Seminar „Robotfußball“ im Sommersemester 2003
Dozent: Dr. Dietmar Lammers
Institut für Informatik
Westfälische Wilhelms-Universität Münster

Selbstorganisation und Lernen

Vorgelegt von: Carsten Keßler
Friesenring 2
48147 Münster
(0251) 2007644
kessler@ifgi.uni-muenster.de

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Ein Hauptaugenmerk bei autonomen Agenten, wie sie z.B. in einer Robotfußball-Mannschaft eingesetzt werden können, ist die Fähigkeit, zu lernen und sich an neue Situationen in der Umgebung anzupassen. Dabei läuft der Lernprozess häufig so ab, dass man von einer indirekten Programmierung sprechen könnte, d.h. dem Agenten wird nach jeder Handlung mitgeteilt, ob er seine Aufgabe zufriedenstellend erledigt hat oder nicht[1]. In diesem Szenario kann man jedoch nicht von wirklicher Autonomie sprechen, da dem Agenten ständig von außen durch einen Trainer o.ä. mitgeteilt werden muss, wie erfolgreich er war, um lernen zu können und sein Verhalten entsprechend anpassen zu können. Außerdem braucht der Agent im Initialzustand einen Anstoß von außen, um überhaupt in Aktion zu treten und mit der Bearbeitung einer Aufgabe anzufangen. Eine weitergehende Forderung wäre daher, dass ein wirklich autonomer Agent aus seinem Startzustand heraus ohne fremdes Zutun beginnt, seine Umgebung zu erkunden und zu lernen.

1.2 Die Vision

Ausgehend von den in der Motivation genannten Forderungen an einen wirklich autonomen Agenten wird sich die Ausarbeitung im Folgenden damit beschäftigen, wie ein solcher Agent in Form eines Roboters realisiert werden kann.
Der Roboter soll durch ein neuronales Netz gesteuert sein, dass sich nach dem Einschalten des Roboters in einem Tabula rasa Zustand befindet, d.h. es sind im Netz keine Gewichte gesetzt[2]. Daraus folgt, dass der Roboter am Anfang auch nicht auf die von seinen Sensoren gemessen Werte reagieren kann. Die Aktivitäten, falls in diesem Initialzustand vorhanden, wären also rein stochastisch. Des Weiteren soll sich dieser Roboter in einer Umgebung zurechtfinden, die sowohl statische als auch dynamische Objekte enthält.
Daraus ergibt sich die Aufgabe, ein internes Ziel für den Roboter zu definieren, das ihn dazu bringt, sich aus dem Initialzustand heraus zu bewegen und während der Bewegung ein eigenes Bild von seiner Umwelt zu entwickeln. Dieses Ziel sollte unabhängig von den Sensoren und der Fortbewegungsart des Roboters sein, so dass es auch zur Steuerung anderen Roboter eingesetzt werden kann.

2 Künstliche neuronale Netze

In diesem Kapitel soll eine kurze Einführung in die grundlegende Funktionsweise von künstlichen neuronalen Netzen gegeben werden, soweit dies zum Verständnis der folgenden Kapitel notwendig ist.

2.1 Das biologische Vorbild

Künstliche neuronale Netze lehnen sich eng an das biologische Vorbild der Nervenzelle an (vgl. Abb. 1). Eine solche natürliche Nervenzelle besteht im Groben aus dem Zellkörper, der den Zellkern enthält, den Dendriten sowie einem Axon. Die Dendriten leiten dabei

[1] „Reinforcement Learning", siehe Kapitel 2.4
[2] Siehe Kapitel 2

Reize in Form von elektrischen Impulsen von anderen Nervenzellen an die Nervenzelle weiter. Diese Reize können anregend (exhibitorisch) oder hemmend (inhibitorisch) wirken. Wird in der Summe eine bestimmt Spannung (der Schwellwert) überschritten, „feuert" die Nervenzelle, d.h. sie gibt ihrerseits über das Axon einen elektrischen Impuls ab, der dann weitere Nervenzellen über deren Dendriten erreicht.
Diese Darstellung der Funktionsweise ist stark vereinfacht und wird einen Biologen kaum zufrieden stellen. Sie reicht jedoch aus, um aufzuzeigen, auf welche Art und Weise sich die künstlichen neuronalen Netze an ihrem biologischen Vorbild orientieren.

