Praktikum Autonome Roboter

—

Projekt III
Visuo-Motorische Koordination - Ein Golf spielender Roboter

Priv.-Doz. Dr. R. Der
Institut für Informatik
Universität Leipzig

30. September 1997

1 Aufgabenstellung

Der autonome Roboter (AR) soll eine der folgenden Verhaltenskompetenzen durch Lernen erwerben:

Einen Ball entweder (i) in ein vorgebenes Ziel (Tor) zu schieben oder (ii) mit einem Schläger durch eine Drehung mit geeigneter Winkelgeschwindigkeit in das Ziel zu schlagen (der AR als Golfspieler). Ball und Ziel können durch die Linearkamera erfaßt werden. Im einfacheren Fall soll die Position des Zieles in Weltkoordinaten bekannt sein, der Ball wird durch die IR–Sensoren erfaßt.

Im Praktikum ist eines der beiden unten geschilderten Lernverfahren einzusetzen, als Vorbild und Anregung für die praktische Realiserung kann die Arbeit [1] dienen, die recht ausführlich Schwierigkeiten und praktische Lösungsmöglichkeiten diskutiert. Alternativ kann auch ein konservativer, d. h. regelbasierter Ansatz versucht und die Ergebnisse miteinander verglichen werden.

Weiterführende Literatur: Vorlesungsskript "Robotik" [2]

2 Diskretisierung von Zustands– und Aktionsraum

Die unten angegebenen Lernverfahren setzen eine geeignete, diskrete Darstellung des Zustands–Aktionsraumes voraus, die zunächst am konkreten Beispiel des Golf spielenden Roboters dargelegt werden soll.

2.1 Der Zustandsraum

Im einfachsten Fall kann man den Arbeitsraum des AR durch ein Gitter überdecken. Dies hat aber den Nachteil, daß man in absoluten Koordinaten arbeiten muß, während die Strategie des AR sich natürlich immer aus seiner Stellung relativ zu Ball und Ziel bestimmt. Geeignete Koordinaten sind (1) der Abstand r des des AR zum Ball und (2) der Winkel α einer geeigneten Achse des AR mit der Verbindungsgeraden zwischen AR und Ball

$$\text{AR} \quad \oslash\!\longrightarrow\!\!\!\!-\!\!\odot \quad \text{Ball}$$

Dazu kommt im allgemeinen Fall (3) der Abstand AR–Ball und (4)der Winkel zwischen der Verbindungsgeraden AR–Ball und Ball–Ziel. Im einfachen Fall des Golfspielers, der nur dem Ball eine bestimmte Geschwindigkeit geben soll, so daß er eine vorgegebene Strecke zurücklegt, ist der Zustandsraum allein durch den Vektor

$$\vec{x} = (r, \alpha)$$

gegeben. Diese Größen sind aus den Sensordaten zu ermitteln und dann ist der Zustandsraum geeignet zu dikretisieren, d. h. man finde eine Abbildung $\vec{x} \to j$, wobei $j = 1, 2, ..., N$ die diskreten Zustände sind. Das kann entweder von Hand oder, wie unter Sesordatenfusion beschrieben, durch eine Kohonenmap realisiert werden.

2.2 Der Aktionsraum

In jedem diskreten Zustand j hat der AR eine diskrete Menge $\mathcal{A}(j)$ von Aktionen $a \in \mathcal{A}$ zur Verfügung. Beispiele sind Bewegung nach N,S,O,W bis in die nächste Zelle des diskretisierten Zustandsraumes, Ausführung der Drehung mit einer aus einer Menge von diskreten Winkelgeschwindigkeiten usw. Weitere Ausführungen zur Wahl der diskreten Aktionen finden sich in [1].

3 Lernverfahren

Im Projekt sollen Methoden der inkrementellen dynamischen Optimierung als (1) Q–Lernen und/oder (2) C–Lernen eingesetzt werden, beides spezielle Realisierungen des Reinforcement–Lernens.

