffentlichungen in diesem Zusammenhang sind [10, 11].

Die Schrittplanung ist ein relativ neues Forschungsthema und es existieren vergleichbar wenig anwendbare Lösungen. Die wichtigsten Vorschläge in [20, 21, 22] und [23] basieren auf dem A*-Algorithmus. Durch die Einführung einer starken Diskretisierung des Zustandraumes und durch die Verwendung einer kleinen, diskreten Menge von Aktionen, planen diese Online-Lösungen lediglich ein paar Schritte voraus und sind in der Lage, mit dynamischen Umgebungen umzugehen. Pläne auf unebenem Grund wurden auch berücksichtigt, so dass die Schrittpläne auch beinhalten können, auf Hindernisse und Treppen zu steigen.

Eine faszinierende, alternative Lösung wurde vor kurzem in [24] präsentiert. Hier wurde eine kurze Sequenz von zukünftigen Schritten als virtuelle kinematische Kette interpretiert. Die Kinematik des Roboters wird durch diese kinematische Kette erweitert, dessen Endeffektor das Ziel erreichen soll. Ihre Position wird mit inverser Kinematik bestimmt. Der Konfigurationsraum und der Aktionsraum wurden dabei nicht diskretisiert, allerdings ist der Algorithmus rechenintensiver. Eine der Rechendauer nach viel versprechende Methode, die, wenn die Umwelt nicht zu überladen ist, auch in wenigen Millisekunden planen kann, wurde in [25] vorgeschlagen. Die Idee ist, das Schrittplanungsproblem im Wesentlichen mit einem Pfadplanungsalgorithmus zu lösen. Tatsächliche Schrittpositionen werden nur an wichtigen Punkten gegeben, wo die Laufgeschwindigkeit des Roboters gleich Null sein muss, zum Beispiel beim Übersteigen eines Hindernisses. Die Mehrheit der Schrittpositionen werden von dem für den HRP-2 entwickelten Bewegungsgenerator entlang des geplanten Weges gelegt [15, 18, 17].

Die am nächsten verwandte Arbeit ist [22], wo ein auf den A*-Algorithmus basierter Schrittplanungsalgorithmus speziell für den humanoiden Roboter ASIMO angepasst wurde. Da der Gang-Algorithmus des ASIMO nicht genau bekannt ist, waren die Autoren gezwungen, einen Schrittvorhersage Algorithmus aus Beobachtungen mit einem Motion Capture System zu entwickeln.

Das Schrittmodell dieser Diplomarbeit wurde bereits von mir in [31] veröffentlicht.

2 Grundlagen

In diesem Kapitel werden die im weiteren Verlauf benötigten Grundlagen beschrieben. Zunächst wird auf die angewendete Hardware und Software eingegangen. Im Anschluss daran werden die in der Arbeit verwendeten Algorithmen vorgestellt. In Abschnitt 2.3 wird der A*-Algorithmus ausführlich beschrieben. Danach wird unter Abschnitt 2.4 das mehrlagige Perzeptron erläutert.

2.1 Dynaped

Abbildung 2.1: Dynaped beim Kicken

Der Roboter, welcher in dieser Arbeit verwendet wurde, ist ein TeenSize-Soccer-Roboter mit einer Größe von 1,05 m und einem Gewicht von 7 kg. Insgesamt besitzt er 13 Freiheitsgrade, welche auf 5 Freiheitsgrade pro Bein, 1 Freiheitsgrad pro Arm und 1 Freiheitsgrad am Kopf verteilt werden. Die Gelenke werden mit Master-Slave-Paaren von Dynamixel EX-106 und RX-64 betrieben. Diese Motoren werden von

einem Microcontroller angesteuert, welcher seine Befehle von einem Sony Vaio Ultramobile-PC UX1XN mit verbautem Intel 1.3 Ghz Core Solo empfängt. Seine Umwelt nimmt er ausschließlich über ein Computer Vision System wahr. Dieses ist mit einer WVGA USB2.0 Camera IDS uEye UI-1226LE mit integrierter Weitwinkellinse ausgestattet. Zudem verfügt er über einen zweiaxialen Beschleunigungssensor ADXL203 und zwei Gyroskope ADXRS. Die Motoren erlauben ihm omnidirektionales Gehen mit einer maximalen Geschwindigkeit von ungefähr 40 cm pro Sekunde.

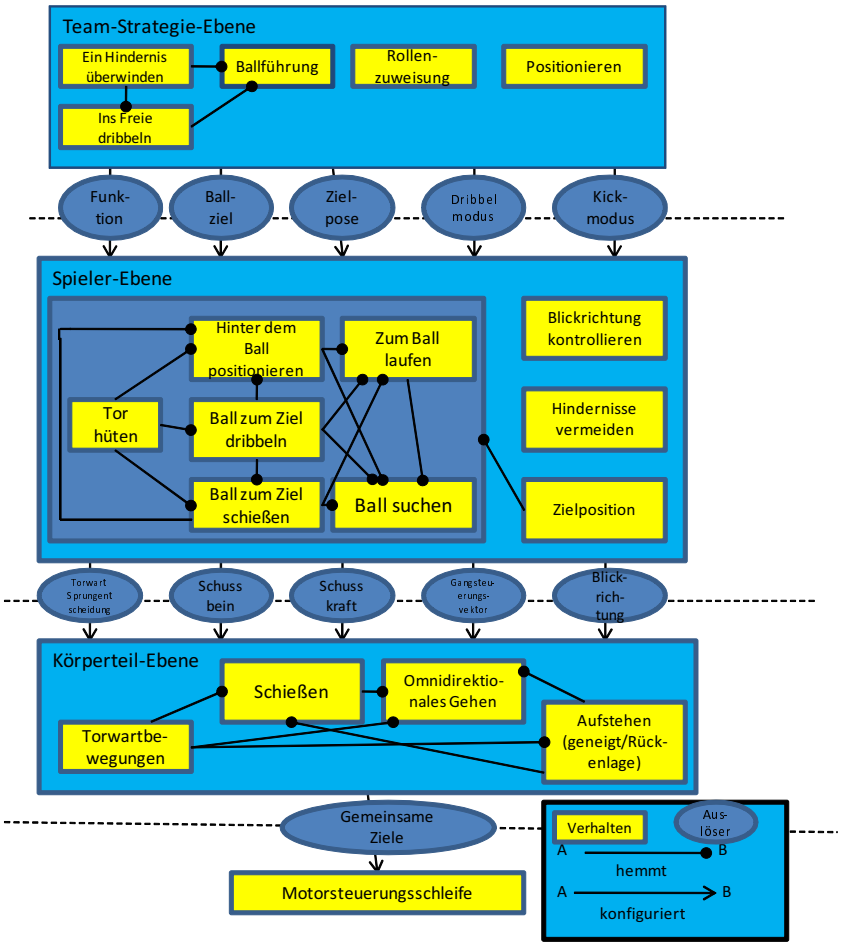

Abbildung 2.2: Übersicht über die drei Ebenen: Körperteil-Ebene $L0$, Spieler-Ebene $L1$ und Team-Strategie-Ebene $L2$ mit den wichtigsten Verhalten auf den Ebenen und den Übergabewerten (Aktuatoren) zwischen den Ebenen.

Geschwindigkeits- und Beschleunigungsbereich

Die Geschwindigkeit und die Beschleunigung des Roboters unterliegen Beschränkungen. Dabei sind die einzelnen Komponenten der Geschwindigkeitsbeschränkungen voneinander abhängig, siehe Abbildung 2.3. Wird eine Komponente erhöht, verringern sich die Anderen. Die Beschleunigungsbeschränkungen hingegen sind unabhängig voneinander.

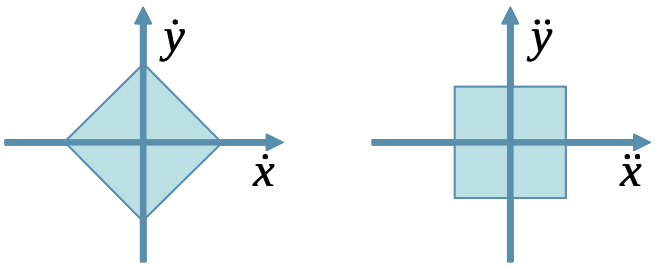

Abbildung 2.3: Zulässiger Beschleunigungs- und Geschwindigkeitsbereich. Komponenten der Beschleunigung sind unabhängig voneinander, die der Geschwindigkeit sind voneinander abhängig.

Position und Pose

In der Arbeit wird häufiger zwischen Position und Pose unterschieden. Eine Position $p_{xy} = x, y$ ist eine zweidimensionale Positionsangabe in kartesischen Koordinaten in Bezug auf einen angegebenen Referenzpunkt. Davon unterscheidet sich die Pose $p = (x, y, \theta)$ als zweidimensionale Position mit zusätzlich angegebener Ausrichtung in Bezug auf die Ausrichtung des Referenzpunktes. Wenn nicht anders bestimmt, werden Posen verwendet.

2.2 Verwendete Software

Der Ultra-Mobile-PC Sony Vaio, welcher die zentrale Recheneinheit des Roboters darstellt, arbeitet mit einer langjährig weiterentwickelten Anwendung unter Windows XP. Diese Anwendung wurde in Microsoft Visual Studio C++ mit Qt von Nokia als Oberfläche entwickelt. Dies ist ein Framework, welches auf unterschiedlichen Ebenen, verschiedene, voneinander unabhängige Verhalten ausführt. Im Einzelnen handelt es sich dort um eine „Team-Strategie-Ebene“ *L2*, welche grundlegende Strategien implementiert, wie z.B. sich zu positionieren. Dieser Befehl vermittelt dem

Roboter, vor dem Spielstart an die Aufstellungspositionen zu gehen. Des Weiteren enthält diese Ebene noch das Standardspielverhalten, welches Strategien für das Fußballspiel definiert und implementiert. Auf der darunter liegenden Ebene $L1$ befinden sich Verhalten, wie sich hinter dem Ball positionieren (GoBehindBall), dribbeln (Dribble), kicken (KickBallToTarget) oder den Ball zu suchen (SearchBall). Diese Verhalten legen Parameter zur Steuerung der untersten Ebene fest. Die unterste Ebene $L0$ implementiert Bewegungen wie das Gehen (DynamicGait) oder die Kickbewegung (QuickKick). Verhalten auf dieser Ebene senden direkte Motorsteuerwerte an den Roboter. Hierbei ist zu bedenken, dass das Kicken auf der untersten Ebene $L0$ nur die eigentliche Kickbewegung darstellt, während das Kickverhalten auf der Ebene ($L1$) dafür Sorge trägt, dass der Roboter richtig ausgerichtet ist. Abbildung 2.2 demonstriert den Aufbau der Ebenenstruktur mit einigen Verhalten. In [12] wird das Framework detailliert beschrieben.

2.3 A*-Algorithmus

In diesem Abschnitt wird der A*-Algorithmus vorgestellt, der in dieser Arbeit des Öfteren zum Einsatz kommt. Das erste Mal veröffentlicht wurde der Algorithmus von Peter Hart, Nils J. Nilsson und Bertram Raphael im Jahre 1968 in [28]. Der Algorithmus gehört zu der Klasse der informierten Suchalgorithmen, bei denen im Gegensatz zu uninformierten Suchalgorithmen eine Schätzfunktion verwendet wird um zielgerichteter zu suchen und um die Laufzeit verringern zu können. Er berechnet den kürzesten Pfad zwischen zwei Knoten innerhalb eines Graphen mit positiven Kantengewichten. Begründet durch den durchschlagenden Erfolg und der breiten Einsatzmöglichkeit sind zahlreiche, verwandte Algorithmen entstanden, die versuchen, unterschiedliche Schwächen des A*-Algorithmus auszugleichen.

Der Algorithmus verwaltet zwei Listen von Knoten. In der offenen Liste sind alle bekannten, noch zu besuchenden Knoten enthalten. Zu diesen Knoten ist ein Weg bekannt, auch wenn dieser noch nicht optimal sein muss. Damit der Pfad zurück verfolgbar bleibt, wird in jedem Knoten ein Verweis auf dessen Vorgängerknoten abgespeichert. Die offene Liste ist sortiert nach dem Wert $f(k)$ des Knoten k. Der Wert $f(k)$ setzt sich zusammen aus $g(k)$, den kumulierten Wegkosten, die notwendig waren, um diesen Knoten zu erreichen, sowie einem heuristischen Wert $h(k)$, welcher eine untere Schranke der verbleibenden Wegkosten angibt:

$$f(k) = g(k) + h(k) \tag{2.1}$$

Die Implementierung der offenen Liste hat massive Auswirkungen auf die Laufzeit des Algorithmus. Meist wird hier eine Vorrangwarteschlange verwendet.

Die zweite Liste ist die geschlossene Liste, die die bereits vollständig evaluierten Knoten beinhaltet. Diese Knoten werden auch expandierte Knoten genannt. Zu jedem dieser Knoten ist der günstigste Weg bekannt. Wie die Knoten der offenen Liste enthalten auch die Knoten in der geschlossenen Liste einen Verweis auf ihren Vorgängerknoten. So lässt sich von jedem Knoten aus der geschlossenen Liste mit Hilfe der Verweise der Pfad bis zum Startknoten zurück verfolgen.

Zusätzlich zu den Knoten, die in diesen beiden Listen verwaltet werden, existieren noch die unbekannten Knoten. Diese werden nicht explizit in einer Liste verwaltet, sondern bei Bedarf aus dem gerade expandierten Knoten berechnet.

Zu Beginn wird die offene Liste mit dem Startknoten initialisiert. Es ist der einzige Knoten, der dem Algorithmus am Anfang bekannt ist.

In der offenen Liste befinden sich die noch nicht besuchten Knoten. Solange ein Knoten in dieser Liste enthalten ist, wird der günstigste Knoten, sprich mit dem kleinsten Wert $f(k)$, zuerst expandiert. Dabei werden die Nachfolgeknoten berechnet, die Heuristik ausgewertet und in die offene Liste einsortiert. Ist der Knoten bereits in der offenen Liste und der Wert $f(k)$ größer als der neu berechnete, wird dieser, sofern es möglich ist, ersetzt. Ist dies nicht möglich, wird der Knoten zusätzlich eingefügt. Erreicht einer der nachfolgenden Knoten einen Zielknoten, so ist der Algorithmus beendet und der Pfad wird zurückgegeben. Dies ist nachweisbar der günstigste Weg zum Ziel. Bedingt durch die Verweise auf die Vorgänger wird der Pfad in der umgekehrten Reihenfolge vom Ziel zum Start ausgegeben. Wird der Zielknoten allerdings nicht erreicht und es befindet sich kein weiterer Knoten mehr in der offenen Liste, bricht der Algorithmus die Berechnung ab. In diesem Fall existiert kein Weg zu den angestrebten Zielknoten.

Definition

Eine Instanz (M, s, Z, F, c, h) oder (M, s, Z, A, t, c, h) dieses Algorithmus ist gegeben durch:

- eine Knotenmenge M, die eine Menge von Zuständen angibt, welche für den A*-Algorithmus gültige Knoten darstellen.
- den Startknoten s, an dem der Algorithmus seine Suche beginnt.
- die Menge der Zielknoten Z, aus der ein Element erreicht werden muss, um eine gültige Lösung zu finden. Oft wird diese Menge als Zielfunktion implementiert.

Abhängig von einem Knoten k:

- eine Menge von Folgeknoten $F(k)$, welche von einem Knoten k durch eine Aktion erreicht werden können. Oft wird diese Menge auch implizit durch eine Aktionsmenge $A(k)$ gegeben, die mit Hilfe einer Transferfunktion $t : A \times M \rightarrow M$ in einen Nachfolgeknoten umgewandelt wird.

- eine Kostenfunktion $c : M \times M \rightarrow R$, die die Kosten eines Schrittes vom Knoten k_t zu seinem Nachfolgeknoten $k_{t+1} \in F(k_t)$ bestimmt.

- eine Heuristik $h(k)$, die die Kosten des restlichen Pfades von dem Knoten k zu einem Zielknoten schätzt.

Heuristiken

Für den A*-Algorithmus gibt es verschiedene Eigenschaften der Heuristiken. Eine Heuristik muss mindestens zulässig sein. Eine strengere Bedingung als die Zulässigkeit ist die monotone Heuristik, welche auch automatisch zulässig ist. In der Regel werden monotone Heuristiken verwendet. Ein Beispiel hierfür ist die Berechnung der Luftlinie zwischen zwei Punkten (Orten) in einer Wegsuche.

Zulässige Heuristik

Eine zulässige Heuristik ist eine, welche die realen Kosten stets unterschätzt. Das bedeutet, wenn die realen Kosten von einem Knoten k zum Ziel $C(k)$ betragen, liegt der Wert der Heuristik $h(k)$ immer unter dem Wert $C(k)$:

$$\forall k \in M : h(k) \leq C(k) \tag{2.2}$$

Zu einem expandierten Knoten muss nicht zwingend der kürzeste Weg bekannt sein, wenn die verwendete Heuristik nur zulässig und nicht monoton ist. Das hat zur Folge, dass bei der zulässigen Heuristik keine geschlossene Liste verwendet werden kann, da es möglich sein muss, einen Knoten mehrmals zu expandieren.

Monotone Heuristik

Eine Heuristik ist nur dann monoton, wenn sie die Kosten für den Weg zum Ziel nie überschätzt, wie auch eine zulässige Heuristik. Dazu kommt noch, dass die Relation

$$h(k) \leq c(k, k') + h(k') \tag{2.3}$$

für jeden Knoten k und dessen Folgeknoten k', gelten muss. $c(k, k')$ stellt die Kosten vom Knoten k zum Folgeknoten k' dar und $h(k)$ die Heuristik vom Knoten k. Diese Formel besagt, dass die Kosten vom Knoten k kleiner sein müssen als die Kosten von einem Knoten zum nächsten $c(k, k')$ plus der Heuristik des Folgeknotens k'. Intuitiv bedeutet das, dass der Wert $f(k) = g(k) + h(k)$ stets ansteigt.

Nullheuristik

Eine Nullheuristik ist eine Heuristik mit der Funktion $h(k) = 0$. In diesem Fall muss der A*-Algorithmus jeden einzelnen Knoten besuchen, da er von der Heuristik keinen geschätzten Kostenwert vermittelt bekommt. Die Heuristik ist zwar zulässig, da die Wegkosten positiv definiert sind und somit niemals geringer als Null sind. Jedoch muss der A*-Algorithmus eine vollständige Breitensuche durchführen, da die Nullheuristik keinerlei Informationen besitzt.

Optimale Heuristik

Eine optimale Heuristik errechnet die exakten Wegkosten von jedem Knoten zum Zielknoten. Durch diese Heuristik muss der A*-Algorithmus keinen überflüssigen Knoten mehr besuchen, da die Heuristik bereits den optimalen Weg kennt. Allerdings ist es oftmals nicht möglich, die optimale Heuristik zu berechnen. Genaue oder sogar optimale Heuristiken sind oft sehr zeitintensiv, so dass sie sich in der Gesamtlaufzeit nicht lohnen.

In der Regel liegen Heuristiken zwischen den Extremen der Nullheuristik und der optimalen Heuristik. Häufig sind mehrere Heuristiken für eine Problemstellung möglich. Eine Heuristik $h_2(k)$ wird von einer anderen Heuristik $h_1(k)$ für die gleiche Problemstellung dominiert, wenn h_1 immer genauere Werte schätzt als h_2.

$$h_1 \textit{ dominiert } h_2 <=> h_1 \textit{ zulässig } \wedge \; h_2 \textit{ zulässig } \wedge \forall k : h_1(k) \geq h_2(k) \tag{2.4}$$

Es wird stets die dominierende Heuristik verwendet, es sei denn sie ist zeitlich zu aufwändig und würde den gesamten Algorithmus verlangsamen.

Inflationsparameter

Sei $h_z(k)$ eine zulässige Heuristik und sei $h_i(k)$ definiert als:

$$h_i(k) = i \cdot h_z(k) \tag{2.5}$$

mit dem Inflationsparameter $i \geq 1$. Der Inflationsparameter bewirkt eine höhere Gewichtung der Heuristik im Vergleich zu den bisherigen Kosten $g(k)$, was zu Folge hat, dass der A*-Algorithmus stärker der Heuristik folgt. Allerdings kann die Heuristik dadurch überschätzen also ist es nicht mehr garantiert, dass die Heuristik zulässig ist. Somit kann auch nicht mehr sicher gestellt werden, dass der A*-Algorithmus den optimalen Weg findet. Jedoch sind die Kosten von dem gefundenen Weg C_i maximal i-mal so groß wie der optimale Weg $C^*(=> C_i \leq i \cdot C^*)$. Der Vorteil des Inflationsparameters ist, dass der A*-Algorithmus unter Umständen deutlich schneller in der Berechnung ist als mit der ursprünglichen Heuristik h_z.

2.4 Das mehrlagige Perzeptron

Das mehrlagige Perzeptron ist ein künstliches, neuronales Netz, wie es in [29] beschrieben wurde. In dieser Arbeit wird das mehrlagige Perzeptron, kurz MLP, als Funktionsapproximator verwendet. Ein mehrlagiges Perzeptron besteht aus mehreren Ebenen. Einer Eingangsebene, in der keine Berechnungen stattfinden, sondern die die Eingabe aufnimmt und an die nächste Ebene weiterleitet, einer oder mehreren versteckten Ebenen und einer Ausgabeebene, die aus der versteckten Ebene die Ausgaben berechnet. Die versteckte Ebene und die Ausgabeebene bestehen aus Neuronen, auch Perzeptrons genannt. In der Ausgabeebene können sich beliebig viele Neuronen befinden. Sind die zu erlernenden Funktionen unabhängig, ist es nicht sinnvoll, diese mit einem MLP zu lernen. Stattdessen werden mehrere MLPs mit jeweils einer Ausgabe für jede Funktion trainiert, was den Lernprozess vereinfacht.

Ein einfaches Perzeptron hat n Eingänge x_i, deren Werte für jedes Neuron in der versteckten Ebene gewichtet aufsummiert werden. Auf diese Summe wird eine Aktivierungsfunktion angewendet. Das daraus resultierende Ergebnis stellt die Ausgabe des Neurons dar. Die Ausgabeneuronen verarbeiten dann die Ergebnisse der Neuronen der versteckten Ebene. Die Ausgabe o_j eines Neurons j ist definiert als

$$o_j = \varphi(net_j) \tag{2.6}$$

mit der Netzeingabe net_j für dieses Neuron

$$net_j = \sum_{i=1}^{n} x_i w_{ij}, \tag{2.7}$$

der Aktivierungsfunktion φ und der Gewichtung w_{ij} zwischen den Neuronen i und j.

Das MLP wird mit einem Verfahren namens Backpropagation trainiert. Dazu werden Beispieldaten benötigt, an deren erwarteten Ausgaben das Ergebnis des MLPs angenähert wird. In einem Durchgang werden die Eingaben eines Beispiels an das MLP

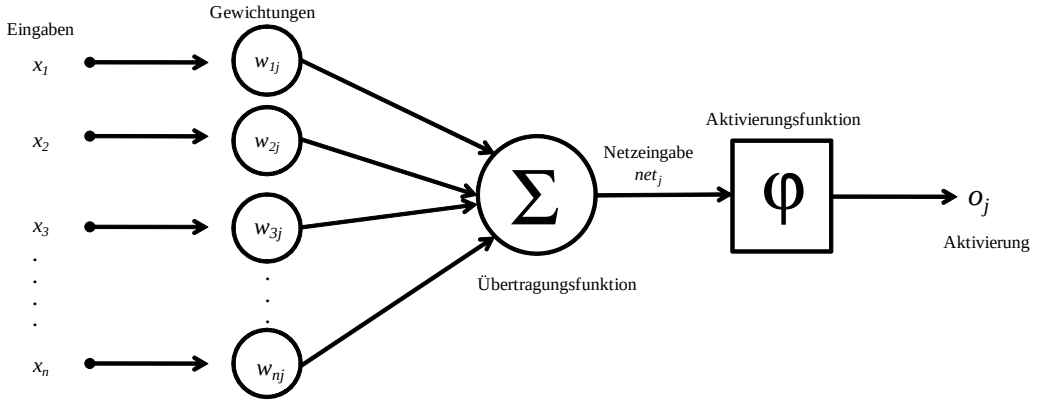

Abbildung 2.4: Struktur eines Neurons

übergeben, berechnet und dessen Ausgabe mit der erwarteten Ausgabe des Beispiels verglichen. Der Unterschied dieser beiden Ausgaben ist der Fehler für das Beispiel. Der berechnete Fehler wird dann von der Ausgabeschicht zur Eingabeschicht zurück propagiert. Dazu werden die Gewichtungen der Neuronen in Abhängigkeit von ihrer Auswirkung auf das Ergebnis angepasst. Bei einer erneuten Berechnung dieses Beispiels ist der Fehler dadurch garantiert geringer. Die Änderung der Gewichtung ist gegeben durch

$$w_{ij}^{neu} = w_{ij}^{alt} + \Delta w_{ij} \tag{2.8}$$

mit

$$\Delta w_{ij} = \eta \delta_j y_i \tag{2.9}$$

$$\delta_j = \begin{cases} \varphi'(net_j)(t_j - o_j), & \text{wenn } j \text{ Ausgabeneuron ist} \\ \varphi'(net_j) \sum_k \delta_k w_{jk}, & \text{wenn } j \text{ verdecktes Neuron ist.} \end{cases} \tag{2.10}$$

Dabei beschreibt η eine Lernrate, mit der die Gewichtsänderungen beeinflusst werden können, y_i die Ausgabe des Neurons i, t_j die erwartete Ausgabe des Neurons j und k den Index der nachfolgenden Neuronen von j.