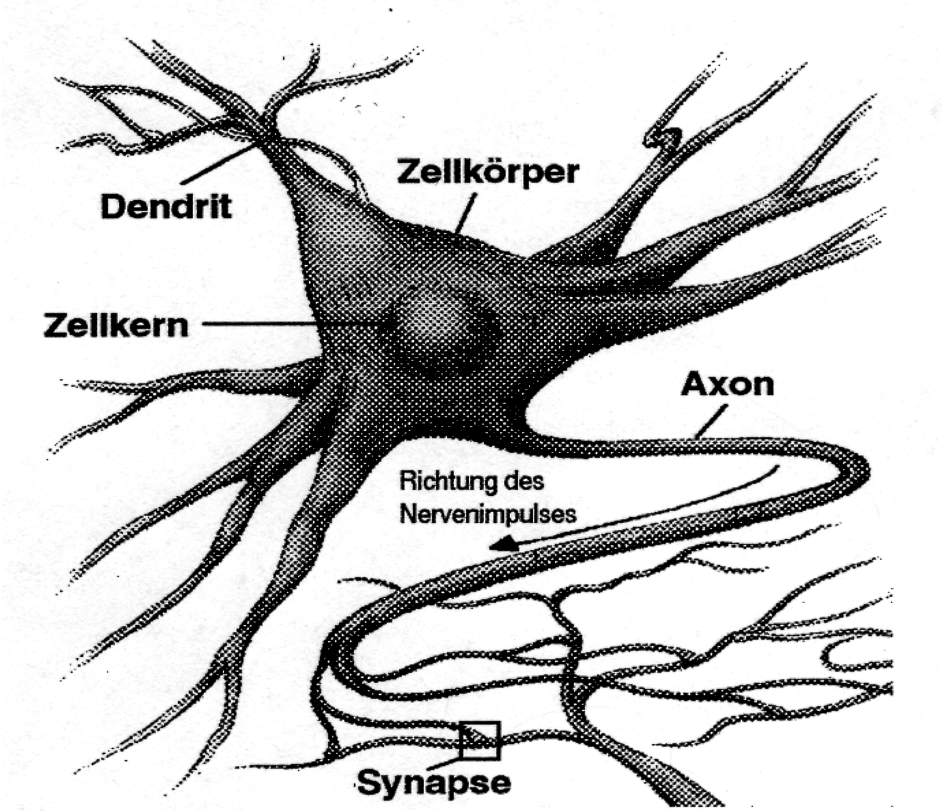

Abb. 1: Nervenzelle. Quelle: Lippe, Skript „Softcomputing"

2.2 Künstliche Neuronen

Künstliche Neuronen sind in der Funktionsweise ihrem biologischen Vorbild sehr ähnlich (Vgl. Abb. 2). Sie erhalten einen Vektor von Eingaben, der in diesem Fall meist aus Zahlen oder anderen zu verarbeitenden Werten besteht (denkbar wären z.B. auch boolesche Werte). Auf diesen Vektor wird die Aktivierungsfunktion angewendet, in der Abbildung im linken Kasten zu sehen. Diese Aktivierungsfunktion[3] berechnet, wann das künstliche Neuron feuert. Ist der Schwellwert für die Aktivierungsfunktion überschritten worden, wird die Ausgabefunktion f(net) auf die Ausgabe der Aktivierungsfunktion angewandt. Das Ergebnis dieser Berechnung wird, parallel zum biologischen Vorbild, ausgegeben. In der Abbildung sind mehrere Ausgaben zu sehen – dies ist nicht ganz korrekt, denn es wird nur eine Ausgabe berechnet, die an beliebig viele folgende Neuronen weitergegeben werden kann (wie bei der natürlichen Nervenzelle auch).
Formal gesehen ist ein künstliches Neuron also ein Tupel, bestehend aus einem Eingabevektor, einem Gewichtsvektor, einer Aktivierungsfunktion und einer Ausgabefunktion.

[3] In der Abbildung ist die Aktivierungsfunktion als gewichtete Summe realisiert. Die gewichtete Summe wird in sehr vielen Fällen zu diesem Zweck benutzt.

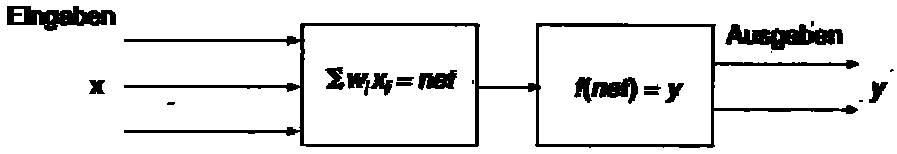

Abb. 2: Schematische Darstellung eines künstlichen Neurons.
Quelle: Dan Patterson – Künstliche Neuronale Netze

