Thema:

Selbstorganisation und Lernen –
Maschinelles Lernen im Robotfußball

Referat

im Rahmen des Seminars

Agenten und Robotfußball

im Fachgebiet Informatik

Themensteller: Prof. Dr. Wolfram-M. Lippe

Betreuer: Dr. Dietmar Lammers

vorgelegt von: Dieter Kerkfeld

Abgabetermin: 23.06.2003

Inhaltsverzeichnis

1 Einleitung

Robotfußball bietet eine ideale Testumgebung für das Benchmarking neuer Hard- und Software-Technologien. Der Soccer Server für die virtuelle Simulationsliga der Robot World Cup Initiative ermöglicht im Speziellen die Forschung an intelligenten Software-Systemen, ohne reale Roboter konstruieren zu müssen[1]. Diese Software-Systeme – im Folgenden intelligente Agenten genannt – zeichnen sich dadurch aus, dass sie ihr Verhalten autonom aus der Beobachtung ihrer Umwelt ableiten und damit ein bestimmtes Ziel verfolgen [Wo99, S. 32]. Die zentrale Anforderung beim virtuellen Robotfußball liegt in der Koordination mehrerer individuell agierender Agenten zur Erreichung eines gemeinsamen Ziels, dem Gewinn des Spiels.

Ziel dieser Ausarbeitung ist es, einen Einblick in maschinelles Lernen in komplexen Multi-Agenten-Umgebungen zu geben. Dieser Einblick orientiert sich am aktuellen Forschungsstand des RoboCup-Teams *Karlsruhe Brainstormers* [RM02a]. Wichtige Komponenten der Agenten dieses Teams werden bereits durch Methoden des Reinforcement Learnings (RL) realisiert. Die Motivation der Brainstormers liegt in der Entwicklung und Anwendung neuer Methoden und Algorithmen des RLs für hochkomplexe Anwendungsgebiete.

In Kapitel 2 wird zunächst der Forschungsansatz der Brainstormers vorgestellt. Anschließend wird in Kapitel 3 die Anwendung des RLs beschrieben. Kapitel 4 bildet den Abschluss der Ausarbeitung mit einem Fazit der dargestellten Inhalte und einem kurzen Ausblick auf mögliche zukünftige Entwicklungen.

2 Methodik der Karlsruhe Brainstormers

In diesem Kapitel wird der zugrunde liegende Forschungsansatz der Brainstormers vorgestellt. In Abschnitt 2.1 wird zunächst eine allgemeine Einführung in das RL gegeben. Anschließend wird in Abschnitt 2.2 die Architektur der aktuellen Brainstormers-Agenten beschrieben.

2.1 Einführung in Reinforcement Learning

Reinforcement Learning ist eine Methode des maschinellen Lernens, bei der ein Agent lernen soll, sich auf Basis eigener, gewonnener Erfahrungen in einer unbekannten Umwelt möglichst optimal zu verhalten. Als einziges Trainingssignal wird nach einer Sequenz von Entscheidungen ein Signal für Erfolg oder Misserfolg gegeben. Um ein

[1] Für eine eingehende Beschreibung des Soccer Servers und der implementierten Welt, Physik und Befehle: siehe [Ro02].

Lernproblem als Reinforcement Learning-Problem formulieren zu können, müssen drei Voraussetzungen erfüllt sein. Erstens muss der Agent über die Möglichkeit verfügen, den Zustand seiner Umwelt zu erfassen. Zweitens muss der Agent Aktionen durchführen können, die den Umweltzustand beeinflussen. Drittens muss dem Agenten ein Ziel vorgegeben werden, d. h. ein gewünschter Endzustand der Umwelt, den der Agent erreichen soll. Zur Lösung von RL-Problemen wurde eine Vielzahl von Verfahren entwickelt. Für einen einführenden Überblick über mögliche Verfahren siehe [SB98].