3 Hierarchische Planung

3.1 Einleitung

Während der Roboter sich bisher für den Schuss positionieren musste und einige Zeit zum Ausrichten benötigte, soll dies nun ohne starken Geschwindigkeitsverlust möglich sein. Um einen gezielten Torschuss erreichen zu können, muss der Roboter seinen Fuß unmittelbar neben den Ball positionieren. Um dies zu ermöglichen, wurde ein Konzept eines hierarchischen Ansatzes für den Ballanlauf des Roboters entwickelt. Zunächst wird ein Überblick über die Ebenen gegeben, um später detailliert auf die einzelnen Ebenen einzugehen. Der Fokus dieser Arbeit liegt auf der Schrittplanungsebene, die in Kapitel 7 genau betrachtet wird, gelegt.

3.2 Prinzip der hierarchischen Planung

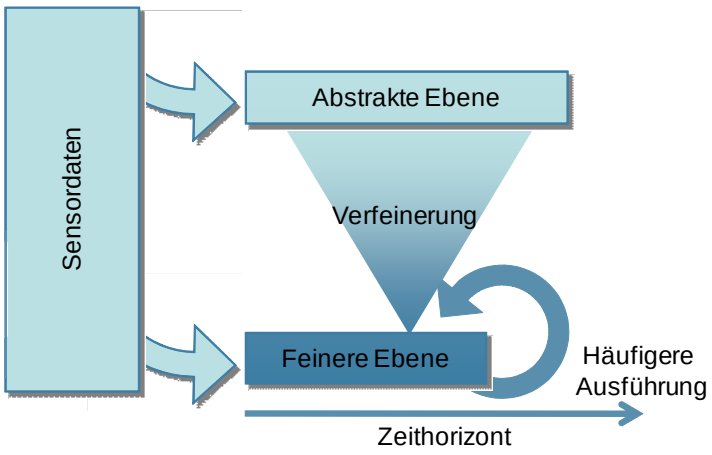

Abbildung 3.1: Der schematische Aufbau einer hierarchischen Planung. Die Ebenen verfeinern ihre Pläne, beginnend von einer abstrakten bis zu einer feinen Ebene. Je feiner die Ebenen umso kürzer wird der Zeithorizont und desto häufiger muss die Ebene neu planen.

In Abbildung 3.1 wird das Prinzip der hierarchischen Planung schematisch dargestellt. Es wird zunächst ein grober Plan erzeugt, welcher dann in den tieferen Ebenen unter Verwendung detaillierterer Informationen verfeinert wird. Der Planungshorizont kann auf den komplexeren, unteren Planungsebenen jeweils verkürzt werden. Durch die Flüchtigkeit der detaillierten Pläne müssen die unteren Ebenen häufig neu geplant werden, während die Pläne höherer Ebenen durch deren Abstraktionsgrad stabiler sind und seltener aktualisiert werden müssen. Ohne hierarchische Planung muss der detaillierteste Plan bis zum Ende berechnet werden, da mit einem beschränkten Zeithorizont kein optimales Verhalten erzeugt werden kann. Ein anderer gröberer Plan liefert nicht das gesuchte detaillierte Ergebnis. Durch die hierarchische Planung können die mittel- und langfristigen Teile des geplanten Verhaltens durch schneller und seltener zu berechnende Pläne ersetzt werden, wodurch eine Reduktion der benötigten Rechenzeit erreicht wird. Das Ergebnis dieser ungenauen Pläne hilft der nächst präziseren Ebene, indem zum einen das Zwischenziel zur Reduktion des Planungshorizonts verwendet werden kann, zum anderen möglicherweise der gefundene Plan als Beschränkung oder als Heuristik in die Planung einfließen kann. Zudem können die von den unteren Ebenen benötigten detaillierten Informationen ohnehin nur auf kurze Entfernung von den Sensoren erfasst werden, was eine vollständige Planung mit hohem Detailgrad zwecklos macht.

3.3 Übersicht über die Ebenen

Der Entwurf sieht vier Planungsebenen und eine ausführende Ebene, wie in Abbildung 3.2 skizziert, vor. Die erste Planungsebene ist die Aktionsplanung, die ei-

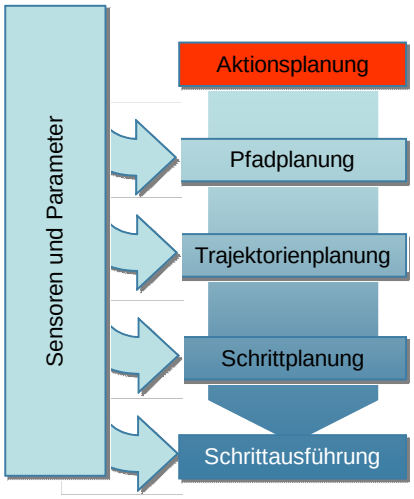

Abbildung 3.2: Schematische Darstellung der verwendeten Ebenen. Der Detailgrad erhöht sich von oben nach unten.

ne grobe Aktion plant. Dies kann beispielsweise ein Torschuss sein. Diese Aktion wird an die nächste Ebene, die Pfadplanung, übermittelt. Die Pfadplanung plant unter Berücksichtigung von Hindernissen einen groben, zweidimensionalen Pfad. Die Trajektorienplanung verfeinert diesen groben, zweidimensionalen Pfad unter Berücksichtigung der Roboterdynamik sowie dessen Geschwindigkeits- und Beschleunigungsbeschränkungen. Die unterste Planungsebene ist die Schrittplanung. Diese Ebene wird in dieser Arbeit genau betrachtet werden. Die Planungsebene plant die einzelnen Schritte des Roboters, damit der Roboter während des Gehens den Ball gezielt schießen kann. Die Ausgabe der Schrittplanung wird an die Schrittausführung übermittelt, welche den Roboter steuert. Da die Schrittausführung bereits existiert, konnte diese von Anfang an verwendet werden.

4 Aktionsplanung

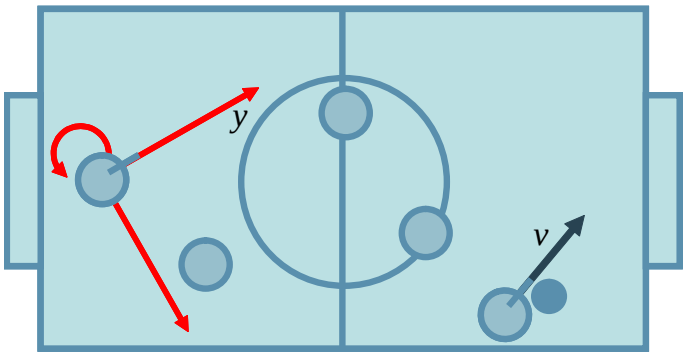

Abbildung 4.1: Das Ergebnis des Aktionsplaners im egozentrischem Koordinatensystem des Roboters r: Eine Zielpose z inklusive Ausrichtung und Geschwindigkeitsvektor v.

Die Aktionsplanung ist nur am Rande Thema dieser Diplomarbeit, da lediglich die Ausgabe benötigt wird. Sie wählt eine grobe Aktion aus, in diesem Fall den Torschuss. Andere mögliche Aktionen wären beispielsweise: ein Schuss in eine Richtung, in eine Richtung dribbeln, blocken oder passen. Als Eingabe könnte diese Ebene Strategieparameter wie zum Beispiel offensives oder defensives Spielverhalten, Torfarbe und weitere Parameter erhalten. Die Ausgabe der Aktionsplanung übergibt der nächsten Ebene eine egozentrische Zielpose $z = (z_x, z_y, z_\theta)$ mit Orientierung z_θ des Roboters, eine egozentrische Zielgeschwindigkeit v_z, wie in Abbildung 4.1 veranschaulicht sowie gegebenenfalls eine Ballaktion a, welche z.B. einen Ballschuss am Zielpunkt auslösen könnte. Egozentrisch bedeutet hier, dass alles aus Sicht des Roboters berechnet wird. Dies hat den Vorteil, dass keine Lokalisierung benötigt wird.

5 Pfadplanung

5.1 Einleitung

Die Aufgabe der Pfadplanungsebene ist das Ausweichen und Umlaufen von Hindernissen. Da der Fokus der Arbeit, wie bereits erwähnt, auf der Planung einzelner Schritte liegt, wurden die hier beschriebenen Verfahren zwar implementiert, doch liegen keine ausführlichen Ergebnisse von Experimenten und Vergleichen vor. Es wurden zwei mögliche Verfahren getestet. Zunächst wird auch hier ein Ansatz mit dem A*-Algorithmus diskutiert, danach wird in Abschnitt 5.3 ein Verfahren vorgestellt, das Hindernisse durch Festlegen von Alternativrouten durch Viapunkte umläuft. Das letztere Verfahren wurde auf dem realen Roboter implementiert und ein Nachweis der Machbarkeit gegeben.

5.2 2D-Pfadplanung mit A*-Algorithmus

Die Pfadplanung erhält als Eingabe die egozentrische Zielpose $z = (z_x, z_y, z_\theta)$ aus dem Aktionsplaner. Die Orientierung z_θ ist bei der 2D-Pfadplanung nicht von Bedeutung. Der Pfadplaner sucht einen möglichst optimalen Pfad im zweidimensionalen Raum $\{(x, y)\} \subseteq \mathbb{R}^2$ praktikabler Positionen zu einer Umgebung um die übergebene Zielposition z. Es genügt, die Umgebung der Zielposition zu erreichen, da die exakte Zielposition aufgrund der Diskretisierung unter Umständen nicht getroffen werden kann. Bei der Pfadplanung werden Hindernisse wie Mitspieler, Gegenspieler oder

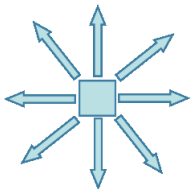

Abbildung 5.1: Neun gleich lange Aktionen des Pfadplaners. Die Ausgangsposition liegt in der Mitte. Die Aktionen zeigen von dieser auf neue Positionen.

gestürzte Roboter einkalkuliert. Allerdings wird weder die Orientierung des Roboters berücksichtigt, noch die roboterspezifische Dynamik und dessen Beschränkungen. Der Zustand des Pfadplaners beschreibt die betrachtete Position p, bestehend aus den Koordinaten x und y, in Bezug auf die ursprüngliche Ausgangsposition des Roboters, als der Algorithmus gestartet wurde. Die Aktionen der Pfadplanung sind Übergänge zu einer neuen Roboterposition. Es werden Übergänge mit einer festgelegten Schrittlänge und Winkelauflösung in alle Richtungen benötigt. Sowohl die Schrittlänge als auch die Winkelauflösung bestimmen die Genauigkeit sowie die Laufzeit des Algorithmus. Es wurde eine Winkelauflösung von neun Aktionen verwendet. Die Aktionsmenge ist in Abbildung 5.1 veranschaulicht dargestellt. Die Kostenfunktion $c(k, k')$ von einem Knoten k zu einem Folgeknoten k' wird definiert als

$$c(k, k') = 1 + \sum_{h \in H} |h - p| \tag{5.1}$$

mit der Menge H der Positionen der Hindernisse.

5.3 Viapunkte

Für einen einfachen Pfadplaner, der Hindernisse umläuft, kann das folgende Verfahren verwendet werden. Das Verfahren arbeitet wie in Abbildung 5.3 dargestellt: Zunächst wird eine Schrittfolge ohne Hindernisse mit dem Schrittplaner erzeugt, wie in 7.4 beschrieben. Danach wird jeder dieser Schritte auf Kollision mit einem Hindernis überprüft. Ein Schritt kollidiert mit einem Hindernis, wenn die Entfernung zwischen Schritt und Hindernis geringer als ein Schwellwert d_{min} ist. Dieser Schwellwert sollte so gewählt werden, dass er die Schwankungen in der Computer Vision ausgleicht und die Größe des Roboters berücksichtigt, so dass die Wahrscheinlichkeit einer realen Kollision vertretbar gering ist. Nachdem eine Kollision festgestellt wurde, werden Viapunkte definiert, indem auf der Achse orthogonal zur Schrittausrichtung des kollidierten Schrittes zwei Viapunkte in einer vorgegebenen Distanz d_{via} vom Hindernis aus in beiden Richtungen platziert werden. Im Prinzip kann $d_{via} = d_{min}$ gesetzt werden. Gegebenenfalls macht es jedoch Sinn, $d_{min} > d_{via}$ als Hysterese zu verwenden, um häufiges Umplanen zu vermeiden. Vom Startzustand aus werden zwei Schrittfolgen zu diesen Viapunkten berechnet, deren Zielausrichtung parallel zur Schrittausrichtung des kollidierten Schrittes ist. Die dort vom Schrittplanungsalgorithmus erreichte Geschwindigkeit und Ausrichtung, sind die Anfangsgeschwindigkeit und –ausrichtung des Pfades vom Viapunkt zum Ziel. Es muss die erreichte Ausrichtung anstatt der am Viapunkt vorgegebenen verwendet werden, da nicht sichergestellt werden kann, dass die vorgegebene Ausrichtung vom Approximator genau erreicht wird.

In dieser Implementierung wurde nur ein Hindernis berücksichtigt, da die verwendete

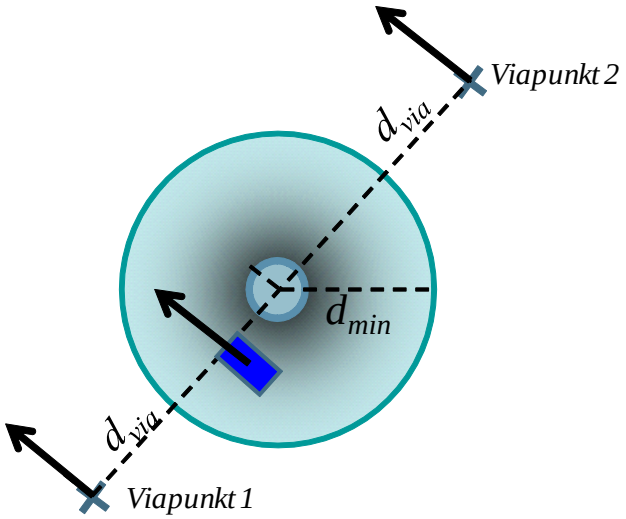

Abbildung 5.2: Zwei Viapunkte werden im Abstand d_{via} vom Hindernis gesetzt, wenn ein Schritt näher als d_{min} an einem Hindernis ist. Die Viapunkte haben die gleiche Ausrichtung wie der kritische Schritt.

Computervision effektiv nur ein Hindernis sicher erkennen kann. Dennoch lässt sich das Prinzip dieser Methode leicht auf beliebig viele Hindernisse ausweiten.

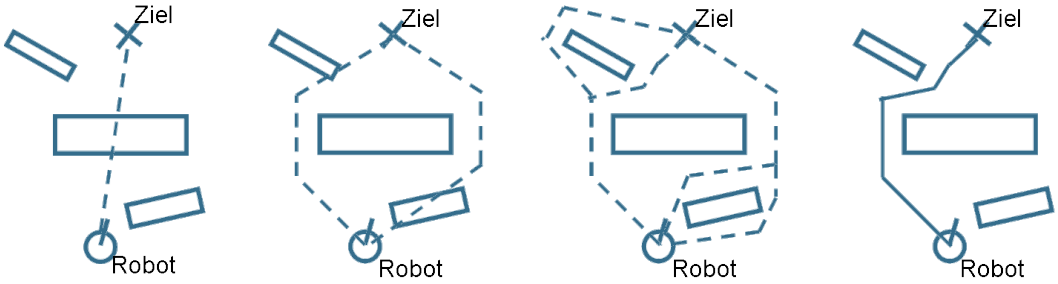

Abbildung 5.3: Veranschaulichung des Verfahrens der Viapunkte mit zwei Hindernissen. Zunächst wird ohne Betrachtung der Hindernisse geplant. Wird danach eine Kollision festgestellt, wird über Viapunkte neu geplant.

Da ein vollständiger Berechnungsdurchlauf des Verhaltens inklusive der Computervision innerhalb eines vorgegebenen Taktes von 12ms abgeschlossen sein muss, waren die Berechnungen in der verwendeten Implementierung zu zeitaufwändig. Deshalb wurde die Berechnung auf mehrere Rechenschritte aufgeteilt. Das Viapunkt-Verfahren, das ein Hindernis umläuft, benötigt im Kollisionsfall fünf Pfade. Die Berechnungen dieser fünf Pfade wurden jeweils auf einen Berechnungsdurchlauf aufgeteilt. Alle fünf Berechnungsdurchläufe wird ein neuer Viapunkt berechnet. Dies

geschieht innerhalb des Zeitraums eines einzelnen Schrittes. Als Steuerbefehl wird, solange kein neues Ergebnis vorliegt, der Gangsteuerungsvektor zum Erreichen des letzten berechneten Viapunktes abgefragt. Sollte das Verfahren keine Kollision berechnen, wird das Ziel als Viapunkt festgesetzt.

6 Trajektorienplanung

6.1 Einleitung

Die Trajektorienplanung dient dazu, eine schnell ablaufbare Trajektorie zu erstellen, die die Dynamik des Roboters berücksichtigt. Anhand dieser Trajektorie soll es möglich sein, Schritte effizient zu platzieren, sei es längst der Trajektorie oder bis zu einem kurzen Zwischenziel. Es wurden verschiedene Ansätze versucht, um dieses Problem zu lösen. Letztendlich kam jedoch keines dieser Verfahren zum Einsatz, da jedes schwerwiegende Probleme aufwies und die Approximation der Schrittplanung, wie in Kapitel 7 besprochen, gelöst wird.

A*-Algorithmus

Eine mögliche Herangehensweise an dieses Problem, ist die Verwendung eines sechsdimensionalen A*-Algorithmus. Dieser sucht die Trajektorie bis zu einem Zwischenziel, welches von der Pfadplanung übergeben wurde. Dabei wird die Geschwindigkeit des Roboters und dessen Orientierung berücksichtigt. Allerdings plant der Pfadplaner nur in zwei Dimensionen. Die Orientierung und Geschwindigkeit kann bei dem übergebenen Ziel nicht mit angegeben werden. Deshalb wird für die Geschwindigkeit an den Endpunkten der maximale Wert angenommen und die Orientierung anhand der Steigung des Pfades geschätzt. Diese Planung berücksichtigt sowohl die Hindernisse als auch die Roboterdynamik mit dessen Beschleunigungs- und Geschwindigkeitsbeschränkungen. Die Kostenfunktion $c(k, k')$ von einem Knoten k zu einem Folgeknoten k' ist identisch mit der Kostenfunktion der Pfadplanung und definiert sich als

$$c(k, k') = 1 + \sum_{h \in H} |h - p| \tag{6.1}$$

mit der Menge H der Positionen der Hindernisse. Der sechsdimensionale Zustand $(x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}) \in \mathbb{R}^6$ besteht aus der betrachteten dreidimensionalen Roboterpose sowie der dreidimensionalen Robotergeschwindigkeit, wobei x, y die kartesischen Koordinaten und θ die Orientierung repräsentieren. Als Aktionsraum wird eine diskrete Untermenge von zulässigen Beschleunigungen unter Einhaltung der Beschleunigungs- und Geschwindigkeitsbeschränkungen verwendet, siehe Abbildung 2.3. Auch wenn

feinere Aktionen genauere Ergebnisse erzeugen können, wurde aufgrund der Laufzeit Aktionen mit der maximalen Beschleunigungen implementiert. Der Algorithmus kann ein Ziel für die Schrittplanung inklusiv der zugehörigen Geschwindigkeit bestimmen (Abbildung 6.1).

Zur Verbesserung der Laufzeit, wurde die Suche lokal auf einen um den zweidimensionalen Pfad liegenden Bereich eingegrenzt. Der Ansatz scheitert dennoch an den zeitlichen Anforderungen, die für einen Großteil der Situationen nicht einzuhalten sind.

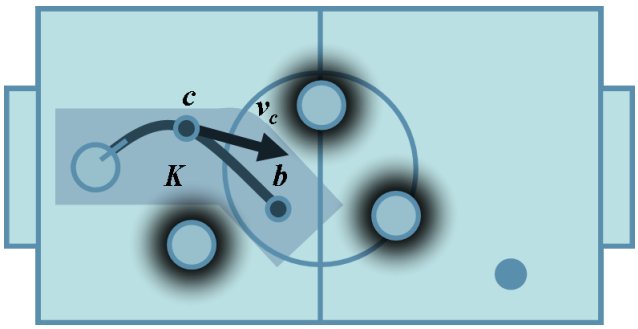

Abbildung 6.1: Ausgabe des Trajektorienplaners: Neues Zwischenziel c für die Schrittplanung inklusiv der zugehörigen Geschwindigkeit v_c.

Splines und Bézierkurven

Eine andere Idee ist das Einsetzen verschiedener Kurven. Jedoch ist die Einhaltung der Roboterdynamik problematisch. Alle geometrischen Kurven besitzen Parameter, die optimiert werden können und so die Roboterdynamik berücksichtigen können. In [1] wird ein Verfahren unter Verwendung zweier funktionaler Splines beschrieben. In dieser Veröffentlichung werden die Parameter iterativ optimiert, um eine Trajektorie mit größtmöglicher Geschwindigkeit unter Einhaltung der Geschwindigkeitsbegrenzungen zu erzeugen. Das Geschwindigkeitsprofil konnte anhand der Extrem- und Wendepunkte bestimmt werden. Ein anderer Kurventyp, der in [2] vorgeschlagen wurde, verwendet einen ähnlichen Algorithmus, jedoch wurden Bézierkurven anstelle von Splines eingesetzt. Allerdings wurden beide Verfahren für die Trajektorienplanung eines omnidirektionalen, holonomen Fahrzeugs entwickelt. Obwohl der verwendete Roboter einen omnidirektionalen Gang besitzt, kann er lediglich durch ein nicht-holonomes Fahrzeug approximiert werden, da das zulässige Geschwindigkeitsprofil durch eine schmale Raute definiert ist. Ein Herausschneiden des holonomen Gangbereiches würde die Geschwindigkeit des Roboters zu stark einschränken, um optimale Trajektorien zu erzeugen. Ein weiteres Problem ist eine Konsequenz aus der holonomen Annahme. Da das System häufig neu planen soll und der Roboter seine Drehung beschleunigen muss, wird eine einstellbare Anfangskrümmung der

Trajektorie benötigt. Zudem ist eine unabhängige Optimierung der Bahnen nicht möglich, da keine Abstraktion des Schrittmodells existiert und somit nicht die Güte der Bahn bestimmt werden kann, ohne die Schritte vorher zu verteilen. Jedoch hat sich das Verteilen der Schritte als aufwendig erwiesen, da alle Schritte voneinander abhängen. Eine funktionierende Möglichkeit, die Schritte zu verteilen, ist mit Hilfe eines A*-Algorithmus. Diesem gelingt es innerhalb von 10 ms, an einem gegebenen Pfad entlang gültige, ablaufbare Schritte zu verteilen oder zurückzugeben, dass dies nicht möglich ist.

6.2 Klothoiden

Eine weitere interessante Kurvenart sind Klothoiden (eng. Clothoids). Die Klothoiden besitzen eine besondere Eigenschaft. Ihr Krümmungsverlauf nimmt linear zu und dient einer ruckfreien Fahrdynamik. Deswegen wird diese Kurvenart bei der Planung von Autobahnen oder Hochgeschwindkeitszuglinien eingesetzt. Zudem wurden Arbeiten veröffentlicht, in denen gezeigt wurde, dass Menschen häufig Klothoiden ähnliche Bahnen laufen. Es ist bisher nicht möglich, Klothoiden effizient für alle möglichen Kombinationen aus Startpose mit Startkrümmung und Endpose mit Endkrümmung zu berechnen. Lediglich unter der Annahme, dass Start- und Endkrümmung 0 seien, kann eine Lösung gefunden werden. Da allerdings in dem vorgesehenen Einsatzgebiet eine häufige Neuplanung aus mehreren Gründen unerlässlich ist, ist die Annahme der Startkrümmung nicht möglich.