2.3 Vernetzung künstlicher Neuronen

Künstliche neuronale Netze entstehen durch die Vernetzung der im letzten Abschnitt vorgestellten Neuronen. Dabei entsteht ein schichtartiger Aufbau (vgl. Abb. 3). Die erste Schicht bezeichnet man als Eingabeschicht, da sie die Eingabe in das Netz aufnimmt. Nach der Eingabeschicht können mehrere verborgene Schichten folgen, die die Eingabe weiter verarbeiten und schließlich an die Ausgabeschicht weiterleiten, die das Ergebnis der Verarbeitung zurückgibt. Das in Abb. 3 gezeigte Netz ist nicht vollständig verbunden, d.h. es ist nicht jeder Knoten einer Schicht mit allen Knoten der darauf folgenden Schicht verbunden. Außerdem findet man in dem gezeigten Netz weder sog. Shortcuts (Verbindungen, die eine oder mehrere Schichten überspringen) noch Rückkopplungen (Verbindungen, die die Ausgabe eines Knotens an einen Knoten in einer vorhergehenden Schicht weitergeben). Solche Konstruktionen sind beim Aufbau von künstlichen neuronalen Netzen jedoch möglich und werden je nach Einsatzgebiet des Netzes auch eingesetzt.

2.4 Lernmethoden

Ein künstliches neuronales Netz kann auf verschieden Arten lernen. Dabei kann auf alle Bestandteile des Netzes Einfluss genommen werden. Es ist möglich, neue Verbindungen zwischen den Zellen anzulegen oder existierende Verbindungen zu löschen. Ebenso können die Zellen selbst gelöscht oder hinzugefügt werden. Innerhalb der Zellen sind Modifikationen der Gewichte, des Schwellwertes oder der Aktivierungs- bzw. Ausgabefunktion durchführbar.
Um dem Netz mitzuteilen, ob und wie es sich anpassen muss, haben sich drei wichtige Lernstrategien herauskristallisiert. Erstens das sog. überwachte Lernen (*supervised learning*), bei dem der Trainer dem Netz nach jedem Durchlauf mitteilt, wie groß der Fehler in der Ausgabe war. Zweitens das bestärkende Lernen (*reinforcement learning*), bei dem der Trainer dem Netz lediglich mitteilt, ob die Ausgabe korrekt war oder nicht. Es erfolgt also keine Angabe darüber, wie groß der Fehler war. Und drittens das unüberwachte Lernen (*unsupervised* oder *self-organized learning*), bei dem das Netz ohne Beeinflussung von außen versucht, die Eingabedaten in Ähnlichkeitsklassen aufzuteilen.