Beim RL interagieren der Agent und seine Umwelt kontinuierlich. Der Agent führt Aktionen aus, die Umwelt reagiert darauf und der Zustand ändert sich. Als Rückmeldung erhält der Agent nur ein numerisches Reinforcement-Signal. Reinforcement-Signale können als Kosten für seine Aktionen aufgefasst werden. Der Agent muss ausprobieren, wie er das vorgegebene Ziel mit minimalen Kosten erreichen kann. Falls der Agent zufällig sein Ziel, einen so genannten positiven Endzustand erreicht, war zumindest die letzte Aktion erfolgreich. Die Sequenz muss aber nicht optimal gewesen sein, da zwischenzeitlich Fehler aufgetreten sein könnten, die wieder ausgeglichen wurden. Falls der Agent erneut in den Zustand der vorletzten Aktion gerät, weiß er nun, welche Aktion zum Erfolg führt. Zusätzlich weiß er, dass die Aktion, die zu diesem Zustand geführt hat, ebenfalls positiv war. Analog zum menschlichen Lernen wird der Agent wiederholt vor die Aufgabe gestellt, sein Ziel zu erreichen. Nach hinreichend vielen Durchläufen kann daraus rückwärtsgerichtet ein Verhalten erlernt werden, das aus möglichst vielen Anfangszuständen zum Ziel führt. Erreicht der Agent einen Zustand, der ihm bekannt ist, kann er eine bewährte Aktion anwenden (exploit) oder eine neue Aktion ausprobieren (explore). Dies ermöglicht ihm, optimales Verhalten zu entwickeln [He98, S. 58].

RL nutzt Markov-Entscheidungs-Prozesse (MDP, von engl.: Markov Decision Process), um RL-Probleme formal zu beschreiben. Eine ausführliche mathematische Darstellung findet sich in [Me99]. Ein MDP wird durch ein Tupel $M := [S, A, r, p]$ dargestellt. Dabei bezeichnet *S* die Menge aller Zustände der Umwelt, *A* die Menge aller möglichen Aktionen eines Agenten. $p(s' \mid s, a)$ beschreibt das Verhalten der Umwelt durch die Wahrscheinlichkeit, mit der das System von Zustand *s* in Zustand s' wechselt, wenn ein Agent die Aktion *a* ausgeführt. $r(s, a)$ gibt schließlich die direkten Kosten an, die dem Agenten für die Anwendung der Aktion entstehen.

Der Agent muss sequentiell Entscheidungen darüber treffen, welche Aktionen er ausführen will. Da eine einzelne Aktion nicht ausreicht, einen MDP entscheidend in Richtung eines gewünschten Zielzustands zu beeinflussen, ist eine Sequenz von Entscheidungen nötig. Eine Sequenz von Entscheidungsfunktionen zur Auswahl von Aktionen wird Taktik genannt. Sie lässt sich als Abbildung $\pi : S \to A$ modellieren. Eine Taktik kann durch eine zugehörige Kostenfunktion $J^{\pi}(s)$ bewertet werden. Jede

Aktion einer Taktik verursacht Kosten. Ziel ist es, eine optimale Taktik π^* zu finden. Die „gierige“ Anwendung der jeweils lokal günstigsten Aktion führt nicht zu einer optimalen Taktik. Eine zunächst „günstige“ Aktion kann anschließend zu hohen Folgekosten führen, während eine „teure“ Aktion später zu niedrigen Kosten führen kann. Die optimalen Kosten in einem Zustand ergeben sich nach dem „Prinzip der Optimalität“ aus der Summe der unmittelbar anfallenden Kosten und den Kosten im Folgezustand, wenn anschließend optimal gehandelt wird [Ri99, S. 324]. Eine optimale Taktik besteht demzufolge aus den Aktionen, die jeweils die Summe aus unmittelbar anfallenden Kosten und erwarteten, kumulierten Kosten, die für zukünftige Aktionen als Resultat einer gewählten Aktion entstehen werden, minimieren.

Mit dem dynamischen Optimierungsverfahren *Value Iteration* lässt sich eine optimale Taktik berechnen [RM02a, S. 5f.]. Wenn der Agent einen Zielzustand erreicht hat, können die erwarteten, kumulierten, zukünftigen Kosten der hinführenden Zustände zurückgerechnet werden. Die optimale Kostenfunktion $J^*(s)$ konvergiert unter schwachen Annahmen [BT96] aus der iterativen Verfeinerung der optimalen erwarteten, kumulierten, zukünftigen Kosten über alle Zustände. Aus der optimalen Kostenfunktion lässt sich anschließend die optimale Taktik ableiten, indem in jedem Zustand die kostenoptimale Aktion[2] ausgewählt wird [RM02a, S. 6].

Aus Effizienzgründen kann die Value Iteration bei den großen Zustands- und Aktionsräumen der Simulationsliga nicht direkt eingesetzt werden. Real Time Dynamic Programming-Verfahren betrachten dagegen nur eine Trajektorie wirklich auftretender Zustände eines Durchlaufs und nicht den gesamten Zustandsraum [Ri99 S. 325]. Daher sind sie effizienter als die Value Iteration.

