Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Intelligente Wegfindung humanoider Roboter am Beispiel des RoboCup

Bachelorarbeit
im Arbeitsbereich Knowledge Technology, WTM
Dr. Sven Magg

Department Informatik
MIN-Fakultät
Universität Hamburg

vorgelegt von
Martin Poppinga
am
23.12.2014

Gutachter: Dr. Sven Magg
Stefan Heinrich

Martin Poppinga
Matrikelnummer: 6318650
Siebentunnelweg 5a
25469 Halstenbek

Abstract

In this thesis I will investigate the path finding of humanoid robots. It takes place in the RoboCup humanoid Kid-Size league environment. It will be tested if a combination of classical potential fields and evolutionary trained CTRNN can generate a good solution. Demanded are a correct orientation at the goal and obstacle avoidance. The tests showed that this goals are reachable. CTRNNs are capable to navigate the robot in a 2D-simulation. The potential fields are able to perform an obstacle avoidance and simple navigation tasks. The combination of both provides good results. With low needs in computation power is the system suitable for robots and other autonomous systems which work with an incomplete model of their enviroment.

Zusammenfassung

In dieser Arbeit wird die Pfadfindung humanoider Roboter untersucht. Als Kontext dient die Humanoide Kid-Size Liga des RoboCup. Dabei wird untersucht, inwiefern eine Kombination von Potential Fields und evolutionär trainierter CTRNN zur Wegfindung geeignet ist. Es wird in einer Simulationsumgebung untersucht, ob die notwendigen Aufgaben der Wegfindung, wie die korrekte Ausrichtung des Roboters oder dem Ausweichen von Gegnern, mit diesem kombinierten Ansatz gelöst werden können. Die Tests zeigten, dass diese Ziele erreichbar sind. CTRNNs lieferten eine gute Wegfindung in der 2D-Simulation. Die Potential Fields können Hindernissen ausweichen und eine einfache Navigation durchführen. Die Kombination beider Verfahren lieferte gute Ergebnisse. Mit seinem geringen Rechenaufwand ist dieses Verfahren geeignet für Roboter und andere autonome Systeme, die mit einem unvollständigen Modell ihrer Umgebung arbeiten.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Der Themenbereich der Robotik ist eins der großen Forschungsgebiete der heutigen Zeit. Im Bereich von selbständig agierenden und lernenden Systemen hat sich in den letzten Jahren viel getan. So sind intelligente Begleiter wie Smartphones allgegenwärtig und aus dem alltäglichen Leben kaum wegzudenken. Auch selbstfahrende Autos sind inzwischen so weit entwickelt, dass sie nicht mehr als utopische Zukunftsgedanken abgestempelt werden, sondern schon auf deutschen Straßen fahren [4]. Bei den Robotern, die klassisch mit dem Begriff des "Roboters"verknüpft werden, den humanoiden Robotern, zeigte sich auch großer Fortschritt in den vergangenen Jahren, sie sind aber dennoch mitten in der Entwicklung[6]. Die Idee ist es, Roboter zu entwickeln, die sich problemlos in der Welt der Menschen zurechtfinden können, während heutige praxisrelevante Systeme speziell eingerichtete Umgebungen wie Warenlager oder Fabrikhallen benötigen. Humanoide, dem Menschen nachempfundene Roboter, sollen dieses neue, komplexe Feld abdecken, indem sie zum Beispiel auf zwei Beinen laufen und so alle Bereiche erreichen können, die auch Menschen zugänglich sind. So sollen sie auch mit Objekten verschiedenster Art interagieren können. Dies ist essenziell für einen Einsatz im Haushalt als Pflegehilfe oder für andere Aufgaben bei denen der Umgang mit Menschen im Zentrum steht.

Die vorliegende Arbeit handelt im Kontext des RoboCup. Der RoboCup ("Robot Soccer World Cup")[18] ist eine internationale Initiative und Wettbewerb im Bereich der Robotik mit einem Fokus auf autonome Systeme. Es forschen Wissenschaftler und Studenten von Universitäten aus einer Vielzahl von Ländern im Bereich der Robotik, um eben diese Problemfelder zu ergründen. Der RoboCup entstand 1997 und wurde mit dem Ziel gegründet, die Forschung international vergleichbar und erlebbar zu machen und so den internationalen Forschungsaustausch voran zu bringen. So lässt sich das Spielen von Fußballrobotern als Standartproblem der Informatik ansehen, wie einst die Herausforderung ein System zu entwickeln um gegen den amtierenden Schachweltmeister zu gewinnen, was 1997 erfüllt wurde [18][13]. So ist es das selbst gesteckte Ziel des RoboCup bis 2050 gegen die amtierenden FIFA-Fußballweltmeister in einem fairen Fußballspiel nach den FIFA-Regeln zu gewinnen [18]. Es gibt jährliche Wettkämpfe in verschieden Teilen der Welt. Eine besondere Bedeutung hat dabei die jährlich in wechseln-

den Ländern stattfindende Weltmeisterschaft. Während 1997 in Japan 40 Teams zusammentrafen, waren es 2013 in Eindhoven über 3000 Teilnehmer in über 400 Teams und mehr als 40.000 Besucher[21] die die Weltmeisterschaft besuchten. Des weiteren gibt es in vielen Ländern kleinere Meisterschaften des RoboCup, die ebenfalls jährlich stattfinden und sich internationaler Beteiligung erfreuen. Diese sind zum Beispiel die GermanOpen in Magdeburg oder die IranOpen in Teheran.

Auch wenn der RoboCup als Wettkampf für den Roboterfußball gegründet wurde, bietet er inzwischen verschiedenen Liegen in unterschiedlichsten Anwendungsbereichen [18]. Die **Rescue Liga** entwickelt Roboter für Katastrophenhilfe und Rettungseinsätze, so wurde Beispielsweise auch schon im RoboCup vorgestellte Hardware während der Katastrophe in Fukushima eingesetzt [27] um Bilder aus unbegehbarem Gebiet zu machen. Auch die Menschenrettung steht im Fokus, wofür die Roboter mit entsprechender Technik wie Wärmebildkameras ausgerüstet sind. Geländegängigkeit der Hardware und Analyse von Signalen auf Hinweise von Verschütteten sind die Kernpunkte der Liga. In der **@Home Liga** wird an Haushaltsrobotern geforscht um beispielsweise körperlich eingeschränkten Menschen Aufgaben im Haushalt abzunehmen. Hier stehen Bereiche wie Objekt- und Spracherkennung im Zentrum, aber auch am Greifen und Transportieren von Gegenständen wird geforscht. In der **Logistik** und der **@Work** Liga geht es um anwendungsbezogene Aufgaben aus dem Bereich der Lagerlogistik sowie Transport und Erkennung von Arbeitsgegenständen.

Den größten Teil des Wettbewerbes bilden die klassischen Ligen des RoboCup, die Soccer bzw. Fußballligen mit ihren unterschiedlichen Unterligen. Hier wird zwischen fahrenden und den humanoiden Robotern unterschieden die sich wiederum jeweils in verschiedene Unterligen nach ihren Größen unterteilen. Jede Liga hat aufgrund ihres Reglements, die auch die Beschaffenheit der Roboter beschreiben, ihre eigene Herausforderungen.

In allen Ligen ist die Wegfindung eine der anzugehenden Problemstellungen, da sich hier beispielsweise in unbekanntem Gelände bewegt werden muss, oder sich Bedingungen in der Umgebung dynamisch verändern. Die Wegfindung beschreibt die Fähigkeit des autonomen Systems, in diesem Kontext des Roboters, einen Weg zu einem vorgegeben Ziel zu finden. Dieser Weg kann auf Eigenschaften wie die benötigte Zeit, Energiekosten, Strecke oder einfache Durchführbarkeit optimiert werden.

Die humanoiden Ligen des RoboCup Soccer bieten mit ihren Regeln ein breites Feld an Herausforderungen, da diese sich nur auf menschenähnliche Sensorik und Aktorik verlassen können. So ist es zum Beispiel nicht möglich präzise Positionsinformationen mittels Infrarot oder Lasertechnik zu gewinnen. Alle Berechnungen müssen auf dem Roboter selbst ausgeführt werden. Die vorliegenden Daten sind mitunter stark verrauscht und es fehlen teils klare und eindeutige Wegmarken die im Blickfeld des Roboters verfügbar sind. Die in diesem Sportszenario erforschten Konzepte können in vielen Bereichen der humanoiden Robotik verwendet werden.

1.1 Motivation

1.1.1 Problemstellung

Einen optimalen Weg für einen Roboter in den humanoiden RoboCup Soccer Ligen zu finden, ist nicht trivial. So sollte möglichst schnell der Ball erreicht werden ohne auf dem Weg dahin mit ihm oder mit einem Hindernis zu kollidieren. Am Ball benötigt der Roboter die richtige Ausrichtung. Erschwerend kommt hinzu, dass oftmals nicht die komplette Umgebung bekannt ist und sich Hindernisse sowie Ball bewegen.

1.1.2 Neuer Ansatz und Ziel dieser Arbeit

Ziel dieser Arbeit wird es sein, eine Möglichkeit zu entwickeln, mit welcher der Roboter in einer simulierten Umgebung einen möglichst effizienten Weg zum Ball findet und sich so am Ball ausrichten kann, dass ein Schuss auf das Tor möglich ist. Dabei wird eine Kombination eines evolutionär trainierten Neuronalen Netzes (genauer, ein CTRNN, siehe: 2.4.3) mit einer klassischen Potential Map (siehe: 2.3) zum Ausweichen von Hindernissen getestet. Das Netz und die Potential Fields können zu jedem Zeitpunkt dynamisch aus einer Eingabe von relativen Positionen von Wegmarken auf dem Spielfeld, wie zum Beispiel Ball und Tor, einen Bewegungsvektor generieren, der vom Roboter umgesetzt werden kann.

Das Trainieren der Netze wird nicht auf dem Roboter selber ausgeführt, da hierfür eine Vielzahl von Durchgängen notwendig sind. Stattdessen kommt eine vereinfachte Simulation zum Tragen, die offline auf einem leistungsstärkeren System geschieht.

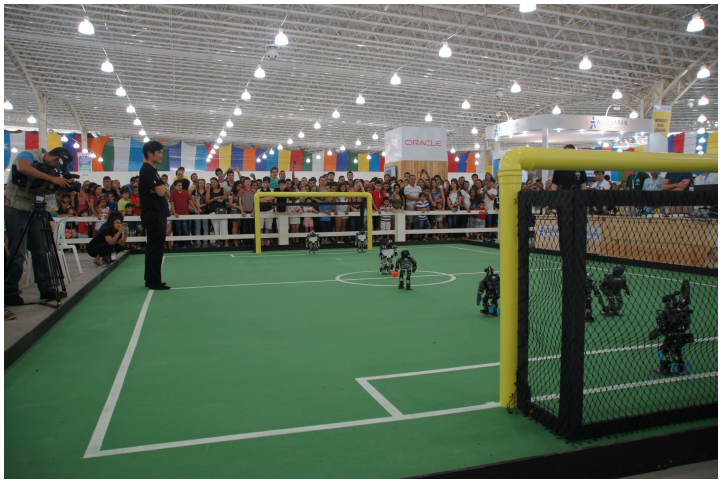
Als Anwendungsszenario wird die RoboCup Soccer Humanoid Kid-Size Liga genommen. Die Arbeit wird so aufgebaut, dass eine Anwendung des Verfahrens in diesem Szenario später möglich ist.

1.2 Umfeld

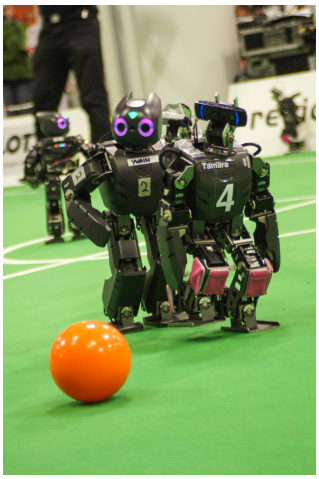
1.2.1 Humanoid Kid-Size Rahmenbedingungen

Die genauen Bedingungen jeder Liga werden in dem jeweiligen Reglement festgelegt. Dieses wird jährlich verändert um dem technologischen Fortschritt zu entsprechen und diesen zu fördern. Für diese Arbeit wird das Reglement von 2014[19] betrachtet.

In der Humanoid Kid-Size Liga wird mit bis zu 4 Robotern pro Team gegeneinander gespielt. In der Regel kommen hier je ein Torwart und 3 Feldspieler zum Einsatz. Die Roboter müssen menschliche Proportionen haben und zwischen 40 cm und 90 cm groß sein. Das Spielprinzip ist dem des klassischen Fußballs nachempfunden. So ist es Ziel des Spiels innerhalb zweier Halbzeiten von je 10 Minuten möglichst viele Tore zu erzielen. Dabei sind zumindest zur Saison 2014 die Tore gelb und der Ball orange um die Erkennung zu vereinfachen. Der Untergrund



(a) Humanoid Kid-Size Spielfeld auf der Weltmeisterschaft 2014 (Brasilien)



(b) Modifizierter DARwIn-OP (vorne) und Original (hinten)

Abbildung 1.1: RoboCup[10]

ist grün mit weißen Spielfeldmarkierungen und beinhaltet ähnliche Elemente wie ein FIFA-Fußballfeld (siehe Abbildung 1.1a). Das Spielfeld ist 6 m mal 9 m groß. Außer den genannten Objekten gibt es keine definierten Landmarken, die zur Positionierung benutzt werden könnten. Neben den 8 Robotern befinden sich ein Schiedsrichter und zwei Robothandler, eine Person pro Team, welche für die Roboter verantwortlich ist, auf dem Feld. Die Roboter spielen vollkommen autonom und müssen alle Berechnungen auf der eigenen Hardware durchführen.