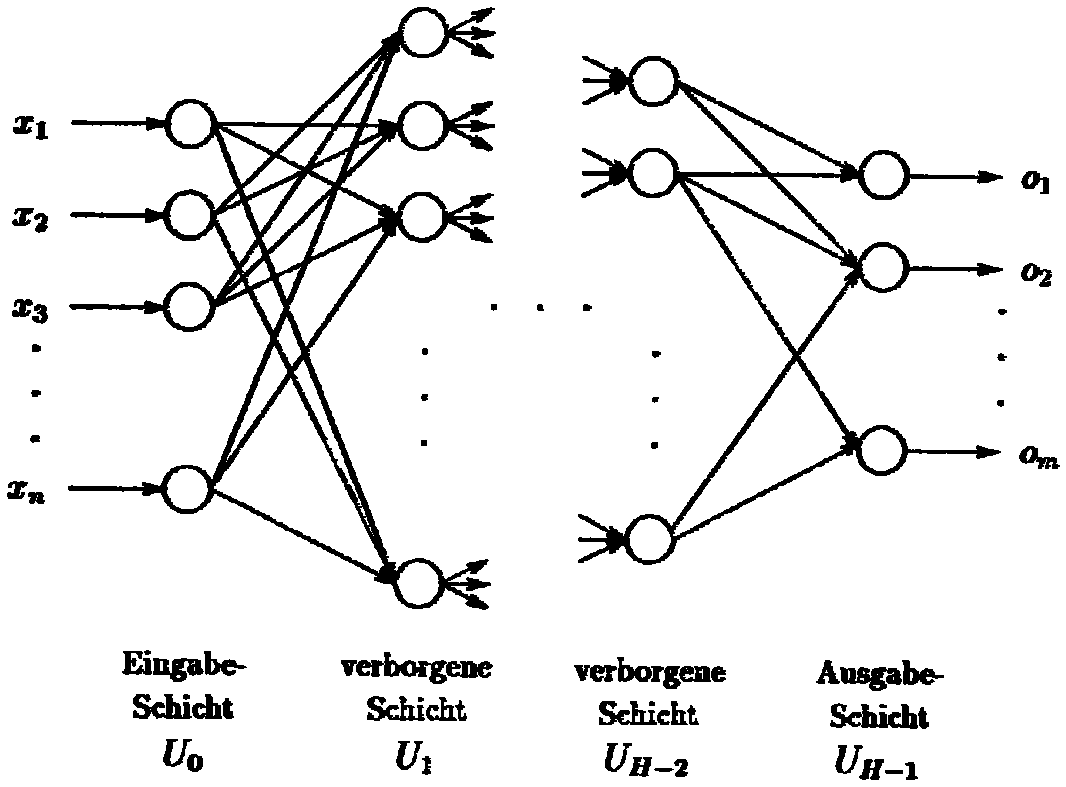

Abb. 3: Neuronales Netz mit Eingabeschicht, Ausgabeschicht und U-2 verborgenen Schicht.
Quelle: Lippe, Skript „Softcomputing"

3 Das Lernmodell

In diesem Kapitel soll erläutert werden, mit welchem Lernmodell das neuronale Netz trainiert werden soll, damit der Roboter autonom, d.h. ohne externen „Lehrer", lernen kann.

3.1 Der Ansatz

Damit der Roboter wirklich autonom lernen kann, ist der Einsatz eines Trainers, wie es die Methoden des überwachten oder bestärkenden Lernens erfordern, ausgeschlossen. Daher wird im Folgenden die Idee entwickelt, dass der Agent selbst seinen eigenen Lernprozess überwacht. Dabei entsteht das Problem, dass der Agent ein Lernmodell braucht, das ihm erlaubt, aufgrund der gemessenen Sensorwerte die im nächsten Zeitschritt zu erwartenden Sensormesswerte vorauszuberechnen. Auf diese Art und Weise kann der Roboter zu jedem Zeitschritt die aktuell gemessenen Sensorwerte mit den im vorherigen Zeitschritt vorausberechneten Werten vergleichen und sein neuronales Netz entsprechend anpassen.

3.2 Controller und Modell

Im Folgenden wird der Controller dazu verwendet, die Lenkung des Roboters zu steuern. Es wird also davon ausgegangen, dass die Vorwärtsbewegung des Roboters nicht

gesteuert werden muss und in diesem Schritt zunächst einmal als Bewegung mit fester Geschwindigkeit angenommen wird[4]. Die Ausgabe des Controllers kann durch die Gleichung

$$y_t = K(x_t; c)$$

beschrieben werden. Dabei ist K die Controller-Funktion, x_t der Vektor der Sensorwerte zum Zeitpunkt t und c der Parameter Vektor. Wie im vorherigen Absatz beschrieben benötigen wir jedoch ein Modell, dass die Sensormesswerte zum Zeitpunkt t+1 vorausberechnet. Dieses Modell wird durch die Gleichung

$$x^P_{t+1} = x_t + M(x_t, y_t; m)$$

dargestellt. x^p_{t+1} steht hier für den zum Zeitpunkt t prognostizierten Vektor der Sensorwerte für den Zeitpunkt t+1. m ist der Parametervektor für das Modell. Da die Vorausberechnung der Sensorwerte für t+1 nie völlig mit den tatsächlich zum Zeitpunkt t+1 gemessenen Werten übereinstimmen kann, wird der Fehler mit Hilfe der Funktion

$$E = \left\| x_{t+1} - x^P_{t+1} \right\|^2$$

bestimmt. Diese Berechnung des Fehlers wird nun genutzt, um die Lernregeln aufzustellen. Die Lernregeln basieren auf der Berechnung des steilsten Gradienten, auf die hier nicht weiter eingegangen werden soll. Wichtig ist, dass die Lernregeln die Parametervektoren für das Modell und den Controller so anpassen, dass der steilste Gradient in der Funktion abgeflacht wird, d.h. der Fehler wird minimiert. Wichtig ist auch, dass nach jedem Durchlauf sowohl das Modell als auch der Controller angepasst werden. Die geschieht mit Hilfe der Funktionen

$$\Delta m = -\eta \frac{\partial}{\partial m} E \quad \text{und} \quad \Delta c = -\varepsilon \frac{\partial}{\partial c} E ,$$

die Modell und Controller entsprechend der Lernraten η und ε nach jedem Durchlauf anpassen.

3.3 Der Khepera Roboter

Um das vorgestellte Modell zu testen, wurde der Khepera-Roboter (vgl. Abb. 4) ausgewählt. Bei diesem Roboter handelt es sich um einen preisgünstigen Desktop-Roboter, der in der Grundausstattung über 8 Infrarotsensoren und 2 Motoren verfügt, mit denen er eine Geschwindigkeit von maximal 1 m/s erreicht. Das Grundmodell ist über eine Schnittstelle auf der Oberseite erweiterbar, z.B. mit einer Videokamera oder einem Greifarm-System.