Falls das Verhalten der Umwelt, in Form der Übergangswahrscheinlichkeit $p()$, nicht bekannt ist, kann *Q-Learning* eingesetzt werden. Dieses Verfahren ermöglicht die fehlenden Werte durch Beobachtung der Umwelt stochastisch anzunähern, um beispielsweise anschließend die Value Iteration anwenden zu können [Me99, S. 11].

2.2 Architektur der Brainstormers-Agenten

Die typische Architektur eines Agenten beim Robotfußball besteht aus zwei Basiskomponenten: ein Modul zur Sensorverarbeitung und ein Modul zur Entscheidungsfindung. In der Simulationsliga stellt die Sensorverarbeitung speziell Anforderungen an die Behandlung vergangener und ungenauer Daten und die Integration der Kommunikation zwischen den einzelnen Agenten. Aufgabe der Sensorverarbeitung ist eine aktuelle Einschätzung des Umweltzustandes. Diese ermittelte Repräsentation des Zustandsraums

[2] Die Aktion, deren Folgezustand die geringsten erwarteten, kumulierten, zukünftigen Kosten aufweist.

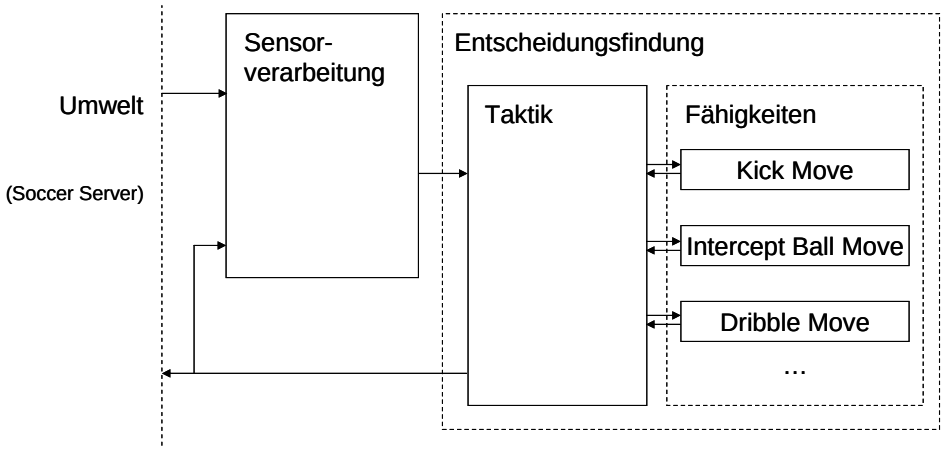
dient als Grundlage für die Entscheidungsfindung. Aufgabe der Entscheidungsfindung ist die Auswahl der Aktionen eines Agenten.

Forschungsschwerpunkt der meisten Teams in der Simulationsliga bildet das Entscheidungsfindungsmodul. Die Sensorverarbeitung scheint hingegen weit entwickelt zu sein und keinen großen Konkurrenzvorteil mehr zu bieten[3]. Als Grund hierfür wird eine geringere Komplexität der Sensorverarbeitung gegenüber der Entscheidungsfindung angenommen. Im Folgenden wird die Entscheidungsfindung betrachtet, da auch die Brainstormers hier ihre Forschung betreiben.

Die hohe Komplexität der Entscheidungsfindung resultiert aus drei Quellen. Die erste Quelle ist die große Anzahl möglicher Zustände der Spielumwelt. Im Soccer Server sind 5400^{23} verschiedene Stellungen der 22 Spieler und des Balls möglich. Der gesamte Zustandsraum mit der jeweiligen Geschwindigkeit aller Objekte und der aktuellen Ausdauer der Agenten[4] ist noch wesentlich größer und daher prinzipiell stetig. Zweitens stehen den elf Agenten eines Teams mindestens 300^{11} mögliche Aktionen pro Zyklus des Soccer Servers zur Auswahl. Drittens müssen ständig neue Entscheidungen für den nächsten Zyklus getroffen werden [RMM$^+$01, S. 2].