1.2.2 Rahmenbedingungen der Hardwareplattform

Die Hardwareplattform auf der die hier entwickelte Software laufen soll entspricht grundlegend der des DARwIn-OP (Dynamic Anthropomorphic Robot with Intelligence–Open Platform)[22]. Diese Plattform wird in der Humanoid Kid Size Liga von mehreren Teams benutzt [20]. Obwohl es in dieser Liga möglich ist, die Roboter komplett selbst zu entwickeln, wird dies nicht von allen Teams wahrgenommen. Da dies einen erheblichen Aufwand darstellt, greifen viele Teams auf diese schon entwickelte Plattform zurück und nehmen teils Modifikationen vor. Auf der hier vorliegenden Plattform wurde eine andere Kamera verbaut (siehe Abbildung: 1.1b).

Durch die geringe Bauhöhe ist es problematisch das ganze Feld inklusive der Feldlinien zu erfassen. Aus der Fläche aufragende Objekte wie der Ball, die Tore oder Hindernisse sind deutlicher zu erkennen. Außerdem setzen die humanoiden Roboter viele kleine Schritte, wodurch es zu vergleichsweise starkem Wackeln und Vibrationen kommt, welche wiederum zu verrauschten Daten führen.

Die Rechenleistung dieser Plattform ist begrenzt [22]. Somit sollte eine Lösung gefunden werden die keinen unverhältnismäßig hohen Rechenaufwand benötigt.

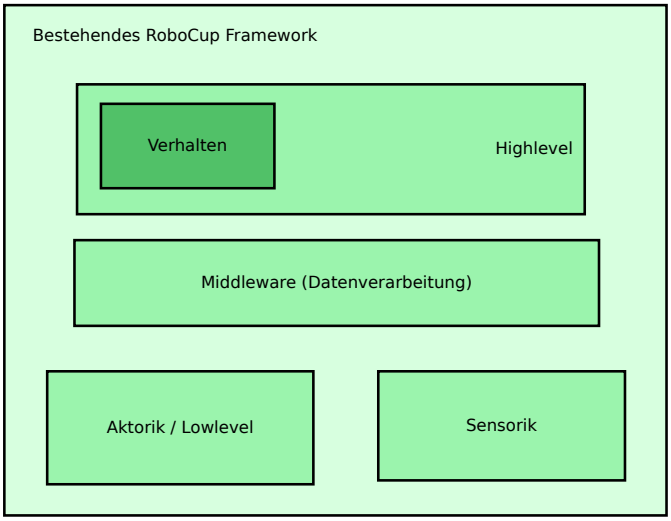


Abbildung 1.2: Schematische Ansicht des RoboCup Frameworks

1.2.3 Die Softwareumgebung

Die Hamburg Bit-Bots sind ein Team der Humanoid Kid-Size Liga, die seit 2011 an dem RoboCup teilnehmen. Die hier geschriebene Software ist zur Einbindung in dieses Framework gedacht.

Die Software, die auf den Robotern der Hamburg Bit-Bots läuft, ist mehrschichtig aufgebaut. So wird auf verschiedenen Ebenen der Software ein anderes Abstraktionslevel erzeugt (siehe Abbildung 1.2). Während sich die unteren Ebenen um Sensorik und Aktorik kümmern, dienen die mittleren Schichten zur Verarbeitung der Daten, wie zum Beispiel der Analyse der Bildinformationen oder der Umsetzung der gewollten Bewegungsrichtung. Diese unteren Ebenen sind zu großen Teilen in den Programmiersprachen C++ und C geschrieben, die mittlere Schicht in Cython (typisiertes Python) um die notwendige Performance der zum Teil mathematisch aufwendigen Verfahren zu erreichen.

Die höherlevelige Architektur, das Verhalten, welches für die abstrakte Planung des Softwaresystems verantwortlich ist, ist in Python 2.7[17] geschrieben, um eine einfache Lesbarkeit und Anpassbarkeit zu gewährleisten. Python gilt als übersichtliche Sprache, die viele Methoden schon bereitstellt und Bereiche wie die Speicherverwaltung intern behandelt. Diese hohe Abstraktion des Frameworks ermöglicht es mit einem vereinfachtem Modell zu arbeiten. So stehen Postionen von Ball und Toren in Vektordarstellung relativ zum Roboter bereit. Außerdem kann so die gewollte Bewegung in drei Werten übergeben werden: der Vor- und Rückwärtsbewegung, der Seitwärtsbewegung und dem Drehen, während sich das Framework um die Umsetzung kümmert. Wie genau diese Bewegungen umgesetzt werden ist sehr unterschiedlich. Es gibt viele Einflussfaktoren wie den Boden, Hindernisse gegen die gelaufen wird oder die Hardwarequalität der einzelnen Roboter. So können zu heiß gewordene Motoren dafür sorgen, dass der Roboter eine Kurve läuft, obwohl dies nicht erwünscht war.

Eine global agierende Lokalisation ist nicht verfügbar, der Roboter weiß nicht

auf welcher absoluten Position des Spielfeldes er sich gerade befindet. Das Lokalisierungsproblem ist eine der größten Herausforderungen in der Humanoiden Liga. Gerade die Symmetrie des Feldes ist hierbei eine der Herausforderungen, da nicht visuell erfasst werden kann auf welcher Seite des Feldes sich der Roboter gerade befindet. Hier müssen andere Konzepte benutzt werden, wie das Merken der Bewegung oder bioinspirierter Verfahren[5].

Kapitel 2

Verwandte Arbeiten und Hintergrundinformationen

2.1 Klassische Wegfindung

Der Bereich der Wegfindung oder auch Motion Planning ist gründlich erforscht und eins der Standardprobleme der Robotik. Es gibt eine Vielzahl von Algorithmen und Ansätzen, die auf unterschiedliche Weise optimale Pfade finden können. Ein Beispiel ist der A*-Algorithmus[11] welcher den kürzesten, beziehungsweise besten Weg zu einem Ziel finden kann. Er bedient sich einer Heuristik welche für eine gute Laufzeit sorgt. Bei Algorithmen wie A* handelt es sich um Suchen in einem Graphen zwischen zwei Knoten. Auch wenn sich in vielen Fällen ein Graph bzw. ein Gitter erstellen lässt, weist dies hier Schwächen auf. Die meisten dieser Algorithmen lassen sich nicht problemlos in dem gegeben Kontext anwenden, da keine Lokalisation beziehungsweise globales Weltmodell mit allen Informationen der Umgebung existiert und so kein korrekter Graph erstellt werden kann. Des Weiteren ändert sich die Umgebung dynamisch, da sich viele der Objekte in der Umgebung vergleichsweise schnell bewegen oder Objekte erst richtig erkannt werden, wenn der Roboter sich ausreichend nah befindet. So müsste die Suche regelmäßig wiederholt werden. Auch lassen sich Eigenschaften wie die Ausrichtung des Roboters hier schlecht abbilden. Aus diesen Gründen sind die klassischen Pfadfindungsalgorithmen in einem Graphen ungeeignet für dieses Anwendungsszenario.

2.2 Bisherige Verfahren und Konzepte im RoboCup

2.2.1 Allgemein

Die Frage der Pfandfindung ist eine der zentralen Punkte im RoboCup. Gerade in den Simulationsligen war dies einige Zeit einer der großen Forschungsschwerpunkte. Inzwischen ist die Software der Simulationsligen soweit ausgereift, dass

das Planen der Bewegungen und Aktionen mehrerer Agenten untereinander im Zentrum stehen [28].

Es gibt hier verschiedene Ansätze. Die meisten dieser Konzepte gehen von einem globalen Weltmodell aus, einem System in dem die Position des Agenten selber und aller Mitspieler und ggf. Gegner bekannt ist. Mit diesen Informationen können komplexe Verhaltensweisen modelliert werden, in denen auch eine klassische Wegfindung erstellt werden kann. Auf dem Spielfeld wird ein Gitter gespannt. Über die nicht durch Hindernisse belegten Felder kann beispielsweise der A*-Algorithmus angewandt werden um zu einem bestimmten Punkt der Karte zu gelangen. Dies ist auch in der Small Size Liga und der Middle Size Liga möglich. Diese besitzen inzwischen eine ausgereifte Lokalisation, da hier im Fall der Small Size Liga eine Draufsicht mittels Deckenkameras erfolgt. Bei der Middle Size Liga ist eine 360 Grad Ansicht verfügbar und dazu können ohne weiteres leistungsstarke Kameras und Computer verbaut werden.

2.2.2 Humanoid Soccer

Im Bereich der humanoiden Roboter ist diese Problemstellung, genau wie die Ligen selber, eher neu. So galten in den ersten Jahren gerade noch das Fortbewegen und die Interaktion mit dem Ball als Herausforderung, so ist inzwischen ein Punkt erreicht, an dem Planung und komplexere Verhaltensweisen relevant werden. Dennoch stellen eine regelmäßige Verschärfung der Regeln, eine oft geringe Rechenleistung und ein begrenztes, dem Menschen nachempfundenes Sichtfeld, verbunden mit einer geringen Bauhöhe die Teams vor neue Herausforderungen. Gerade Kameraqualität und Rechenleistung sind im Bereich der Objekterkennung und Positionsbestimmung in vielen Fällen begrenzende Faktoren. So sind weiterhin nur wenige Teams der humanoiden Ligen in der Lage sich fehlerfrei zu Lokalisieren und entsprechende komplexe Verhaltensweisen zu benutzten. Durch erschwerte Bedingungen durch neue Regeln in vielen Bereichen, wie der Spielfeldgröße, der Landmarken oder dem Untergrund, sind Bereiche wie eine globale Positionierung in vielen Fällen noch nicht Spielentscheidend. Deshalb wird sich oftmals nur auf die aktuelle visuelle Eingabe verlassen um die Bewegung zum Ziel durchzuführen.

2.2.3 Bisherige Lösung bei Hamburg Bit-Bots

Die Hamburg Bit-Bots benutzen ein vereinfachtes Modell der Umgebung das relativ zum Agenten aufgestellt wird. Es sind die Positionen des Balles und der Tore relativ zum Roboter vorhanden und, sofern sich ein Hindernis im Sichtfeld befindet, auch die Position dessen. Hindernisse sind in diesem Fall andere Roboter und Menschen auf dem Feld. Bisher werden in der Software des RoboCup Teams der Universität Hamburg, in deren Kontext diese Arbeit entsteht, arithmetisch Berechnungen für die Bewegung des Roboters in der Vertikalen, der Horizontalen sowie der Rotation berechnet. Die geschieht abhängig von dem Abstand und Winkel zu den einzelnen Landmarken wie Tor und Ball.

Die aktuelle Wegfindung unterteilt sich in vier Phasen, die abhängig von der Position des Roboters zum Ball sind. Die erste Phase kommt bei einer großen Distanz zum Ball zum Tragen, in der sich der Roboter mit hoher Geschwindigkeit vorwärts bewegend in Richtung des Balles dreht. Wenn der Roboter in die Nähe des Balles kommt, wird die Geschwindigkeit verringert und nicht mehr rotiert, um eine genauere Positionierung zu ermöglichen. Liegt der Ball hinter dem Roboter, geht dieser gerade zurück und anschließend seitwärts bis den Ball vor sich hat. Abschließend dreht der Roboter sich um den Ball bis er zu einem Tor ausgerichtet ist.

Dabei traten verschiedene Probleme auf. So waren zum Beispiel die verwendeten harten Grenzen für die Einteilung der Entfernung zu den einzelnen Landmarken in einzelne Zonen problembehaftet, da die Daten zum Teil nicht präzise genug sind. Es erwies sich oft als schwierig, auf den Eingabewerten gute Funktionen zu finden die einen schnellen Pfad erzeugen. Für ein gutes Ergebnis musste häufiger die Pfadplanung mit angepasst werden, wenn sich andere Bereiche des Systems veränderten, wie kleine Änderungen an der Hardware oder dem Walking. Auch individuell mussten sich die Berechnungen sich zum Teil unterscheiden, da sich jeder Roboter etwas unterschiedlich bei der Bewegungsgenauigkeit verhält. Außerdem erwies es sich als schwierig den Pfad so zu wählen, dass der Roboter sofort richtig am Ball stand. Darum war in der Regel noch eine Ausrichtungsphase vonnöten. Das Ausweichen von Hindernissen war nicht eingebaut, wodurch oftmals Roboter ineinander gelaufen sind und sich somit gegenseitig behindert oder zum Sturz gebracht haben. Auch die Mess- und Vergleichbarkeit verschiedener Varianten war hier nur schwer gegeben. Ein Vergleich mit der bisherigen Wegfindung ist unter 4.4.1 zu finden.