[4] Der Steuerung der Geschwindigkeit widmet sich Kapitel 4.

Abb. 4: Der Khepera-Roboter. Quelle: http://www.k-team.com

3.4 Das Experiment

Wie bereits erwähnt wird für das erste Experiment angenommen, dass der Roboter sich mit konstanter Geschwindigkeit vorwärts bewegt und der Output des Controllers lediglich die Drehrichtung und –Geschwindigkeit angibt. Die Modellierung des Controllers erfolgt dabei über ein einzelnes Neuron, das die gemessenen Sensorwerte mit der Funktion

$$y = \tanh\left(\sum_{i=1}^{n} c_i x_i\right)$$

verarbeitet[5]. Es wird also der Tangens Hyperbolicus auf die gewichtete Summe der Sensorwerte angewandt.
Mit dieser Konfiguration wurde nun der Khepera Roboter verschiedenen Experimenten unterzogen, um zu untersuchen, wie das eingesetzte neuronale Netz[6] sich in bestimmten Situationen entwickelt. Zunächst wurde der Roboter an eine Wand gestellt, so dass er die Wand links von sich sieht.
Abbildung 5 zeigt die Verläufe der Controller Parameter c_1 bis c_6. Es ist leicht zu erkennen, dass die Parameter nach einer kurzen Einschwingphase alle jeweils recht konstante Werte annehmen. Zunächst ist der Roboter noch schräg auf die Wand zugefahren, um dann immer näher an die Parallele zur Wand zu kommen und schließlich tatsächlich parallel zu fahren. Dieses Verhalten ist darauf zurückzuführen, dass die Controller Parameter so angepasst werden, dass der Fehler der Vorausberechnung minimiert wird. Dieser Fehler ist genau dann minimal, wenn der Roboter parallel zur Wand fährt, weil er dann genau die Sensormesswerte vorausberechnen kann, die im nächsten Zeitschritt auch gemessen werden.

[5] Dieses Neuron hat lediglich eine Ausgabefunktion und keine Aktivierungsfunktion, da in jedem Falle eine Ausgabe zur Steuerung der Drehbewegung benötigt wird.
[6] Eigentlich handelt es sich nicht wirklich um ein Netz, da ein einzelnes Neuron zur Steuerung eingesetzt wird.

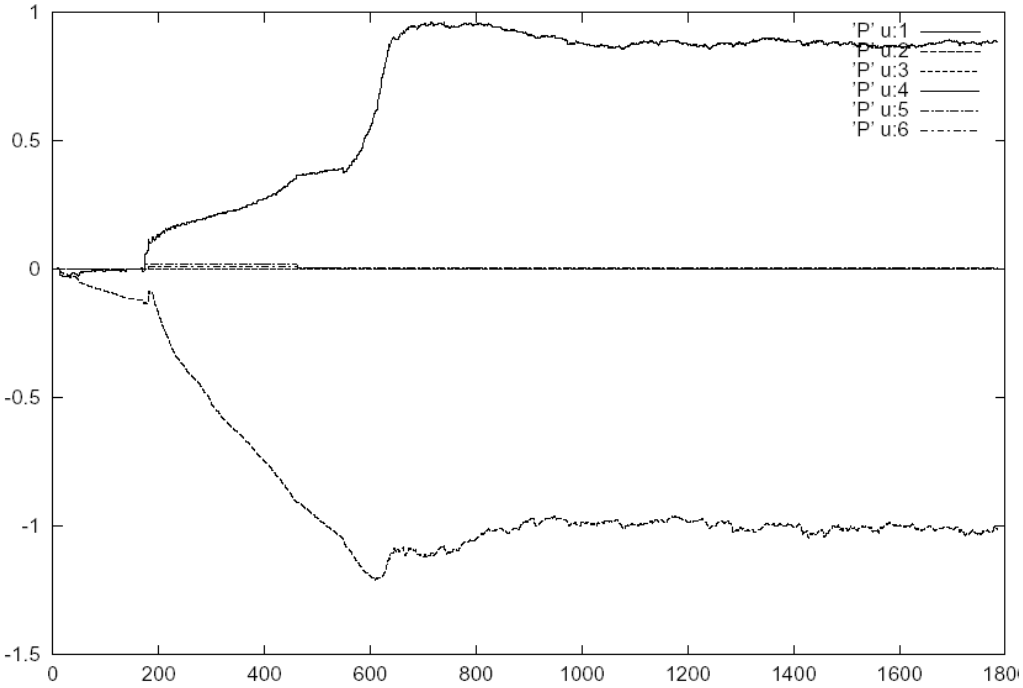

Abb. 5: Entwicklung der Controller Parameter bei der Fahrt entlang einer Wand.
Quelle: Der & Liebscher, True autonomy from self-organized adaptivity