6.3 Zusammenfassung

Es wurden vielerlei Kurvenarten betrachtet, die allesamt schwerwiegende Probleme aufwiesen. Zum einen ist stets die Laufzeit problematisch, zum anderen waren die Ansätze nach aktuellem Stand der Wissenschaft überhaupt nicht lösbar. Somit musste eine Möglichkeit gefunden werden, Schritte ohne eine vorberechnete Kurve effizient berechnen zu können.

7 Schrittplanung

7.1 Einleitung

Der Fokus dieser Arbeit liegt auf der Planung einzelner Schritte des Roboters um aus dem Laufen den Ball schießen zu können. In dem folgenden Kapitel wird die Schrittplanung beschrieben und analysiert. Die Planung erfolgt mit Hilfe eines A*-Algorithmus, welcher im Anschluss daran in Abschnitt 7.3 erläutert wird. Dieser sucht optimale Pfade zum Ball im Raum der möglichen Steuerungsvektoren. Ein Schrittmodell ermöglicht die Schätzung der Schrittpositionen aus gegebenen Gangsteuerungsvektoren. In Abschnitt 7.2 werden zwei Schrittmodelle hergeleitet und miteinander verglichen. Es stellt sich heraus, dass die Performance der Schrittplanung (A*-Algorithmus) nicht ausreichend ist und den Geschwindigkeitsanforderungen eines Livesystems nicht gerecht werden kann. Deshalb werden im darauf folgenden Abschnitt Pfade offline vorberechnet, die in Abschnitt 7.5 mit unterschiedlichen Funktionsapproximatoren angenähert werden. Die verschiedenen Approximatoren werden abschließend miteinander verglichen. In Abschnitt 7.5.5 wird auf die Implementierung in der verwendeten Software eingegangen.

7.2 Schrittmodell

7.2.1 Einleitung

Auf zentralem Mustergenerator basierende Methoden (CPG) und auf inverse Kinematik basierende Methoden sind zwei erfolgreiche Herangehensweisen, um ein kontrolliertes, dynamisches Gehen für zweibeinige, humanoide Roboter zu implementieren, auch wenn sich diese stark in ihren Kernelementen unterscheiden. Auf zentralem Mustergenerator basierende Methoden [13], auch Limit Cycle Walking genannt [14], generieren einen abstrakten, periodischen Signalfluss, welcher in Motorbefehle übersetzt wird. Daraus resultiert eine rhythmische Gewichtsverlagerung und Beinschwungbewegung. Inverse Kinematik basierte Lösungen [15, 16, 19] berechnen Trajektorien in kartesischen Koordinaten für wichtige Körperteile, wie dem Becken

und den Füßen, vor. Diese Trajektorien werden in Motorbefehle umgewandelt, indem die inverse Kinematik, mit den gegebenen Trajektorien als Beschränkung, gelöst wird. Im Fall der inversen Kinematik basierten Methode sind die Schrittpositionen im vornherein bekannt: Sie werden durch den Schnitt der Fußtrajektorie mit dem Boden ermittelt. Jedoch sind im Fall der zentralen mustergenerierten Methode die Schrittpositionen nicht zwangsläufig erhältlich, da sie nur indirekte Ergebnisse der Amplituden und Frequenzen von abstrakten Signalmustern sind. Das Ziel ist es, die Schrittpositionen von einem zentralen mustergenerierten Gehen vorherzusagen und diese Vorhersagen für die Umsetzung einer genaueren Schrittplanung zu benutzen. In dieser Arbeit werden zwei verschiedene Ansätze zur Schätzung der Schrittpositionen präsentiert und ihre Leistung in experimentellen Versuchen verglichen.

Nach einem Überblick über verwandte Arbeiten wird eine kurze Einführung in die verwendete Methode in Abschnitt 7.2.2 gegeben. Dann wird eine Übersicht über den Schrittvorhersage-Algorithmus in Abschnitt 7.2.3 präsentiert, die dann zu den ausführlichen Beschreibungen der zwei unterschiedlichen Ansätze, die umgesetzt wurden, führt: ein vorwärtskinematischer Ansatz in Abschnitt 7.2.3 und ein Motion Capture Ansatz in Abschnitt 7.2.3.

7.2.2 Dynamisches Gehen auf der Basis eines zentralen Mustergenerators

In den folgenden Kapiteln werden die grundlegenden Konzepte der auf einem zentralen Mustergenerator basierenden Ganggenerationsmethode vereinfacht eingeführt. Nun wird sich auf die Kernelemente konzentriert, die wichtig für das Verständnis des Schrittvorhersagemodells sind.

Beinschnittstelle

Die Beinschnittstelle ist eine untere Abstraktionsebene, die eine intuitive Kontrolle über ein Bein mit drei Parametern ermöglicht. Der Beinwinkel Θ_{Leg} definiert den Winkel des Beines in Bezug auf den Rumpf, der Fußwinkel Θ_{Foot} definiert die Neigung des Fußes in Bezug auf die Traversalebene und die Beinverlängerung η definiert den Abstand zwischen dem Fuß und dem Rumpf (Abbildung 7.1). Die Ausgaben der Beinschnittstelle sind Gelenkwinkel für die Hüfte, Knie und Fußgelenk. Die Beinschnittstelle ermöglicht die unabhängige Kontrolle der drei Parameter und kapselt die Berechnung der koordinierten (auf einander abgestimmten) Gelenkwinkel.

$$L(\Theta_{Leg}, \Theta_{Foot}, \eta) = (\Theta_{Hip}, \Theta_{Knee}, \Theta_{Ankle}) \tag{7.1}$$

Die Beine können in Roll-Nick-Gier-Richtung geneigt werden. Wie in Abbildung 7.1 (rechts) dargestellt, bewegt ein positiver Wert des Θ^{roll}_{Leg} Parameters den rechten

Fuß in x Richtung vom Rumpf aus nach außen, ein positiver Wert des Θ_{Leg}^{pitch} den rechten Fuß vorwärts in y Richtung und ein positiver Wert des Θ_{Leg}^{yaw} dreht den Fuß im Uhrzeigersinn. Am Wichtigsten ist, dass der Fuß um seine eigene Achse gedreht wird. Für den linken Fuß werden die positiven Richtungen an der sagittalen Ebene gespiegelt. Die drei Richtungskomponenten können unabhängig voneinander kontrolliert werden. Weiter wird dies in [13] veranschaulicht.

Der zentrale Mustergenerator (CPG)

Der CPG erzeugt Muster von rhythmischen Aktivierungen aus einer periodischen internen Uhr, genannt Gangphase $-\pi \leq \phi < \pi$. Die Muster kodieren die Wellenform der Parameter der Beinschnittstelle. Insbesondere die Beinverlängerung wird mit einer sinusförmigen Funktion aktiviert, wohingegen die Phase des linken Beines um π in Bezug auf das rechte Bein verschoben ist (Abb. 7.2, oben links).

$$P_w = sin(\phi) \tag{7.2}$$

Die entgegengesetzte Verkürzung und Verlängerung der Beine verursacht eine rhythmische, seitliche Verlagerung des Körpergewichts, die abwechselnd ein Bein aus seiner Unterstützungspflicht entlässt. Dieses Bein kann geschwungen werden. In Abstimmung mit dem Signal der Beinverlängerung P_w, generiert das CPG eine zweite Aktivierung, um das freie Bein zu schwingen.

$$P_s = sin(\phi - \frac{\pi}{2}), \quad -\pi \leq \phi < 0 \tag{7.3}$$

$$P_s = 1 - \frac{\phi}{\pi}, \quad 0 \leq \phi < \pi \tag{7.4}$$

Wie in Abbildung 7.2 oben rechts gezeigt, wird das Bein mit einer sinusförmigen Bewegung vorwärts geschwungen und mit einer linearen Bewegung in der Unterstüt-

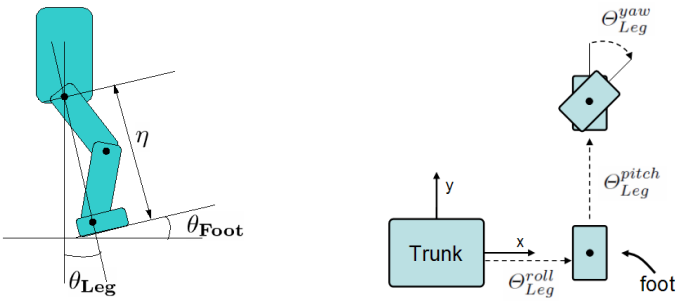

Abbildung 7.1: Die Beinschnittstelle erlaubt die unabhängige Kontrolle der drei abstrakten Parameter: den BeinwinkelΘ_{Leg}, den Θ_{Foot}, und die Beinverlängrung η (left). Das Bein kann in Roll-Nick-Gier-Winkel geneigt werden (rechts).

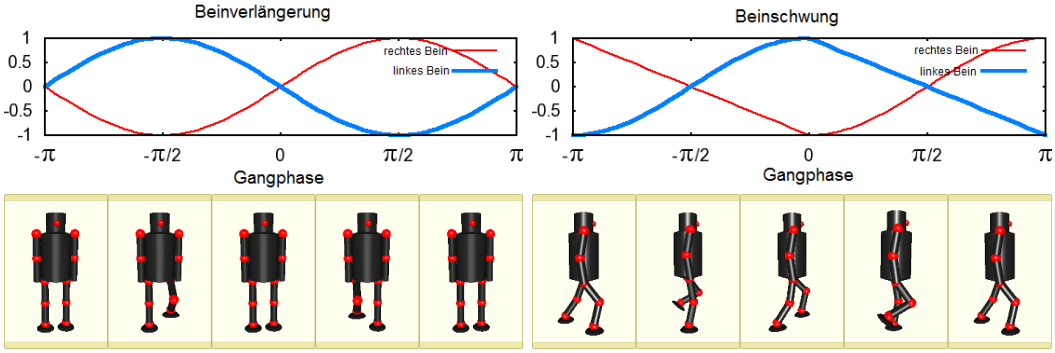

Abbildung 7.2: Die Wellenform des zentralen Mustergenerators kodiert die Muster der Beinverlängerung (oben links) und des Beinschwungs (oben rechts). Die untere Zeile veranschaulicht die Posen des Roboters in der entsprechenden Gangphase.

zungsphase zurückgezogen. Der Standfußwechsel wird voraussichtlich in der Gangphase $\phi = 0$ von rechts nach links und in der Gangphase $\phi = -\pi$ von links nach rechts eintreten.

Omnidirektionale Gangsteuerung

Laufrichtung und Schrittgröße werden kontrolliert durch die Modulation der Amplitude der Beinschwungsaktivierung P_s mit einem Gangsteuerungsvektor $g \in [-1, 1]^3$ und der Anwendung dieser Modulation auf die Roll-Nick-Gier-Komponente des Beinwinkels Θ_{Leg}. Omnidirektionales Gehen wird durch die gleichzeitige Anwendung des Schwungsignals in allen drei Richtungen mit verschiedenen Intensitäten erreicht. Zum Beispiel resultiert eine Mischung der Nick- und Gier-Komponenten in einem gebogenen Gang vorwärts, wo die Intensität der Gier-Komponente die Krümmung des Pfades bestimmt. Die modulierten Signale werden dann durch einen Konfigurationsvektor $c \in \mathbb{R}^3$ transformiert, der eine Abbildung des CPG Signal Raums in den Raum der Beinwinkel, ausgedrückt in Radiant, ist. c kann zur Anpassung der gleichen CPG Muster auf Robotern von verschiedenen Größen benutzt werden, zum Beispiel für die KidSize und die TeenSize Klasse und zur Feinabstimmung der einzelnen Roboter. Zusammenfassend beschreiben diese Gleichungen die Generierung der Gang-Trajektorie:

$$\Theta_{Leg}^{roll} = P_s \cdot g_x \cdot c_x + |g_x| \cdot c_x \tag{7.5}$$

$$\Theta_{Leg}^{pitch} = P_s \cdot g_y \cdot c_y \tag{7.6}$$

$$\Theta_{Leg}^{yaw} = P_s \cdot g_z \cdot c_z + |g_z| \cdot c_z \tag{7.7}$$

Am Ende der Gangkontrollkette konvertiert die Beinschnittstelle die Beinwinkel und Erweiterungen in Gelenkwinkel, wie in Abbildung 7.3 dargestellt.

Insbesondere die Gleichung der sagittalen Richtung (7.6) unterscheidet sich von denen der lateralen und Gier-Richtungen (7.5, 7.7). In sagittaler Richtung schwingen die Beine vollständig von ganz vorne bis ganz hinten. Positive und negative Beinwinkel sind gleichfalls erlaubt. In lateraler Richtung jedoch würden die Beine kollidieren. Zur Vermeidung negativer Beinwinkel wird ein positiver Wert auf die Roll-Komponente der Beinwinkel proportional zu der seitlichen Komponente des Gangsteuerungsvektors hinzuaddiert, um zu bewirken, dass sich die Beine spreizen, wenn man in die laterale Richtung geht. Als Ergebnis treten zwei verschiedene Schrittgrößen auf: ein langer Schritt, wenn das führende Bein in die Richtung, in der sich der Roboter bewegt, geschwungen wird und ein kleiner Schritt, wenn das andere Bein nachgezogen wird, um das führende Bein bei einem Beinwinkel nahe null zu treffen. Das gleiche gilt für die Gier-Richtung.

Abbildung 7.3: Die Gangtrajektorien-Generationskette. Der CPG ist immer aktiv und erzeugt periodische Aktivierungssignale getaktet durch die Gangphase ϕ. Nach der Modulation durch den Gangsteuerungsvektor g werden die Signale in den Beinwinkel-Raum mit der Konfiguration c transferiert. Die Beinschnittstelle konvertiert die Beinwinkel in Gelenkwinkel, die an den Roboter weitergegeben werden.

7.2.3 Das Schrittvorhersage-Modell

Für die Durchführung eines Schrittplanungsalgorithmus wird ein Vorwärtsmodell

$$F(g_x, g_y, g_z, \phi) = (p_x, p_y, p_\theta) \tag{7.8}$$

benötigt, das einen Gangsteuerungsvektor g auf eine erwartete Schrittposition und Ausrichtung $p \in \mathbb{R} \times \mathbb{R} \times [-\pi, \pi]$ in kartesischen Koordinaten abbildet. p wird als die Position und Ausrichtung des Schrittes in dem lokalen Koordinatensystem des Standfußes definiert. Wie im vorherigen Abschnitt skizziert, kann ein Gangsteuerungsvektor g zwei verschiedene Schrittgrößen produzieren. Die Gangphase ϕ muss als Parameter zum Entfernen der Zweideutigkeit mit einbezogen werden. Der Wert von ϕ wird bestimmt durch das Merken des Standfußes und dem Wissen, in welcher Gangphase der nächste Schritt auftreten wird.

Es wurden zwei Strategien entwickelt, um diese Abbildung zu erhalten. Als einen analytischen Ansatz wurde ein kinematisches Modell des Roboters zur Berechnung der Vorwärtskinematik aus den gegebenen Gelenkwinkeln benutzt. Alternativ wurden Trainingsdaten mit einer Motion-Capture-Anlage gesammelt und lineare Regression benutzt, um die Abbildung F zu erlernen. Beide Ansätze sind ausführlich in den folgenden Abschnitten beschrieben.

Kinematisches Modell

Das kinematische Modell erfordert ein genaues Skelett des Roboters, das aus den CAD-Konstruktionsplänen gewonnen wurde. Die Vorhersagen werden durch die Anwendung der Gelenkwinkel zum Zeitpunkt des Schrittes auf das kinematische Modell und die Berechnung der Position und der Ausrichtung des Schwungfußes relativ zum Standfuß im kartesischen Raum mit der Denavit-Hartenberg Transformation berechnet [27] (7.5). Um die Gelenkwinkel zum Zeitpunkt des Schrittes für einen spezifischen Gangsteuerungsvektor g zu verstehen, wird die entsprechende Gangphase ϕ (0 oder $-\pi$) eingesetzt und die komplette Gangtrajektorien-Erstellungskette ausgeführt. Der CPG gibt die gleichen Signale aus, die er zum Zeitpunkt des Schrittes erzeugen würde. Mit der Konfiguration c eines spezifischen Roboters und des Gangsteuerungsvektor g werden die gewünschten Gelenkwinkel (Abbildung 7.3) erhalten.

Durch ihre analytische Natur können die Vorhersagen sehr effizient berechnet werden. Jedoch sind einige Fehlerquellen unvermeidlich. Ungenauigkeiten im Skelett können nicht vollständig vermieden werden sowie die Tatsache, dass der Roboter nicht perfekt die befohlenen Gelenkwinkel ausführt. Außerdem gibt es immer mechanischen Verschleiß, Spiel in den Zahnrädern und unerwünschte Elastizitäten, die ein Abweichen des physikalischen Systems von der Theorie verursachen. Deswegen wurde ein alternativer Ansatz gewählt und diese Abbildung aus den von dem tatsächlichen physikalischen System gesammelten Daten gelernt. Dieser Ansatz wird im nächsten Abschnitt präsentiert.

Maschinell gelerntes Modell

Als ein alternatives Vorgehen wurden Daten mit einem Motion-Capture-System gesammelt, um die Schritte zu lernen, wie sie tatsächlich passieren. Zunächst wurden zwei KidSize-Roboter und später für die Schrittplanung der TeenSize-Roboter Dynaped, wie in Abbildung 7.4 gezeigt, mit reflektierenden Markierungen in Dreier- oder Vierergruppen auf dem Kopf, der Hüfte und den Füßen ausgestattet. Mit allen Robotern wurden ungefähr fünf Minuten mit mehr oder weniger zufälligen Laufge-

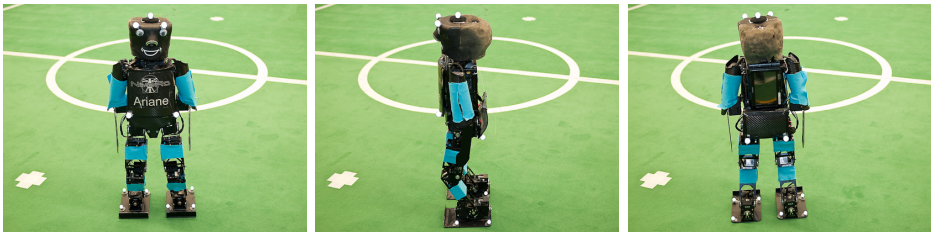

Abbildung 7.4: Gruppen aus reflektierenden Markern wurden zur Identifikation des Kopfes, der Hüfte und der beiden Füße verwendet.

schwindigkeiten und Richtungen aufgezeichnet. Mit diesen Daten wird versucht, den gesamten Gangsteuerungsraum zu untersuchen.

Die Ausgabe der Motion-Capture-Anlage besteht aus Trajektorien der reflektierenden Marker, die mit einer Aufnahme des Gangsteuerungsvektors synchronisiert wurden. Bei der Datennachverarbeitung wurde ein nicht starres Skelett in die Punktwolke der Marker eingebettet, indem die Schwerpunkte jeder Markergruppe berechnet wurden und der Kopf mit der Hüfte und die Hüfte mit den Füßen verbunden wurden. Das Skelett wurde für die weitere Verarbeitung genutzt. Aus der Markergruppe der Hüfte wurde die Orientierung des Roboters in Bezug auf die globale vertikale Achse berechnet. Die Ausrichtung beschreibt, in welche Richtung der Roboter in dem Welt-Koordinatensystem ausgerichtet ist. Dies muss aber nicht notwendigerweise mit der Laufrichtung übereinstimmen. Mit den Markergruppen der Füße wurde die Ausrichtung der beiden Füße relativ zu der globalen Ausrichtung des Roboters

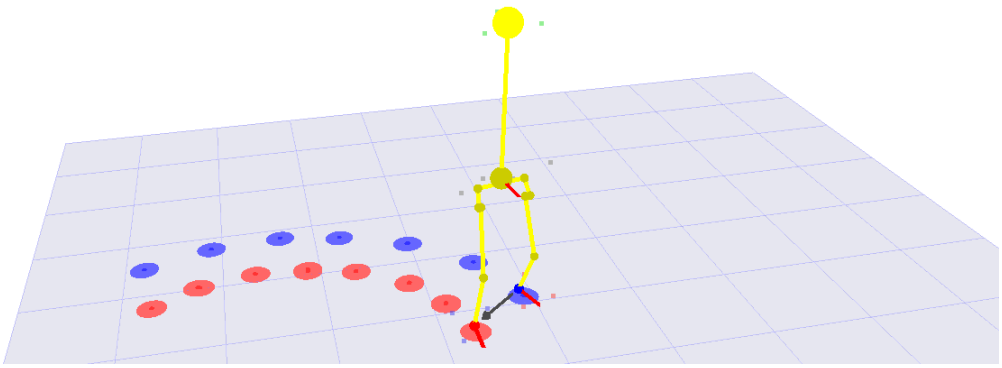

Abbildung 7.5: Visualisierung der von der Motion-Capture-Anlage aufgenommenen Daten: Die Markerwolke, das kinematische Modell, welche in die Wolke eingebettet wurde, die Orientierung des Roboters und der Füße und einige der extrahierten Schrittpositionen auf dem Boden. Der Pfeil zwischen den Füßen veranschaulicht den Schrittvektor, der vom kinematischen Modell extrahiert wurde.

berechnet.

Zur Herstellung der Trainingsdaten mussten einzelne Schritte identifiziert werden. Die Punktfüße des vereinfachten Skelettes werden genutzt, um zwei Merkmale zu extrahieren: Die Höhe der Füße h_l und h_r und die Geschwindigkeiten der Füße v_l und v_r werden aus zwei aufeinanderfolgenden Aufnahmen berechnet. Ein Schritt wird erkannt, wenn die Füße etwa die gleiche Höhe und die gleiche Geschwindigkeit haben:

$$|h_l - h_r| + |v_l - v_r| < 0.01\,. \tag{7.9}$$

Abbildung 7.5 zeigt eine Visualisierung der Markerwolke, das kinematische Modell, welches in der Wolke eingebettet ist, die Ausrichtung des Rumpfes und der Füße und einige der extrahierten Schrittpositionen auf dem Boden. Insgesamt wurden rund 3000 Schritte und passende Gangsteuerungsvektoren identifiziert.

Wie in Abschnitt 7.2.2 erwähnt, implementiert die Beinschnittstelle eine unabhängige Kontrolle der Fußposition und Ausrichtung. Dies ermöglicht die Annahme, dass statt der dreidimensionalen Funktion F (7.8) drei unabhängige eindimensionale Funktionen gelernt werden können. Jedoch ist, wenn ein Schritt ausgeführt wird, der feste Bezugspunkt der Standfuß und nicht das Koordinatensystem des Rumpfes. Die g_z Komponente des Gangsteuerungsverktors wendet eine Rotation auf die Schrittposition um einen Winkel α an und folglich hängen p_x und p_y von g_z ab. Dies wird in Abbildung 7.6 (a) gezeigt.

Zur Bewältigung dieses Problems wird eine Schrittposition $q = (q_x, q_y, \alpha)$ im Koordinatensystem des Rumpfes eingeführt (Abbildung 7.6 (b)). q_x und q_y sind definiert als die Entfernungen zwischen den Füßen in x und y Richtung und α ist die Ausrichtung des Schwungfußes in Bezug auf den Rumpf. In diesem Koordinatensystem hängt q_x nur von g_x, q_y nur von g_y und α nur von g_z ab. Das sind die Zuordnungen, die gelernt werden. Es werden (q_x, q_y, α) von den identifizierten Schritten aus den Motion-Capture-Daten erhalten. Das Koordinatensystem des Rumpfes, aus der Markergruppe der Hüfte bekannt, wird berechnet (q_x, q_y) aus der Differenz zwischen den Fußkoordinaten des eingebetteten Skeletts. α ist gleich der Orientierung des Schwungfußes, die aus der Markergruppe des Fußes berechnet wurde. Die Ausrichtung des Standfußes und die des Schwungfußes werden für symmetrisch genug befunden, so dass sie nicht getrennt modelliert werden müssen.