2.3 Potential Fields

Schon in den Neunziger Jahren hat Ronald C. Arkin das Modell der Potential Fields[2] im Bereich der Roboternavigation vorgestellt. Dieses Modell ist deutlich besser für die Wegplanung von Robotern geeignet und wurde schon in vielen Szenarien mit autonomen Agenten getestet. Es handelt sich bei Potential Fields um Karten auf denen Bewegungsvektoren erzeugt werden, die die Agenten in die so bestimmten Richtungen steuern lassen. Einzelne Positionen auf dieser Karte haben ein bestimmtes Potential, sodass es einem Agenten einfacher ist, sich in eine bestimmte Richtung zu bewegen. Es bildet sich eine Art Landschaft mit Bergen und Tälern. Die Höhenunterschiede werden als Vektoren dargestellt, die Richtung und Stärke darlegen. Um die Vektoren zu erzeugen gibt es Attraktoren, die die Vektoren auf sich zeigen lassen und Repulsoren die abstoßend wirken (siehe Abbildung: 2.1a). Dabei ist die Stärke, mit der ein einzelner Repulsor oder Attraktor auf eine bestimmte Position der Karte wirkt, abhängig von der Entfernung. Je näher der Agent sich an einem Repulsor befindet, umso stärker ist dessen Kraft. Diese Kraft kann beschränkt sein, indem sie auf unendlich steigt um eine Barriere zu erstellen, oder bei größerer Entfernung auf null gesetzt wird. Die Funktionen

zur Stärke können entsprechend der Aufgabe unterschiedlich aussehen. In diesem Beispiel ist die Stärke ($force$) des Vektors von der Distanz zum Objekt ($dist$) abhängig. Die Stärke wächst quadratisch an. $\vec{x_v}, \vec{y_v}$ sind die Vektoren des Feldes. Es werden die Komponenten des Einheitsvektors, welche aus dem Abstand des Objektes pro Ache (x_o, y_o) und der euklidischen Distanz gebildet werden mit der Kraft multipliziert (siehe Formel: 2.2 und 2.3).

$$force = \frac{1}{dist^2} \tag{2.1}$$

$$\vec{x_v} = \frac{-x_o}{dist} * force \tag{2.2}$$

$$\vec{y_v} = \frac{-y_o}{dist} * force \tag{2.3}$$

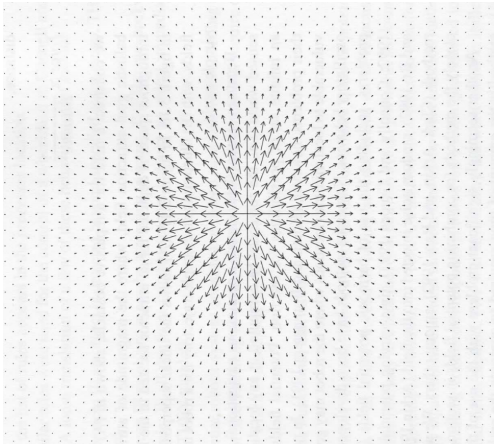
Dabei gibt es verschiedene Arten wie diese Felder aussehen können um ein bestimmtes Verhalten zu erzeugen. So können die erzeugten Vektoren unabhängig von ihrer Position einen uniformen Wert haben um einen Grunddruck zu erzeugen. Die Felder können von den Rändern entlang einer Achse in die Mitte der Karte zeigen um einen Korridor (siehe Abbildung: 2.1b) zu erzeugen indem sich der Roboter bewegen soll. Durch ein sich um einen Punkt drehendes Feld können Präferenzen angegeben werden auf welcher Seite zum Beispiel der Roboter ein Hindernis passieren soll. Diese Felder werden auch Schema oder Verhalten genannt.

Die Felder können beliebig kombiniert werden(siehe Abbildung: 2.1c), indem man die Vektoren aufsummiert. So verstärken sich Vektoren oder heben sich gegenseitig auf. Mit diesen verschieden Feldern können schon umfangreiche Verhalten bzw. Schemata erzeugt werden.

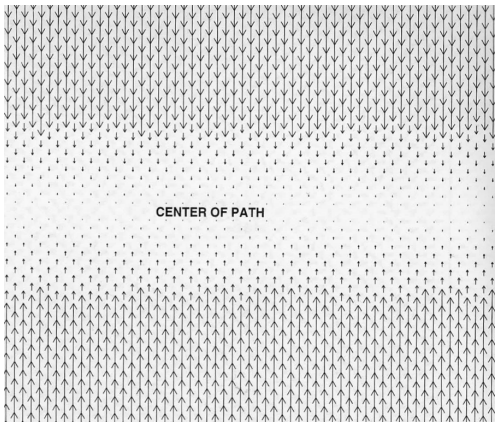
$$\vec{v} = \sum_{i=1}^{N} \vec{v_i} \tag{2.4}$$

Die einzelnen Komponenten des Vektors $\vec{v}$ bilden sich aus der Summe der Komponenten der Vektoren der einzelnen Felder $\vec{v_i}$ (siehe Formel: 2.4). Einfache Repulsor- und Attraktorfelder können mit geringem Rechenaufwand erstellt werden, weswegen sie sich gut für eine simple Steuerung eines Roboters eignen.

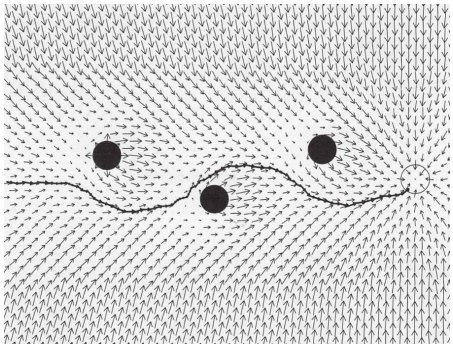
Problematisch wird es zum Beispiel, wenn eine bestimmte Ausrichtung erwünscht wird, die nicht mit der Bewegungsrichtung übereinstimmt, da sich dieses nicht direkt abbilden lässt. Auch kann es schnell dazu kommen, dass der Roboter in ein lokales Maxima gerät und so das Ziel nicht erreicht. Durch ein zusätzliches randomisiertes Feld kann dies bei kleinen Maxima, wie dem direkten zusteuern auf ein Hindernis kompensiert werden. Bei sackgassenartigen Situationen ist dies allerdings nicht mehr ausreichend. In diesem Fall könnten Felder aufgespannt werden, die abstoßend sind von Orten an denen man schon einmal war. Das Aufstellen solcher Felder ist komplex, erfordert entsprechend einen hohen Rechenaufwand und ist ohne ein globales Wissen der Umgebung oft nicht durchzuführen.



(a) Repulsor



(b) Korridor



(c) Kombination von mehreren Potential Fields

Abbildung 2.1: Verschiedene Potential Fields[2]

2.4 Künstliche Neuronale Netze

Der Bereich der Künstlichen Intelligenz ist schon seit vielen Jahren eng mit der Robotik verknüpft. Sie soll so auf die Umgebung reagieren, dass der größte mögliche Nutzen bzw. Erfolg daraus gezogen wird [25]. Dabei wird darauf geachtet, dass der intelligente Agent auch in unbekannten Umgebungen beziehungsweise in unbekannten Szenarien gute Leistungen erzielt. So kommt ein Mensch auch in Umgebungen zurecht die er noch nie gesehen hat. Genauso soll ein Agent die Umwelt wahrnehmen und aus dieser lernen. Viele dieser Konzepte sind bioinspiriert, also der Natur nachempfunden.

Künstliche Neuronale Netze sind ein bioinspiriertes Verfahren der künstlichen Intelligenz. Sie orientieren sich an den biologischen Neuronalen Netzen wie sie im Nervensystem von Lebewesen zu finden sind, wie zum Beispiel im menschlichen Gehirn. Sie zeichnen sich durch eine sehr umfangreiche Vernetzung und eine sehr hohe Parallelität aus[12][1]. Die künstlichen neuronale Netze sind diesen nachempfunden und können deutlich schneller arbeiten, besitzen aber auf klassischer Computerhardware mit bis zu 8 Prozessoren nur in einem geringen Rahmen die notwendige Parallelität um die biologischen Vorbilder in vollem Umfang nachzubilden.

Künstliche neuronale Netze bestehen aus künstlichen Neuronen und den Übergängen zwischen ihnen. Die einzelnen Neuronen bekommen Eingaben von anderen Neuronen, die durch den Übergang gewichtet wurden und berechnen auf Grundlage dessen ihre Ausgabe den sie an die nächste Schicht oder die Umwelt weitergeben. Durch das Anpassen der Kantengewichte kann das Netz lernen, bzw. sich verändern. [23] Neuronale Netze werden in verschiedenen Bereichen eingesetzt. Sie können komplexe Funktionen nachbilden und benötigen dabei oft deutlich weniger Rechenzeit als andere Modelle. Neuronale Netze sind in der Anwendung sehr Laufzeiteffizienz, müssen allerdings vorher trainiert werden. Auch können sie dazu benutzt werden, Funktionen abzubilden, zu denen keine einfachen arithmetischen Funktionen bekannt sind. So zum Beispiel in der Geräuschverarbeitung oder der Objekterkennung. Es gibt verschiedene Arten von Netzen, die allen diesem Grundprinzip folgen, aber unterschiedlich aufgebaut sind und differenzierte Eigenschaften besitzen.

2.4.1 Topologie und Arten von Netzen

Die einfachste Art von Netzen sind Perzeptrone[24], bzw. One-Layer-Feed-Forward Netze. Netze dieser Art bestehen neben der Eingabeschicht nur aus der Schicht der Ausgabeneuronen. In den meisten Fällen sind Ein- und Ausgabeschicht von Ein- nach Ausgabe voll vernetzt. Da die Verbindungen in dieser Art von Netzen nur in einer Richtung erfolgen, also die Informationen nur vom In- zum Output fließen, nennen sich diese Netze Feed-Forward Netze. Man kann hier beliebig viele interne Schichten einfügen, die jeweils zur nächsten Schicht vernetzt sind, diese Schichten werden auch versteckte Schichten (Hidden Layer) genannt, da sie keine direkte Interaktion mit der Umwelt haben, sondern Ein- und Ausgabe nur über andere Neuronen erfolgt. Hiermit können auch komplexere Funktionen abgebildet werden. Werden die Neuronen mit sich oder mit vorherigen Schichten verknüpft, handelt es sich um rekurrente Netze. Hierbei ist zum Beispiel auch eine Vollverknüpfung möglich (alle Neuronen mit allen verknüpft). Da der Input der Rekurrenz im Allgemeinen erst in der nächsten Iteration von den einzelnen Neuronen betrachtet werden kann, ist mit diesen Netzen ein Speichern von Informationen über mehrere Iterationen möglich.

Diese Netze haben eine feste Anzahl von Neuronen und Kanten. Andere Konzepte wie Growing Neural Gas[8] haben die Möglichkeit, zur Laufzeit Neuronen und Kanten hinzuzufügen und zu entfernen. Auch kann bei Netzen dieser Art die Position in einem Koordinatensystem der Neuronen wichtig sein, was bei den klassischen Netzen nicht betrachtet wird.

Es gibt eine Vielzahl verschiedener Netze, die sich in verschiedensten Konzepten unterscheiden. In der Umsetzung der Arbeit wurde hauptsächlich mit CTRNN (siehe: 2.4.3) gearbeitet.

2.4.2 Funktionsweise und Eigenschaften

Ein Netz besteht aus einer Menge von Neuronen. Je nach Topologie haben diese unterschiedliche Eigenschaften, sind dennoch ähnlich aufgebaut.

$$a_i = \sum_{j=1}^{N} w_{ij} x_j \tag{2.5}$$

Die Aktivierung des Neurons a_i ist die Summe der mit w_{ij} gewichteten Eingaben x_j für alle N vorhergehende Neuronen j.

Aus dieser Aktivierung wird nun ein Ausgabewert berechnet. Dieser wird durch die Aktivierungsfunktion Φ bestimmt. Die kann im einfachsten Fall linear sein ($\Phi(a_i) = ka_i$ mit einer Konstante k), bei der die Aktivierung ausgegeben wird, oder bei einer Stufenfunktion bei der nur ein konstanter Wert weitergegeben wird, wenn ein bestimmter Threshhold Θ überschritten wurde (siehe Formel: 2.6).

$$\Phi(a_i) = \begin{cases} 1 & a_i > \Theta \\ 0 & \text{ansonsten} \end{cases} \tag{2.6}$$

Des Weiteren kann der Wertebereich der Ausgabe mithilfe einer Sigmoidfunktion (Beispielsweise Formel 2.7) eingeschränkt werden. Hier werden beliebig große Werte stetig in den Wertebereich zwischen 0 und 1 abgebildet. Es kann hier noch ein sogenannter Bias θ zum Tragen kommen. Hierbei handelt es sich um einen Wert der zusätzlich noch auf die summierte Eingabe gerechnet wird. So kann die Aktivierungsfunktion in eine bestimmte Richtung verschoben werden.

$$\Phi(a_i) = \frac{1}{1 + e^{-ka_i}} \tag{2.7}$$

Die Ausgabe $y_i = \Phi(a_i)$ wird an alle nachfolgenden Neuronen weitergegeben.