3.1 Reinforcement–Lernen

Beim Reinforcement–Lernen ist die einzige Information, die der AR von außen erhält, die Nachricht über das Erreichen des Zieles, d. h. das reinforcement Signal ist $r = 1$ falls der AR das vorgegebene Ziel (oder ein Teilziel) erreicht hat und $r = 0$ sonst. Bei mehreren Teilzielen kann das Reinforcement auch abgestuft erteilt werden. Zusätzlich kann auch negatives Reinforcement (Strafe) $r < 0$ ausgegeben werden, falls der AR eine Handlung mit einem verbotenen oder unerwünschten Ergebnis ausgeführt hat. Das Verhalten des AR ist durch die sog. policy Funktion π definiert, die in jedem Zustand j die Aktion $a \in \mathcal{A}(j)$ des AR festlegt, $\pi : j \to a$ oder

$$a = \pi(j) \tag{1}$$

Aufgabe ist das Auffinden einer mögichst optimalen policy Funktion π.

Die Güte der Policy wird an dem Reinforcement gemessen, das der AR unter dieser Policy in der Zukunft zu erwarten hat. Ist der AR zum aktuellen Zeitpunkt im Zustand j und sind $j', j'', ..., j^{(n)}, ...$ die Folgezustände, d. h. die Zustände die der AR unter der Policy π in den folgenden Zeitschritten durchläuft, so ist die Güte der Policy für den Zustand j

$$V(j) = \sum_{k=0}^{\infty} \gamma^k r\left(j^{(k+1)}\right) \tag{2}$$

Unter der gegebenen Policy π ist diese Größe gleichzeitig auch ein Maß für den Wert (value) das Zustandes j im Hinblick auf die Menge von Reinforcement, das der AR in der Zukunft aufsammeln wird, wenn er von diesem Zustand aus startet. Die Größe γ in 2 ist die sog. discount Rate, die wegen $\gamma < 1$ zukünftiges Reinforcement geringer wichtet als das unmittelbar erhältliche. Damit wird eine Policy um so höher bewertet, je schneller sie den AR zum Ziel (Quelle für Reinforcement) führt.

Weitere Ausführungen finden sich im Vorlesungsskript [3].

3.2 Q–Lernen

Während die Wertfunktion $V(j)$ nur den Wert des Zustandes j angibt, bewertet die sog. Q–Funktion $Q(j, a)$ Zustands–Aktions–Paare (ZAP) (j, a) und legt damit zugleich die Policy fest. Unter einer deterministischen Policy wird dabei im aktuellen Zustand j immer die bestbewertete Aktion ausgeführt, d. h.

$$a = \pi(j) = \arg\max_{a'} Q(j, a') \tag{3}$$

und damit $Q(j, a) \geq Q(j, a')\ \forall a'$. Die vor allem in der Frühphase des Lernens erforderliche Exploration des Zustands–Aktionsraumes (ZAR) wird durch eine stochastische Policy realisiert, wobei die Aktion a mit einer Wahrscheinlichkeit

$p_a(j)$ ausgeführt wird. Die Wahrscheinlichkeiten können etwa durch eine Boltzmannvereilung in Termen der Q–Funktion definiert werden

$$p_a(j) = \frac{e^{Q(j,a)/T}}{\sum_{a'} e^{Q(j,a')/T}} \tag{4}$$

Die fiktive Temperatur T bestimmt die Explorationsrate, diese wird im Laufe des Lernens "abgekühlt", für $T \to \infty$ wird schließlich der deterministische Fall erreicht.

Die Q–Funktion legt offensichtlich des Verhalten des Systems vollständig fest. Das Q–Lernen, d. h. die schrittweise inkrementelle Verbesserung einer Anfangsschätzung der Q–Funktion wird durch den Algorithmus von Watkins realisiert:

1. Initialisierung der Q–Funktion, z. B. $Q(j,a) = 0\,\forall j,a$ oder $Q(j,a) = random\,\forall j,a$. Start des AR in einem beliebigen Zustand j.