Die rumpforientierte Schrittposition q wird dann zur standfußorientierten Schrittposition p mit einer Schritttransformationsfunktion T, wie in Abbildung 7.6 (c) und (d) dargestellt, umgewandelt. (q_x, q_y) wird im Koordinatensystem des Standfußes um den Winkel α gedreht. Dann rotiert der Schwungfuß um den Winkel α um seinen

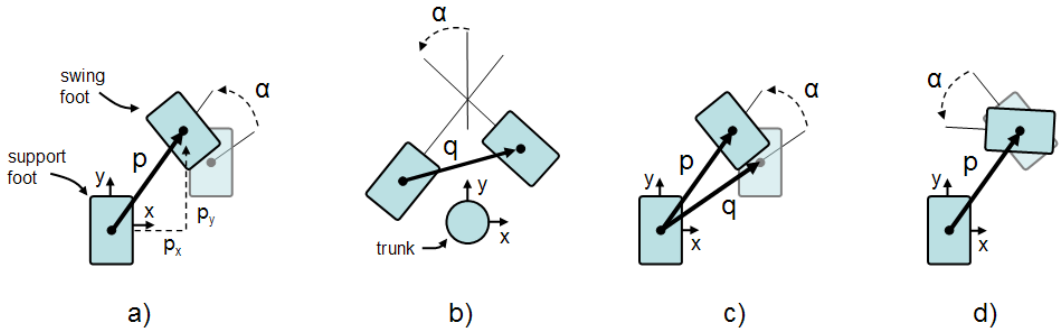

Abbildung 7.6: Die g_z-Komponente des Gangsteuerungsvektors rotiert die Schrittposition mit einem Winkel α um den Standfuß und beeinflusst p_x und p_y (a). Jedoch ist der Schritt q im Koordinatensystem des Rumpfes unabhängig von der Rotation (b). q wird auf die Standfuß bezogene Schrittposition p abgebildet, mit einer Rotation um den Winkel α (c) und einer anschließenden Rotation des Schwungfußes wieder um den Winkel α (d).

eigenen Ursprung. Die Transformationsfunktion T ist gegeben durch:

$$(p_x, p_y) = (q_x, q_y) \cdot R(\alpha) \tag{7.10}$$

$$p_\theta = 2\alpha \tag{7.11}$$

wobei R eine Rotationsmatrix bezeichnet.

Darüber hinaus wurde eine weitere Vereinfachung der Lernaufgabe implementiert. Abbildung 7.7 zeigt eine Aufsplittung der Funktion F. Ist g und ϕ als Eingabe gegeben, kann die Gangsteuerungskette (Abbildung 7.3) zur Berechnung der Beinwinkel zum Zeitpunkt des nächsten Schrittes verwendet werden. Die Schrittfunktion S ist der aktuelle, physische Schritt. Sie bildet die Beinwinkel auf eine Schrittposition q im Koordinatensystem des Rumpfes ab. Die Schritttransformation T übersetzt q zur Schrittposition p im Koordinatensystem des Standfußes. Nur S muss gelernt werden und kann durch drei einfache, unabhängige und eindimensionale Funktionen approximiert werden:

$$q_x = q_x(\Theta^{roll}_{Leg}), \tag{7.12}$$

$$q_y = q_y(\Theta^{pitch}_{Leg}), \tag{7.13}$$

$$\alpha = \alpha(\Theta^{yaw}_{Leg}). \tag{7.14}$$

Diese Abkürzung ermöglicht auch die Nutzung verschiedener Roboterkonfigurationen durch einfaches Austauschen dieser Konfiguration in der Gangsteuerungskette. Eine bessere Anpassung dieses Algorithmus auf verschiedenen Individuen des gleichen Robotertyps wird erwartet.

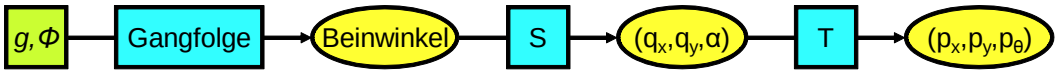

Abbildung 7.7: Eine Zerlegung der Funktion F. Gegeben g und ϕ, die Gangsteuerungskette berechnet die Beinwinkel im Moment des nächsten Schrittes. Der physikalische Schritt S bildet die Beinwinkel auf eine Schrittposition q im Koordinatensystem des Roboters ab. Die Schritttransformation T übersetzt q in den Schritt p im Koordinatensystem des Standfußes.

7.2.4 Experimentelle Ergebnisse

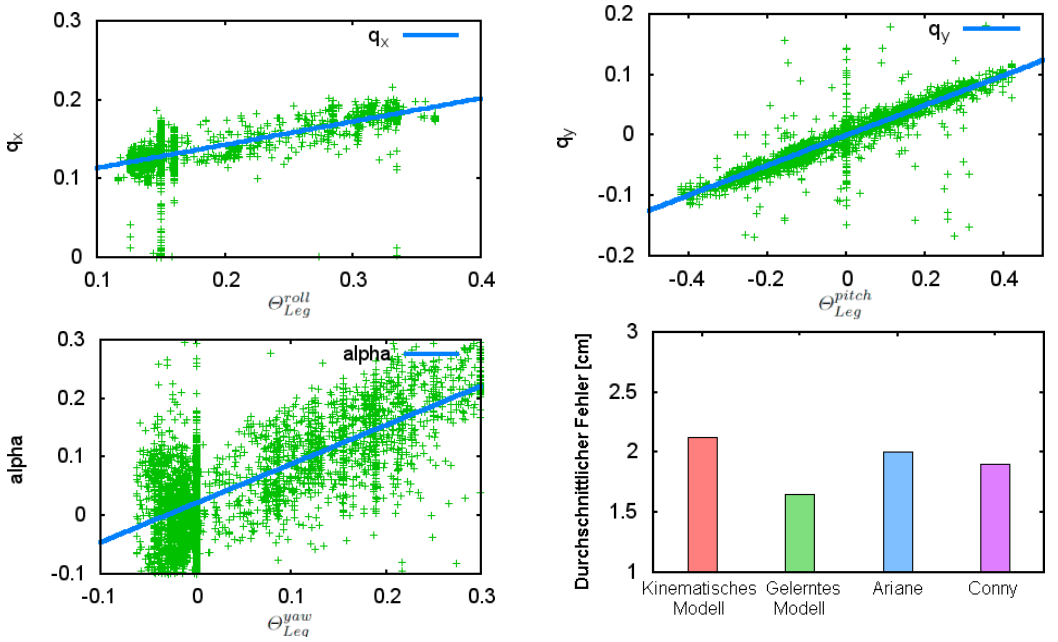

Abbildung 7.8: Darstellung der gesammelten Daten und der approximierten Funktionen für q_x (oben links), q_y (oben rechts), and α (unten links). Alle drei Beziehungen zeigen einen starken linearen Charakter und können einfach mit einer linearen Funktion approximiert werden. Die Auswertung der Vorhersagegenauigkeiten des analytischen Modells, des maschinell gelernten Modells mit Kreuzvalidierung, Arianes Modell angewendet auf Conny und Connys Modell angewendet auf Ariane werden in dem Säulendiagramm unten rechts dargestellt.

Abbildung 7.8 zeigt die Daten und die approximierten Funktionen für q_x (7.12), q_y (7.13), und α (7.14). Alle drei Beziehungen zeigen einen starken linearen Charakter und können in der einfachsten Art problemlos mit linearen Funktionen approximiert werden. Selbstverständlich würde man die Abbildung von Beinwinkel nach

Schrittweite als sinusförmig erwarten. Die geeignetste Erklärung für den offensichtlich linearen Zusammenhang ist, dass unsere Roboter relativ kleine Schritte machen und solange ihr Argument nahe Null ist, können Sinusfunktionen gut mit linearen Funktionen approximiert werden. Die Funktionsabbildung Θ_{Leg}^{yaw} nach α ist beinahe die Identität. Die geringe Abweichung von der Identität muss von dem Fehler des physikalischen Systems herrühren, wenn ein Schritt ausgeführt wird. Mit dem Lernen dieser Abweichung, kann eine Verbesserung der Schrittvorhersage erwartet werden.

Mit den Motion-Capture-Daten wurde die Performance des vorwärtskinematischen Ansatzes und des maschinell gelernten Ansatzes bewertet. Der Fehler einer einzelnen Vorhersage wurde mit der euklidischen Distanz zwischen dem vorhergesagten und dem realen Schritt aus den Motion-Capture-Daten gemessen. In Abbildung 8 unten rechts werden die Zahlen des durchschnittlichen Fehlers gemessen und in vier unterschiedlichen Experimenten präsentiert. Das vorwärtskinematische Modell wurde auf dem kompletten Datensatz von ungefähr 3000 Schritten ausgewertet. Die Leistung des maschinell gelernten Ansatzes wurde mit einem vierfachen Kreuzvalidierungsverfahren auf dem kompletten Datensatz gemessen. Zusätzlich wurde eine Kreuzvalidierung zwischen zwei Robotern ausgeführt: Ariane und Conny. Ein Modell wurde für jeden Roboter mit 500 zufällig ausgewählten Schritten aus dem eigenen Datensatz trainiert. Mit der dazugehörigen roboterspezifischen Konfiguration c wurden beide Modelle ausschließlich auf dem Schrittsatz des anderen Roboters getestet.

Alle trainierten Modelle übertreffen den vorwärtskinematischen Ansatz. Die besten Vorhersagen wurden von dem, auf den Gesamtdaten trainierten Modell, mit einer Genauigkeit von ungefähr 1,6 cm erreicht. Die durchschnittlichen Fehler der roboterindividuellen Kreuzvalidierungen sind schlechter als das Modell, welches auf den gesamten Schrittdaten trainiert wurde, aber übertreffen immer noch das kinematische Modell, auch wenn sie nur auf wenigen Schrittdaten trainiert wurden. Das zeigt, dass die Vorhersagen robust und zwischen den Roboter desselben Typs übertragbar sind, ungeachtet der Unterschiede in der Gangkonfiguration. Die Unterschiede in der Genauigkeit der beiden Roboterexperimente können durch die Unterschiede in den individuellen Gehstilen begründet werden. Da die Arme näher am Körper sind, bevorzugt Ariane kleinere Schritte und hat eine deutlich seitliche Schwäche verglichen mit Conny, die einen breiteren Bereich von Schrittgrößen abdeckt und die daher ein besseres Trainingsset erzeugt.

Um die Vorhersagegenauigkeit zu verbessern, wurden weitere Möglichkeiten untersucht. Es wurde überlegt, die Unabhängigkeitsannahme fallenzulassen und ein nichtlineares Regressionsverfahren zu nutzen um die Funktion

$$f(\Theta_{Leg}^{roll}, \Theta_{Leg}^{pitch}, \Theta_{Leg}^{yaw}) = (q_x, q_y, \alpha) \tag{7.15}$$

in einem Durchlauf zu lernen. Damit dieser Versuch erfolgreich sein kann, muss eine Abhängigkeit zwischen den drei Parametern und jeder der Ausgabedimensionen

	Δq_x	Δq_y	$\Delta\alpha$
Θ_{Leg}^{roll}	0.022	0.043	0.037
Θ_{Leg}^{pitch}	0.070	0.028	0.020
Θ_{Leg}^{yaw}	0.241	0.013	0.001

Tabelle 7.1: Korrelationskoeffizienten zwischen den Beinwinkeln in Roll-, Nick- und Gierrichtung sowie den Abweichungen der Schrittpositionen q im Koordinatensystem des Rumpfes.

bestehen. Es wurde die Korrelation zwischen den Beinwinkeln, die vom Gang erzeugt werden, und der Abweichung Δq untersucht, welche die Differenz zwischen dem egozentrischen Schritt q und dem erwarteten Schritt ist, der von dem linearen Approximator vorhergesagt wurde. Die Tabelle 7.1 enthält die Korrelationskoeffizienten. Abgesehen von einem kleinen Einfluss von Θ_{Leg}^{yaw} auf q_x, konnte keine nennenswerte Korrelation zwischen den Parametern und der Schrittabweichung identifiziert werden. Folglich hat der lineare Ansatz das Lernproblem ausgereizt. Es kann nicht erwartet werden, dass ein komplizierter Ansatz deutlich bessere Ergebnisse auf den gleichen Daten erzeugt.

7.3 Schrittplanung mit A*-Algorithmus

Die Schrittplanungsebene soll einzelne Schritte von der aktuellen Roboterpose zum Ball planen, so dass dieser ohne vorheriges Ausrichten mit dem linken Fuß bewegt wird. Die Pfade sollen die Dynamik des Roboters durch Verwendung des Schrittmodells und unter Einhaltung der Geschwindigkeits- und Beschleunigungsbeschränkungen berücksichtigen, damit ausschließlich gültige, ablaufbare Schrittfolgen entstehen. Diese Ebene wird mit Hilfe des A*-Algorithmus realisiert. Sie muss im Raum der möglichen Gangsteuerungsvektoren planen, da nur der Vektor beeinflusst werden kann. Die Auswirkungen dieser Aktionen finden aber im kartesischen Raum statt, so dass das Schrittmodell zur Vorhersage verwendet werden muss.

Definition

Wie in den Grundlagen beschrieben, definiert sich eine Instanz des A*-Algorithmus (M, s, Z, A, t, c, h) durch die Zustandsmenge M, den Startzustand s, die Zielmenge Z, die Aktionsmenge A, die Transferfunktion t, die Kostenfunktion c und die Heuristik h. In diesem Abschnitt wird der verwendete A*-Algorithmus vollständig definiert.

Zustandsmenge M

Ein Zustand $z \in M$ aus der Zustandsmenge $M = L \times G$ in diesem Suchalgorithmus ist definiert durch die Pose $l \in L$ des linken Fußes und des aktuellen GCVs $g \in G$, der äquivalent zur momentanen Geschwindigkeit des Roboters ist. Die Roboterpose wird indirekt durch die Pose des linken Fußes bestimmt und wird nicht explizit benötigt. Somit ergibt sich als Zustandsmenge M die Menge aller Posen $l \in L = \mathbb{R} \times \mathbb{R} \times [-\pi, \pi]$ und aller gültigen GCVs $g \in G$. Gültige GCVs sind GCVs aus dem Intervall $G = [-1, 1]^3$ die den Geschwindigkeitsbeschränkungen des Roboters genügen.

Würde nur einmal geplant werden und anschließend dieser Plan ausgeführt, benötigte der Zustand lediglich die Position. Da aber häufig neugeplant werden soll, ist die Position alleine nicht mehr eindeutig, sondern der Zustand muss zudem den aktuellen GCV beinhalten.

Der Startzustand s

Der Startzustand $s \in M$ definiert den Zustand, mit welchem der Algorithmus die Suche beginnt.

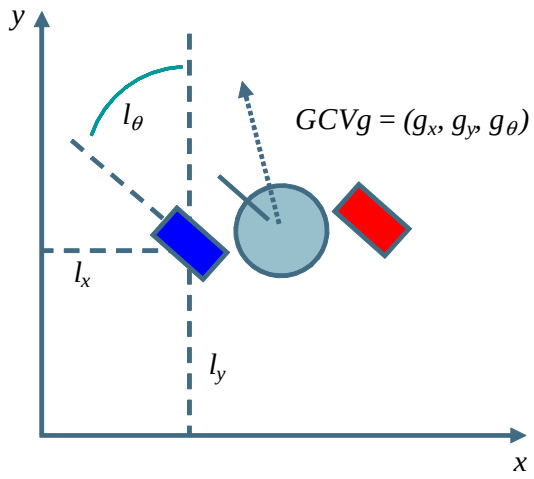

Abbildung 7.9: Startzustand s der Schrittplanung mit A* bestehend aus der Pose $l = (l_x, l_y, l_\theta)$ des linken Fußes des Roboters und dem letzten Gangsteuerungsvektor $g = (g_x, g_y, g_\theta)$.

Die Zielmenge Z

Die Zielmenge $Z \subseteq M$ ist ein kleiner Bereich um das Ziel $z = (z_x, z_y, z_\theta, g_x^*, g_y^*, g_\theta^*) \in Z \subseteq M$ mit $z_{xy} = (z_x, z_y)$ und $g_{xy}^* = (g_x^*, g_y^*)$ in dem normierten Koordinatensystem mit dem Radius r, der maximalen Ausrichtungsabweichung Δ_θ und der maximalen Geschwindigkeitsabweichung Δ_g. Das bedeutet, dass eine Schrittpose $l = (l_x, l_y, l_\theta) \in M$ mit der Position $l_{xy} = (l_x, l_y)$, dem Ausrichtungswinkel l_θ und dem Gangsteuerungsvektor $g = (g_x, g_y, g_\theta)$ mit $g_{xy} = (g_x, g_y)$ genau dann zur Zielmenge gehört, wenn die Bedingungen

$$||l_{xy} - z_{xy}||_2 = \sqrt{(l_x - z_x)^2 + (l_y - z_y)^2} < r \tag{7.16}$$

$$|l_\theta - z_\theta| < \Delta_\theta \tag{7.17}$$

$$||g_{xy} - g_{xy}^*||_2 = \sqrt{(g_x - g_x^*)^2 + (g_y - g_y^*)^2} < \Delta_g \tag{7.18}$$

mit $r = 10$, $\Delta_\theta = 10$ und $\Delta_g = 0,1$ erfüllt sind. Die erste Bedingung (7.16) sorgt dabei dafür, dass die Fußposition sich in der Nähe des Zieles befindet. Durch die zweite Bedingung (7.17) muss die Fußausrichtung ähnlich der Zielausrichtung sein. Diese beiden Bedingungen sind einzeln formuliert, damit r und Δ_θ unabhängig und intuitiv gewählt werden können, da die Position und die Ausrichtung andere Maßstäbe und Bedeutungen haben. Die dritte Bedingung (7.18) lässt eine leichte Abweichung der x und y Komponenten des aktuellen Gangsteuerungsvektors zum optimalen GCV zu. Dabei wird die Drehkomponente g_θ ignoriert.

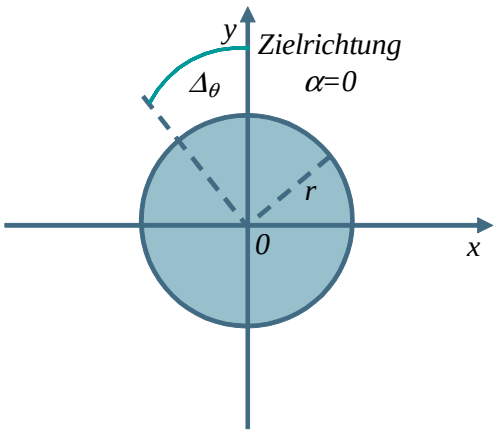

Abbildung 7.10: Der Zielbereich der Schrittplanung um den Ursprung mit der Größe r, mit der Zielausrichtung in Richtung der y-Achse und maximaler erlaubter Abweichung der Ausrichtung von Δ_θ

Die Aktionsmenge A

Die Aktionsmenge A ist eine diskrete Untermenge der realen, zulässigen Beschleunigungen. Diese werden mittels der Limitierungsfunktion aus dem Gangverhalten und der Konfigurationsdatei des Roboters ermittelt. In der Implementierung, verwendete der A*-Algorithmus fünf Beschleunigungsaktionen, jeweils eine nach vorne, nach rechts, nach links, sowie Drehung nach rechts und nach links. Da mit maximaler Geschwindigkeit das Ziel erreicht werden soll, gibt es keine Aktion, die abbremst. Allerdings bremst das Modell automatisch beim Drehen ab. Alle Aktionen werden mit voller Beschleunigung ausgeführt. Es sind somit keine Zwischenstufen möglich. Dies sollte aber nicht so weit von der Optimalität wegführen. Des Weiteren werden stets Doppelschritte berechnet, also beginnend vom linken Standfuß, einen rechten Schritt und danach wieder einen linken, so dass ein gültiger Zustand immer den linken Fuß als Standfuß verwendet.

Die Transferfunktion t

In der Transferfunktion werden diese Aktionen zweimal auf den aktuellen GCV angewendet mit Berücksichtigung der Beschleunigung und insbesondere der Geschwindkeitslimitierungen. Durch die erste Anwendung kommt der GCV g_r zustande, der die linke Schrittposition in eine rechte Schrittposition überführt, durch die zweite Anwendung kommt der GCV g_l zustande, der wiederum die rechte Schrittposition in eine linke Schrittposition überführt. Hierbei ist zu beachten, dass es sich um eine einzelne Aktion handelt, die auf beide Schritte angewendet wird. Es ist also nicht möglich, im Links-rechts-Schritt zu stark rechts zu drehen und im rechts-links-Schritt wieder links. Mit der Pose der dadurch entstandenen linken Schrittposition und dem GCV g_l, der zu dieser Pose geführt hat, wird der Folgezustand definiert.

Sei $a = g_a \in A$ eine Aktion aus der Aktionsmenge, $z = (l_z, g_z) \in M$ der momentane Zustand und $m(g)$ das Schrittmodell, dann ist die Transferfunktion

$$t : M \times A \to M \tag{7.19}$$

gegeben durch:

$$lz := l_z + m(g_z + g_a) + m(g_z + 2g_a) \tag{7.20}$$

$$g_z := g_z + 2g_a \tag{7.21}$$

Die Kostenfunktion c

In diesem A*-Algorithmus wird eine einfache Kostenfunktion verwendet. Die Kostenfunktion $c(k)$ gibt für jeden Knoten k 1 zurück. Intuitiv bedeutet dies, dass jeder

Schritt für den Roboter gleich teuer ist. Die Kosten ergeben für einen Pfad kumuliert die Anzahl der benötigten Schritte vom Startpunkt zum betrachteten Knoten k. Die Gesamtkosten $C(P)$ eines Pfades P mit $|P|$ Schritten betragen:

$$C(P) = |P| \tag{7.22}$$

Die Heuristik h

Die Heuristik $h(k)$ schätzt die zum Erreichen des Zieles benötigten Kosten ab. Die geschätzten Kosten liegen dabei immer unter den realen Kosten. Der A*-Algorithmus nutzt die Heuristik, indem er den Knoten als nächstes expandiert, der von der Heuristik als kostengünstig eingeschätzt wird. Dies hat zur Folge, dass je besser die Heuristik die Wegkosten schätzt, desto geringer fällt die Laufzeit des Algorithmus aus. Für die Heuristik ist die Geschwindigkeit als Distanz pro Schritt definiert, so dass sie nicht die benötigte Zeit, sondern die benötigten Schritte als Kosten ausgibt. Die Kosten sind dadurch mit der Kostenfunktion des A*-Algorithmus kompatibel.

In die Entwicklung der Heuristik wurde viel Zeit investiert, da der A*-Algorithmus bei ausreichender Genauigkeit extreme Laufzeiten aufwies. Die Genauigkeit, mit der das Ziel erreicht werden kann, ist variabel. Das heißt, dass der Radius, in welchem das Ziel liegt, beliebig vergrößert bzw. verkleinert werden kann. Je größer der Radius ist, desto einfacher kann der A*-Algorithmus das Ziel zwar treffen, allerdings ist die Berechnung erfolgreich abgeschlossen, sobald der Roboter sich in diesem Radius positioniert. Dies kann jedoch eine Position sein, welche noch zu weit vom eigentlichen Ziel entfernt ist. Deshalb wird versucht, den Radius um das Ziel so klein wie möglich zu halten, auch wenn dies längere Laufzeiten des A*-Algorithmus zur Folge hat. Dafür wird jedoch das Ziel sehr genau getroffen.

Zunächst wurde eine einfache Heuristik für den 6D Posen-Geschwindigkeitsplaner verwendet mit

$$h_l(d) = \frac{d}{v_m},\ d = ||p_1 - p_2|| \tag{7.23}$$

für zwei Punkte p_1 und p_2 und der maximalen Geschwindigkeit v_m. Diese Heuristik ist gültig, weil sie stets die benötigten Kosten unterschätzt, da die Maximalgeschwindigkeit als durchschnittliche Geschwindigkeit verwendet wird. Aufgrund dessen, dass diese Heuristik aber für große Maximalgeschwindigkeiten im Vergleich zur Beschleunigung sehr stark unterschätzt, werden andere Heuristiken benötigt.

Man geht davon aus, dass der Roboter bereits zum Ziel gedreht ist und eine Anfangsgeschwindigkeit v_0 besitzt. Diese Annahmen vereinfachen das Problem und unterschätzen stets die reale Zeit, da der Roboter zusätzlich Zeit zum Drehen braucht.

Nun beschleunigt der Roboter mit maximaler Beschleunigung a, bis er die Maximalgeschwindigkeit v_m erreicht hat. Somit ist:

$$v(t) := \begin{cases} v_0 + a \cdot t, & \text{wenn } t < f \\ v_m, & \text{sonst} \end{cases} \quad \text{mit } f := \frac{v_m - v_0}{a} \tag{7.24}$$

Grafisch wird dieser Vorgang so abgebildet:

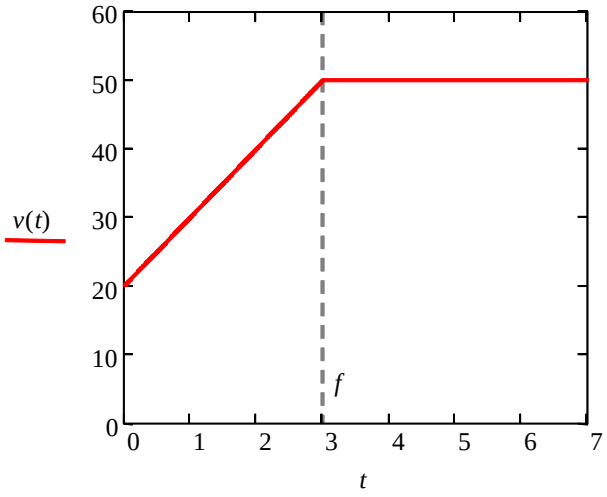

Abbildung 7.11: Geschwindigkeitsprofil. Erst wird linear beschleunigt und sich danach mit konstanter Geschwindigkeit fortbewegt.