2.4.3 Continous-Time Recurrent Neuronal Network

Das Continous-Time Recurrent Neuronal Network (CTRNN)[7] ist ein Netz das auch in der Robotik eingesetzt wird.[14] Hier handelt es sich um eine spezielle Art von rekurrenten neuronalen Netzen die noch einen Zeitfaktor mit betrachten. Eine mögliche Darstellung eines CTRNNs wie sie auch in dieser Arbeit verwendet wurde ist in Formel 2.8.

$$a_i = a_{n-1} + \frac{1}{\tau}\left[-a_{n-1} + \sum_{j=1}^{N} w_{ji} y_j + \theta_i\right] \tag{2.8}$$

Die Variable τ ist die Zeitkonstante. Durch sie kann das Netz lernen wie schnell die Neuronen reagieren sollen. Ein hohes τ sorgt für ein träges Neuron. Anschließend wird wie bei anderen Netzen auch eine Sigmoidfunktion angewendet.

2.4.4 Modifikation von Netzen

Damit neuronale Netze die gewünschten Resultate liefern und gute Lösungen erzeugen, müssen sie angepasst werden. Hierbei werden Teile des Netzes verändert. Das betrifft meist die Gewichte der Kanten und den Bias der Neuronen (der Bias kann auch als Kante von einem Neuron mit einer konstanten Ausgabe betrachtet werden). Es gibt verschiedene Konzepte zum Optimieren der künstlichen neuronalen Netze. Man Unterscheidet zwischen unüberwachtem Lernen(unsupervised learning)[26], überwachtem Lernen(supervised learning)[23]und bestärkendem Lernen(reinforcement learning)[3]. Des Weiteren gibt es die Möglichkeit Netze mittels evolutionärer Verfahren zu trainieren[9].

Die einfachste Art zu lernen ist die des unüberwachten Lernens. Hier bekommt das Netz keine Rückmeldung aus der Umwelt. Dem Netz ist somit nicht bekannt, was die erwarteten Werte sind. Stattdessen versucht das Netz selber Strukturen in der Eingabe zu erkennen. Dies wird zum Beispiel beim Clustern oder anderen Einordnungsaufgaben verwendet.

Beim überwachten Lernen wird ein Satz von Trainingsdaten ausgewählt wofür die korrekte bzw. erwartete Lösung bekannt ist. Dies kann zum Beispiel eine mathematische Funktion sein. Nach dem Lernen kann die Qualität mit einem Satz von Testdaten geprüft werden. Diese Daten sind dem Netz noch unbekannt, sollten dennoch zur richtigen Ausgabe führen. Dadurch, dass immer die gewünschte Lösung bekannt ist kann der Fehler der aktuellen Berechnung für die Ausgabe berechnet werden. Durch Backpropagation kann der Fehler auch für ggf. versteckte Schichten ermittelt werden. Mithilfe des Fehler können nun die Gewichte der Kanten angepasst werden. Beim überwachten Lernen werden für alle Trainingsdaten die exakte Lösung benötigt. Wenn Agenten in unbekannten Umgebungen arbeiten, ist dies in vielen Fällen nicht direkt möglich. Für diese Fälle bietet sich oftmals bestärkendes Lernen an. Hier wird nicht jeder Schritt der Fehler berechnet, sondern über mehrere Schritte hinweg die Qualität beurteilt. Angelehnt an Bestärkung im Rahmen der Biologie. So hat das Netz das Ziel eine möglichst große Belohnung zu bekommen. Es gibt hierbei verschiedene Konzepte wie dieses Wissen nun in das Lernen besserer Netze umgesetzt wird.

2.5 Evolutionäre Algorithmen

Ein Verbesserung in der bioinspirierten künstlichen Intelligenz kann auf verschieden Arten erfolgen. Eine davon ist die Verwendung genetischer bzw. evolutionärer Algorithmen. Ein solcher Algorithmus ist an die Evolution der Natur angelehnt und dient der Optimierung der Individuen [9]. Es gibt eine Population bestehend aus einer Menge von Individuen. Diese Individuen müssen veränderbar sein; beispielsweise eine Kette von Aktionen als Lösungsweg oder ein Graph bestehend aus verschiedenen Knoten und Kanten. Es werden zunächst eine große Anzahl zufälliger Individuen erstellt und auf ihre Fitness geprüft. Die Fitness gibt die Qualität der Individuen wieder. Hierbei muss ein guter Weg gefunden werden die Qualität

zu messen.

Eine gute Fitnessfunktion steht im Zentrum eines evolutionären Ansatzes, da diese definiert, ob eine Lösung besser ist als eine andere. Dabei sollte darauf geachtet werden, dass die Funktion möglichst stetig ist, sodass die Evolution nicht in lokale Mini- bzw. Maxima gelangt. Wie so eine Fitnessfunktion aussieht ist sehr von der gestellten Aufgabe abhängig.

Unter Anderem anhand der Qualität der einzelnen Lösungen wird entschieden welche Individuen weiter Bestandteil der Population bleiben können. Ein anderes Kriterium zur Auswahl wäre zum Beispiel das Alter des Individuums. Durch Rekombination, dem verknüpfen zweier Individuen und Mutation, zufälliger Veränderung einzelner Merkmale, entstehen neue Individuen, die in die nachfolgende Generation übergehen. Die Art der Evolution kann meistens sehr frei angepasst werden. So wird in der Auswahl der Individuen, der Art der Mutation und Rekombination umfangreich variiert. Hierbei wird der Genotyp, die interne Repräsentation, verändert, dies führt in der Regel zu einem veränderten Verhalten.

Auch neuronale Netze kann man mit evolutionären Verfahren trainieren. Eine Rekombination der Netze ist in vielen Fällen nicht möglich, da die Netze in der Regel so verknüpft sind, dass eine Kombination zweier Netze zu keinem sinnvollen neuen Netz führt. Allerdings ist eine Mutation der Netze recht simpel möglich, indem die Eigenschaften der Netze zufällig mutiert werden.

Bei jedem Netz aus der Population wird diese Mutation angewendet, dazu wird über jede Kante des Netzes iteriert und mit einer bestimmten Wahrscheinlichkeit p diese Kante mutiert. Bei der Mutation wird ein zufälliger Wert r addiert. r ist ein zufälliger Wert, der beispielsweise durch die Gaußfunktion generiert wird. Er kann positiv oder negativ sein. So kommt es meist zu kleinen Mutationen, welche mit einer Selektion die Fitness nach oben treiben. Es kann aber auch zu größeren Sprüngen kommen, die helfen sollen kleineren lokalen Maxima zu entkommen. Diese Werte haben meist obere und untere Grenzen. Werden diese mit der Zufallszahl überschritten, wird der Zufallswert von der überschrittenen Grenze abgezogen (Siehe Formel 2.9). Dies soll verhindern, dass genau die Grenzwerte angenommen werden.

$$w_i = \begin{cases} max - r & w_{i-1} + r > max \\ min - r & w_{i-1} + r < min \\ w_{i-1} + r & ansonsten \end{cases} \tag{2.9}$$

Das Gewicht w_i ist der neue Wert, w_{i-1} beschreibt das vorherige Gewicht. Bei r handelt es sich um den Zufallswert der addiert wird und bei min und max um die Grenzen.

Um auszuwählen, welche Individuen in die nächste Generation übernommen werden, gibt es verschiedene Verfahren, so können einfach die Individuen mit der besten Fitness übernommen werde. Eine andere Möglichkeit ist eine Selektion auf Basis der Fitness oder dem Rang des Individuums[29].

Die Wahrscheinlichkeit ob ein Individuen ausgewählt wird ist hier abhängig von der Fitness. Entweder indem die Wahrscheinlichkeit des Auswählens direkt mit der Fitness im Zusammenhang steht oder durch eine entsprechende Sortierung. Dadurch haben Individuen mit einer hohen Fitness eine gute Chance in die Folgegeneration zu kommen und dies auch mehrmals. Bei Individuen mit schwacher Fitness ist dies unwahrscheinlicher allerdings nicht ausgeschlossen. Diese Individuen können helfen lokale Maxima zu vermeiden, da so auch alternative Lösungen in der Population verbleiben.

Kapitel 3

Konzept der Software

3.1 Konzept

Potential Fields bieten eine gute Möglichkeit den Roboter zu steuern, zeigen allerdings einige Schwächen. So sind vorhandene Daten der Ball- und Torposition oftmals stark verrauscht und nicht präzise genug. Außerdem ist es schwer, mit Potential Fields die genauen Parameter für die Bewegung zu erfahren. So ist die Frage der Geschwindigkeit und ob ein Drehen des Roboters oder Seitwärtsschritte besser geeignet sind, nicht direkt zu Beantworten. Auch zeigte es sich als schwierig zu modellieren, dass die verschiedenen Bewegungsarten unterschiedlich effizient sind (zum Beispiel vorwärts schneller als rückwärts). Ein lernendes Verfahren sollte dieses leisten. So fiel die Wahl auf ein evolutionär trainiertes CTRNN für die Wegsteuerung in Verbindung mit den Potential Fields um Hindernissen auszuweichen. Das neuronale Netz ersetzt in diesem Sinne das Attraktorfeld der Potential Fields und übernimmt die fehlende Dimension (Drehung) der Ansteuerung. Da die Software später auf einem hohen Abstraktionslevel laufen soll und viele Evolutionsschritte notwendig sind, fand das evolutionäre Training in einer 2-Dimensionalen Simulationsumgebung statt.

3.2 Anbindung an Softwareumgebung

Die Simulation und die anderen Teile der hier entwickelten Software sind eigenständig von der bisherigen Software der Hamburg BitBots und wurden so konzipiert, dass eine Einbindung des neuronalen Netzes und der Potential Fields in das Framework einfach machbar ist. So wurden die selben Schnittstellen für Ein- und Ausgabe gewählt wie sie schon vorhanden sind. Auch die Programmiersprache ist identisch, sodass die Kernkomponenten, das neuronale Netz und die Potential Fields, einfach in die Modulstruktur eingepflegt werden können und nur noch kleine Anpassungen notwendig sind.

3.3 Vereinfachungen und Vorgaben dieser Arbeit

In Rahmen dieser Arbeit werden einige Annahmen getroffen und Gegebenheiten abstrahiert.

Es wird davon ausgegangen, dass die Software keine Objekte fälschlicherweise als Bälle, Tore oder Hindernisse erkennt. So kommt es in der Realität häufiger zu false Positives bei der Objekterkennung. Dessen Behebung ist mittels verschiedenen Verfahren lösbar, aber nicht Teil dieser Bachelorarbeit. Der Fehler auf den Daten wird mittels einer normalverteilten Zufallsverteilung auf die Daten gerechnet um das Rauschen der Daten durch Bewegungen o.Ä. zu simulieren. Die Erkennung der Objekte erfolgt in einer 360° Ansicht, das heißt der Roboter kann diese erkennen auch wenn sie hinter ihm liegen, da zum Beispiel die Kopfbewegung nicht mit simuliert wird. Auch wird das Symmetrieproblem der Umgebung hier abstrahiert, der Roboter weiß zu jedem Zeitpunkt welches Tor das des Gegners ist. Des Weiteren wird innerhalb der Simulation davon ausgegangen, dass Ball und Hindernisse ihre absolute Position nicht verändern. Da die Berechnungen allerdings Dynamisch für jeden Zeitschritt ausgeführt werden, sollte dies keinen Unterschied im Resultat des Verfahrens erzeugen und die Netze sowie die Potential Fields damit umgehen können.

Kapitel 4

Umsetzung und Ergebnisse

4.1 Software

Die entwickelte Software lässt sich in verschiedene Bereiche unterteilen, die im Folgenden vorgestellt werden. Der grobe Aufbau ist in Abbildung 4.1 zu sehen. Eine Vielzahl von Parametern sind in einer Konfigurationsdatei anpassbar.

4.1.1 Simulation

Neben den eigentlichen Verfahren entstand weitere Software, die für ein funktionierendes System notwendig ist. Die Simulation ist für die Bewertung des Verfahrens maßgeblich und rechnet die Szenarien mit den entsprechenden Potential Fields und neuronalen Netzen durch. Sie dient als Schnittstelle zwischen den Verfahren und der Umgebung und beinhaltet maßgeblich den Teil, der die Realität simuliert. Die Simulation ist in mehrere Teile gegliedert.

Die **grafische Ausgabe** ist für das Plotten der einzelnen Simulationen zuständig, dabei wird nach jeder Generation der Evolution, das Individuum mit der besten Fitness dargestellt. Das Plotten der Pfade geschieht mit der Pythonbibliothek "matplotlib" [15].