2. Im aktuellen Zustand j: Auwahl einer Aktion a gemäß der Wahrscheinlichkeitsverteilung $\{p_a(j)\}$.

3. Ausführung von a: AR führt Übergang von j in den Folgezustand j' durch. [1]

4. Update $Q(j,a)$ für das ZAP (j,a) gemäß[2]

   $$\Delta Q(j,a) = \epsilon\left[r(j') + \gamma V(j') - Q(j,a)\right] \tag{5}$$

   wobei $r(j')$ das im Folgezustand j' erhaltene Reinforcement und

   $$V(j') = \max_{a'} Q(j',a') \tag{6}$$

   der Wert des Folgezustandes ist.

5. j' wird neuer aktueller Zustand

6. Abbruch? GOTO 2

Die Konvergenz des Algorithmus (unter einer geeigneten Abkühlungsstrategie für ϵ und T) gegen die optimale Strategie kann exakt gezeigt werden, s. [**?**].

3.3 C–Lernen

Ein einfacher zu implemetierendes und in verschiedener Hinsicht dem Q–Lernen überlegenes Verfahren ist das von uns entwickelte sog. C–Lernen. Dieses verzichtet auf die Einführung einer Bewertung von Zustands-Aktionspaaren sondern verbessert stattdessen inkrementell direkt eine Schätzung des Wertes $V(j)$ der Zustände und der Wahrscheinlichkeiten $p_a(j)$ für die Aktionen selbst. Wesentlicher Unterschied ist die Update–Regel, s. Punkt 4.

[1] Der Fall $j' = j$ bedarf in den praktischen Anwendungen besonderer Aufmerksamkeit.

[2] ΔQ ist die Änderung (Inkrement) von Q im Updateschritt (Lernschritt).

3.3.1 Der Algorithmus

Der Algorithmus umfaßt die folgenden Schritte:

1. Initialisierung der Wertfunktion, z. B. $V(j) = 0\,\forall j$ und $p_a(j) = 1/A(j)$ wobei $A(j) = \#\,\mathcal{A}(j)$ die Zahl der verfügbaren Aktionen im Zustand j ist (Gleichwahrscheinlichkeit aller Aktionen). Wenn a priori Domänen–Wissen über die besten Aktionen vorhanden ist, kann dieses direkt durch dezidierte Wahl der Wahrscheinlichkeiten integriert werden.

2. Im aktuellen Zustand j: Auwahl einer Aktion a gemäß der Wahrscheinlichkeitsverteilung $\{p_a(j)\}$.

3. Ausführung von a: AR führt Übergang von j in den Folgezustand j' durch.

4. Update $V(j)$, $p_a(j)$ für das ZAP (j, a) gemäß

$$\Delta V(j) = \epsilon\left[r(j) + \gamma V(j') - V(j)\right] \tag{7}$$

$$\Delta p_a(j) = \frac{\epsilon}{K}\left[r(j) + \gamma V(j') - V(j)\right]$$

   wobei $r(j')$ das im Folgezustand j' erhaltene Reinforcement ist. Die Parameter sind günstigerweise im Bereich $0.1 < \epsilon < 0.3$ und $3 < K < 10$ zu wählen.

5. j' wird neuer aktueller Zustand

6. Abbruch? GOTO 2

Das Verfahren zeichnet sich gegenüber dem Q-Lernen durch eine Reihe von Vorteilen aus. Diese betreffen (i) die Selbsteinstellung der Explorationsrate, d. h. der Übergang zur deterministischen Strategie wird nicht global geregelt sondern stellt sich anhand der lokalen Unterschiede in den Werten der angesteuerten Folgezustände individuell von selbst ein. Damit ist die Abkühlungsstrategie des Q-Lernens eingespart worden. (ii) Die Übernahme von Vorwissen erfolgt durch die Vorgabe von Werten und Wahrscheinlichkeiten der einzelnen Zustände bzw. Aktionen und kann damit unmittelbar etwa aus vorhandenem Regelwissen übernommen werden, s. a. [5].