Um die Komplexität der Entscheidungsfindung beherrschbar zu machen, verfolgen die Brainstormers einen modularen Ansatz. Die Entscheidungsfindung wird in zwei Subprobleme geteilt. Zum einen werden Entscheidungen betrachtet, die individuelle Fähigkeiten wie „Ball annehmen“ oder „passen“ ermöglichen. Diese Fähigkeiten werden im Folgenden *Moves* genannt. Für jeden Move werden sein Ziel und die Anfangszustände, in denen er angewendet werden kann, definiert. Das Ziel eines Moves sollte nicht zu komplex gewählt werden, beispielsweise würde ein Move „Schieße ein Tor“ die Entscheidungsfindung nicht vereinfachen. Weiterhin verringert sich durch die Anwendung eines Moves die Zahl der vom Agenten zu treffenden Entscheidungen. Zum anderen werden Entscheidungen behandelt, die durch Verwendung der Moves als Aktionen ein Mannschaftsspiel auf taktischer Ebene ermöglichen. Da nur Moves definiert werden, die sinnvoll im Fußball angewendet werden können, reduzieren sich die möglichen Aktionen der Agenten auf taktischer Ebene beträchtlich. Taktik bezieht sich in diesem Zusammenhang auf die Anwendung von Spielzügen im Fußball. Sie darf hier nicht mit einer Sequenz von Aktionen [siehe Kapitel 2.1] verwechselt werden. Abbildung 1 stellt die modulare Architektur eines Brainstormers-Agenten dar.

[3] Die Implementierung der Sensorverarbeitung haben die Brainstormers direkt vom Carnegie Mellon University-Team übernommen [RMM$^+$01, S. 6].
[4] In der Simulationsliga verfügen die Agenten über eine bestimmte Ausdauer [Ro02, S. 36].



Quelle: vgl. RMH+03, S. 9

Abb. 1: Architektur der Brainstormers-Agenten aus dem Jahr 2002

3 Anwendung des Reinforcement Learnings

Nachdem in Kapitel 2 die verwendete Methodik der Brainstormers erläutert wurde, wird nun die Anwendung des RLs beschrieben. Zunächst wird in Abschnitt 3.1 der Lernvorgang für die Fähigkeiten der einzelnen Agenten dargestellt. Anschließend werden in Abschnitt 3.2 zwei Ansätze zur Realisierung eines taktischen Mannschaftsspiels aufgezeigt.

3.1 Lernen von Fähigkeiten

Ein Move ist eine Taktik, d.h. eine Sequenz von Aktionen bzw. Basiskommandos (z. B. *kick*, *dash*, *turn*) an den Soccer Server, die einen Anfangszustand in mehreren Zyklen in einen Endzustand überführt. Für jeden Move sind die Anfangszustände, in denen er eingesetzt werden kann, und das Ziel, das er erreichen soll, definiert. Ein Move endet, wenn ein möglicher Endzustand erreicht wurde. Die Endzustände können positiv/erwünscht oder negativ/nicht erwünscht sein. Beispielsweise endet der Move *Intercept ball* positiv, wenn der Agent den Ball in seiner Schussreichweite hat, oder negativ, wenn der Agent keine Möglichkeit mehr hat, den Ball zu erreichen. Die meisten Teams, die ein Move-bezogenes Konzept anwenden, implementieren die nötigen Basiskommandos und ihre Parameter manuell fest [RM02a, S. 10]. Dieses Vorgehen ist sehr aufwändig und unflexibel.

Die Brainstormers formulieren das Lernen eines Moves hingegen als dynamisches Optimierungsproblem. Die Aufgabe ist das automatisierte Lernen einer Taktik, die mit minimalen kumulierten Kosten aus möglichst vielen Anfangszuständen das Ziel eines Moves erfüllt. Um die Komplexität zu begrenzen, werden nur die Zustände betrachtet, die während eines Moves auftreten. Jede Aktion, die ein Agent ausführt, verursacht

Kosten $r(s,a) = c > 0$. Die Kosten für die Aktionen sind konstant und hängen daher insgesamt linear von der Anzahl der benötigten Aktionen ab. „Kürzere“ Taktiken haben entsprechend geringere Kosten und werden bevorzugt. Falls ein negativer Endzustand eintritt, werden maximale Kosten angesetzt. Es handelt sich folglich um eine Art „kürzester Pfad“-Problem zwischen einem Anfangszustand und einem gewünschten Endzustand, wobei „Kürze“ hier niedrigen Kosten entspricht. Aufgrund des prinzipiell stetigen Zustandsraums kann die Kostenfunktion $J^{\pi}(s)$ aber nicht berechnet werden. Stattdessen wird sie durch künstliche neuronale Netze (KNN) angenähert. Die Input-Schicht empfängt den aktuellen Zustand, die Output-Schicht repräsentiert die erwarteten, kumulierten, zukünftigen Kosten in einem Zustand [siehe Kapitel 2.1]. Eine ausführliche Darstellung der verwendeten KNN findet sich in [Me99, Kap. 7]. Ein zusätzlicher Vorteil der KNN ist ihre Generalisierbarkeit, d. h. sie liefern auch für nicht gelernte Zustände einen sinnvollen Output [Ri99]. Der Lernvorgang einer optimalen Taktik für einen Move verläuft wie folgt [RM02a, S. 11]:

1. Start in einem zufälligen, für den Move gültigen, Anfangszustand.
2. Auswahl einer Aktion durch den Agenten. Dazu wird entweder die nach bisherigem Wissen optimale Aktion ausgewählt oder zufällig eine neue Aktion ausprobiert[5].
3. Beobachtung des Nachfolgezustands der Umwelt durch den Agenten.
4. Aktualisierung der Kostenfunktion für den Vorgängerzustand. Die neue Schätzung der Kostenfunktion im Vorgängerzustand besteht aus der Summe der erwarteten, kumulierten, zukünftigen Kosten im Nachfolgezustand und den Kosten der durchgeführten Aktion. Dadurch entsteht eine kontinuierlich besser werdende Schätzung der erwarteten, kumulierten, zukünftigen Kosten in allen durchlaufenen Zuständen. Die neue Kostenfunktion kann einem KNN aber nicht direkt zugewiesen werden. Stattdessen werden die Gewichte im KNN angepasst, um den Fehler zwischen alter und neuer Schätzung der Kostenfunktion zu minimieren [Ri99, S. 4].

Beispielhaft wird im Folgenden das Lernen des *Kick*-Moves kurz beschrieben [RM02a, S. 10–12]. Verwendet wird ein KNN mit vier Input-Neuronen für die Position und Geschwindigkeit des Balls, einem Output-Neuron für die erwarteten, kumulierten, zukünftigen Kosten und 20 verborgenen Neuronen für das nicht lineare Mapping zwischen den Input- und Output-Schichten. Nach etwa 45 Lernzyklen mit 2000 verschiedenen Anfangszuständen pro Lernzyklus stabilisiert sich das KNN bei einer Leistung von ca. 95 %. Das heißt, in 95 % aller gültigen Anfangszustände ist der Agent in der Lage, den Ball anzunehmen und mit einer bestimmten Geschwindigkeit in die gewünschte Richtung zu schießen. Die Fehlschüsse in den verbliebenen Zuständen führen die Brainstormers auf die stochastische Umwelt der Simulationsliga zurück

[5] Vorschläge zu Verhältnissen zwischen *Exploration* und *Exploitation*: siehe [Me99, S. 12 und 41].

[RM02a, S. 11]. Eine einzelne Taktik bzw. ein einzelnes KNN ist jedoch nicht ausreichend, um einen Move in allen Zuständen zielgerecht auszuführen [RBM+00, S. 3]. Daher wird die Implementierung einer Move-Routine auf mehrere Submoves verteilt, die jeweils über ein eigenes KNN verfügen. Die Routine für den Kick-Move besteht aus drei KNN für drei verschiedene Geschwindigkeiten. Will der Agent in einem Spiel einen Move ausführen, kann er über das entsprechende KNN die Kosten, die aus den möglichen Aktionen resultieren, bestimmen und die optimale Sequenz von Aktionen auswählen.

Insgesamt wurden die Moves: *intercept ball, go in one of eight directions, dribble in one of eight directions, shoot to goal (kick)* (drei Geschwindigkeiten), *pass ball to teammate* (10 Möglichkeiten) und *wait at position* erlernt [RMM+01]. Ein einzelner Move wird in den meisten Fällen aber nicht direkt zu einem Tor führen. Der folgende Abschnitt befasst sich daher mit der Aufgabe, RL auch für die Koordination mehrerer Agenten und ihrer Moves anzuwenden.

3.2 Lernen von taktischen Entscheidungen

Eine Mannschaft ist nur erfolgreich, wenn alle Teammitglieder miteinander kooperieren. Das gemeinsame Ziel ist der Gewinn des Spiels. Dieses Ziel wird in zwei Subziele unterteilt. Einerseits sollen eigene Tore geschossen werden, andererseits sollen gegnerische Tore verhindert werden. Im Robotfußball muss dafür die Entscheidungsfindung von elf Agenten koordiniert werden. Dieses Vorhaben ist wesentlich komplexer als das Lernen einzelner Fähigkeiten. Um die Komplexität zu reduzieren, stehen den Agenten auf taktischer Ebene nur die gelernten Moves als Aktionen zur Verfügung und keine Basiskommandos.