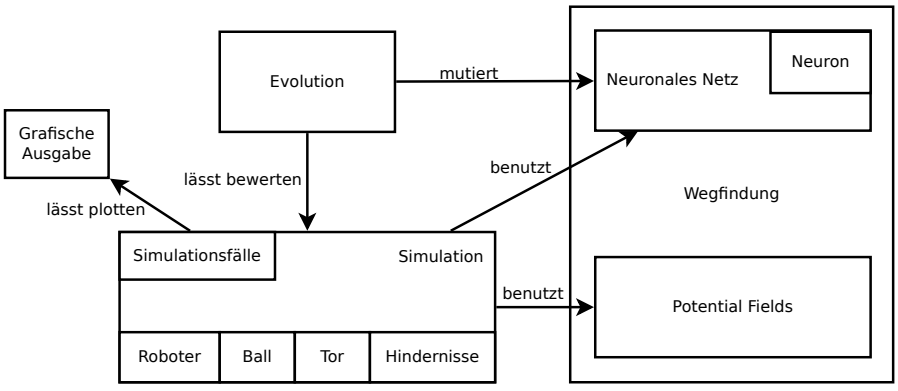


Abbildung 4.1: Aufbau der Software

Die **Simulationsfälle** beschreiben den Aufbau der Simulationsumgebung. Sie können entweder vorgegeben werden um spezielle Randfälle abzuprüfen oder zufällig generiert werden um zu verhindern, dass die Evolution keine allgemeingültige Lösung findet, sondern auf die speziellen Fälle trainiert. Es können die Torposition, die Ballposition, die Gegnerpositionen und deren Anzahl sowie die Position und die Orientierung des Roboters angeben werden.

Das **Weltmodell** wird so erstellt, wie der Simulationsfall es vorgibt. Es beinhaltet die globalen Informationen der Welt und hält die Objekte, die zu der Welt gehören. Dazu gehören der Ball, das Tor und eine Liste von Hindernissen beliebiger Anzahl.

Während diese Objekte nur ihre Position halten, hält das Objekt des **Roboters** weitere Informationen wie die aktuelle Orientierung. Des Weiteren besitzt es Informationen zum Zustand, wie der aktuellen Bewegung und speichert den Bewegungsverlauf ab, der später für die Berechnung der Fitness relevant ist.

In jedem Zeitschritt werden die relativen Positionen zu den anderen Objekten des Weltmodells berechnet. Die interne Repräsentation erfolgt in u (entspricht der y Achse) und v (entspricht der negativen x Achse). Dieses sind die Werte, mit denen der Roboter die weiteren Berechnungen anstellt. Diese sind an die Werte angelehnt, die auf der echten Plattform verfügbar sind. Bei deren Generierung werden außerhalb der Simulation keine globalen Positionsinformationen benötigt. Innerhalb der Simulation werden die Werte wie in Formel 4.1 und 4.2 gezeigt berechnet.

$$u = -(x_r - x_o) * cos(\varphi) + (y_r - y_o) * sin(-\varphi) \tag{4.1}$$

$$v = (x_r - x_o) * sin(\varphi) - (y_r - y_o) * cos(\varphi) \tag{4.2}$$

Wobei x_r, y_r die globale Position des Roboters, φ die Orientierung des Roboters und x_o, y_o die globale Position des Objektes ist.

Des Weiteren berechnet das Objekt die Bewegungsgeschwindigkeit und aktualisiert die Position auf dem Spielfeld.

$$s_n = \begin{cases} \max & s_g >= \max \\ \min & s_g <= \min \\ (s_{n-1} * d + s_g)/(d+1) & \text{ansonsten} \end{cases} \tag{4.3}$$

s_g, der gewünschte Geschwindigkeitswert
s_n, die Geschwindigkeit zum Zeitpunkt n
d, eine Konstante zur Verzögerung

Bei der Berechnung der Geschwindigkeit wird eine gewisse Verzögerung erzeugt, die der echten Umgebung nachgebildet ist. Es wurde für die Tests $d = 4$ gesetzt. Diese Berechnung der Geschwindigkeiten gilt für die Vor- und Rückwärtsbewegung des Roboters sowie der Rotation. Die Minimal- und Maximalwerte (min und max)

werden einzeln festgelegt. Anhand der Bewegung können die neuen Positionen auf dem Feld berechnet werden.

$$x_n = x_{n-1} + s_n * cos(\varphi) * c \tag{4.4}$$

$$y_n = y_{n-1} + s_n * sin(\varphi) * c \tag{4.5}$$

Wobei c eine Konstante ist um die Relationen einzuhalten. Und x_n, y_n die globalen Koordinaten des Roboters zum Zeitpunkt n.

Es gibt noch Funktionen die als Abbruchkriterien dienen. Sie überprüfen, ob der Roboter nah genug am Ball, richtig ausgerichtet und langsam genug ist. Erst wenn alle Bedingungen stimmen gilt das Ziel als erreicht und die aktuelle Runde terminiert. Dabei wurde ein quadratischer Bereich von ca 20cm vor dem Ball und ein Winkel zum Tor von weniger als 60° als Ziel angegeben. Dieser Winkel reicht, um beispielsweise einen Seitwärtsschuss in Richtung des Tores auszuführen.

4.1.2 Neuronales Netz

Das nächste Modul ist das neuronale Netz. Es ist für das Finden des Pfades zum Ball und der korrekten Ausrichtung zuständig.

Ein Neuron hält den aktuellen Bias der mit 0 initialisiert wird und den Zeitwert τ, der mit 3 initialisiert wird. Außerdem hat jedes Neuron seine Aktivierungsfunktion. Diese Funktionen beschreibt die Ausgabe des Neurons.

Es wird ein CTRNN benutzt, wie unter 2.4.3 gezeigt. Ist das τ hoch gewählt, werden mehr Zeitschritte benötigt bis sich die Ausgabe verändert. Bei einem niedrigen τ reagiert das Neuron schnell auf neue Eingaben. Hierdurch ist eine Art selektiver von Filterung der Daten möglich. In diesem Fall bedeutet es, dass der Roboter nicht sofort die Richtung ändern will, wenn die Daten der Ballposition einen Ausreißer enthalten.

Als Sigmoidfunktion kommt Formel 4.6 zum Einsatz. a ist die Aktivierung des Neurons, Φ bildet diese auf den Wertebereich zwischen -1 und 1 ab.

$$\Phi(a) = \frac{a}{1 + |a|} \tag{4.6}$$

Das Netz besitzt eine mehrschichtige Topologie. Es gibt eine Eingabeschicht mit 7 Neuronen und eine versteckte Schicht mit 7 Neuronen. Die Ausgabeschicht besteht aus 3 Neuronen, welche die Bewegung des Roboters angeben (siehe Abbildung 4.2). Die Eingabeneuronen geben die Eingaben direkt weiter nachdem die Sigmoidfunktion angewandt wurde, die anderen beiden Schichten berechnen ihre Ausgaben mit der Aktivierungsfunktion des CTRNN.

Als Eingabe dienen die Entfernungen zum Ball und zum Tor in Metern mit je beiden Achsen sowie die Vektoren des Potential Fields der letzten Iteration. Diese spiegeln die Differenz zwischen der gewollten Bewegung des Netzwerkes und der tatsächlichen Bewegung wieder.

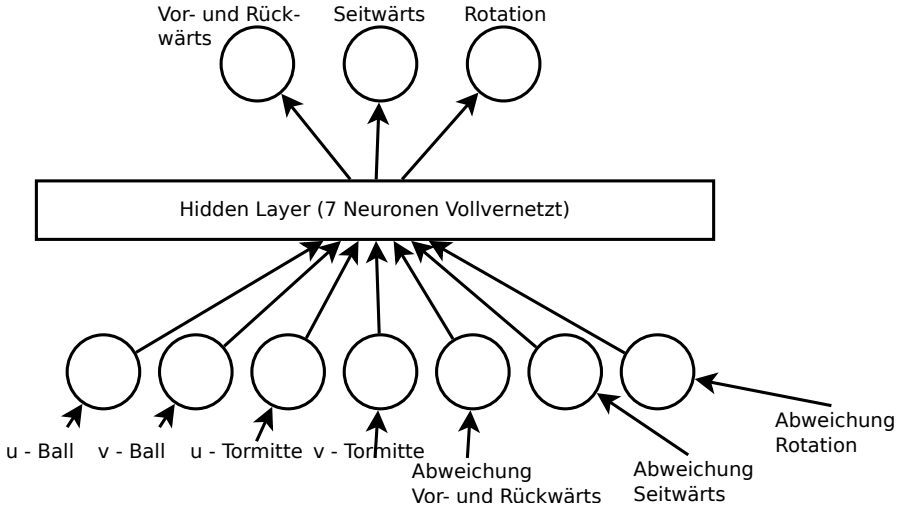


Abbildung 4.2: Aufbau des Netzes

4.1.3 Potential Field

Das neuronale Netz bildet die wichtigste Schicht für das Potential Field (siehe: 2.3). Die anderen Schichten bestehend aus den Repulsoren durch die Hindernisse, werden hier generiert. Das Potential Field wird für jeden Schritt dynamisch generiert, da keine globalen Positionsinformationen verfügbar sind. Da hier keine komplexen Funktionen benötigt werden ist das Feld trotzdem performant. Die Berechnung erfolgt wie in 2.3 beschrieben. Als Funktion der Stärke zeigte sich $force = (\frac{800}{dist})^2$ als gute Wahl.

Die Vektoren der einzelnen Netze werden alle aufsummiert und normiert wodurch sich der Vektor für den Roboter bildet(siehe Formel: 2.4). Dieser wird nun je nach Einstellung direkt auf die Vorwärts-, Rückwärts- und Seitwärtsbewegung addiert.

Je näher der Roboter gerade an einem Hindernis ist umso stärker wirkt die Kraft des Felds auf den Roboter. Dadurch kann der Roboter ausweichen.

Der Wert der Rotation wird nicht angepasst, da so beispielsweise der Roboter in der Nähe des Balls vom Ball weggedrückt werden würde und eine Ausrichtung so deutlich erschwert wird (siehe auch Abschnitt 4.3.3).

Felder werden zum Einen von Hindernissen erzeugt, denen so ausgewichen werden kann. Zum Anderen erzeugt der Ball ein Repulsorfeld. So soll verhindert werden, dass der Roboter gegen den Ball läuft, während er sich ausrichtet. Des Weiteren sorgt dieses Feld dafür, dass der Roboter abbremst bevor er das Ziel erreicht. Die Kraft ist hier geringer als bei Gegnern. Außerdem erzeugt der Ball wahlweise noch einen konstanten Vektor in seine Richtung, dies war bei den Durchläufen mit dem CTRNN nicht mehr notwendig.

4.1.4 Evolution

Die Evolution bearbeitet eine zuvor eingestellte Höchstzahl von Generationen oder bricht bei einer ausreichenden Genauigkeit ab. Vor dem ersten Durchlauf werden gemäß der Größe der Population zufällige Individuen bzw. neuronale Netze erstellt. Die Gewichte der Kanten werden dabei zufällig gesetzt während der Bias und der Tau-Wert (τ) mit null bzw. drei initialisiert werden.

Bei allen Zufallsoperationen wird ein Seed aus einem logischen Zeitstempel und einem gesetzten Ausgangsseed generiert. Damit können trotz Zufallszahlen Durchläufe exakt wiederholt werden.

Zentral für die Evolution ist die Bewertung der jeweiligen Netze mit einer Fitnessfunktion. Die entsprechende Werte bekommt das Modul aus der Simulation, die die Szenarien schrittweise abarbeitet.

Es reicht nicht, die Zeit zu messen, die der Agent zum erreichen des Zieles benötigt. Gerade die Ausrichtung zum Tor könnte nur durch Zufall gelingen, wenn es keinen evolutionären Druck gäbe, die richtige Orientierung einzunehmen. So wird das Netz belohnt, wenn es in der Nähe das Balles die richtige Orientierung annimmt. Außerdem wird das Netz bestraft, wenn es zu nah an Hindernisse kommt (siehe Formel: 4.7).

$$b = \sum_{i=0}^{N} \left[k_1 + k_2 * \frac{d_i}{d_0} + k_3 * |w_i| * \left(\frac{500}{d_i} \right)^{1.75} + k_4 * \sum_{h=1}^{H} \left(\frac{1}{d_i^h} \right)^{2.75} \right] \tag{4.7}$$

b : Bestrafung für das Netz
k_1 : Konstante Strafe pro Bewegungsschritt
k_2 : Konstanter Wert für Gewicht der Distanz
k_3 : Konstanter Wert für Gewicht der Ausrichtung
k_4 : Konstanter Wert für Gewicht der Hindernissentfernung
w_i : Winkel zum Tor zum Zeitpunkt i
d_i : Distanz zum Ball zum Zeitpunkt i
d_i^h : Distanz zum Hinderniss h zum Zeitpunkt i
N : Anzahl der Bewegungschritte
H : Anzahl der Hindernisse

$$Fitness = \frac{1}{b} \tag{4.8}$$

Dabei zählen mehrere Faktoren in die Berechnung der Fitness ein. Die Summe der Entfernungen des Roboters zum Ball, die Ausrichtung zum Tor in Zusammenhang mit der aktuellen Distanz zum Ball sowie die Distanz zu den Hindernissen.

Die hier gewonnene Fitness wird über alle Szenarien bestimmt und aufsummiert. So erlangen Netze eine gute Fitness die gute Ergebnisse in allen oder möglichst vielen Szenarien erreichen.

Als Auswahlkriterium wird die Fitness betrachtet und eine Rang basierte Selektion angewendet. Dabei wurde mit einer Wahrscheinlichkeit von $p = 0.2$ das

gerade betrachtete Element in der nach Fitness sortierten Menge ausgewählt und von Beginn der Menge neu begonnen ein Element auszuwählen. Die Fitness wurde außerhalb der Reihenfolge nicht betrachtet.