Nun existiert eine Funktion $v(t)$, die die Geschwindigkeit v zum Zeitpunkt t angibt. Es ist aber die benötigte Zeit für eine Distanz d erforderlich. Also wird zunächst die Funktion integriert.

$$d(t) = \int_0^t v(t)\, dt = \begin{cases} v_0 t + \frac{1}{2} a t^2, & \text{wenn } t < f \\ v_0 f + \frac{1}{2} a f^2 + v_m t - v_m f, & \text{sonst} \end{cases} \tag{7.25}$$

Zunächst wird die invertierte Stelle b benötigt. Diese entsteht durch das Einsetzen von f in $d(t)$:

$$b = d(f) = \frac{v_m^2 - v_0^2}{2a} \tag{7.26}$$

Durch Invertieren der Funktion

$$t = h(d) = \begin{cases} -\frac{p}{2} + \sqrt{\frac{p^2}{4} - q}, & \text{wenn } d < b \\ \frac{d - v_0 f - \sqrt{2af^2 + v_m f}}{v_m}, & \text{sonst} \end{cases} \quad \text{mit } p := \frac{2v_0}{a}, q := \frac{-2d}{a} \tag{7.27}$$

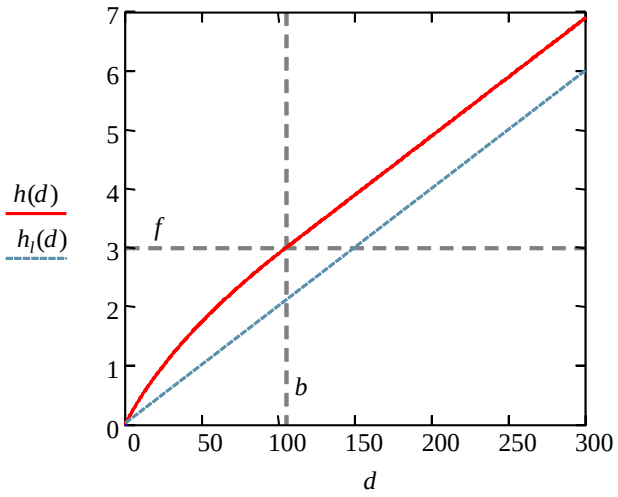

Abbildung 7.12: Die Heuristiken im Vergleich: Die Distanzheuristik wird von der beschleunigten Heuristik dominiert

Grafisch wird diese Heuristik (oberer Graph) im Vergleich zur alten Heuristik (unterer Graph) wie folgt dargestellt:

Es ist offensichtlich, dass die neue Heuristik weiterhin unterschätzt, aber stets höhere, also genauere Schätzwerte liefert, als die einfache Distanzheuristik. Leider betrachtet diese Heuristik nur das eindimensionale Problem, also das Beschleunigen, mit Maximalgeschwindigkeit fortbewegen und abbremsen. Sie berücksichtigt nicht die zweidimensionalen Besonderheiten, wenn der Roboter eine Kurve zum Ball laufen muss. Dies ist maßgeblich der Grund, warum der A*-Algorithmus teilweise immer noch hohe Rechenzeiten benötigt, um eine Lösung zu finden. Dies ist gerade in Fällen mit kurzen Distanzen, hoher Geschwindigkeit und starken Ausrichtungen in eine entgegengesetzte Richtung zu beobachten.

Zeitanalyse der Schrittplanung mit A*-Algorithmus

Die Graphen in Abbildung 7.13 zeigen die Erfolgsquote der Schrittplanung mit A*-Algorithmus in Abhängigkeit der maximal investierten Zeit. Der linke Graph zeigt deutlich, dass für eine Erfolgsquote von 60% über 30 Sekunden eingeplant werden müssen. Andererseits können 50% der Fälle in 5 Sekunden berechnet werden. Der Verlauf der Kurve lässt darauf schließen, dass mehr als 30 Sekunden Berechnungsdauer keinen bedeutenden Mehrwert erzeugen. Allerdings ist die Berechnungsdauer auf den möglichen Startposen nicht gleichverteilt, sondern es existieren leicht und schwer zu berechnende Bereiche. Wird eine möglichst vollständige Abdeckung

des Konfigurationsraums angestrebt, werden längere Berechnungszeiten benötigt. Im

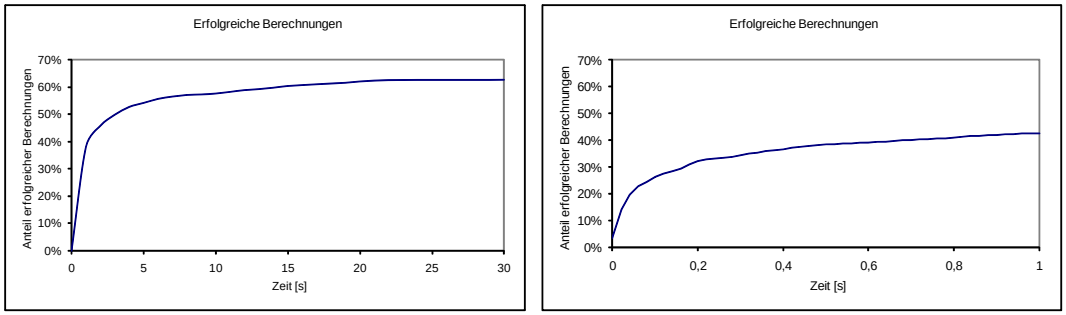

Abbildung 7.13: Erfolgsquote der Schrittplanung mit A* in Abhängigkeit der maximalen Berechnungsdauer mit unterschiedlichen Zeithorizonten.

rechten Graph wird diese Quote in einem kleineren Zeitintervall von einer Sekunde dargestellt. Hier wird sichtbar, dass bei einer maximalen Berechnungsdauer von 100 ms ein Anteil von 25% berechnet werden kann.

Fazit

Die Darstellung der Berechnungsdauer im vorherigen Abschnitt macht deutlich, dass ein Online-Betrieb in der jetzigen Umsetzung nicht möglich ist. Daher muss eine Alternative gefunden werden, die benötigte Online-Berechnungszeit drastisch zu kürzen.

7.4 Schrittplanung mit Funktionsapproximator

Da die Laufzeit die vornehmlich beschränkte Ressource ist und der A*-Algorithmus des Schrittplaners zu lange Rechenzeiten für den Online-Betrieb aufweist, wurde eine Möglichkeit entwickelt, diesen zur Verbesserung der Online-Laufzeit vorzuberechnen und danach mit einem Funktionsapproximator anzunähern. Dazu werden mit dem A*-Algorithmus $n_p = 17761$ zufällige Pfade aus einer Definitionsmenge (7.4.1) zum Ziel berechnet. Diese Pfade sind im Rahmen des vom A*-Algorithmus verwendeten diskreten Modells optimal. Es entstehen ausschließlich gültige, ablaufbare Schrittfolgen. Die einzelnen Schritte, genauer die GCVs dieser Schritte, werden dann vom Funktionsapproximator trainiert.

Während der Laufzeit erhält die Schrittplanung die Position eines Ziels im egozentrischen Koordinatensystem und berechnet daraus den Gangsteuerungsvektor des

nächsten Schrittes. Der GCV wird an die ausführende Schicht übergeben. Doch anstatt einen kompletten Pfad online mit dem A*-Algorithmus zu berechnen, wird der Approximator abgefragt und dessen Rückgabewert als Ausgabe weitergeleitet. Durch das Vorberechnen möglicher Pfade ist der Approximator keineswegs rein reaktiv, sondern verwendet Wissen aus dem Schrittmodell. Es wurden mehrere Funktionsapproximatoren getestet und letztendlich wird ein mehrlagiges Perzeptron eingesetzt.

Durch das mehrlagige Perzeptron wird die Funktion $s : g_t, r, z \rightarrow g_{t+1}$ mit aktuellem GCV g_t, Roboterpose r, Zielpose z und dem nächstem GCV g_{t+1} approximiert. Diese Funktion hat neun Dimensionen, welche sich aus drei Dimensionen für den GCV, weiteren drei für die Roboterpose und nochmals drei für die Zielpose ergeben. Um dieses Problem zu vereinfachen, wurden durch Normierung des Koordinatensystems die drei Eingangsdimensionen des Ziels eingespart. Dafür wird das Koordinatensystem so definiert, dass sich das Ziel im Koordinatenursprung befindet und die Zielausrichtung $\alpha = 0$ in Richtung der y-Achse zeigt. Das Ziel wird mit dem linken Fuß angesteuert. Durch die folgende Transformation N ist es möglich, jede egozentrische Situation wie sie aus der Computervision kommt in dieses System abzubilden.

Transformation N:

$$(r_{bx}, r_{by}) = R(-z_\theta) \cdot (-z_x, z_y)^T \tag{7.28}$$

$$r_{b\theta} = -z_\theta \tag{7.29}$$

Mit $R(\alpha)$ als Rotationsmatrix und Zielpose z.

Das auf den Ball zentrierte Koordinatensystem hat den Vorteil, dass alle einzelnen Schritte von den berechneten Pfaden ohne Transformation als Trainingsbeispiele verwendet werden können, anstatt nur die Anfangsposition und dessen nächster Schritt. Es soll die Funktion $s(g_t, r_n) \rightarrow g_{t+1}$ abgebildet werden, mit der in Bezug auf den Ball normierten Roboterpose r_n, dem momentanen Gangsteuerungsvektor g_t und einem neuen zukünftigen GCV g_{t+1}. Somit bestehen die Datenpunkte aus den drei Komponenten (g_t, r_n, g_{t+1}). Diese können aus jedem Schritt gewonnen werden, indem der GCV des letzten Schrittes, also der Schritt, der zu der aktuellen Position geführt hat, die Position des linken Fußes, sowie der neue GCV des nächsten Schrittes verwendet werden. Natürlich muss dabei zwischen einem rechten und einem linken Schritt unterschieden werden. Deshalb werden die GCVs beider Schritte abgebildet. Die Extraktion der Datenpunkte ist im Gegensatz dazu bei einem Koordinatensystem, welches auf das Ziel zentriert wurde, nicht ohne Koordinatenumrechnung möglich. Das Verwenden aller Schritte als Datenpunkt ermöglicht eine deutlich geringere Anzahl berechneter Pfade und legt zudem den Fokus auf dem Ball nahe Positionen, da dort alle Pfade entlang führen und Datenpunkte erzeugen.

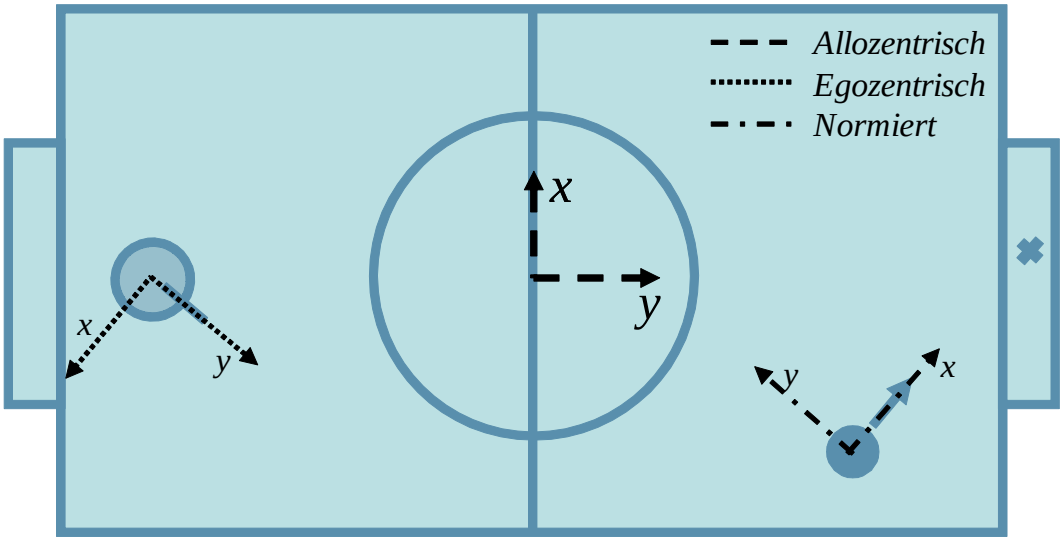

Abbildung 7.14: Die drei Koordinatensysteme: Das globale allozentrische, das auf den Roboter zentrierte egozentrische und das auf den Ball bezogene normierte Koordinatensystem. Die Computervision bietet die entdeckten Objekte in egozentrischen Koordinaten. Die Schrittplanung arbeitet im auf den Ball normierten Koordinatensystem.

7.4.1 Definitionsbereich der Startpositionen

Der Ball befindet sich im Ursprung des Koordinatensystems mit der Zielausrichtung nach links. In dem Quadranten, welcher links entgegengesetzt der Ausrichtung des Balles liegt, werden gleichmäßig n_{xy} viele Positionen mit einem maximalen Abstand von $L = 3$ Meter verteilt, siehe Abbildung 7.15. Dazu werden Punkte auf einem Raster in Abständen von 30 cm verteilt und überprüft, ob sie innerhalb des Radius L liegen. Jeder dieser Punkte ist zudem mit Ausrichtungen und Startgeschwindigkeiten ausgestattet. Die Ausrichtungen sind mit einer Diskretisierung im Bereich +- 90° von der Nullausrichtung definiert. Die Auflösung dieser Diskretisierung beträgt $n = 10$. Die besagte Nullausrichtung ist als Ausrichtung zum Ball definiert. Dies stellt eine Vereinfachung dar, um die benötigten A*-Pfade einzuschränken und uninteressantere Fälle wegzulassen. Da der Roboter, wenn er den Ball nicht sieht, ein vorhandenes Verhalten „SearchBall“ aufruft, um den Ball zu suchen, sollte sich der Ball für die Berechnung mit dem A*-Algorithmus im Sichtbereich des Roboters befinden. Die Menge der Startgeschwindigkeiten ist eine diskrete Untermenge aller möglichen GCVs mit einer komponentenweisen Auflösung von $n_{GCV} = 10$.

Das Winkelmaß, das die Ausrichtung des Roboters beschreibt, ist zyklisch, die Approximation hingegen nicht. Deshalb entsteht eine Unstetigkeitsstelle beim Approximieren. Die berechneten Pfade haben bei der Drehung auch eine Unstetigkeitsstelle,

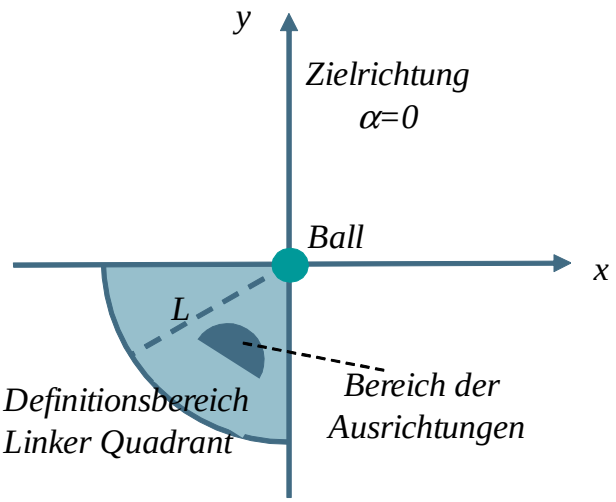

Abbildung 7.15: Definitionsbereich der Startpositionen. Der grün-blaue Teil zeigt den Definitionsbereich, innerhalb dieses Bereiches sind die Roboterpositionen gleichmäßig verteilt. Jede dieser Positionen hat Ausrichtungen im Bereich +-90 Grad Unterschied von der Ballausrichtung.

nämlich ungefähr dann, wenn der Roboter mit dem Rücken zum Ziel steht und es keinen Zeitunterschied macht, ob er sich links oder rechts herum zum Ziel dreht. Es hat sich gezeigt, dass gute Ergebnisse geliefert werden, wenn die Unstetigkeitsstelle nach unten zeigt.

Erläuterung der Datenerhebung

Für die Berechnung wurden die Startpositionen im Definitionsbereich definiert, gemischt und dann in $n_r = 8 \cdot 3 = 24$ Pakete aufgeteilt. Auf drei Rechnern wurden dann 8 Instanzen eines Programms zur Pfadberechnung mit jeweils einem Paket der Startpositionen gestartet und über ein Wochenende 17761 Pfade berechnet. Da mehr Startpunkte definiert wurden, als jemals berechnet werden könnten, wird eine zufällige Aufteilung der Startpunkte in den Dateien benötigt, um eine gleichmäßige Verteilung der berechneten Pfade sicherzustellen. Das Programm speichert nach der Beendung einer A*-Pfadsuche stets die Ergebnisse der Suche. Die gefundenen Pfade werden von dem Programm in der Datenbank als gefunden markiert. Die Suche eines jeden Pfades wird beim Erreichen eines einstellbaren Limits von A*-Iterationen abgebrochen. Diese Pfade werden dann intern als fehlgeschlagen markiert und die maximalen Iterationen abgespeichert. Bei Bedarf kann die Berechnung der Pfade unterbrochen und zu einem späteren Zeitpunkt fortgesetzt werden. Dabei werden Startpunkte, die bei gleichem oder höherem Limit fehlgeschlagen sind, nicht wieder-

holt, während Startpunkte mit niedrigerem Limit wiederholt werden. Startpunkte, die bereits in der Datenbank als berechnet markiert wurden, werden nicht erneut berechnet. So kann auch bei einem Fehler im Programm die Suche weitergeführt werden.

Jeder gefundene Pfad bekommt eine Paketnummer, eine Durchlaufnummer und eine Pfadnummer. Die Paketnummer gibt an, in welchem Paket sich die Startposition befand. Die Durchlaufnummer gibt die Möglichkeit, jedem Durchlauf eine Nummer zu vergeben. So können unterschiedliche Durchläufe zusammengefasst werden, ohne dass der Index zerbricht. Die Pfadnummer ist eine fortlaufende Nummer für jeden Pfad eines Pakets. Diese ist nur in einem Paket eindeutig definiert und kommt insgesamt mehrfach vor. Die Kombination aus Paket-, Durchlauf- und Pfadnummer ergeben einen immer eindeutigen Index. Daran kann ein berechneter Pfad eindeutig identifiziert werden, auch beim Zusammenführen mehrerer Durchläufe.

7.4.2 Zieldifferenz der Pfade korrigieren

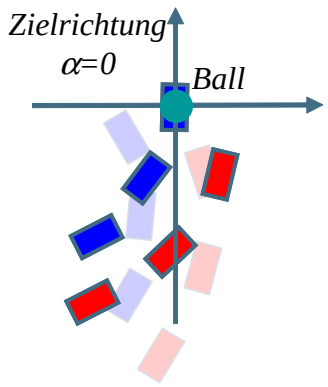

Abbildung 7.16: Die Schrittpfade des A*-Algorithmus treffen nicht ganz genau das Ziel. Durch eine Transformation kann dies korrigiert werden.

Die Ausgangssituation ist ein von der A*-Schrittplanung berechneter Pfad, der die letzte Fußposition in die Nähe des Zieles setzt, allerdings nicht zwingend genau auf das Ziel. Diese Situation ist in Abbildung 7.16 skizziert. Zur Erhöhung der Genauigkeit am Ziel wird ein Verfahren angewendet, welches erlaubt, den Fehler am Ziel zu korrigieren. Dies geschieht durch Anwendung einer Translation und Rotation auf dem gesamten Pfad, so dass die letzte Fußposition des Roboters exakt auf das Ziel verschoben wird. Aus dieser Verschiebung resultiert, dass der gesamte Pfad mitverschoben wird, wie in Abbildung 7.16 demonstriert. Da die Startposition des Roboters nicht relevant ist, sondern der Fokus ausschließlich auf die Abdeckung des Definitionsraumes gelegt wird, ist diese Transformation möglich. Da die Abweichung

am Ziel den gesamten Pfad betrifft, erzeugen ähnliche Startposen um diese Abweichung verrauschte Pfade. Durch die Transformation wird dieses Rauschen entfernt und somit die Genauigkeit der approximierten Pfade am Ziel deutlich erhöht. Der vorgestellte Vorgang wird wie folgt definiert:
Für einen Pfad P wird folgende Transformation mit $z = (z_x, z_y, z_\theta) \in P$ als der letzte Schritt des berechneten Pfades P definiert als:

$$\Delta_{xy} = (-z_x, -z_y) \tag{7.30}$$
$$\Delta_\theta = -z_\theta \tag{7.31}$$

Sei nun $p = (p_x, p_y, p_\theta) \in P$ ein Schritt im Pfad P inklusiv des letzten Schrittes und sei $p_{xy} = (p_x, p_y)$ die 2D-Position ohne Orientierung, dann wird für jeden Schritt p die Transformation angewendet

$$\forall p \in P :$$
$$p_{xy} = (p_{xy} + \Delta_{xy}) \cdot R(\Delta_\theta) \tag{7.32}$$
$$p_\theta = p_\theta + \Delta_\theta \tag{7.33}$$

mit der Rotationmatrix R.

7.4.3 Trainingsbeispiele für die andere Seite

Es wurden nur Pfade für den Anlauf von dem linken, hinter dem Ball liegenden Quadranten des Koordinatensystems des Balles mit dem A*-Schrittplaner berechnet, siehe Abbildung 7.17. Um nun ebenso von dem rechten Quadranten aus zum Ball anlaufen zu können, werden Pfade von der rechten Seite aus als Trainingsbeispiele benötigt. Diese werden erzeugt, indem die Pfade von dem linken Quadranten auf den rechten gespiegelt werden, sowie auch die x- und θ-Kompontene des GCVs. Bedauerlicherweise ändern sich dabei die Schritte: sowohl der Zielschritt, mit dem das Ziel erreicht wird, als auch der Startschritt, mit dem los gelaufen wird. Das bedeutet, dass die Schritte, die zuvor mit dem linken Fuß ausgeführt wurden, nun mit dem rechten Fuß ausgeführt werden und umgekehrt. Um dies zu verhindern, wird der Pfad des linken Quadranten anstatt mit dem linken Fuß, mit dem rechten Fuß gespiegelt, welcher, wie in Abschnitt 7.4.4 beschrieben, berechnet wurde.
Die Transformation, die einen GCV $g = (g_x, g_y, g_\theta)$ spiegelt, ist gegeben durch:

$$g_x = g_x \cdot -1 g_\theta = g_\theta \cdot -1 \tag{7.34}$$

7.4.4 Trainingsbeispiele für das rechte Bein

Sowohl zum Spiegel der Quadranten als auch zum Anlaufen mit dem rechten Fuß werden Trainingsbeispiele für den Anlauf mit dem rechten Fuß benötigt. Eine Mög-

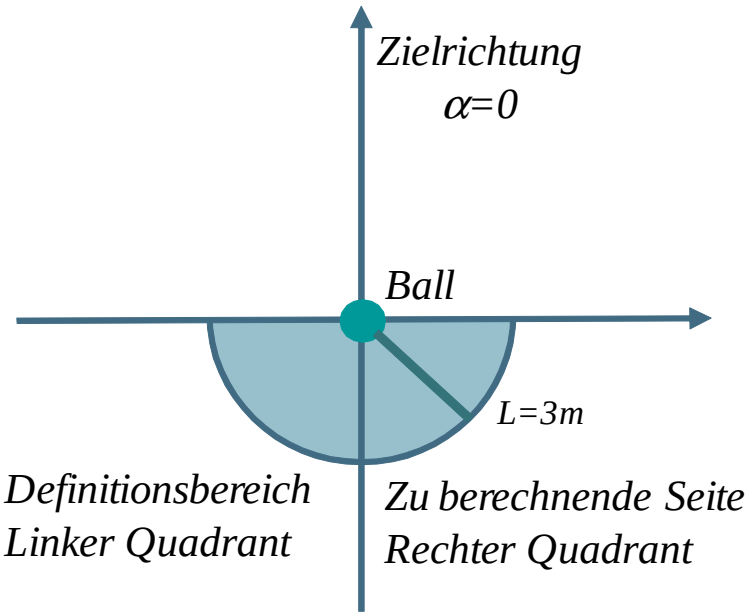

Abbildung 7.17: Definitionsbereich der Schrittplanung mit A*-Algorithmus (links) und Übertragung auf die andere Seite (rechts)

lichkeit wäre, ähnlich viele Pfade für diesen Anlauf mit dem A*-Schrittplaner durchzurechnen. Eine andere weitaus elegantere Möglichkeit ist es, diese aus den vorhandenen Pfaden zu errechnen.