Das verwendete Beschreibungsmodell der MDP lässt sich auf diesen Multi-Agenten-Fall erweitern. Ein MMDP (Multi-agent Markov Decision Process) kann dann als Tupel $M_n := [S, A, r, p]$ definiert werden. Im Gegensatz zum einfachen MDP gibt es hierbei für jeden Agenten ein Set möglicher Aktionen und ein eigenes Reinforcement-Signal. Das Reinforcement-Signal eines Agenten kann folglich von den Aktionen der anderen Agenten abhängen. Auch im Multi-Agenten-Fall werden optimale Taktiken angestrebt. Für den allgemeinen Fall unkorrelierter Reinforcement-Signale aller Agenten kann kein Optimalitätsmaß definiert werden [RM02a, S. 7]. Es werden daher zwei Spezialfälle bezüglich der Reinforcement-Signale betrachtet. Der erste Spezialfall stellt kooperierendes Verhalten durch ein gemeinsames Reinforcement-Signal dar. Dagegen wird im zweiten Spezialfall ein opponierendes „Nullsummen"-Verhalten zweier Agenten durch gegensätzliche Reinforcement-Signale erreicht. Der Erfolg des einen Agenten hat den entsprechenden Misserfolg des anderen Agenten zur Folge. Die Existenz optimaler Taktiken kann für beide Spezialfälle nachgewiesen werden. Für die Herleitung und zu

den zugehörigen Kostenfunktionen siehe [RM02b]. Auf den Robotfußball übertragen, lässt sich ein Fußballspiel als Nullsummen-Spiel zwischen zwei Teams untereinander kooperierender Agenten beschreiben. Die Berechnung optimaler Taktiken erweist sich jedoch als problematisch, da lokal optimales Verhalten der einzelnen Agenten nicht global optimal sein muss[6].

Bislang werden die taktischen Entscheidungen bei den Brainstormers noch fest mit Regeln und verschiedenen Prioritäten der Moves implementiert. Nähere Informationen finden sich in [RBM+00, S. 3]. Im Folgenden werden zwei Ansätze beschrieben, mit denen die Brainstormers derzeit versuchen, das Lernen von taktischen Entscheidungen zu realisieren. Prinzipiell wird ähnlich vorgegangen wie beim Lernen von Fähigkeiten. Wie bereits ausgeführt, sind die einzelnen Aktionen hier aber Moves, um die Komplexität zu begrenzen. Ein positiver Endzustand ist mit einem erzielten Tor erreicht. Ein negativer Endzustand tritt ein, wenn der Gegner den Ball erlangt[7]. Ein negativer Endzustand wird mit hohen Strafkosten belegt. Um eine Kooperation zwischen den Agenten zu erreichen, bekommt jeder Agent in jedem Zyklus die gleichen konstanten Kosten zugewiesen. Kostenoptimale Taktiken beinhalten folglich wahrscheinlich die Kooperation mehrerer Agenten. Schließlich ist davon auszugehen, dass ein Solospiel eines Agenten in den meisten Fällen – wenn überhaupt – nur mit höheren Gesamtkosten zum Erfolg führt.

Joint Action Learners

Der erste Ansatz geht von Agenten als Joint Action Learners aus. Diese Agenten haben jederzeit Kenntnis über die Aktionen aller Teammitglieder. Daher kann jeder Agent die erfolgten Zustandsübergänge beobachten, und daraus bestimmen, nach welcher Übergangswahrscheinlichkeit sie stattgefunden haben. Im Unterschied zum einfachen MDP hat ein Agent dabei nur Einfluss auf seinen Teil der Gesamtaktionen. Da alle Agenten außerdem dieselben Reinforcement-Signale erhalten, kann jeder Agent einzeln als einfacher MDP modelliert werden. Durch Beobachtung ausreichend vieler Zustandsübergänge können wie im beim Lernen der Moves die erwarteten, kumulierten, zukünftigen Kosten für die Anwendung einer Aktion angenähert werden. Hieraus kann wiederum eine optimale Gesamttaktik abgeleitet werden. Die einzelnen Taktiken für die Agenten ergeben sich dann durch entsprechende Projektionen von der optimalen Gesamttaktik [RM02a, S. 14].

Diese Vorgehensweise ist mit zwei Problemen verbunden. Zum einen wächst der Aktionsraum exponentiell mit der Zahl der beteiligten Agenten. Die Lernvorgänge der KNN konvergieren dann nicht mehr, da es zu viele mögliche Aktionen gibt. Zum

[6] Für ein Beispiel diesbezüglich: siehe [RM02b, S. 3].
[7] Andere Fälle wie „Ball im Aus“ oder Foules etc. werden von den Brainstormers noch nicht betrachtet.

anderen ist dieses Verfahren an eine fixe Zahl an Agenten gebunden. Ein spontan zusätzlich in die Abwehr eingreifender Agent ist dann beispielsweise nicht möglich.