3.3.2 Negatives Reinforcement in fehlerintoleranter Umgebung

Im Prinzip funktioniert die Update–Regel 7 auch für negatives Reinforcement. Allerdings erlernt der AR dabei nicht, sich von den negativen Quellen möglichst weit entfernt zu halten, wie das in einer fehlerintoleranten Umgebung sinnvoll wäre. Ein verbesserter Algorithmus ersetzt die Regel 7 durch die folgende

$$\text{Update–Regel} \quad \begin{aligned} \Delta V(j) &= \epsilon \left[r(j) + \gamma \left(V(j') - V\left(j^{(-)}\right) \right) - 2V(j) \right] \\ \Delta p_a(j) &= \tfrac{\epsilon}{K} \left[r(j) + \gamma \left(V(j') - V\left(j^{(-)}\right) \right) - 2V(j) \right] \end{aligned}$$

wobei $j^{(-)}$ der Vorgängerzustand zum aktuellen Zustand j ist. Mit dieser Regel konnten in einer Reihe von Anwendungen gute Ergebnisse erzielt werden. Zu beachten ist, daß ϵ hier nict den Wert $\epsilon = 1/2$ überschreiten darf.

3.4 Effektivierung der Lernverfahren

Neben der geeigneten Darstellung der relevanten Gebiete des Zustands-Aktions-Raumes steht die Effektivierung des Lernverfahrens selbst als unabdingbare Voraussetzung für den Erfolg des Projektes. Aus unseren bisherigen Erkenntnissen können wir die folgenden Lösungsansätze schon jetzt angeben, weitere Erfahrungen im Projekt werden die nötige Spezifizierung und Weiterentwicklung dieser Erkenntnisse bringen:

- **Initialisierung durch Vorwissen:** Unter "academic world" Bedingungen kann man es sich leisten, den Lerner mit einer *tabula rasa* Initialisierung zu starten. Der weitaus größte Teil der Lernzeit wird dann allerdings bis zum Erreichen einer suboptimalen Strategie verbraucht, dagegen ist das Feinlernen der optimalen Strategie weit weniger zeitkritisch. Im Projekt soll versucht werden, den Lerner durch "Vormachen" einer zumindest suboptimalen Strategie zu initialisieren, so daß nur noch das Feinlernen eigentlich geleistet werden muß. Damit läßt sich hoffentlich das Lernen nicht nur in der Computersimulation in endlicher Zeit realisieren.
- Die Leistungsfähigkeit des Lernens hängt in entscheidender Weise von der adäquaten Diskretisierung des Raumes ab. Es wäre deshalb anzustreben, die Diskretisierung während des Lernens den Erfordernissen des Lerners anzupassen. Anregungen dazu finden sich in [4].
- Interpolation: Durch die Diskretisierung ist die Genauigkeit der Aktionen begrenzt, die Diskretisierung entspricht einer stückweise konstanten Approximation der Policy–Funktion π. Durch geeignete Interpolation zwischen den Stützstellen kann die Genauigkeit erhöht werden.

Literatur

[1] M. Asada. Purposive behaviour acqusition for a real robot by vision based reiforcement learning. *Machine Learning*, 23:279 – 304, 1996.

[2] R. Der. Vorlesung Robotik. Universität Leipzig Institut für Informatik. http://www.informatik.uni-leipzig.de/ der/Vorlesungen/skrpt-rob.html.

[3] R. Der. Vorlesung Robotik. UniversitätLeipzig Institut für Informatik Kap. 2.3: Die Planungskomponente. http://www.informatik.uni-leipzig.de/ der/Vorlesungen/rob-plan.ps.Z.

[4] R. Der and M. Herrmann. Efficient q-learning by division of labour. In *Proc. International Conference on Artificial Neural Networks - ICANN95*, pages 129 – 134, 1995.

[5] R. Der and M. Herrmann. Self-adjusting reinforcement learning. In *Nonlinear Theory and Applications - NOLTA 96*, pages 441 – 444, 1996. Internet: http://www.informatik.uni-leipzig.de/ der/Veroeff/nolta96.ps.gz.