Ein weiteres Experiment wurde mit dem Khepera Roboter mit aufgesetztem Greifarm durchgeführt. Es sollte herausgefunden werden, ob der Controller in der Lage ist, zu lernen, einen Ball zwischen den zwei Greifarmen zu balancieren. Dazu wurde vor dem Roboter für eine gewisse Strecke ein Tischtennisball mit der Hand geführt. Nach ca. 40 cm war der Roboter in der Lage, den Ball so lange zwischen den Greifarmen zu führen, bis der Ball zu schnell wurde und der Roboter nicht mehr folgen konnte. Abbildung 6 zeigt die Verläufe der Controller Parameter c_1 bis c_5 . Es fällt wiederum auf, dass die Parameter nach einer Einschwingphase jeweils gegen einigermaßen konstante Werte konvergiert, die jedoch aufgrund des dynamischen Verhaltens des Balles nicht so konstant sein können wie bei dem Experiment mit der Wand.

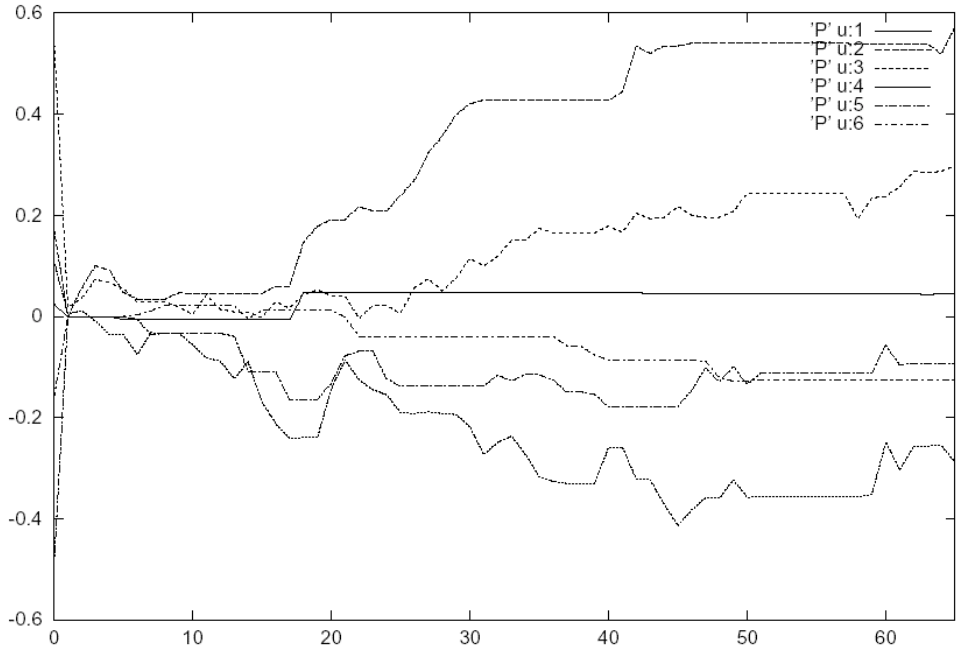

Abb. 6: Entwicklung der Controller Parameter beim Führen eines Balles.
Quelle: Der & Liebscher, True autonomy from self-organized adaptivity

4 Roboter mit Eigeninitiative

Nachdem in Kapitel 3 der Einsatz eines Neurons zur Lenkung des Khepera Roboters vorgestellt wurde, soll im Folgenden gezeigt werden, wie das Modell angepasst werden muss, damit es zur Entwicklung von Eigeninitiative bei dem Roboter geeignet ist. Außerdem wird gezeigt, wie die Steuerung der Geschwindigkeit realisiert wird.

4.1 Das Problem

Bislang war bei den Experimenten von einer gegebenen, konstanten Vorwärtsbewegung des Roboters ausgegangen worden. In Kapitel 1 war jedoch gefordert worden, dass echtes autonomes Verhalten auch die Entwicklung von Eigeninitiative beinhalten muss. Mit dem bislang verwendeten Modell ist dies jedoch nicht möglich, da es möglichst konstant bleibende Sensormesswerte anstrebt. Setzt man also einen Roboter aus, dessen Geschwindigkeit durch das Modell und den Controller aus Kapitel 3 gesteuert wird, wird er keine Regung zeigen, da die Sensormesswerte konstant bleiben und somit die Controller Parameter nicht angepasst werden.
Der Controller muss also so angepasst werden, dass der Roboter einen Anreiz zur Aktivität bekommt, ohne dass äußere Einflüsse auf ihn wirken.