In einem ersten Experiment der Brainstormers spielten zwei lernende Stürmer gegen einen bzw. zwei manuell, fest programmierte Verteidiger. Das Ziel der Stürmer war, möglichst schnell ein Tor zu schießen. Für jeden Move wurde ein eigenes KNN angelernt, das die erwarteten, kumulierten, zukünftigen Kosten bei Anwendung des jeweiligen Moves repräsentiert [RMM+01, S. 5]. Als Input dienen die Positionen der Spieler und des Balls und die Geschwindigkeit des Balls. Das Experiment ergab gegenüber einer manuellen Implementierung der Stürmer eine wesentlich höhere Erfolgsrate durch das RL[8]. Während des Experiments kam es zu kooperativen taktischen Spielzügen der Stürmer, unter anderem auch zu Doppelpässen. Aufgrund der exponentiell anwachsenden Aktionsräume ist es den Brainstormers aber nicht gelungen, dieses Lernverfahren erfolgreich in Szenarien mit weiteren Agenten einzusetzen.

Independent Learners

Bei diesem Ansatz wird ein schwächerer Agententyp angenommen. Die so genannten Independent Learner kennen nur ihre eigenen Aktionen und nehmen die anderen Agenten nicht wahr. Der Einfluss, den die Aktionen der anderen Agenten ausüben, kann nicht von stochastischen Einflüssen der Umwelt unterschieden werden. Bei Independent Learnern gibt es keine theoretische Garantie mehr, dass eine optimale Taktik durch Lernen gefunden werden kann [RMH+03, S. 3].

Der Lernalgorithmus läuft wie folgt ab [RMH+03, S. 3f.]:

1. Es gibt eine zu Beginn leere Sammlung von Beispielen. Ein Beispiel besteht aus einem Zustand und den zugehörigen erwarteten, kumulierten, zukünftigen Kosten.
2. Eine zufällige Anfangstaktik bestimmt dann die Aktionen der Agenten. Die gewählten Aktionen werden gespeichert. Dies wird solange wiederholt, bis ca. zehn erfolgreiche Sequenzen gespeichert wurden. Alle nicht erfolgreichen Aktionssequenzen werden dann gelöscht, bis auf die finalen Zustände, die zu einem negativen Ausgang führten.
3. Die verbliebenen Zustände werden um Duplikate gefiltert und in die Beispiel-Sammlung eingefügt.
4. Mit den Beispielen wird ein KNN trainiert. Dies erfolgt analog zum Lernen der Moves [siehe Kapitel 3.1]. Als Input dienen die Positionen der Spieler und des Balls und die Geschwindigkeit des Balls.

[8] 85 % Erfolg mit RL zu 35 % bei einem Verteidiger, bzw. 55 % zu 10 % bei zwei Verteidigern.

Das KNN im vierten Schritt wird mittels des RPROP-Algorithmus angelernt. Hierbei handelt es sich um eine spezielle Variante des Backpropagation-Algorithmus, bei dem keine günstige Lernrate gesucht werden muss. Nähere Informationen zu diesem Algorithmus finden sich in [RB93]. Dieser Lernalgorithmus durchläuft 5.000 Iterationen der Schritte zwei bis vier. Die alten Beispiele zu behalten, verhindert das Vergessen der bereits durchlaufenen erfolgreichen Sequenzen [RMH+03, S. 4].

Im Speziellen wurde der zuvor beschriebene Lernalgorithmus für die Positionierung von Stürmern, die nicht im Ballbesitz oder in Ballnähe sind, beim Spiel gegen Verteidiger und Torwart angewandt. Das weitere Verhalten der Stürmer wurde in einem hybriden Ansatz aus der in bisherigen Wettbewerben bewährten regelbasierten Implementierung übernommen. Die Verteidiger waren ebenfalls regelbasiert. Der Lernalgorithmus führte zum einen zu einer signifikanten Verbesserung des Sturms[9]. Zum anderen zeigten weitere Versuche, dass das gelernte Verhalten generalisierbar war. Dementsprechend konnte es auch auf andere Anfangszustände und andere Teams als Gegner angewendet werden.

4 Fazit und Ausblick

Abschließend ist zu sagen, dass die Anwendung von RL-Methoden im Robotfußball sinnvoll und Erfolg versprechend ist. RL bringt deutlich bessere Resultate als eine manuelle feste Implementierung, die dazu noch sehr aufwändig ist. Überdies ist die Anwendung von RL flexibler und in der Lage, ein optimales Verhalten zu generieren. Während das Lernen der Fähigkeiten für einzelne Agenten bereits erfolgreich angewendet wird, existieren aufgrund der Komplexität des Multi-Agenten-Falls für taktische Entscheidungen bislang nur Ansätze.