Dazu wird der letzte linke Schritt aus der Schrittsequenz eines jeden Pfades gelöscht, um mit einem rechten Fuß den Pfad zu beenden. Da jedoch stets Doppelschritte, also rechts-links-Paare trainiert werden, muss zudem noch der erste Schritt gelöscht werden, um einerseits auf eine gerade Anzahl an Schritten zu kommen und andererseits mit einem rechten Fuß zu beginnen. Die so erhaltenen Pfade werden wie in Abschnitt 7.4.2 beschrieben, transformiert, so dass der letzte Schritt wieder das Ziel trifft.

7.5 Funktionsapproximatoren

Da der A*-Algorithmus zum Erlangen einer gültigen Schrittfolge, wie bereits erwähnt, extreme Laufzeiten aufwies, wird ein Funktionsapproximator eingesetzt, der es erlaubt, ähnliche Werte innerhalb kürzester Zeit zu erzeugen. Als Funktionsapproximator können verschiedene Verfahren eingesetzt werden. In dieser Arbeit wurde ein Interpolationsverfahren basierend auf dem k-nächste-Nachbarn-Verfahren, ein

linearinterpolierendes Gitter und letztlich ein mehrlagiges Perzeptron eingesetzt.

Der A* erzeugt Schrittfolgen von einem Startpunkt aus dem Definitionsbereich zum Ziel, welches im Koordinatenursprung liegt. Diese Schrittfolgen sind Aneinanderreihungen von linken und rechten Fußpositionen mit dazugehörigen GCVs. Jeweils ein GCV erzeugt den Schritt mit dem linken Fuß und ein anderer GCV erzeugt einen Schritt mit dem rechten Fuß. Die Funktionsapproximatoren erhalten als Eingabe die aktuelle Pose des linken Fußes und die aktuelle Geschwindigkeit des Roboters gegeben durch den GCV. Als Ausgabe erzeugen diese Approximatoren zwei GCVs: Einen für die rechte Schrittposition (GCV g_r) und einen für die linke Schrittposition (GCV g_l). Ein Beispiel ist ein Datenpunkt dieser Funktion. Somit ist ein Beispiel definiert als Eingabe $(l, g_v) \Rightarrow$ Ausgabe (gl, gr).

7.5.1 Das k-nächste-Nachbarn-Verfahren

Zunächst wurde ein k-nächste-Nachbarn-Verfahren für die Approximation der Schritte getestet. Zu Beginn wurden alle mit dem A*-Algorithmus vorberechneten Datenpunkte einem k-d-Baum (mit k=6) in einer Vorberechnungsphase hinzugefügt. Für die Abfrage eines Gangsteuerungsvektor für eine Startpose im Koordinatensystems des Balles werden die $n_n = 200$ nächsten Nachbarn aus dem k-d-Baum abgefragt. Durch diese Datenpunkte, die sich im sechsdimensionalen Eingangsraum (GCV g_v und Pose l) befinden, werden sechs sechsdimensionale Hyperebenen

$$h_{ix}(l_j, g_{vj}) = g_{ixj} \tag{7.35}$$
$$h_{iy}(l_j, g_{vj}) = g_{iyj} \tag{7.36}$$
$$h_{i\theta}(l_j, g_{vj}) = g_{i\theta j} \tag{7.37}$$

mit $i \in [l, r]$ und $j \in [1, n_n]$ eine für jede Komponente der beiden Gangsteuerungsvektoren g_r und g_l mit der Methode der kleinsten Quadrate eingebettet. Durch Berechnen des Wertes der Hyperebene am Anfragepunkt (g_v, l) werden zwei GCVs ermittelt: Der GCV g_l und der GCV g_r. Diese GCVs müssen mit den gegebenen Roboterlimitierungen begrenzt werden, da nicht sichergestellt werden kann, dass bei der Interpolation und Extrapolation keine ungültigen, nicht ablaufbaren GCVs erzeugt werden.

Für die einfache Steuerung des Roboters reichen die berechneten GCVs aus. Doch wird der gesamte Pfad benötigt, müssen die Schrittpositionen berechnet werden. Diese können mit dem Vorwärtsmodell aus der Pose und den GCVs berechnet werden, indem das Modell mit dem GCV g_l auf die linke Fußpose angewendet wird. Dadurch entsteht die Fußpose r des rechten Fußes. Wird darauf das Modell mit g_r angewendet, erhält man die neue Pose des linken Schrittes. Eine aufeinanderfolgende Ausführung dieser Methode erzeugt einen ablaufbaren Pfad zum Ziel.

Dieses Verfahren erzeugt geeignete, glatte Pfade. Der einzige Nachteil, den dieses Verfahren aufweist, ist die zwar deutlich verkürzte, aber immer noch zu lange Rechenzeit von durchschnittlich 80ms pro Pfad, siehe Abbildung 7.23.

7.5.2 Linearinterpolierendes Gitter

Da das Hauptproblem des k-nächsten-Nachbar-Ansatzes die lange Laufzeit ist, wurde eine Möglichkeit implementiert, die die Gangsteuerungsvektoren g_l und g_r in konstanter Zeit berechnen kann. Dies wird mit einer Interpolation zwischen Gitterpunkten in einem Gitter erreicht. Es können hier verschiedene Gitterstrukturen eingesetzt werden, solange die Zugriffszeit der Werte eines Gitterpunktes in $O(1)$ liegt. Diese können zum Beispiel ein homogenes Gitter, ein logarithmisches Gitter oder ein Polargitter sein. Man beachte hierbei die hohe Eingangsdimension. Im Polargitter entstehen ab der zweiten Dimension Polstellen. Je mehr Dimensionen existieren, desto mehr Polstellen entstehen. Im Sechsdimensionalen ist der Umfang der Polstellen immens.

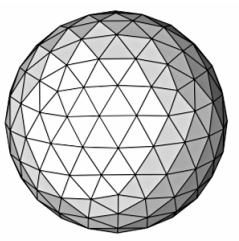

Abbildung 7.18: Bei einer geodätischen Kuppel sind alle Punkte auf der Kugeloberfläche gleich weit von ihrem nächsten Nachbarn entfernt.

Viele Gitterpunkte konzentrieren sich auf kleinstem Raum. Um diesem Effekt entgegenzuwirken, müsste eine sechsdimensionale, geodätische Kuppel (Abbildung 7.18) eingesetzt werden, die schon im dreidimensionalen Raum schwer zu implementieren ist. Daher wird hier das homogene Gitter näher betrachtet.

Das homogene Gitter mit d_e Eingangsdimensionen und d_a Ausgangsdimensionen ist definiert als eine Abbildung

$$G : \mathbb{N}^{d_e} \to \mathbb{R}^{d_a} \tag{7.38}$$

die in $O(1)$ abgefragt werden kann. Jedem Gitterpunkt ist eine kartesische Koordinate in Bezug auf den Ball zugeordnet.

Zunächst müssen den Gitterpunkten Werte zugewiesen werden. Dies wird erreicht, indem die zwei Gangsteuerungsvektoren g_l und g_r für die kartesischen Koordinaten

der Gitterpunkte mit dem k-nächste-Nachbarn-Verfahren, wie in Abschnitt 7.5.1 beschrieben, berechnet werden.

Zur Laufzeit müssen die Gangsteuerungsvektoren g_l und g_r aus einem übergebenen Ziel berechnet werden. Eigentlich werden dazu die kartesisch-benachbarten Gitterpunkte benötigt. Doch ist es oft einfacher, den Anfragepunkt in das Koordinatensystem des Gitters zu transferieren und dort die Gitterzelle zu bestimmen, in der sich der Anfragepunkt befindet. Eine Gitterzelle besteht, je nach Definition des Gitters, aus n Eckpunkten $E \subset \mathbb{N}^{d_e}$. Da ein m-dimensionaler Hyperwürfel 2^m Ecken besitzt, verfügt im Falle des homogenen und polaren Gitters mit der Dimension $d_e = 6$ eine Zelle über $n = 2^{d_e} = 64$ Eckpunkte.

Die Abbildung (7.38) kann leicht als ein eindimensionales Feld definiert werden. In dem Fall wandelt die Indextransferfunktion $I(e)$ (7.39) den Indexvektor $e \in \mathbb{N}^{d_e}$ in den entsprechenden Index des eindimensionalen Feldes um.

$$I(e) = \sum_{i=0}^{d-1} d_e^i \cdot e_i \tag{7.39}$$

mit d_e-dimensionalen Gitterindex $e \in \mathbb{N}^{d_e}$.

Der Wert jeder dieser Eckpunkte $e \in E$ wird mit $G(e)$ abgefragt. Bei dem homogenen Gitter kann durch Auf- und Abrunden einzelner Komponenten des Anfragepunktes zu einem Gitterpunkt navigiert werden. Durch die binäre Darstellung $i_j \in \{0, 1\}$ mit $j \in [0, d_e]$ eines Laufindizes $i \in [0, 2^{d_e}]$ kann das Auf- ($i_j = 1$) und Abrunden ($i_j = 0$) systematisch erfolgen. Zwischen diesen Eckpunkten wird linear interpoliert, indem für jede Ausgabedimension eine sechsdimensionale Hyperebene mit den Eckpunkten als Datenpunkte eingebettet wird. Durch Auswerten der Ebene an der Anfragestelle erhält man den Interpolationswert.

Dieser Approximator hat bei hohen Dimensionen den Nachteil, dass der Speicheraufwand stark mit der Auflösung des Raums ansteigt, so dass nur geringe Auflösungen möglich sind. Viele der Gitterpunkte sind nicht relevant, da die Auflösungsdichte der Beispiele im weitentfernten Raum abnimmt. Überlegungen zum Speicheraufwand: Das System verwendet sechs Dimensionen. Jeder Datenpunkt benötigt zwei GCVs, die aus drei 32-Bit-*float*-Werten bestehen, also $3 \cdot 4$ Byte = 12 Byte. Wird jede Achse mit zehn Datenpunkten aufgelöst, so benötigt das Gitter mindestens $10^6 \cdot 2 \cdot 12$ Byte = 23 Megabyte.

7.5.3 Mehrlagiges Perzeptron

Ein anderer sehr speichereffizienter und zudem laufzeiteffizienter Funktionsapproximator ist das mehrlagige Perzeptron (MLP). Es wurden insgesamt vier Netze trainiert, davon eins pro Quadrant (siehe Abbildung 7.15) für den Anlauf mit dem linken Fuß sowie für den Anlauf mit dem rechten Fuß. Obwohl sich der Algorithmus also prinzipiell aussuchen könnte, mit welchem Fuß er anlaufen möchte, wird in der verwendeten Implementierung nur mit dem linken Fuß angelaufen. Es ist anzunehmen, dass hier Verbesserungen möglich sind.

Für den Anlauf mit dem linken Fuß werden zwei mehrlagige Perzetrons verwendet, eins für jeden Quadranten in 7.15. Das neuronale Netz für den Anlauf aus dem linken Quadranten wurde mit den Schritten der korrigierten Pfade, die mit dem A*-Algorithmus vorberechnet wurden, trainiert. Das Netz für den Anlauf aus dem rechten Quadranten wurde mit Schritten der Pfade trainiert, die, wie in 7.4.4 beschrieben, umgerechnet wurden.

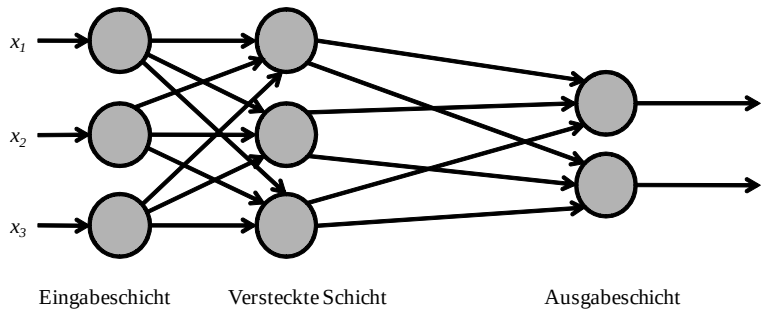

Abbildung 7.19: Aufbau eines mehrlagigen Perzeptrons

Die verwendeten mehrlagigen Perzeptrons besitzen eine versteckte Schicht mit 20 Neuronen. Dazu besitzen sie sechs Eingangsneuronen für die sechs Eingangsdimensionen, drei für die Zielpose $p_z = (z_x, z_y, z - \theta)$ und drei für die aktuelle Geschwindigkeit repräsentiert durch den Gangsteuerungsvektor $g = (g_x, g_y, g_\theta)$. Eigentlich besäße das Netz sechs Ausgangsneuronen für die sechs Ausgangsdimensionen für zwei GCVs g_r, g_l. Doch um das neuronale Netz besser trainieren zu können, wurden die Ausgangsdimensionen in eigenständige Netze aufgespalten, so dass sechs MLPs mit jeweils einem Ausgangsneuron entstehen. Da die Ausgaben unabhängig von einander sind, nachgewiesen in Abschnitt 7.2.4, kann das Trainieren eines Netzes mit sechs Ausgängen gegenüber sechs Netzen mit einem Ausgang keinen Vorteil haben. Somit ist das Trainieren der sechs Netze einfacher und führt zu besseren Ergebnissen.

Lernkurve

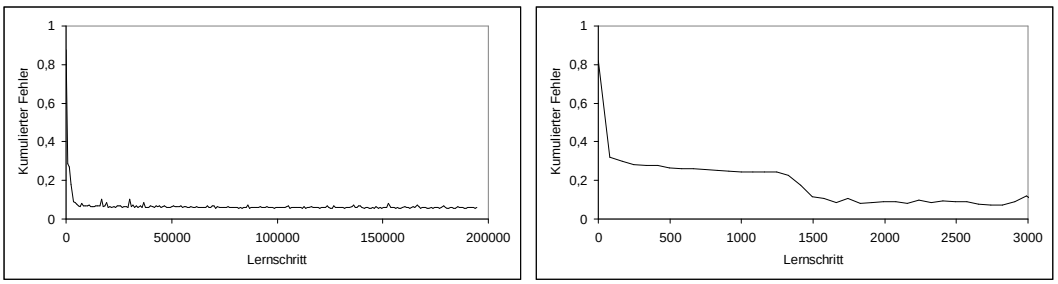

Abbildung 7.20: Die Lernkurve beim Lernen des kompletten Datensatzes (links) und eines verkürzten Datensatzes (rechts).

In Abbildung 7.20 wird der Lernfehler pro Lernschritt dargestellt. Im linken Diagramm wurde der komplette Datensatz aller mit der A*-Schrittplanung vorberechneten Doppelschritte trainiert. Dort ist erkennbar, dass der Fehler schnell auf ein konstant niedriges Niveau sinkt. Im rechten Diagramm wurde ein verkürzter Datensatz zum Lernen verwendet. Der Fehler wurde aufgrund der Vergleichbarkeit auf dem kompletten Datensatz berechnet. Anhand dieses Diagramms ist erkennbar, dass ein kleiner Datensatz vollkommen ausreichend ist.

7.5.4 Vergleich der Approximatoren

Zunächst werden die Fehler der reinen Approximationswerte der unterschiedlichen Verfahren miteinander verglichen. Danach werden die beiden Verfahren anhand der Treffergenauigkeit am Ball gemessen. Dazu wird für jeden Datensatz der komplette Pfad bis zum Erreichen des Balls berechnet. Die Genauigkeit, mit der der Ball getroffen wird, ist das Ergebnis dieses Vergleichs. Zudem wird die Zeit für das Abfragen eines einzelnen Schrittes als auch die Zeit für die Berechnung des kompletten Pfades gemessen. Da die Zeit die stark beschränkte Ressource ist, wird dem letzten Vergleich viel Bedeutung zugemessen. Für die Vergleiche wurde eine Testmenge bestimmt, die nicht Teil der Trainingsmenge ist. Es wird die in dem Kontext des Vergleichs betrachtete Differenz zwischen dem Wert des Funktionsapproximators und dem dazugehörigen realen Wert des Datenpunktes berechnet. Aus den Einzeldifferenzen ergeben sich folgende Kennzahlen: Für N Datensätze und dem dazugehörigen Fehler im Kontext des jeweiligen Vergleichs δ_i für $i \in [0..N]$ definiert sich

- der Mittelwert als

$$\mu := \frac{\sum_{n=0}^{N} \delta_i}{N} \tag{7.40}$$

- der absolute Mittelwert als

$$\frac{\sum_{n=0}^{N} |\delta_i|}{N} \tag{7.41}$$

- und die absolute Standardabweichung als

$$\sigma := \sqrt{\frac{\sum_{n=0}^{N} \delta_i^2}{N} - (\frac{\sum_{n=0}^{N} |\delta_i|}{N})^2} \tag{7.42}$$

Zunächst wird der Fehler der einzelnen Approximatoren verglichen. Dazu wurde jeder einzelne Schritt der Testmenge von dem Funktionsapproximator ausgewertet und die Differenz zum Sollwert des Datenpunktes berechnet.

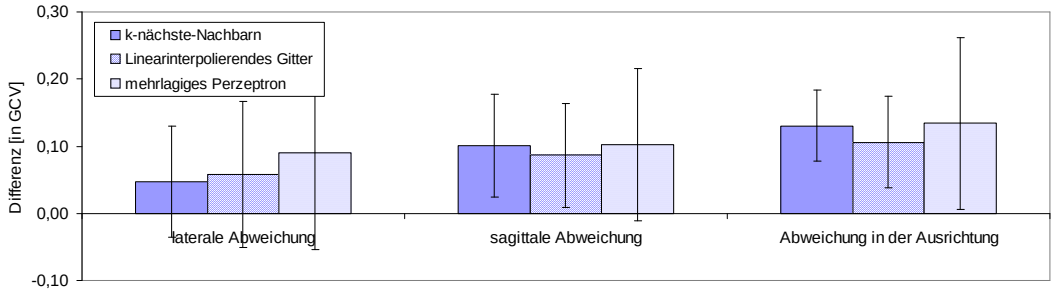

Abbildung 7.21: Durchschnittliche Abweichung des berechneten GCVs der Approximatoren von dem realen GCV im Vergleich.

Die Abbildung 7.21 vergleicht die Abweichung des approximierten vom realen GCV in den einzelnen Komponenten. Da das k-nächste-Nachbar-Verfahren auf der Beispielmenge arbeitet und das Gitter auf Stützstellen, die vom k-nächste-Nachbar-Verfahren gefüllt werden, ist die Abweichung beim mehrlagigen Perzeptron wie erwartet am stärksten. Erstaunlich hingegen ist die geringere Abweichung in der y- und θ-Komponente des GCVs des linearinterpolierenden Gitters verglichen mit der Abweichung des k-nächsten-Nachbarn-Verfahrens.

Als nächstes wird die Genauigkeit der Funktionsapproximatoren am Zielpunkt miteinander verglichen. Dazu wurden die Pfade der drei Approximatoren von allen Datenpunkten der Testmenge aus berechnet und geprüft, wie genau die Zielpose erreicht wird. Abbildung 7.22 veranschaulicht die Ergebnisse. Es wird schnell deutlich, dass alle Approximatoren ähnliche Genauigkeiten in der lateralen Abweichung sowie in der Abweichung der Zielausrichtung aufweisen. Lediglich in der sagittalen Abweichung wird deutlich, dass, wie erwartet, das k-nächste-Nachbarn-Verfahren, welches ohne Training direkt auf den Eingangsdaten arbeitet, die besten Ergebnisse erzielt. Das linearinterpolierende Gitter ist nur etwas ungenauer, gefolgt vom mehrlagigen

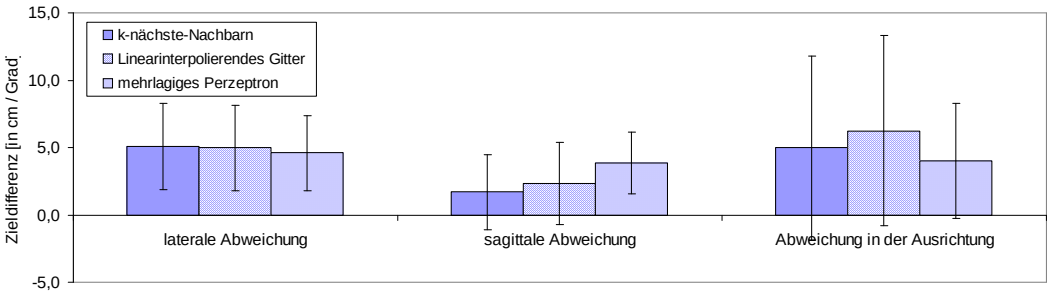

Abbildung 7.22: Genauigkeit der Pfade der verglichenen Funktionsapproximatoren am Ziel gemessen durch die laterale und sagittale Abweichung sowie der Abweichung in der Ausrichtung.

Perzeptron, welches die höchste Ungenauigkeit am Zielpunkt aufweist. Obwohl es zu erwarten wäre, dass bei der Pfadgenauigkeit ähnliche Ergebnisse wie beim Vergleich der reinen Funktionsapproximatoren entstehen, sind Unterschiede zu beobachten. Diese Unterschiede können dadurch erklärt werden, dass die Funktionsapproximatoren ähnliche Fehler ausfweisen und sich diese Approximationsfehler unterschiedlich auswirken können.

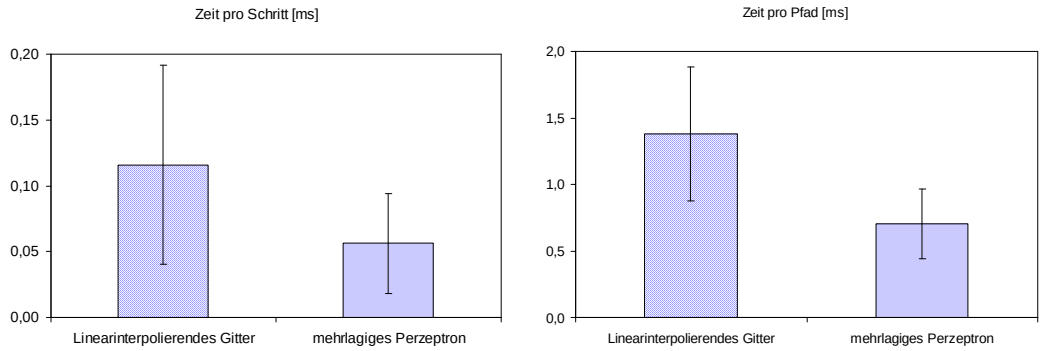

Abbildung 7.23: Benötigte Zeit für die Berechnung einzelner Schritte (links) und ganzer Pfade (rechts). Das linearinterpolierende Gitter ist in beiden Vergleichen langsamer als das mehrlagige Perzeptron. Das k-nächste-Nachbarn-Verfahren ist hier nicht abgebildet, da die Dauer zur Berechnung eines Schrittes durchschnittlich 6 ms und die zur Berechnung eines ganzen Pfades 80 ms beträgt.

In Abbildung 7.23 wird die durchschnittlich benötigte Zeit zur Berechnung eines einzelnen Schrittes sowie zur Berechnung eines ganzen Pfades dargestellt. Gemessen wurden die angegebenen Zeiten auf einem Intel Core 2 Duo T8300 bei 2.4GHz unter Windows 7. Es ist leicht ersichtlich, dass das entwickelte k-nächste-Nachbarn-

Verfahren extreme Laufzeiten aufweist. Durch Verwenden des linearinterpolierenden Gitters reduziert sich die Laufzeit erheblich, doch benötigt das Gitter viel Arbeitsspeicher. Das mehrlagige Perzeptron weist die schnellste Laufzeit auf und benötigt zudem deutlich weniger Speicher, während die Genauigkeit des MLPs gegenüber dem Gitter nicht wesentlich abnimmt. Da die benötigte Rechenzeit die wichtigste Charakteristik ist, wurde in der Implementierung das mehrlagige Perzeptron verwendet.

7.5.5 Umsetzung in der verwendeten Software

Die verwendete Software ist in drei unterschiedliche Ebenen strukturiert:

- Ebene $L2$, welcher grundlegende Strategien implementiert, wie z.B. Positionieren und Standardspielverhalten
- Ebene $L1$, beinhaltet Fußballverhalten für einen einzelnen Roboter, wie sich hinter dem Ball positionieren, kicken oder den Ball zu suchen.
- Ebene $L0$, implementiert Bewegungen, wie das Gehen oder die Kickbewegung.

Auf diesen Ebenen werden unterschiedliche Verhalten unabhängig voneinander ausgeführt. Die Schrittplanung wird auf Ebene $L1$ angesiedelt. Weitere wichtige Verhalten auf dieser Ebene sind:

- „GoBehindBall“, das den Roboter für einen Schuss hinter dem Ball positioniert.
- „DribbleBallToTarget“, einem reaktiven Verhalten, das den Ball dribbelt.
- „KickBallToTarget“, das nach der richtigen Positionierung hinter dem Ball den Ball Richtung Tor schießt.
- „SearchBall“, das den Ball sucht, wenn er nicht gesehen wird.
- „AvoidObstacle“, das versucht Hindernissen auszuweichen.