4.2 Anreiz zur Aktivität

Den Anreiz, den der Roboter braucht, um aus dem Tabula rasa Zustand heraus Aktivität zu zeigen, bekommt er durch eine Modifikation des in Kapitel 3 vorgestellten Modells. Diese Anpassung beinhaltet einen Trick: Im Vorhersagemodell, das die Sensormesswerte für den Zeitpunkt t+1 vorausberechnet, wird „die Zeit umgedreht". Wie dies geschieht, verdeutlicht die Gleichung für die Zeitschleife

$$\hat{\boldsymbol{x}}_t = \boldsymbol{x}_{t+1} + \mathcal{M}^{(-)}(\boldsymbol{x}_{t+1}).$$

Es wird also nicht mehr länger x_{t+1} aufgrund von x_t berechnet, sondern es wird zum Zeitpunkt t+1 zurückgerechnet, welcher Sensormesswert aufgrund des aktuellen Modells im vorherigen Zeitschritt t hätte gemessen werden müssen. Dabei entsteht wiederum ein Fehler, der durch

$$F = \|\boldsymbol{u}\|^2 \qquad \boldsymbol{u} = \boldsymbol{x}_t - \hat{\boldsymbol{x}}_t$$

mit

Beschrieben wird. Daraus folgt, dass der Fehler klein wird, wenn das aktuelle Verhalten des Roboters gut durch sein Modell repräsentiert wird. Dies allein bringt also noch nicht den gewünschten Anreiz zur Aktivität, da das Verhalten des Roboters gut durch sein Modell repräsentiert wird, wenn er nichts tut.

4.3 Dynamik

Den entscheidenden Anstoß bekommt der Roboter durch die Dynamik, die bislang nicht im Modell enthalten ist. Dies können z.B. Störungen der Sensoren, Messfehler oder auch das Rauschen sein, mit dem die Messungen der Sensoren stets belegt sind. Diese Dynamik lässt sich als ξ_t in die Zeitschleife einfügen:

$$\hat{x}_t = x_{t+1} + \mathcal{M}^{(-)}\left(x_t + \mathcal{M}\left(x_t\right) + \xi_t\right)$$

Die Sensorwerte werden aufgrund dieser Dynamik nie exakt konstant sein. Durch die Umkehrung der Zeit im Modell ergibt sich der Effekt, dass sich Fehler in der Zeit rückwärts fortpflanzen, was dazu führt, dass der Modellfehler minimiert wird, wenn die Sensormesswerte instabil sind. Ist einmal eine kleine Abweichung zwischen den Sensormesswerten zweier Zeitpunkte t und t+1 gegeben, was de facto in jedem Zeitschritt der Fall sein wird, wird eine Kettenreaktion ausgelöst, die den Fehler immer größer werden lässt und somit eine immer schneller werdende Anpassung der Controller Parameter auslöst. Dies führt wiederum dazu, dass der Roboter aus dem Initialzustand heraus anfährt und bis zur maximalen Geschwindigkeit beschleunigt.
Neben dem Beschleunigen aus dem Initialzustand heraus ohne Fremdeinwirkung zeigt das verwendete Modell noch einen weiteren interessanten Aspekt. Wenn der Roboter auf ein Hindernis stößt, wird der Messfehler im Modell sprunghaft so groß, dass der Roboter umdreht und in die entgegengesetzte Richtung weiterfährt. Dies kann bei der Orientierung in unbekannten Umgebungen durchaus gewünscht sein.

5 Visuelle Sensoren

Nachdem gezeigt wurde, wie Geschwindigkeit und Lenkung in Abhängigkeit von den Infrarotsensoren des Khepera Roboters mit Hilfe von neuronalen Netzen gesteuert werden können, soll im folgenden der Einsatz einer Videokamera zur Orientierung in der Umgebung vorgestellt werden.