Außerdem werden die drei bestbewerteten Individuen übernommen um ein zufälliges Verlieren einer guten Lösung zu verhindern. Somit wurden relativ stark die besten Individuen selektiert. Dies sorgt für ein schnelles Ansteigen der Fitness. Trotz dieses hohen Elitismusses gab es keine Probleme mit lokalen Maxima.

Die Mutation passt Bias, τ und die Kantengewichte an, es kommt die Gaußfunktion mit $\sigma = 1$ zur Anwendung.

4.2 Anbindung

Der Quelltext der Potential Fields und des Neuronalen Netzes kann direkt in das Framework des Roboters eingefügt werden, da es hier keine Abhängigkeiten gibt. Die Klassen können im Verhalten einfach aufgerufen werden. Als Input dienen weiterhin die Entfernungen zu Ball und Tor, sowie die Abweichung beim letzten Zeitschritt. Die ausgegebenen Geschwindigkeitswerte werden mit den Potential Fields verrechnet und sofort an das Framework weitergegeben, welches diese in Bewegung umsetzt. Die Simulation entfällt auf dem Roboter, da hier die Bewegung tatsächlich umgesetzt wird. Auch das Training soll zumindest nicht hauptsächlich auf dem Roboter stattfinden. Vielmehr wird ein Netz mittels des Pythonmoduls cPickle aus der Simulation auf einem PC gespeichert und auf den Roboter übertragen, sodass das trainierte Netz dort verfügbar ist.

4.3 Ergebnisse

Es zeigte sich, dass die Wahl verschiedener Parameter bezüglich Evolution und Neuronaler Netze auf das Endergebnis oftmals wenig Einfluss hatte, auch wenn Ergebnisse teils schneller erzielt wurden. So erwies sich eine Netztopologie mit 7 Eingabeneuronen, 7 Versteckten und 3 Neuronen zur Ausgabe als effizienteste Wahl. Sofern nicht anders angegeben erfolge das Training der Netze mit bestimmten Parametern, die im Anhang unter A.1 zu finden sind.

4.3.1 Lernfortschritt

In Abbildung 4.3 ist der Vergleich der jeweiligen besten Netze zum in der Evolution in Abhängigkeit zur Anzahl von Neuronen im internen Layer zu sehen. Bei sieben Neuronen zeigte sich das beste Bild. Dabei ist zu erkennen, dass die Fitness über die in etwa ersten 100 Generationen das größte Wachstum zeigt und sich danach nur noch langsam verbessert. Die Durchläufe mit neun Neuronen erreichten eine Qualität auf dem selben Niveau, benötigten allerdings deutlich mehr Generationen und mehr Rechenzeit pro Iteration. Fünf Neuronen zeigten ein minimal schlechteres Wachstum der Fitness.

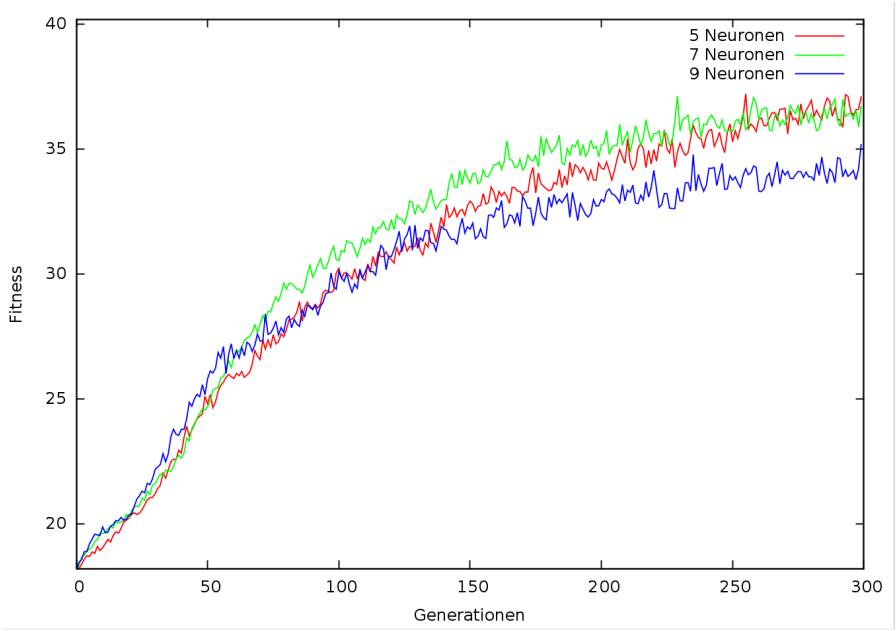
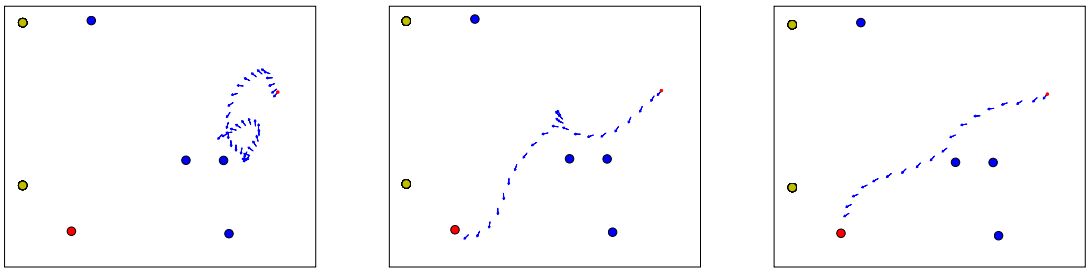


Abbildung 4.3: Verlauf der höchsten Fitness pro Generation bei 5,7 und 9 Neuronen in der verstecken Schicht (Mittelwerte aus 3 Durchläufen)



(a) Nach 1 Generation (b) Nach 51 Generationen (c) Nach 101 Generationen

Abbildung 4.4: Lernfortschritt der Evolution

Bei Abbildung 4.4 sind die Pfade nach 1(a), 51(b), 101(c) Generationen der Evolution zu sehen. In Tabelle 4.1 findet sich die Legende zu den nachfolgenden Abbildungen.

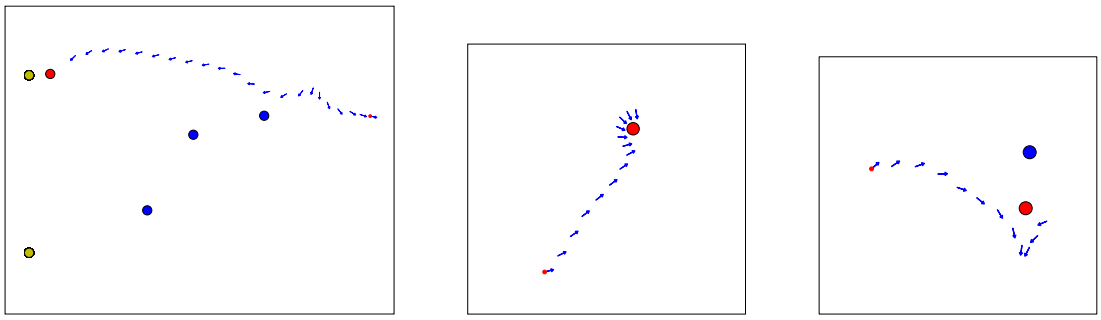
4.3.2 Funktionen der Software

Das erzeugte Netz in Kombination mit den Potential Fields generierte in den meisten Fällen gute Ergebnisse. Es wurde das gewünschte Verhalten erreicht und in Teilen sogar übertroffen.

So wird zuverlässig zum Ball gefunden und sich in Richtung des Tores ausgerichtet. Dabei erfolgt die Ausrichtung zum Tor meistens schon vor Ankunft am Ball (siehe Abbildung: 4.5a). War dies nicht möglich, erfolgte die Ausrichtung direkt am Ball (Abb. 4.5b). Auch wenn ein Hindernis den Weg versperrte, konnte

Tabelle 4.1: Legende für Grafiken

Großer roter Punkt	Ball
Blaue Punkte	Hindernisse
Gelbe Punkte	Torpfosten
Kleiner roter Punkt	Startposition
Blaue Pfeile	Position und Ausrichtung des Roboters über die Zeit (jeder 3. Zeitschritt)



(a) Ausrichtung auf dem Weg (b) Ausrichtung am Ball (c) Rückwärts Ausrichten

Abbildung 4.5: Ausrichten

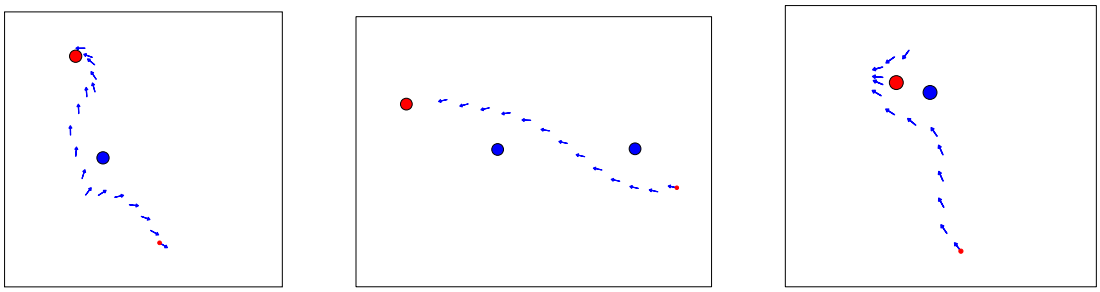
in den meisten Fällen ein guter Weg als Alternative gefunden werden (Abb. 4.6c). Auch wenn kein Hindernis am Ball war zeigte sich in einigen Fällen ein ähnlicher Pfad um den Ball herum. Auch wenn dies auf den ersten Blick nach einem Umweg aussieht, ist diese Option zeitsparend, da so der Roboter sich so weniger drehen muss, besonders in den Fällen, in denen der Roboter zunächst am Ball vorbei lief (siehe Abbildung: 4.5c).

Es zeigte sich, dass das Netz schnell lernt, dass Vorwärtslaufen effizienter ist als Rückwärtslaufen, ein anfängliches Wenden aber vergleichsweise lange dauert. Falls der Ball zum Start hinter dem Roboter lag zeigte sich, dass sich der Roboter bei günstigen Gelegenheiten effizient umdrehen kann, sofern dies nötig war. (siehe Abbildung: 4.5a).

Durch die Potential Fields ist der Roboter in der Lage Hindernissen auf dem Pfad auszuweichen (Abb.: 4.6) oder die Positionierung am Ball so zu ändern, dass ein anderer Weg gefunden wird. Dabei kann es auch mit einer hohen Anzahl von Hindernissen umgehen. Auch bestimmte Aktionen wie das Wenden konnten meistens ohne großen Effizienzverlust durchgeführt werden.

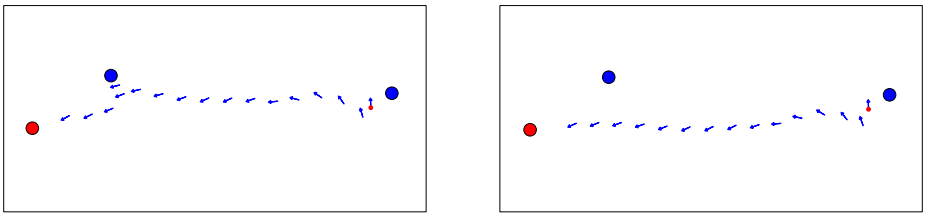
Der Roboter war nicht nur in der Lage Hindernissen auszuweichen, auch gelang es durch einen schwächeren Repulsor sehr gut den Ball so zu umlaufen, dass dieser nicht aus Versehen berührt werden würde(Abbildung: 4.6c). Dieses Feld sorgte des Weiteren schon alleine für ein Abbremsen des Roboters vor dem Ball .

Einstellbar ist unter anderem die Stärke der Repulsoren. Hier musste eine gute Mitte gefunden werden. Waren die Felder zu schwach kamen die Roboter den Hin-



(a) (b) (c)

Abbildung 4.6: Ausweichen



(a) Fitnessfunktion ohne Hindernisse (b) Fitnessfunktion mit Hindernisse

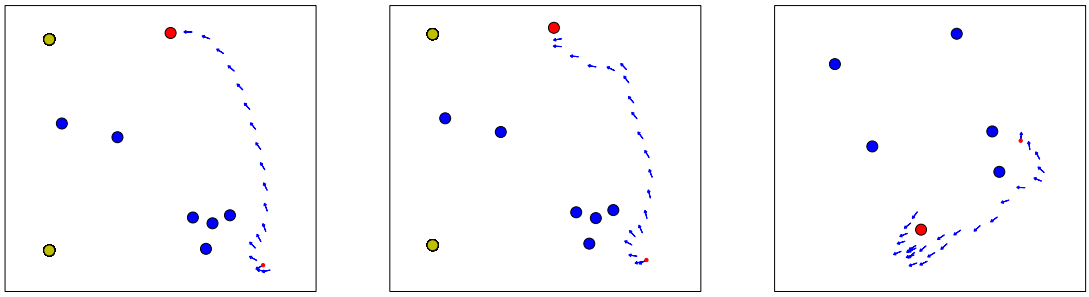
Abbildung 4.7: Druck zum Hindernis, mit verschiedenen Fitnessfunktionen trainiert

dernissen zu nah. War der Druck zu stark kam es teils zu großen Umwegen, da der Roboter nicht mehr zwischen zwei Hindernissen entlanglief (siehe Abbildung: 4.9c).