Nach der Evaluierung der Verhalten auf einer Ebene werden ihre Ergebnisse mit ihrer Aktivierung gewichtet, aufsummiert und an die nächste Ebene weitergereicht. Jedes Verhalten kann unterschiedliche Ergebnisse festlegen. In der Software werden diese Ergebnisse Aktuatoren genannt. Es gibt eine ganze Reihe von Aktuatoren pro Ebene. Für die Implementierung der Schrittplanung sind die folgenden beiden Aktuatoren wichtig:

- „GaitControlVector“, für die Festlegung des Gangsteuerungsvektor
- „GazeOrientation“, zum Einstellen der Kopfausrichtung in Radiant

Ein Verhalten ist eine Klasse, die im Wesentlichen aus zwei überschriebenen Funktionen besteht:

- Die Aktivierungsfunktion, die errechnet, wie stark das Verhalten aktiv ist. Die Aktivierung kann einen Wert zwischen 0 und 1 annehmen. Sind mehrere Verhalten aktiv, werden deren Wünsche gewichtet aufsummiert. Im Normalfall ist auf Ebene $L1$ allerdings stets nur ein Verhalten aktiv, mit Ausnahme der „AvoidObstacle“. Daher wählt die Aktivierungsfunktion in der Regel zwischen aktiv 1 und inaktiv 0 und gibt somit die Bedingung an, mit der das Verhalten aktiv wird und die Zielfunktion ausgeführt wird.
- Die Zielfunktion, die verschiedene Werte in die Aktuatoren schreiben kann. Hier wird das eigentliche Verhalten implementiert.

Die einzelnen Verhalten werden in einen Verhaltensmanager eingebunden. Hier besteht die Möglichkeit, andere Verhalten bei Aktivierung zu hemmen oder von anderen Verhalten gehemmt zu werden. Der Manager achtet dabei darauf, dass keine zirkulären Bezüge entstehen.

Das Schrittplanungsverhalten benötigt eine Bedingung, wann es aktiv wird. Diese wird wie oben beschrieben in der Aktivierungsfunktion der Klasse implementiert. Das Verhalten soll aktiv werden, wenn die folgenden Bedingungen erfüllt sind, siehe Abblidung 7.24:
Sei die egozentrische Repräsentation des Balls $B = (b_x, b_y)$ und das Schussziel $T = (t_x, t_y)$ gegeben und definiere $Winkel(Vektor)$ den Winkel zwischen y-Achse und dem angegebenen Vektor.

- Der Roboter ist Feldspieler
- Der Ball wird gesehen
- Das Ziel soll sich vor dem Roboter befinden

$$Winkel(T) < \frac{\pi}{2} \tag{7.43}$$

- Der Roboter muss sich im Definitionsbereich des A*-Algorithmus aus Abschnitt 2.3 befinden.

$$Winkel(T - B) - Winkel(B) < \frac{\pi}{2} - 0.2 \tag{7.44}$$

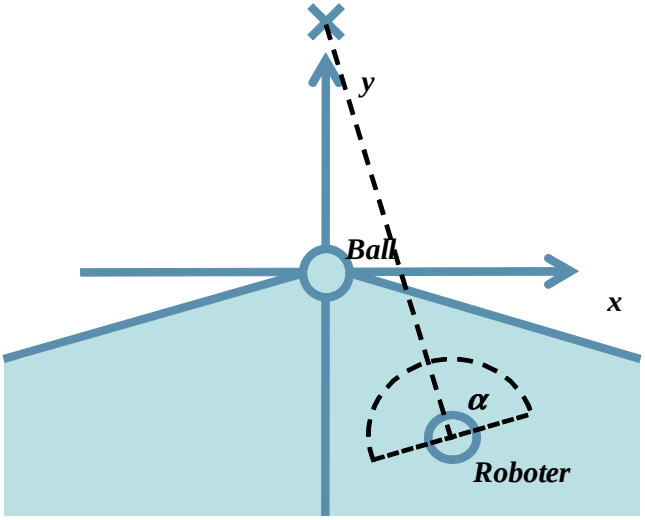

Abbildung 7.24: Das Verhalten der Schrittplanung aktiviert sich, wenn das Ziel vor dem Roboter liegt, also der Roboter in dem Winkelbereich α ausgerichtet ist und er sich im Koordinatensystem des Balles in dem dunklen Bereich befindet.

In der Zielfunktion der Klasse wird das eigentliche Verhalten implementiert. Hier können Aktuatoren gesetzt werden, die das Verhalten des Roboters beeinflussen. Wie bereits erwähnt, sind die beiden interessanten Aktuatoren in diesem Kontext der Aktuator „GaitTargetVector“ sowie der Aktuator „GazeOrientation“.

Die Blickorientierung errechnet sich in Abhängigkeit der egozentrischen Ballposition $B = (b_x, b_y)$ wie folgt:

$$\alpha = max(min(Winkel(b), 1), -1) \tag{7.45}$$

mit $Winkel(Vektor)$ als der Winkel zwischen dem Vektor und der y-Achse. Der Roboter soll immer Richtung Ball blicken, da der Distanzfehler in der Computervision zu den Seiten zunimmt. Dabei ist der Blickwinkel auf 1 Rad pro Seite begrenzt.

Zur Berechnung des Gangsteuerungsvektors wird eine Zielpose $z = (z_x, z_y, z_\theta)$ benötigt. Da die Zielpose mit dem linken Fuß angesteuert wird, kann die Zielposition $Z = (z_x, z_y)$ als Ballposition $B = (b_x, b_y)$ definiert werden. Als Ausrichtung des Balls wird die Zielrichtung definiert. Die Zielrichtung ergibt sich aus dem Vektor zwischen der Ballposition B und dem Ballziel T. Damit definiert sich das Ziel als

$$z_x = b_x \tag{7.46}$$
$$z_y = b_y \tag{7.47}$$
$$z_\theta = Winkel(B - T) \tag{7.48}$$

Nach Berechnung des Ziels wird die Situation in das Koordinatensystem des Algorithmus abgebildet. In der Computervision werden alle Posen egozentrisch dargestellt, d.h. im Koordinatensystem des Roboters. In diesem Koordinatensystem ist die Roboterpose $r = (r_x, r_y, r_\theta) = (0, 0, 0)$. Die Roboterpose, die Ballposition und gegebenenfalls Hindernisse müssen in das auf den Ball zentrierte Koordinatensystem des Algorithmus transformiert werden. Die transformierte Roboterpose ergibt sich:

$$(r_x, r_y) = R(-z_\theta) \cdot (-z_x, r_y)^T \tag{7.49}$$

$$r_z = -z_\theta \tag{7.50}$$

mit $R(\alpha)$ als Rotationsmatrix

Da der Algorithmus mit dem Mittelpunkt vom Fuß die Position anläuft, die Roboterpose des Frameworks allerdings auf dem Zentrum des Roboters liegt, muss die Roboterpose verschoben werden. Zum einen muss diese in Vorwärtsrichtung verschoben werden, da zu dem die Computervision einen systematischen Fehler in der Entfernungsschätzung macht. Dazu wird ein Ballanlauf durchgeführt und das Minimum der Entfernungsschätzung y_{min} betrachtet. Diese gibt an, wann der Ball vom Roboter bewegt wird. Also muss die Pose um diese Entfernung nach hinten verschoben werden. Zum anderen muss die Pose um die Schrittbreite des Roboters in seitlicher Richtung verschoben werden. Dazu wird das Schrittmodell m mit einem Nullschritt GCV $g = (0, 0, 0)$ ausgewertet. Dies ergibt die Differenz der Fußpositionen im Stillstand. Die Differenz der seitlichen Komponenten der Fußpositionen ergibt den Abstand der Füße. Die Hälfte des Abstandes ist die Position des Zentrums des Roboters. Um diesen muss die Pose verschoben werden. Damit wird folgende Translation $v = (v_x, v_y, v_\theta)$ auf die Roboterpose des Frameworks angewendet: Sei

$$m(g, s) : GCV \to \mathbb{R}^3 \tag{7.51}$$

das Modell in Abhängigkeit eines Gangsteuerungsvektors g, Standfuß $s \in \{l :=$ linker Fuß $, r :=$ rechter Fuß$\}$ und definiere $g_0 = (0, 0, 0)$ den Null-GCV, dann ergibt sich die Translation v als

$$v = (\frac{m_x(g_0, l)}{2}, y_{min}, 0) \tag{7.52}$$

Das mehrlagige Perzeptron wird für den Eingangsvektor bestehend aus der umgerechneten Roboterposition und dem momentanen Gangsteuerungsvektor ausgewertet. So erhält man die zwei Gangsteuerungsvektoren g_l für den linken Schritt und g_r für den rechten Schritt. Anhand der Gangphase ϕ wird berechnet, ob als Nächstes ein linker oder ein rechter Schritt ausgeführt wird. Die Gangphase wird vom zentralen Mustergenerator erzeugt und liegt dem gesamten „DynamicGait“ zugrunde. Hierbei handelt es sich um ein sinusoides Signal. Bei ungefähr $\phi = \pi$ setzt das rechte Bein auf, also beginnt der linke Schritt. Bei $\phi = 0$ beginnt der rechte Schritt.

Der GCV wird in einem kleinen Fenster um diese Zeitpunkte vom „DynamicGait“-Verhalten abgeholt. Um diese Zeitpunkte einfach zu treffen, kann der Wechsel um $\frac{\pi}{2}$ phasenverschoben werden. Also ist der verwendete GCV g

$$g = \begin{cases} g_r, & \text{wenn } -\frac{\pi}{2} \leq \phi \leq \frac{\pi}{2} \\ g_l, & \text{sonst} \end{cases} \tag{7.53}$$

Da diese von einem mehrlagigen Perzeptron berechnet werden, kann nicht gewährleistet werden, dass sie allen Beschränkungen genügen. Daher muss der berechnete GCV g limitiert werden. Dies geschieht durch Ausführung der bereits im „Dynamic Gait“-Verhalten enthaltenen Limitationsroutine.

8 Ergebnisse

8.1 Einleitung

In diesem Kapitel werden die Ergebnisse des implementierten Algorithmus bewertet. Zunächst wird der Schrittplaner mit dem vorhandenen reaktiven Verhalten verglichen. Dazu wird erst einmal der Aufbau des Versuches erläutert, um zu zeigen, wie die Daten beschafft wurden. Die Erklärung der Daten folgt ebenfalls in diesem Abschnitt. Im Anschluss daran werden die Ergebnisse zusammen getragen und diskutiert. Dazu kann ermittelt werden, wie sehr sich einzelne Merkmale verbessert haben. In dem nächsten Abschnitt werden die verschiedenen Trajektorien verglichen. Zum Schluss wird die Stabilität der Pläne debattiert.

8.2 Aufbau

Die Leistungsfähigkeit der Schrittplanung kann durch einen Vergleich mit dem vorhandenen Verfahren bewertet werden. Das vorhandene Verhalten „DribbleBallToTarget“ ist ein reaktives Verhalten, welches den Ball Richtung Tor dribbelt. Dieses Verhalten besitzt einige Parameter, die manuell eingestellt werden müssen. Das reaktive Verhalten wurde für diesen Test so eingestellt, dass es den Ball mit dem linken Fuß anläuft und es wurde viel Zeit in die Optimierung investiert, um das neue Verhalten mit dem bestmöglich konfigurierten Verhalten zu vergleichen. Das reaktive Dribbelverhalten ist besser eingestellt als zum Zeitpunkt der Weltmeisterschaft in Singapur, bei welcher unser Team den Weltmeistertitel erreichte. Zu Beginn wurden 15 verschiedene Startsituationen, bestehend aus einer Roboterpose und einer Ballposition, definiert. Abbildung 8.1 zeigt die Startkonfigurationen.

Drei Roboterposen befinden sich ungefähr auf der Mittellinie und sind in Richtung Tor ausgerichtet, zwei weitere, die jeweils nach innen ausgerichtet sind, wurden an den äußeren Positionen platziert. Es befinden sich drei Ballpositionen in Höhe des Elf-Meter-Punktes, eine in der Mitte und die anderen beiden rechts und links davon. Aus allen möglichen Kombinationen der fünf Roboterposen und der drei Ballpositionen, ergeben sich 15 Konfigurationen. Für jede Konfiguration wurden un-

gefähr sieben Anläufe mit der Motion-Capture-Anlage aufgenommen. Beschädigte Aufnahmen, in denen die Motion-Capture-Anlage wichtige Trackingpunkte verloren hat, mussten entfernt werden. So entstanden für beide Verfahren jeweils ca. 100 Anläufe.

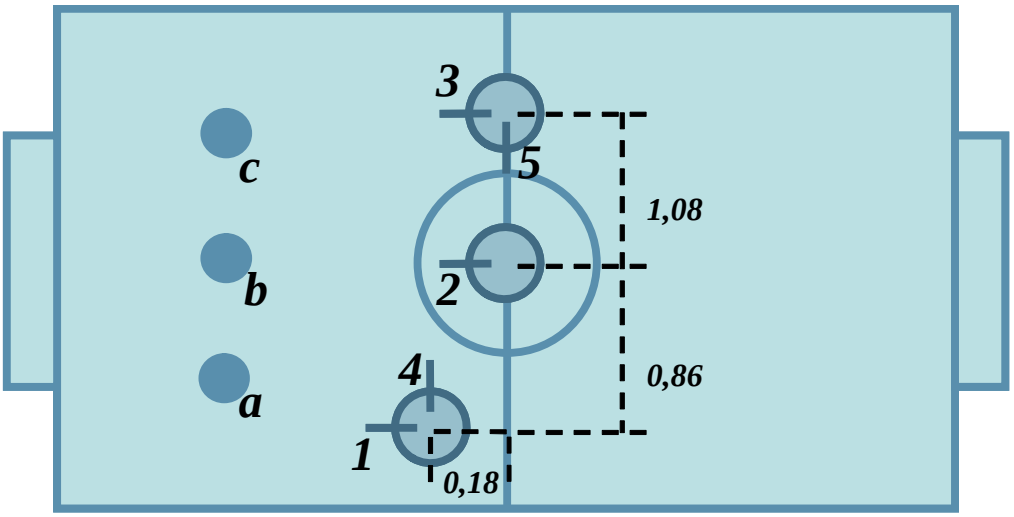

Abbildung 8.1: Die Kombinationen der Startsituationen aus Roboterpose [1..5] und Ballposition [a,b,c].

Es wurden Markergruppen an den Füßen, der Hüfte und dem Ball befestigt. Es wurde versucht, diese asymmetrisch zu gestalten, um es der Motion Capture Anlage zu ermöglichen, die einzelnen Gruppen gut voneinander zu unterscheiden. Zu den Markerpositionen von der Motion Capture Anlage wurden folgende Werte vom Roboter zeitsynchron aufgenommen:

- Ein Zeitstempel für jede Aufnahme.
- CMD: Das Startsignal des Roboters. Ist der Roboter pausiert, ist CMD gleich 1, läuft der Roboter an, ist CMD gleich 0.
- Der aktuell anliegende Gangsteuerungsvektor des Roboters.
- Ballwinkel, Balldistanz: die egozentrische Position des Balles, wie sie vom Roboter wahrgenommen wird, angegeben in Polarkoordinaten, bestehend aus Winkel und Distanz. Diese Ballposition weicht erwartungsgemäß von der realen Position des Balls ab.
- Zielwinkel, Zieldistanz: die egozentrische Position des Ballzieles. Diese Position ist im verwendeten System die Mitte des Tores.

Aus der gesamten Aufnahme wurden die einzelnen Anläufe extrahiert, indem der Startzeitpunkt, der Zeitpunkt des Ballkontaktes sowie ein Endzeitpunkt definiert wurden. Größtenteils konnten diese automatisch errechnet werden.

Für die automatische Erkennung ist der Startpunkt definiert durch den Wechsel des CMD-Signals von 1 auf 0. Danach wird der Zeitpunkt des Ballkontaktes bestimmt, indem der Zeitpunkt gesucht wird, an dem sich der Ball um einen festgelegten Schwellwert seit der letzten Aufnahme bewegt hat. Dieser Schwellwert liegt bei 0.005 Meter. Das Hauptproblem bei der Erkennung des Ballkontakts ist, dass der Ball in einigen Aufnahmen nicht erkannt und mit dem rechten Fuß verwechselt wurde. Da dies von einer Aufnahme auf die andere passierte, wurde jeder Schwellwert dadurch überschritten. Dadurch schlug die automatische Erkennung fehl und es musste manuell nachgeholfen werden, indem die Aufnahmen mit der falschen Ballposition übersprungen wurden. Der Endzeitpunkt wird nach 100 Aufnahmen vom Zeitpunkt des Ballkontaktes festgesetzt, was einem Zeitraum von einer Sekunde entspricht.

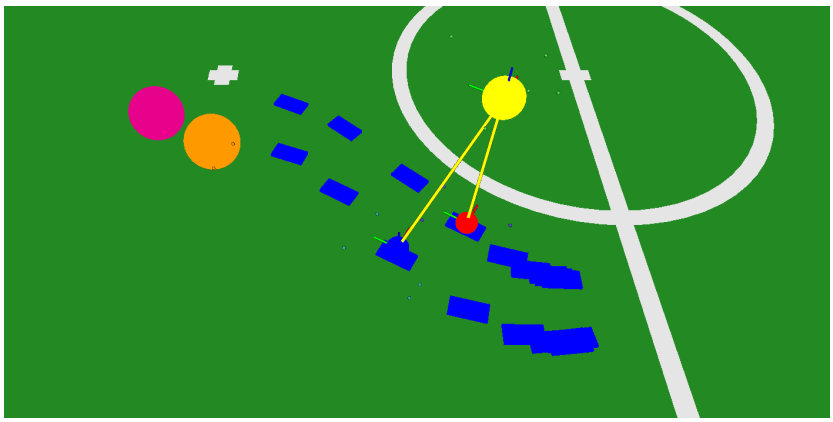

Abbildung 8.2: Visualisierung der Motion-Capture-Daten mit Visualisierung des Roboters, seinen Füßen, der Schritte, die er in dem betrachteten Anlauf zum Ball geht (blau), den Ball (orange) und den gesehenen Ball (magenta) .

An einigen Stellen musste manuell korrigiert werden, da durch fehlerhafte oder fehlende Trackingpunkte oder ganz fehlende Sequenzen in den Aufnahmen die automatische Erkennung fehlschlug. Teilweise war die Festlegung des Endzeitpunktes fehlerhaft, da der Ball bereits für die nächste Aufnahme wieder bewegt wurde oder aus anderen Gründen seine Position veränderte. Dies musste durch Plausibilitätsprüfungen erkannt und anschließend durch eine Verschiebung des Endzeitpunktes behoben werden.

8.3 Ergebnisse

Es wurden folgende Kennzahlen zur Bewertung der Verfahren ermittelt.

- Ballgeschwindigkeit v:
  Die Ballgeschwindigkeit ist definiert als der Differenzenquotient aus der Differenz zwischen der Ballposition zum Endzeitpunkt und zum Kontaktzeitpunkt sowie der Differenz der beiden Zeitpunkte in Sekunden.
  Definiert b den Zeitpunkt des Ballkontaktes und e den Endzeitpunkt der Ballbewegung, sowie d_i die Balldistanz in Meter und t_i die Zeit in Sekunden zum Zeitpunkt i, dann ist die Ballgeschwindigkeit definiert als:

$$v := \frac{d_e - d_b}{t_e - t_b} \tag{8.1}$$

- Winkelabweichung σ:
  Die Differenz zwischen Zielwinkel α_{target} („ballTargetAngle“) und realisiertem Schusswinkel definiert die Winkelabweichung.
  Mit den obigen Definitionen und mit B_i als Position des Balles zum Zeitpunkt i wird die Winkelabweichung σ in Grad definiert als:

$$\sigma := |Winkel(B_e - B_b) - \alpha_{target}| \tag{8.2}$$

- Schrittanzahl n:
  Die Schrittanzahl zählt die Schritte zwischen Startzeitpunkt und dem Zeitpunkt des Ballkontaktes.

Wie in den Abbildungen 8.4, 8.5 und 8.3 dargestellt, kann größtenteils eine Verbesserung beobachtet werden. Die Stärke des Algorithmus ist die Winkelgenauigkeit,

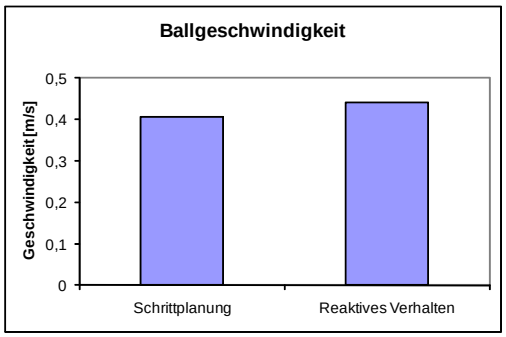

Abbildung 8.3: Vergleich der durchschnittlichen Ballgeschwindigkeit der Schrittplanung und des reaktiven Verhaltens.

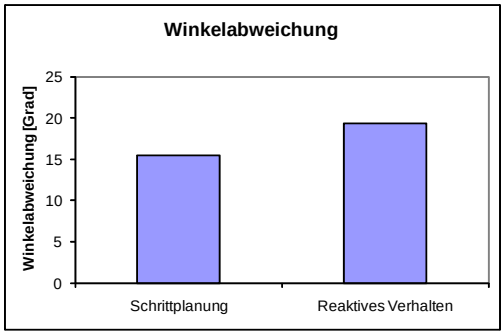

Abbildung 8.4: Verbesserung der durchschnittlichen Winkelabweichung durch die Schrittplanung im Vergleich zum reaktiven Verhalten.

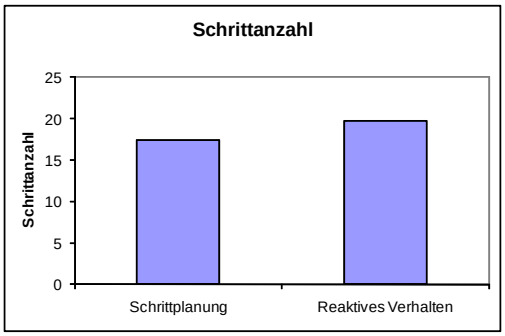

Abbildung 8.5: Vergleich der durchschnittlichen Schrittanzahl die für den Anlauf zum Ball von der Schrittplanung und dem reaktiven Verhalten benötigt wurden.

dargestellt in Abbildung 8.4. Während das reaktive Verhalten durchschnittlich 23 Grad Abweichung zwischen Ballbewegungswinkel und Zielwinkel hat, reduziert sich diese Abweichung auf ungefähr 18 Grad, das entspricht einer Reduktion um knapp 22 %. Die Schrittanzahl reduziert sich um 11,5 % von 19,7 Schritten im Durchschnitt über alle Anläufe bei dem vorhandenen reaktiven Verhalten auf 17,5 Schritte bei der Schrittplanung (Abbildung 8.5).

Wie in Abbildung 8.6 dargestellt, benötigt der Schrittplaner verglichen mit dem reaktiven Verhalten entweder gleich viele oder deutlich weniger Schritte. Intuitiv, bei einfachen Situationen, wie zum Beispiel 1a, 2a und 2b, in denen der Roboter lediglich geradeaus laufen muss, benötigt der Schrittplaner eine vergleichbare Schrittanzahl wie das reaktive Verhalten. Mit anderen Worten, in den einfachen Fällen ist das reaktive Verhalten genauso gut. In vielen anderen Fällen jedoch benötigt der Schrittplaner bis zu sechs Schritte weniger als das reaktive Verhalten. In einem einzigen Fall (5b) schlägt das reaktive Dribbelverhalten die Schrittplanung in der Anzahl der Schritte doch auf Kosten der Winkelgenauigkeit. Bei genauer Analyse

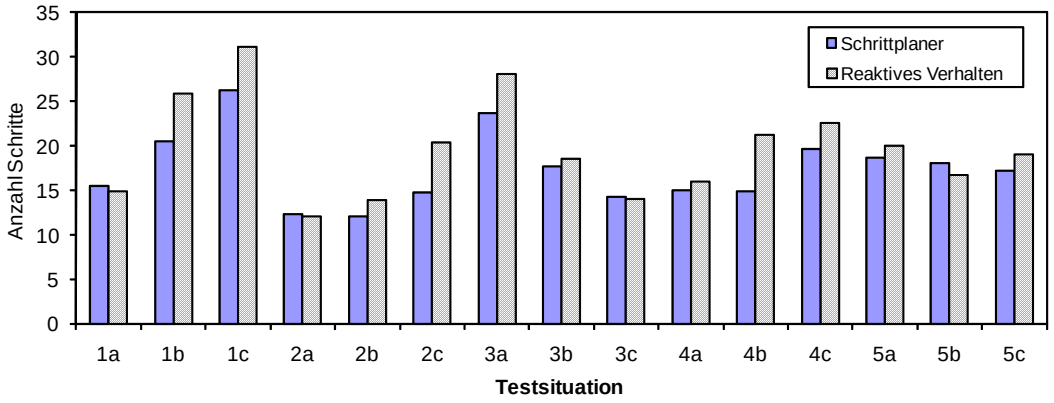

Abbildung 8.6: Vergleich der durchschnittlichen Anzahl der Schritte, die das reaktive Verhalten und der Schrittplaner für die unterschiedlichen Testsituationen benötigt haben.

des Anlaufs in diesen Testdurchläufen wird deutlich, dass das reaktive Verhalten in der Nähe des Balles sich nicht in Richtung Tor dreht und somit einen erheblichen Winkelfehler in diesem Test erzielt.