5.1 Vorbereitung der Eingabedaten

Für die Experimente mit visuellen Sensoren wurde nicht mehr der Khepera Roboter, sondern ein Modell von Pioneer (vgl. Abb. 7) verwendet. Die Daten, die der Roboter misst, können durch

$$x=(v_l, v_r, s_1, \ldots, s_k)$$

beschrieben werden. v_l und v_r sind dabei die gemessenen Geschwindigkeiten der beiden Räder und s_i die Pixelwerte der Kamera. Die Pixelwerte sind im R^3, da die Kamera RGB-Farbaufnahmen macht.
Die von der Kamera aufgenommenen Werte werden zunächst einem Preprocessing unterzogen, bevor sie in das neuronale Netz gegeben werden. Zuerst erfolgt eine binäre Klassifizierung aller Pixel, ob sie der Farbe des Balles entsprechen oder nicht. Anschließend wird das Bild auf eine Auflösung von 32 x 32 Pixel herunterskaliert, um im abschließenden Schritt aus 2 aufeinander folgenden Bildern einen Bewegungsvektor im R^2 zu erzeugen. Dieser Bewegungsvektor beschreibt die Bewegungsrichtung des Balles und wird im Folgenden für als Eingabe für das neuronale Netz verwendet.

Abb. 7: Der Pioneer Roboter folgt dem Ball.
Quelle: Der & Liebscher, True autonomy from self-organized adaptivity

5.2 Der Controller

Der Controller für den Roboter soll nun zwei Aufgaben erfüllen: Er soll den Roboter lenken und dessen Geschwindigkeit regeln. Daher besteht der Controller auch aus zwei Neuronen, von denen das erste die Geschwindigkeit regelt. Dieses bekommt als Eingabe die Sensorwerte der beiden Räder sowie die Geschwindigkeit des Balles in Fahrtrichtung. Diese Geschwindigkeit ist die eine Komponente aus dem Bewegungsvektor, der im Preprocessing errechnet wurde. Das Neuron funktioniert wie das in Kapitel 3 zur Lenkung eingesetzte, d.h. es ist auf die Minimierung des Fehlers ausgerichtet, und zwar in Bezug auf den Ball. Dadurch soll erreicht werden, dass der Roboter seine Geschwindigkeit möglichst der des Balles anpasst.
Das zweite Neuron steuert die Lenkung und Nutzt dazu die Komponente des Vektors, die senkrecht zur Fahrtrichtung des Roboters steht. Auch hier wird wieder das bekannte Modell verwendet, so dass der Roboter in der Summe sowohl Lenkung und als auch Geschwindigkeit an den Ball anpassen kann.

5.3 Ergebnisse

Nachdem das Verhalten des Roboters beim Verfolgen eines Balles untersucht wurde, konnte festgestellt werden, dass das Modell sowohl für die Lenkung als auch für die Steuerung grundsätzlich geeignet ist. Die Steuerung der Geschwindigkeit arbeitet ähnlich gut wie beim Khepera Roboter. Der Einsatz des Controllers für die Lenkung funktioniert grundsätzlich auch, es gibt jedoch noch einige technische Probleme zu lösen, beispielsweise müsste die Bildwiederholrate der Kamera erhöht werden, da in den Tests das Problem auftrat, dass sich kein Bewegungsvektor erzeugen ließ, da der Ball zwischen 2 Bildern das Sichtfeld der Kamera verlassen hatte.

6 Zusammenfassung

Im der vorliegenden Ausarbeitung wurde gezeigt, wie Roboter mit Hilfe von neuronalen Netzen bzw. einzelnen Neuronen lernen können, sich in einer unbekannten Umgebung zu orientieren, und zwar auf eine Art und Weise, die keinen externen Trainer erfordert. Außerdem wurde ein Modell vorgestellt, dass es dem Roboter erlaubt, aus dem Initialzustand heraus, also ohne vorher gesetzte Gewichte, anzufahren und sich in seiner Umgebung zu bewegen. Außerdem wurde ein Ansatz vorgestellt, wie der Einsatz von Videokameras in diese Modelle eingearbeitet werden kann.
Der Bezug zum Roboterfußball ist bei den hier vorgestellten Experimenten noch eher schwach, sie zeigen aber, in welche Richtung sich die Steuerung und Lernfähigkeit der Agenten beim Roboterfußball entwickelt wird. Momentan dominieren dort noch vorwiegend Ansätze aus dem Bereich der klassischen künstlichen Intelligenz.

Literaturverzeichnis

DER, LIEBSCHER (2002): *True autonomy from self-organized adaptivity*, Bristol.

PATTERSON (1997): *Künstliche neuronale Netze*, München.

LIPPE (2003): *Skript zur Vorlesung Softcomputing*, Münster. Online unter: http://wwwmath.uni-muenster.de/math/inst/info/ Professoren/Lippe/lehre/ skripte/nnscript/index.html (abgerufen am 02.06.2003).

K-Team (o.J.): Online unter: http://ww.k-team.com/ (abgerufen am 07.06.2003).