Es zeige sich bei ersten Tests auch nicht vorausgesehenes Verhalten. So lernte das Netz schnell, dem Druck des Potential Fields zum Ausweichen des Gegners entgegen zu wirken, da so geringere Umwege beim Training gelaufen werden mussten. Dies war zunächst auch erwünscht, da so das Netz flexibel auf den Input der Potential Fields reagieren kann. Dadurch kam es jedoch gelegentlich auch zu Bewegungen auf Hindernisse zu, wenn dieses gerade passiert wurden (siehe Abbildung: 4.7a). Dies konnte durch ein Anpassen der Fitnessfunktion behoben werden (siehe Abbildung: 4.7b). Es wurde die Entfernung zu den Hindernissen mit einbezogen, was zu besseren Ergebnissen führte. Andere Teile des Pfades waren davon nicht betroffen

Überraschend gut konnte das Netz mit stark verrauschten Daten umgehen wenn es hier rauf trainiert wurde(siehe Abbildung: 4.8b). Es wurde ein zufälliger, normalverteilter Wert auf die Eingabedaten des CTRNNs addiert. Dabei war als Standardabweichung $\sigma = 0.3$ gesetzt. Dies entspricht 30 cm. Im Vergleich zeigt sich, dass das Netz sehr gut damit umgehen kann. Wenn das Netz wie in Abbildung 4.8c ohne Rauschen trainiert wurde, kam es bei der Ausrichtung etwas häufiger zu Problemen.

Gelegentlich, gerade bei vielen Hindernissen, kam der Roboter aufgrund der Bewegungsverzögerung zu nah an ein Hindernis und musste ein kleines Stück zurücklaufen, bevor er seinen Pfad fortsetzten konnte. Dies kostete Zeit, verhinderte



(a) Sehr geringes Rauschen (b) Starkes Rauschen (c) Probleme auf dem Letzten Stück

Abbildung 4.8: Starkes Rauschen auf den Daten

aber erfolgreich eine Kollision (Abbildung: 4.9a). Durch weitere Anpassungen, bei der die Abweichung der Bewegungen in Zusammenhang mit der aktuellen Bewegungsrichtung gestellt werden, könnte dies auch vermieden werden, indem so vorausschauender seitlich zur Bewegungsrichtung ausgewichen wird.

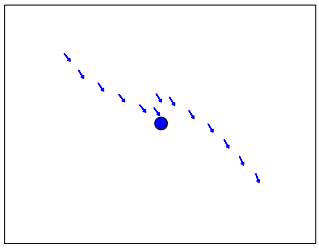
An einigen Stellen legte die evolutionäre Optimierung Schwächen der Simulation offen, indem sie zum Beispiel ausnutzte, dass die Maximalgeschwindigkeit für vorwärts und seitwärts einzeln berechnet werden. So lernte das System leicht schräg zu laufen um so eine erhöhte Gesamtgeschwindigkeit zu haben. Dies spiegelt zwar die existierenden Grenzwerte im echten Framework wider, dort würde der Roboter allerdings instabiler laufen, was in der Simulation zu diesem Zeitpunkt nicht betrachtet wird.

Wenn sich mehrere Hindernisse in der Nähe des Balles befanden, wurde zum Teil ein zu umständlicher Weg eingeschlagen(siehe Abbildung: 4.9b), da die einzelnen Hindernisse in der Simulation nicht getrennt betrachtet werden. Dies Zeigt eine der Schwächen des Verfahrens auf. Solche Situationen kommen allerdings selten vor und sind auch mittels anderer Verfahren schwer lösbar.

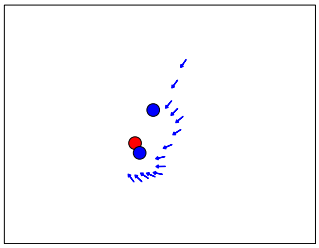
4.3.3 Potential Fields

Bei den Potential Fields gab es zwei unterschiedliche Möglichkeiten die Bewegung des Roboters zu beeinflussen, da der Roboter sowohl seitwärts laufen, als sich auch drehen kann. In Abbildung 4.12 sind die Methonden im Vergleich zu sehen. In Abbildung 4.12a kam ausschließlich das Potential Field ohne Rotation zum Tragen. Es wurde sich entsprechend nicht am Ball ausgerichtet und die Orientierung während des gesamten Pfades nicht verändert. In Abbildung 4.12b wurde stattdessen die Rotation benutzt. In vielen Szenarien wurde so auch ein Pfad gefunden. In 4.12c ist die bewährte Kombination von Potential Fields (ohne Rotation) und CTRNN im Vergleich zu sehen. Diese Kombination hat Vorteile im Gegensatz zur Rotation. Wie in Abbildung 4.11 zu sehen ist war das Ausweichen in einigen Situationen nicht sonderlich effektiv. Außerdem kam es zu Problemen beim Ausrichten zum Ball wie unter Abbildung 4.9d

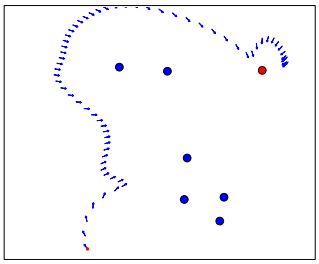
Es ist, wie gezeigt, auch möglich, mittels der Repulsoren und einem Attraktor



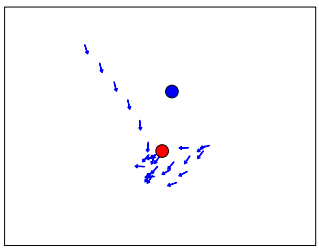
(a) Muss bei Hindernis zurück



(b) Hindernis im Weg

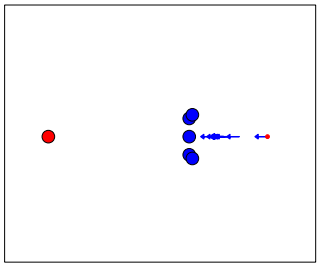


(c) Zu starkes Feld

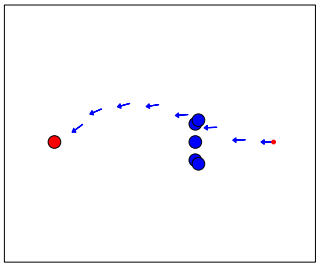


(d) Schafft Ausrichtung nicht

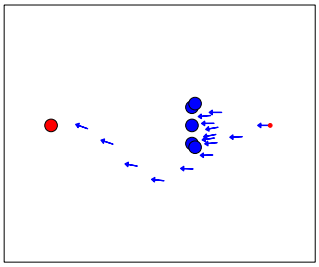
Abbildung 4.9: Aufgetretene Probleme



(a) Potential Field



(b) Neuronales Netz



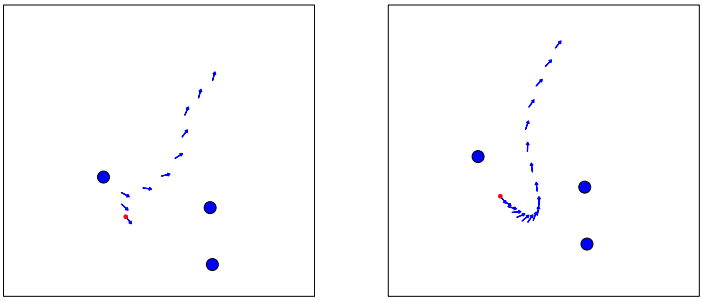
(c) Kombinierte Lösung

Abbildung 4.10: Lokale Extrema

zum Ball zu finden. Allerdings ist es wahrscheinlicher in lokale Maxima zu geraten (Abbildung: 4.10a) als bei einer hybriden Lösung. (Abbildung: 4.10c). Durch die Kombination mit den CTRNN konnte der Sackgasse entkommen werden und es wurde den Hindernissen ausgewichen. Dabei ist zu erwähnen, das hier nur Repulsoren und Attraktoren zum Tragen kamen. Mittels komplexerer Felder oder einem zufälligen Feld kann dies auch in einem begrenzten Rahmen verhindert werden.

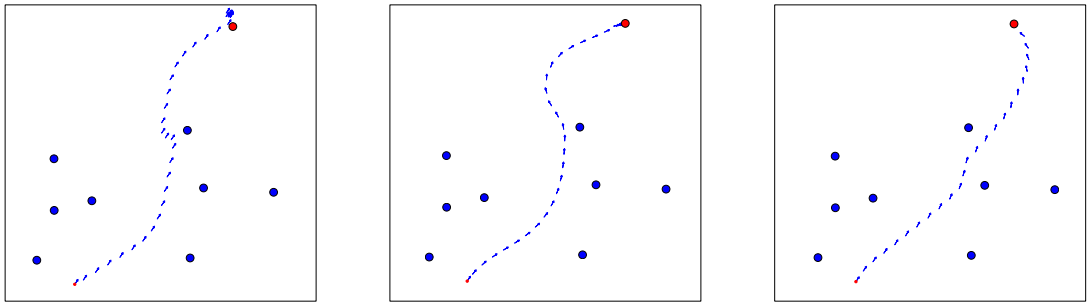
4.3.4 Neuronales Netz

Auch eine Lösung mit ausschließlich neuronalen Netzen ist möglich. So wurden geeignete Wege gefunden. Hier ist allerdings das Ausweichen von Hindernissen nicht direkt umsetzbar (siehe Abbildung: 4.13). Außerdem kommt es ggf. zu Kollisionen



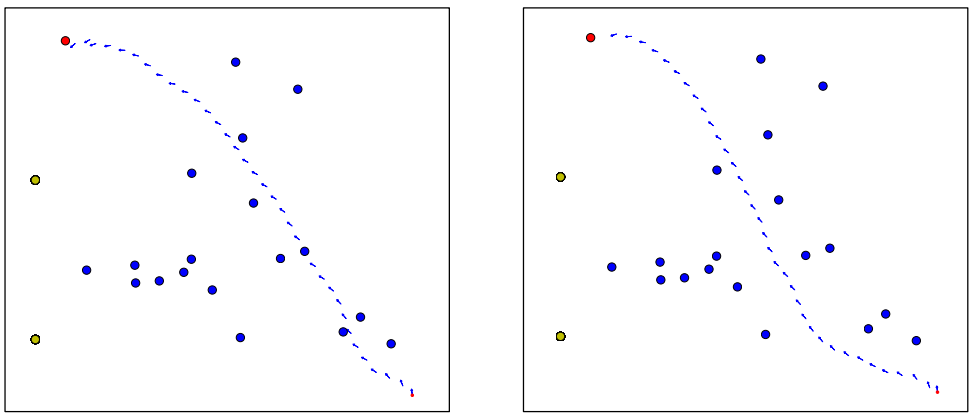
(a) Seitwärts (b) Drehen

Abbildung 4.11: Ausweichen mit Potential Fields



(a) Seitwärts (b) Drehen (c) Seitwärts mit Neuronal

Abbildung 4.12: Potential Fields



(a) Nur CTRNN (b) CTRNN und Potential Fields

Abbildung 4.13: CTRNN

mit dem Ball beim Ausrichten. Ließ man diese Bereiche außer Acht, zeigten sich schon schöne Ergebnisse mit einer Intelligenten Pfadfindung mit einem effizienten Ausrichten am Ziel.

4.4 Diskussion

4.4.1 Einordnung

Zu anderen Verfahren

Wie in 4.3 gezeigt bietet eine Kombination der beiden Verfahren Vorteile gegenüber den einzelnen Konzepten im Einzelnen. So hatte die Vorgehensweise mit Potential Fields Probleme mit Sackgassen (lokalen Maxima) und der richtigen Ausrichtung. Dafür entfiel das aufwendige Erstellen der komplexen Felder und es konnte auf simple Repulsoren zurückgegriffen werden. CTRNN leisten dies, brauchen aber eine Möglichkeit, auf eine variable Anzahl von Hindernissen einzugehen. Das kombinierte Verfahren zeigte Pfade auf, die effizient erscheinen und mit herkömmlichen Methoden wohl nicht entstanden wären.

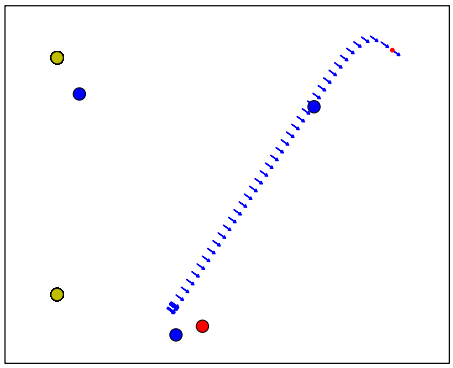
Zu bisherigem Verfahren

Zumindest in der simulierten Umgebung zeigte sich dieses Verfahren als deutlich effizienter als das Bisherige. Der Ball wurde deutlich schneller erreicht. Auch wenn der Ball hinter dem Roboter lag konnte sich schnell ausgerichtet werden, außerdem geschah die Ausrichtung zum Tor in den meisten Fällen schon auf dem Weg zum Ball und nicht erst bei Ankunft. Abbildung 4.14 zeigt einen Vergleich vom alten Verfahren, dem alten Verfahren mit den hier benutzten Potential Fields sowie dem neuen Verfahren mit CTRNNs und den Potential Fields.