8.4 Trajektorienvergleich

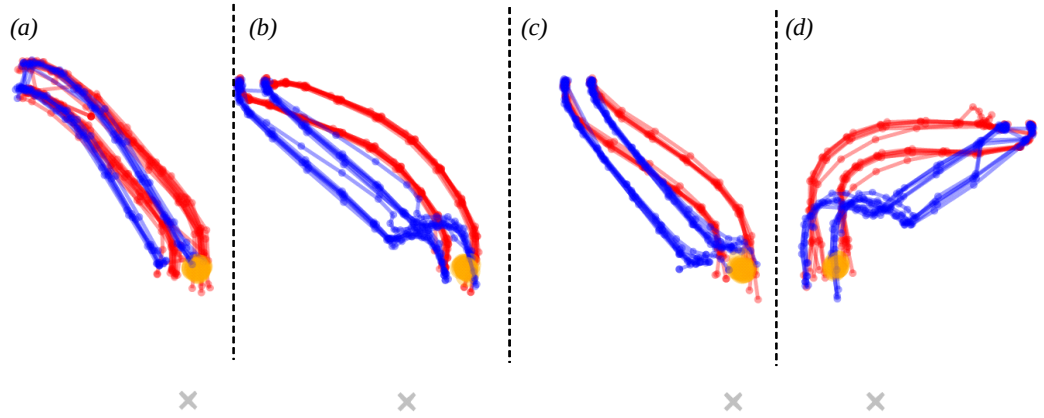

Abbildung 8.7: Bahnen, die das reaktive Verhalten (blau / dunkel) und die Schrittplanung (rot / hell) mit dem realen Roboter abgelaufen ist.

Mit der Motion-Capture-Anlage wurden mehrere Anläufe aufgenommen, sowohl von der erarbeiteten Schrittplanung als auch von dem vorhandenen reaktiven Verhalten.

Die aufgezeichneten Schritte wurden anhand der Gangphase identifiziert. Die Trajektorien der linken und der rechten Schritte beider Verfahren wurden für jeden Anlauf visualisiert. Diese Visualisierung ermöglicht es nun die Trajektorien zu analysieren. Bisher wurden die Trajektorien des reaktiven Verhaltens noch nicht analysiert. In Abbildung 8.7 wurden alle verwendbaren Pfade der Anläufe von vier unterschiedlichen Aufstellungssituationen dargestellt.

Anhand der Darstellungen in 8.7 (b) wird deutlich, dass das rein reaktive Verhalten zuerst gerade zum Ball läuft und sich dann unmittelbar vor diesem ausrichtet. Dieses Verhalten ist unabhängig von der Position und Ausrichtung, wie auch in (d) deutlich zu sehen. In (c) kann man erkennen, dass das reaktive Verhalten nicht immer in der Lage ist sich richtig auszurichten. Das geht soweit, dass es in (a) direkt gegen den Ball läuft und den Ball in eine falsche Richtung dribbelt. Dadurch kommt das reaktive Verhalten in diesen Fällen deutlich schneller zum Ball als die Schrittplanung, die auf die Ausrichtung Wert legt. Diese Erkenntnis erklärt auch, warum die Schrittanzahl der Schrittplanung nicht so deutlich gesenkt wurde, wie es vielleicht erwartet werden könnte. In allen vier Darstellungen läuft die Schrittplanung stets sehr glatte Pfade ab.

Wie erwartet, liegen die Schrittpositionen einer Serie von Anläufen der Schrittplanung aufeinander, da der Versuchsaufbau die gleiche Startpose des Roboters und die gleiche Ballpose vorschreibt. Gäbe es eine deutliche Streuung, wäre das ein Indiz für eine äußerst unstabile Planung. Diese Instabilität könnte aus Sensordatenrauschen herrühren.

8.5 Schrittanzahl

Für die Auswertung eines Anlaufes werden zwei verschiedene Schrittanzahlen benutzt. Zum einen die Anzahl der realen, noch zu gehenden Schritte zum anderen die Schritte, die zum gleichen Zeitpunkt zum gesehenen Ball geplant werden. Diese beiden Schrittanzahlen werden gezählt und im Anschluss daran miteinander verglichen. Die Differenz ist der Fehler, den die Schrittplanung macht. Abbildung 8.8 zeigt diesen Fehler in Abhängigkeit der Entfernung zum Ziel. Eine Abhängigkeit des Fehlers von der Entfernung wird deutlich. Je näher der Roboter dem Ziel kommt, desto genauer werden die Pläne. Das liegt unter anderem daran, dass die geschätzte Balldistanz mit der Entfernung überproportional zunimmt. Daher erscheint das Ziel mit steigender Entfernung weiter weg, als es tatsächlich ist. Die Ungenauigkeit am Ziel kann dadurch erklärt werden, dass der vom Roboter gesehene Ball auch am Ziel von Bild zu Bild variiert. Ist der Ball zum Zeitpunkt der Berechnung zu weit weg, werden noch zwei Schritte berechnet, obwohl der Roboter den Ball bereits berührt. Weiter fällt die hohe Varianz der Schrittanzahl auf. Zu dieser Varianz führen zwei

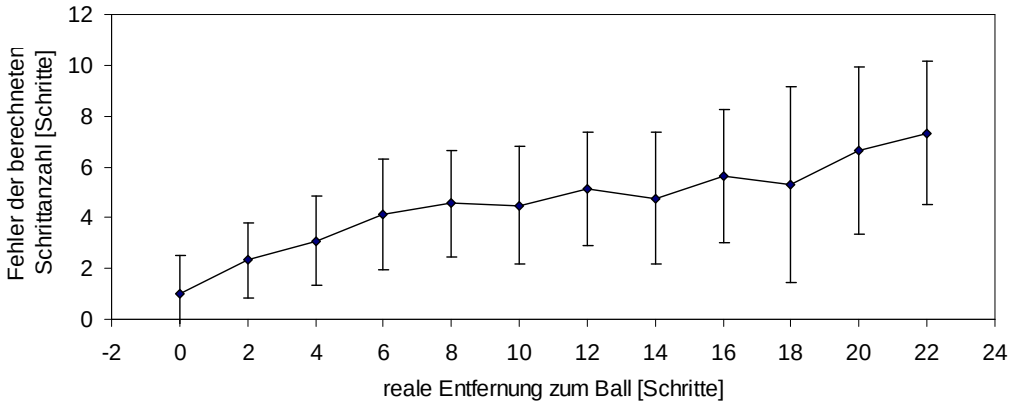

Abbildung 8.8: Berechnungsfehler der Schrittplanung in Abhängigkeit der Entfernung zum Ziel

Effekte. Zum einen geht die Instabilität des gesehen Balles in die Schrittpläne ein. Zum anderen nimmt die Schrittanzahl mal vor und mal nach dem Zeitpunkt des tatsächlichen Schrittes ab.

8.6 Begründung der Ballgeschwindigkeit

Aus den Aufnahmen der Versuche zur Evaluierung der beiden Verhalten wurde der Zeitpunkt der Ballberührung durch den Roboter genau bestimmt. Zu diesem Zeitpunkt wurde die Gangphase und der Gangsteuerungsvektor extrahiert, sowie das Geschwindigkeitsprofil des Balls über die Einzelaufnahmen hinweg. In dem Geschwindigkeitsprofil, Abbildung 8.9, wurde eine Gerade mit linearer Regression eingebettet, um Messungenauigkeiten zu entfernen, denn oft beginnt das Profil mit Schwankungen, doch es stabilisiert sich schnell wieder. Irgendwann verliert die Anlage den Ball und es können keine Aussagen über die Geschwindigkeit gemacht werden. In dem stabilen Teil des Profils wurde eine Gerade eingebettet. Zwar ist der Geschwindigkeitsverlauf nicht linear, doch als Näherung ist diese Approximation ausreichend. Die Ballgeschwindigkeit ergibt sich als der Wert dieser Geraden zum Zeitpunkt der Berührung.

Abbildung 8.10 zeigt links die Ballgeschwingkeit in Abhängigkeit der y-Komponente des Gangsteuerungsvektors, welcher die aktuelle Geschwindigkeit des Roboters darstellt. In dem Intervall $g_y \in [0,5; 0,9]$, in dem Datenpunkte enthalten sind, wird eine Abhängigkeit deutlich sichtbar. Unter $g_y = 0,5$ sind keine Datenpunkte enthalten, da in den Aufnahmen der Verhaltentests kein Anlauf mit so geringer Geschwin-

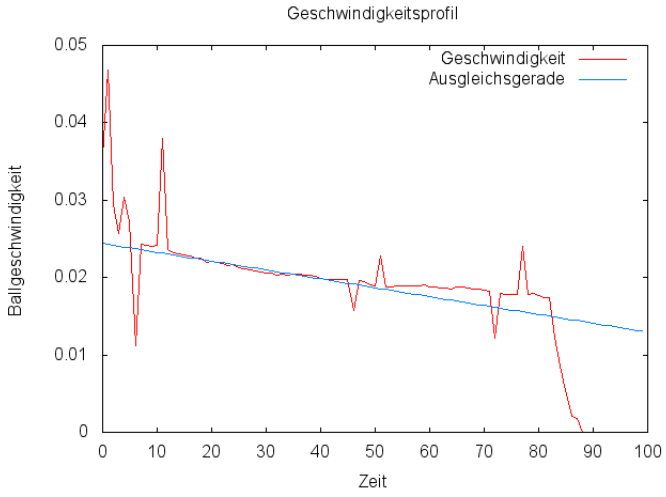

Abbildung 8.9: Geschwindigkeitsprofil eines Anlaufes: Das Profil beginnt oft mit Schwankungen und stabilisiert sich über die Zeit. Irgendwann verliert die Motion-Capture-Anlage den Ball.

digkeit vor kam. Um dies zu evaluieren müssten eigene Aufnahmen hierzu erstellt werden. Der Korrelationskoeffizient bestätigt die Vermutung. Demnach korrelieren Gangsteuerungsvektor und Ballgeschwindigkeit mit $0,781$

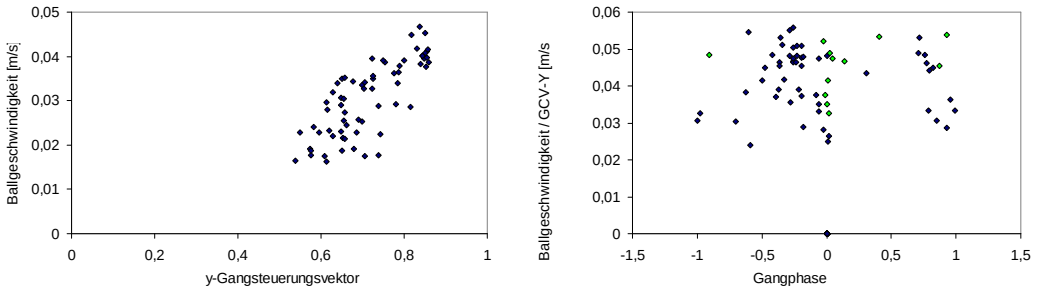

Abbildung 8.10: Links Ballgeschwindigkeit in Abhängigkeit der y-Komponente des Gangsteuerungsvektors g, rechts Ballgeschwindigkeit

Die rechte Abbildung zeigt die Ballgeschwindigkeit in Abhängigkeit der Gangphase ϕ. Die Ballgeschwindigkeit wurde durch die y-Komponente des Gangsteuerungsvektors dividiert, um den bereits erkannten Zusammenhang herauszufiltern. Manchmal berührt der Roboter beim Anlauf den Ball leicht. Dabei rollt der Ball langsam nach vorne und der Roboter trifft ihn mit dem nächsten Schritt gut. Diese Schüsse werden in der Abbildung grün dargestellt mit der Gangphase der ersten Berührung und der Ballgeschwindigkeit des Nachschusses. Somit wird gezeigt, dass diese ungünstigen Gangphasen ebenso schnelle Ballgeschwindigkeiten hevoru-

fen können. Der Raum ist nicht repräsentativ ausgefüllt, doch es lässt sich erahnen, dass keine klare Abhängigkeit zwischen Gangphase ϕ und der Ballgeschwindigkeit vorhanden ist.

Demnach ist es sinnvoll eine hohe Geschwindigkeit zu forcieren, doch der Geschwindigkeitsgewinn durch das Treffen einer genauen Gangphase ist gering.

9 Zusammenfassung und Diskussion

9.1 Zusammenfassung

In dieser Diplomarbeit wurde eine hierarchische Schrittplanung erarbeitet, welche dem Roboter ermöglicht, aus dem Laufen heraus den Ball zu dribbeln. Das System besteht aus vier Planungsebenen und einer ausführenden Ebene. Die erste Ebene ist die Aktionsplanung, welche eine mögliche Aktion, wie zum Beispiel einen Torschuss, auswählt.

Für die zweite Ebene, die Pfadplanung, wurden Viapunkte entwickelt. Diese ermöglichen es dem Roboter Hindernisse zu umgehen. Dafür wird zuerst ein Pfad wie ursprünglich geplant. Sollte dieser mit einem Hindernis kollidieren, werden neben dieses Hindernis mehrere Viapunkte gesetzt, über die der Roboter neue Pfade plant, den besten davon auswählt und diesen beschreitet.

Die nächste Ebene ist die Schrittplanung, welche den eigentlichen Schwerpunkt dieser Arbeit darstellt. Für die Schrittplanung wurde aus Motion-Capture-Daten ein Schrittmodell gelernt. Das erlernte Schrittmodell bildet Steuerbefehle auf kartesische Schrittkoordinaten ab. Mit diesem Modell ist es möglich die für den Ballanlauf benötigten Steuerbefehle automatisiert zu erzeugen. Hierfür wurden mit der Schrittplanung viele Pfade unter Verwendung des A*-Algorithmus berechnet. Zur Verbesserung der Online-Laufzeiten des Systems wurde ein Funktionsapproximator eingesetzt. Drei unterschiedliche Funktionsapproximatoren wurden getestet und miteinander verglichen. Diese waren das k-nächste-Nachbar-Verfahren, ein linear interpolierendes Gitter und das mehrlagige Perzeptron. Es stellte sich heraus, dass sich die Approximatoren ähnlich verhalten. Allerdings wies das mehrlagige Perzeptron den wenigsten Speicherplatz sowie die geringste Laufzeit auf.

Das System wurde implementiert und auf dem realen Roboter getestet. Es stellte sich heraus, dass das System in einer Echtzeitumgebung eingesetzt werden kann, da die benötigte Online-Laufzeit mit 0,05 ms minimal ist. Mit der Motion-Capture-Anlage wurden circa 100 Anläufe aufgenommen, sowohl für dieses System als auch für das vorhandene reaktive Verhalten. Die beiden Methoden wurden miteinander verglichen. Es wurde gezeigt, dass die Winkelabweichung und die Schrittanzahl mit dem entwickelten System verbessert wurden.

9.2 Diskussion

Wie in Abschnitt 8.6 gezeigt, kann die Ballgeschwindigkeit nicht wesentlich durch eine Optimierung der Gangphase erhöht werden. Lediglich durch eine höhere Robotergeschwindigkeit kann diese verbessert werden. Eine Möglichkeit zur Verbesserung des Systems wäre, unter Verwendung der Schrittplanung mit A*-Algorithmus, eine hohe Endgeschwindigkeit zu forcieren oder die A*-Pfade anschließend zu optimieren.

Die Gangphase kann optimiert werden, indem die letzten beiden Schritte vor dem Ballschuss online optimiert werden. Dazu wird der letzte Schritt in dem gewünschten Abstand vor den Ball gesetzt, so dass beim Schwung dieses Beines der Ball mit der richtigen Gangphase getroffen wird. Der Schritt davor wird mittig zwischen der aktuellen Fußposition und der eben definierten Schrittposition gesetzt. Die so bestimmten Positionen werden mit dem invertierten, linearen gelernten Modell in Gangsteuerungsvektoren umgerechnet. Allerdings müssen diese auf Gültigkeit geprüft werden und gegebenenfalls angepasst werden. Wird eine Beschränkung verletzt, wird das Maximum dieser Beschränkung ausgenutzt. Daraus folgt aber, dass die Gangphase nicht mehr genau getroffen werden kann. Da der Approximator in den meisten Fällen das Ziel gut erreicht, besteht die Hoffnung, dass die Beschränkungen nicht häufig verletzt werden.

Die Performance der Schrittplanung mit A*-Alogrithmus kann durch Verbesserung der Heuristik erhöht werden. Eine bessere Heuristik als die verwendete berücksichtigt die Drehung und die Trägheit des Roboters. Ein Modell, welches für eine solche Heuristik verwendet werden könnte, wurde in [30] beschrieben. Hier werden 2D-Trajektorien für Flugkörper berechnet und die Zeit zum Erreichen des Zieles mit angegeben. Dafür werden verschiedene eindimensionale Fälle betrachtet, die einzeln gesehen ein ähnliches Beschleunigungsmodell entwickeln wie in 7.3 beschrieben. Das Modell muss noch so angepasst werden, dass es immer unterschätzt, was nicht sichergestellt ist.

Danksagung

Ich bedanke mich in erster Linie bei meiner Familie, die es mir ermöglicht hat mich über meine gesamte Studienzeit hinweg unbeschwert auf mein Studium zu konzentrieren. Ich bedanke mich bei meiner Freundin Ann-Katrin und meinem Bruder Thomas für die geistige und seelische Unterstützung. Mein Dank gilt auch Prof. Dr. Sven Behnke für die Korrektur und für den positiven und aufklärenden Einfluss in der Endphase. Ich danke Prof. Dr. Rolf Klein für die Zweitkorrektur. Weiter bedanke ich mich sehr herzlich bei meinem Betreuer Marcell Missura für den tatkräftigen Beistand, den er tagtäglich leistete und die fürsorgliche und kompetente Betreuung meiner Diplomarbeit und auch dafür, dass er mir mit persönlichen Ratschlägen zur Seite stand und ein Mentor für mich war.

Literaturverzeichnis

[1] M. Lepetic, G. Klancar, I. Skrjanc, D. Matko, B. Potocnik, *Time optimal path planning considering acceleration limits*, Robotics and Autonomous Systems, vol. 45, Issues 3-4, 31.12.2003, pp. 199-210

[2] K.G. Jolly, R. Sreerama Kumar, R. Vijayakumar, *A Bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits*, Robotics and Autonomous Systems, vol. 57, Issue 1, 3.01.2009, pp. 23-33,

[3] Arechavaleta, G.; Laumond, J.-P.; Hicheur, H.; Berthoz, A., *An Optimality Principle Governing Human Walking*, In Robotics, IEEE Transactions on Robotics, vol.24, no.1, pp.5-14, Feb. 2008

[4] Arechavaleta, G.; Laumond, J.-P.; Hicheur, H.; Berthoz, A., *The nonholonomic nature of human locomotion: a modeling study*, In Biomedical Robotics and Biomechatronics, 2006. BioRob 2006. The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, pp.158-163, 20-22 Feb. 2006

[5] J-P. Laumond, G. Arechavaleta, T.-V.-A Truong, H. Hicheur, Q.-C. Pham and A. Berthoz, *The words of the human locomotion*. In Robotics Research: The 13th Symposium (ISRR-2007). Springer STAR Series.

[6] D. S. Meek, D. J. Walton, *The use of Cornu spirals in drawing planar curves of controlled curvature*, Journal of Computational and Applied Mathematics, vol 25, Issue 1, January 1989, pp. 69-78

[7] D.S. Meek, R.S.D. Thomas, *A guided clothoid spline, Computer Aided Geometric Design*, vol. 8, Issue 2, May 1991, pp.163-174

[8] D. S. Meek, D. J. Walton, *Clothoid Spline Transition Spirals* Mathematics of Computation Vol. 59, No. 199 (Jul., 1992), pp. 117-133

[9] D.J. Walton, D.S. Meek, *A controlled clothoid spline*, Computers and Graphics, Volume 29, Issue 3, June 2005, Pages 353-363

[10] Joshua H. Henrie, Doran K. Wilde, Planning, *Continuous Curvature Paths Using Constructive Polylines*, Journal of Aerospace Computing, Information, and Communication, Dec 2007, vol. 4, pp. 1143-1157.

[11] Dong Hun Shin and Sanjiv Singh, *Path Generation for Robot Vehicles Using Composite Clothoid Segments*, tech. report CMU-RI-TR-90-31, Robotics Institute, Carnegie Mellon University, December, 1990

[12] S. Behnke, J. Stuckler, M. Schreiber, H. Schulz, M. Bohnert, K. Meier, *Hierarchical reactive control for a team of humanoid soccer robots*, Humanoid Robots, 2007 7th IEEE-RAS International Conference on , pp.622-629, Nov. 29 2007-Dec. 1 2007

[13] S. Behnke, *Online Trajectory Generation For Omnidirectional Biped Walking*, In ICRA, 2006

[14] M. Zhao, J. Zhang, H. Dong, Y. Liu, L. Li and X. Su, *Humanoid Robot Gait Generation Based on Limit Cycle Stability*, In RoboCup 2008: Robot Soccer World Cup XII, 2008, pp. 403-413, Springer

[15] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada and K. Yokoi, *Biped walking pattern generation by using preview control of zero-moment point*, 2003, In ICRA, 2003, pp. 1620-1626

[16] Q. Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi and K. Tanie, *Planning Walking Patterns for a Biped Robot*, In IEEE Transactions on Robotics and Automation, 2001, Vol. 17, pp. 280-289

[17] K. Kaneko and F. Kanehiro and S. Kajita and H. Hirukawa and T. Kawasaki and M. Hirata and K. Akachi and T. Isozumi, *Humanoid Robot HRP-2*, In ICRA, 2004, pp.1083-1090

[18] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi and H. Hirukawa, *The 3D linear inverted pendulum mode: a simple modeling for a bipedwalking pattern generation*, In IROS, 2001, Vol. 1, pp. 239-246

[19] M. Friedmann, J. Kiener, S. Petters, H. Sakamoto, D. Thomas and O. von Stryk, *Versatile, high-quality motions and behavior control of humanoid soccer robots*, In Workshop on Humanoid Soccer Robots of the 2006 Humanoids, 2006, pp. 9-16

[20] J. Chestnutt and J. Kuffner, *A Tiered Planning Strategy for Biped Navigation*, In Humanoids, 2004

[21] J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba and H. Inoue, *Online Footstep Planning for Humanoid Robots*, In ICRA, 2003, pp. 932-937

[22] J. Chestnutt, M. Lau, K.M. Cheung, J. Kuffner, J.K. Hodgins and T. Kanade, *Footstep Planning for the Honda ASIMO Humanoid*, In ICRA, 2005

[23] J.S. Gutmann, M. Fukuchi and M. Fujita, *Real-time path planning for humanoid robot navigation*, In Proc. of 19th Int. Conf. on Artificial intelligence, 2005, pp. 1232-1237

[24] O. Kanoun, E. Yoshida and J.P. Laumond, *An Optimization Formulation for Footsteps Planning*, In Humanoids, 2009

[25] Y. Ayaz, T. Owa, T. Tsujita, A. Konno, K. Munawar and M. Uchiyama, *Footstep Planning for Humanoid Robots Among Obstacles of Various Types*, In Humanoids, 2009

[26] M. Vukobratovic and B. Borovac, *Zero-Moment Point - Thirty Five Years of its Life*, In journal Humanoid Robotics, 2004, pp. 157-173

[27] J. Denavit and R.S. Hartenberg, *A kinematic notation for lower-pair mechanisms based on matrices.* Journal of Applied Mechanics, vol. 22, 1955, pp. 215-221

[28] P. E. Hart, N. J. Nilsson, B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics SSC4 (2), pp. 100-107, 1968.

[29] D. E. Rumelhart, J. L. McClelland, *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 1: Foundations* The MIT Press, Cambridge, MA, 1986.

[30] M. Missura, *Berechnung einer Flugbahn in 3D über beliebige Wegpunkte* 9. October 2007

[31] A. Schmitz, M. Missura, S. Behnke, *Learning Footstep Prediction from Motion Capture* In Proceedings of RoboCup International Symposium, Singapore, June 2010.