Als Zwischenschritt zum vollständig lernenden Agenten dient die hybride Verbindung von maschinellem Lernen und statischen Komponenten. Als nächstes Ziel gilt es, die Kommunikation in die Kooperation der Agenten zu integrieren. Agenten könnten dadurch nicht nur ihre Aktionen, sondern auch ihre weiteren Absichten kommunizieren.

Bis zu einem Agent, der sein Verhalten für die Entscheidungsfindung vollständig durch RL-Methoden erlernt, ist es aber noch ein weiter Weg.

[9] Bei sieben Stürmern gegen sieben Verteidiger und einen Torwart stieg die Erfolgsquote von 18 % auf 29 % [RE02a, S. 15f.].

Literaturverzeichnis

[BT96] Bertsekas, Dimitri P.; Tsitsiklis, John: *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[He98] Herrmann, J. Michael: *Neuronale Netze* (Vorlesungsskript). http://www.gwdg.de/~mherrma/script/v.ps.gz [Abrufdatum: 05.05.2003]

[Me99] Merke, Arthur: *Reinforcement Lernen in Multiagentensystemen*. Diplomarbeit 1999, Universität Karlsruhe (TH)

[RB93] Riedmiller, Martin; Braun, Heinrich: *A direct adaptive method for faster backpropagation learning: the RPROP algorithm*. In: Proceedings of the IEEE International Conference on Neural Networks, San Francisco, 1993, S. 586–591. http://lrb.cs.uni-dortmund.de/~riedmill/publications/riedml.icnn93.ps.Z [Abrufdatum: 20.05.2003]

[RBM+00] Riedmiller, Martin; Buck, Sebastian; Merke, Artur; Ehrmann, Ralf; Thate, Ortwin; Dilger, Sergio; Sinner, Alex; Hoffmann, Andreas; Frommberger Lutz: *Karlsruhe Brainstormers - Design Principles*. In: RoboCup 1999: Robot Soccer World Cup III. Hrsg.: Veloso, M.; Pagello, E.; Kitano, H. LNAI 1856, Springer, 2000, S. 588–591.

[Ri99] Riedmiller, Martin: *Concepts and Facilities of a neural reinforcement learning control architecture for technical process control*. In: Neural Computation and Application Journal, (1999) 8. S. 323–338.

[RM02a] Riedmiller, Martin; Merke, Artur: *Using Machine Learning Techniques in Complex Multi-Agent Domains*. In: Perspectives on Adaptivity and Learning. Hrsg.: Stamatescu, I.; Menzel, W.; Richter, M.; Ratsch, U. LNCS, Springer, 2002.

[RM02b] Riedmiller, Martin; Merke, Artur: *Karlsruhe Brainstormers – a Reinforcement Learning approach to robotic soccer II*. In: RoboCup 2001: Robot Soccer World Cup V. Hrsg.: Birk, A.; Coradeschi, S.; Tadokoro, S. LNAI 2377, Springer, 2002, S. 435-440.

[RMH+03] Riedmiller, Martin; Merke, Artur; Hoffmann, Andreas; Withopf, Daniel; Nickschas, Manuel; Zacharias, Franziska: *Brainstormers 2002 - Team Description*. In: RoboCup 2002: Robot Soccer World Cup VI, LNCS,

Springer, (noch nicht erschienen). http://lrb.cs.uni-dortmund.de/~riedmill/publications/riedml.merke.robocup02.ps.gz [Abrufdatum: 30.04.2003]

[RMM+01] Riedmiller, Martin; Merke, Artur; Meier, David; Hoffmann, Andreas; Sinner, Alex; Thate Ortwin; Ehrmann Ralf: *Karlsruhe Brainstormers – a Reinforcement Learning approach to robotic soccer*. In: RoboCup 2000: Robot Soccer. World Cup IV. Hrsg.: Stone, P.; Balch, T.; Kraetzschmar G. LNAI 2019, Springer, 2001, S. 367-372.

[Ro02] Robocup.org: *Users Manual RoboCup Soccer Server*, 2002. http://sserver.sourceforge.net/docs/manual.pdf [Abrufdatum: 20.05.2003]

[SB98] Sutton, Richard S.; Barto, Andrew G.: *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*, MIT Press, 1998. http://www-anw.cs.umass.edu/~rich/book/the-book.html [Abrufdatum: 20.05.2003]

[Wo99] Wooldridge, Michael. *Intelligent agents*. In: Multi Agent Systems: A Modern Approach to Distributed Artificial Intelligence. Hrsg.: Weiss, Gerhard. MIT Press, 1999, S. 27–78.