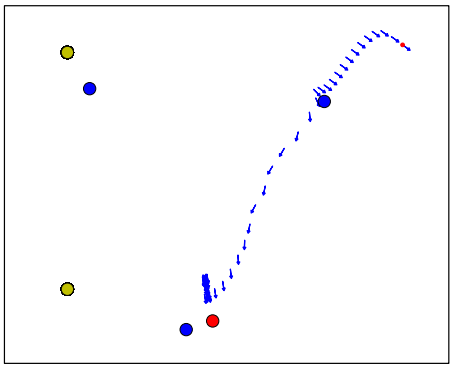
Der Teil zur Ausrichtung in Richtung des Tores des alten Verhaltens wird nicht dargestellt, da dieser Teil nicht direkt in der Simulation anwendbar ist. Bei der zweiten Variante wird der Pfad durch das Hindernis gestört. Dadurch gerät der Roboter in eine andere Phase der Wegfindung, für den Fall, dass der Ball vor dem Roboter liegt.

Zum theoretischen Optimum

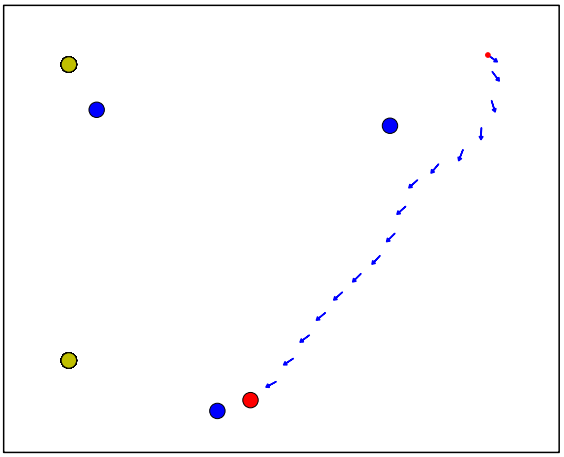
Ein theoretisches Optimum ist schwer zu definieren, da es einen großen möglichen Lösungsraum gibt. In den meisten Fällen wurde ein guter Weg gefunden. Bei dem Ausweichen der Hindernisse zeigte sich ebenfalls meist eine gute Lösung auch wenn es Fälle gab wie das direkte auf ein Hindernis zu laufen bei denen der Agent zu stark abbremste oder lieber einen anderen Weg hätte wählen sollen. Teils wurden sogar Lösungen gefunden die besser waren als der Pfad den man intuitiv gewählt hätte wenn man den Roboter manuell steuern würde (siehe Abbildung: 4.5c).



(a) Das alte Verfahren



(b) Altes Verfahren mit Potentieal Fields



(c) Das neue Verfahren

Abbildung 4.14: Vergleich zum bisherigen Verhalten

4.4.2 Softwarelaufzeit

Das Trainieren des Netzes war im Vergleich zur Laufzeit des Netzes selber sehr aufwendig. Potential Fields und das neuronale Netz können ohne Probleme auf dem Roboter ausgeführt werden, ohne dass es zu massiven Performanzproblemen kommt. Das Berechnen des CTRNN mit 7 Neuronen dauerte mit Python2.7 ca. 0.18 ms (5 Neuronen: 0.12ms). Die Potential Fields brauchten pro Zyklus ca. 0.03 ms bei 8 Hindernissen (0.06 ms bei 18).

Das Training hingegen muss offline auf einer anderen Hardware, wie beispielsweise einem PC geschehen, da hier viel Rechenzeit benötigt wird. Das Verwenden eines anderen Interpreters wie dem pypy-Interpreter[16] beschleunigte das Verfahren stark (ca. Faktor 5). Das Lernen war in diesem Fall je nach Parametern nach einigen Minuten für kleine Testsätze und einigen Stunden bei großen Testsätzen abgeschlossen (übliche PC-Hardware, amd64 Architektur, 2.9 GHz). Diese Zeiten, sowohl Training als auch auf dem Roboter können durch gezielte Programmoptimierungen, Parallelisierung oder die Wahl einer anderen Programmiersprache

weiter reduziert werden.

4.4.3 Anwendbarkeit im aktuellen Framework

Auch wenn die Kombination beider Verfahren gute Ergebnisse in der Simulation zeigten ist dies zum jetzigen Zeitpunkt außerhalb der Simulation noch nicht direkt im RoboCup Framework der Hamburg Bit-Bots anwendbar, da nicht durchgehend Positionsinformationen aller relevanten Objekte auf dem Feld vorhanden sind. Wenn ein gutes lokales Weltmodell existiert, dass Daten zu Toren und Anderem dauerhaft verlässlich darstellt auch wenn diese außerhalb des Sichtbereiches liegen, sollte es möglich sein, nach Anpassung einiger Parameter, das trainierte Netz auf dem Roboter für eine effiziente Wegfindung zu benutzen.

Kapitel 5

Fazit und weitere Arbeit

Ziel dieser Arbeit war es mittels einer Kombination von Potential Fields und einem bioinspirierten Verfahren eine gute Wegfindung in einer Simulationsumgebung zu erreichen. Dies ist gelungen. Es konnten im Rahmen einer Simulation gute Ergebnisse erzielt werden (siehe: 4.3.2). So wurde der Ball mit nur wenigen Ausnahmen, fast immer erreicht. Auch eine korrekte Ausrichtung am Ziel war möglich sofern kein Hindernis die Position blockierte. Das Ausrichten erfolgte schon auf dem Weg und war in der Regel sehr effizient.

Mittels CTRNNs lassen sich gut komplexe Wege und Verhaltensweisen trainieren (siehe: 4.3.4). Dabei bieten diese Netze auch gleich Möglichkeiten zur Filterung der Daten. Gerade in dem gegebenen Kontext ist eine Filterung sehr hilfreich, da teils mit vielen Ausreißern zu rechnen ist. Während das Training der Netze rechenaufwendig ist, verhalten sich die Netze in ihrer Funktion sehr performant und benötigen nur wenig Rechenzeit (siehe: 4.4.2). Das evolutionäre Training hatte keine merklichen Probleme mit lokalen Extrema und schaffte es immer gute Lösungen zu finden. Das CTRNN alleine erzeugte geeignete Wege, jedoch ohne die Hindernisse zu betrachten.

Potential Fields wurden ergänzend eingesetzt Sie können auch mit einer hohen und variablen Anzahl von Repulsoren umgehen und so eine effektive Obstacle Avoidance umsetzten (siehe: 4.3.3). Auch das Berühren des Balles lies sich so verhindern. Dabei kamen nur simple Felder wie Repulsoren zum Tragen. Auch hier war wenig Rechenleistung notwendig. Durch den geringen Rechenaufwand ist dieses Verfahren gut für Roboter bzw. autonome Systeme geeignet.

Davon ausgehend, dass alle Objektpositionen relativ zum Roboter bekannt sind, ließen sich mit diesem Verfahren auch ohne eine globale Lokalisation gute Ergebnisse erzielen.

Auch wenn das Verfahren bereits gute Lösungen zeigt kann beispielsweise weiter an der Fitnessfunktion gearbeitet werden, um noch bessere Pfade oder eine kürzere Trainingszeit zu erzeugen. Ebenso wichtig ist eine möglichst realitätsnahe Simulation. So könnte eine 3D-Simulation zum Tragen kommen um Themen wie Stabilität mit zu betrachten.

In weiteren Arbeiten könnte das hier trainierte Netz weiter verbessert werden,

indem das Training individuell auf die Roboter abgestimmt wird. So könnte ein ständiges links Driften eines Roboters aufgrund ermüdeter Hardware kompensiert oder, je nach Untergrund, mehr oder weniger gedreht werden. Das Netzwerk müsste dies auch schon leisten können, wenn zum Beispiel Feedback mittels einer Deckenkamera gesammelt wird. So könnte das Netz trainiert werden mit einem Abdriften umzugehen. Dies ist allerdings noch zu testen.

Denkbar wäre auch eine Lösung, die dies während des Produktivbetriebes flexibel erlernen kann. Beispielsweise indem mittels bestimmter Verfahren der Fehler von gewollter und tatsächlicher Bewegung zum Lernen benutzt wird um noch bessere Ergebnisse zu bekommen.

Es wäre auch denkbar, das System auf mehrere Agenten zu erweitern und so zu versuchen, komplexe Verhaltensweisen zu erlernen. So wäre es möglich, mittels weiterer Eingaben in das Netzwerk, Entfernungen anderer Roboter zum Ball zu betrachten um so ein abstimmendes Verhalten zu erzeugen.

Ebenso kann das Konzept der Potential Fields in diesem Kontext erweitert werden, hier gibt es viele Möglichkeiten. So könnten die Seitenlinien als Repulsoren dienen um zu verhindern, dass der Roboter ungewollt das Spielfeld verlässt. Dazu ist allerdings eine weiterführende Lokalisation notwendig.

Anhang A

Parameter

Tabelle A.1: Parameter für CTRNN:

Kantengewicht	$[-5, 5]$
τ	$[0.5, 4.5]$
Bias	$[-6, 6]$
versteckte Neurone	7

Tabelle A.2: Parameter für Evolution:

Populationsgröße	70
Mutationswahrscheinlichkeit	0.02
Selektionswahrscheinlichkeit	0.2
Garantiert übernommene Individuen	3
Anzahl Testszenarien pro Generation	400
Anzahl Generationen	300

Tabelle A.3: Parameter für Simulation:

Abbruchkriterien	Winkel zm Tor, Position am Ball
Rauschen auf den Daten	Gauß-Verteilung $\sigma = 0.001$ (in m)

Literaturverzeichnis

[1] Daniel J Amit. *Modeling brain function: The world of attractor neural networks*. Cambridge University Press, 1992.

[2] Ronald C Arkin. *Behavior-based robotics*. MIT press, 1998.

[3] Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[4] Team Berlin. Spirit of berlin: An autonomous car for the darpa urban challenge hardware and software architecture. *retrieved Jan*, 5:2010, 2007.

[5] Marc Bestmann. Bachelorarbeit: Biologically inspired localization on humanoid soccer playing robots. (eingereicht).

[6] J Falconer. Honda developing disaster response robot based on asimo. *IEEE Spectrum*, 11, 2013.

[7] Dario Floreano and Claudio Mattiussi. *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press, 2008.

[8] Bernd Fritzke et al. A growing neural gas network learns topologies. *Advances in neural information processing systems*, 7:625–632, 1995.

[9] David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Number 2. Addison-Wesley, Reading, MA, 1989.

[10] Hamburg Bit-Bots. Fotos von Wettbewerben, 2014. `http://www.bit-bots.de/` [Online; Zugegriffen 17.12.2014].

[11] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[12] Suzana Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in human neuroscience*, 3, 2009.

[13] Feng-hsiung Hsu. Ibm's deep blue chess grandmaster chips. *IEEE Micro*, 19(2):70–81, 1999.

[14] Sven Magg and Andrew Philippides. Gasnets and ctrnns–a comparison in terms of evolvability. In *From Animals to Animats 9*, pages 461–472. Springer, 2006.

[15] matplotlib. Python plotting, 2014. `http://matplotlib.org/` [Online; Zugegriffen 21.12.2014].

[16] pypy. alternative implementation of the Python language, 2014. `http://pypy.org/` [Online; Zugegriffen 21.12.2014].

[17] Python Software Foundation. Python 2.7. `https://www.python.org/` [Online; Zugegriffen 19.12.2014].

[18] RoboCup Federation. A Brief History of RoboCup. `http://www.robocup.org/about-robocup/a-brief-history-of-robocup/` [Online; Zugegriffen 20.12.2014].

[19] RoboCup Federation. HumanoidLeagueRules2014. `http://www.informatik.uni-bremen.de/humanoid/pub/Website/Downloads/HumanoidLeagueRules2014-07-05.pdf` [Online; Zugegriffen 21.12.2014].

[20] RoboCup Federation. HumanoidLeagueTeams2014. `http://www.informatik.uni-bremen.de/humanoid/bin/view/Website/Teams2014` [Online; Zugegriffen 21.12.2014].

[21] RoboCup Federation. RoboCup2013 - Final Report. `http://www.robocup2013.org/final-report-robocup2013-available/` [Online; Zugegriffen 21.12.2014].

[22] Robotis. DARWIN-OP. `http://www.robotis.com/xe/darwin_en` [Online; Zugegriffen 17.12.2014].

[23] Raúl Rojas. *Neural networks: a systematic introduction*. Springer, 1996.

[24] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[25] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25, 1995.

[26] Terence D Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural networks*, 2(6):459–473, 1989.

[27] Tomoichi Takahashi and Masaru Shimizu. How can the robocup rescue simulation contribute to emergency preparedness in real-world disaster situations?

[28] Amir Tavafi, Narges Majidi, Michael Shaghelani, and Amir Seyed Danesh. Optimization for agent path finding in soccer 2d simulation. In *Mobile Communication and Power Engineering*, pages 109–114. Springer, 2013.

[29] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

Erklärung der Urheberschaft

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

Ort, Datum Unterschrift

Erklärung zur Veröffentlichung

Ich erkläre mein Einverständnis mit der Einstellung dieser Bachelorarbeit in den Bestand der Bibliothek.

Ort, Datum Unterschrift