Seminar - Computational Intelligence

Learning from Demonstration by Constructing Skill Trees

Manuel Zellhöfer
Mat.Nr. 4596223

SS 2012
16. August 2012

Inhaltsverzeichnis

Das »Berechnen« (oder auch die mathematische Repräsentation) von intelligentem Verhalten war das Thema des Seminars mit dem schönen Titel »Computational Intelligence«. Verschiedene Ansätze die in sehr unterschiedliche Richtungen vordringen wurden vorgestellt. Diese Ausarbeitung zum Seminarvortrag befasst sich mit einer Technik die den Namen »Constructing Skill Trees« trägt. Es geht darum, einem Agenten »intelligentes« Verhalten zu ermöglichen. Es soll die Methodik und Theorie des vorgestellten Ansatzes erläutert, dieser bestmöglich bewertet und vielleicht auch kritisiert und mit anderen im Seminar vorgestellten Ansätzen in Bezug gesetzt werden.

1 Vorgestellter Ansatz

»Constructing Skill Trees« ist genau gesagt nur ein Teil des Ansatzes, der eine Reihe anderer Methoden und Techniken kombiniert und damit versucht, die Idee des Reinforcement Learning auf eine neue Ebene zu hieven.

Für das klassische Problem des Reinforcement Learning (*RL*), wird ein Lösungsansätz vorgeschlagen, der durch Einführung einer Hierarchie (*Skill Framework*), Methoden zur Dimensionsreduktion und dem effizienten Verwerten von Beispielverhalten (*Learning from Demonstration, LfD*) intelligentes Verhalten außerhalb von »diskreten« Labyrinthen ermöglicht. Also vielseitige Agenten (z.B. mobile Roboter, die über Endeffektoren verfügen) zu befähigen, in komplexen Umgebungen zu agieren und dabei Demonstrationen eines Experten zuhilfezunehmen.

Zur Erläuterung soll zuerst letztgenanntes »klassisches« RL-Problem ins Gedächtnis gerufen werden. Zu den Schwierigkeiten die sich daraus ergeben werden dann die betrachteten Lösungsansätze erklärt. Für diese wird auch der Großteil der zugrundeliegenden Theorie erläutert.

1.1 Reinforcement Learning

Dem RL liegt das Markow-Entscheidungsproblem (engl. *Markow-Decision-Problem, MDP*) zugrunde, ein stochastisches Modell für Agenten in definierten Zuständen, dem bestimmte Aktionen zur Verfügung stehen für die er Belohnungen oder Strafen (*Return*) erhält. Formal besteht ein solches Problem aus

- einer Menge von Zuständen $S \subset \mathbb{R}^d$,
- einer Menge von Aktionen $A \subset \mathbb{R}^n$,
- den Übergangswahrscheinlichkeiten $P^a_{ss'} = P(s'|s,a)$ und
- dem Return[1] $R : S \times A \times S \to \mathbb{R}$

[1] Jedem Zustandsübergang $s \to s'$ durch eine Aktion a wird eine Belohnung r zugewiesen

Die Lösung eines MDPs besteht im Finden einer Strategie (engl. *Policy*)

$$\pi : S \to A$$

die für jeden Zustand s die Aktion wählt, die den *erwarteten* zukünftigen Return maximiert. Ein Hilfsmittel um die optimale Strategie π^* zu finden ist die Bewertungsfunktion (engl. *Valuefunction*)

$$V_\pi(s) = E\left\{\sum_i \gamma^i r_i | s_0 = s\right\}$$

Der Faktor $\gamma \in [0, 1]$ dient der Abwertung zukünftiger Returns.[4]

Die Aktionen werden so gewählt, dass sie für jeden Zustand diese Bewertungsfunktion maximieren. Da in einer unbekannten Umgebung nicht genug Information vorhanden ist, um sie komplett zu berechnen, wird sie approximiert. Im vorgestellten Ansatz geschieht dies durch ein lineares Modell

$$\hat{V}(s) = \sum_{i=1}^{k} w_i \phi_i(s)$$

mit Gewichten $w_1, \ldots, w_k$ und Basisfunktionen $\Phi = [\phi_1, \ldots, \phi_k]$ die hier aus Sinusschwingungen unterschiedlicher Frequenzen bestehen. Diese Art der Basisfunktionen soll sich laut Konidaris [3] in [2] besonders gut für dieses Problem eignen.

Für einfache MDPs, also solche die wenige Aktionen beinhalten, die eine überschaubare Zahl von (diskreten) Zuständen definieren gibt es etablierte Methoden (Dynamische Programmierung, TD-Learning und Derivate). Diese lassen den Agenten durch wiederholte, teils zufällige Interaktion mit seiner Umwelt (Exploration) nach einer optimalen Lösungsstrategie suchen. [5]

Der vorgestellte Ansatz hat sich nun zur Aufgabe gemacht folgende Schwierigkeiten anzugehen:

- *Komplexe Strategien*
  Das MDP wird schnell unhandlich oder unlösbar, wenn komplexe Handlungen zur Findung des gewünschten Ziels nötig sind. Durch die Einführung einer hierarchischen Struktur soll dieses Problem gelöst werden (Abschnitt 1.2). Ferner werden Demonstrationen der Lösung verwendet um den Agenten bei schwierigen Aufgaben das Ziel überhaupt finden zu lassen (Abschnitt 1.4).
- *Kontinuierliche, hochdimensionale Zustände und Aktionen*
  Vorhandene Lösungen funktionieren mit niedrigdimensionalen, diskreten Zuständen und Aktionen. Für kontinuierliche muss eine andere Herangehensweise gewählt werden. Auch für mehrdimensionale Zustands- und Aktionsräume wird eine Methode vorgeschlagen, die Komplexität zu reduzieren (Abschnitt 1.3).

- *Wiederverwenden und Verbessern von bereits Gelerntem*
  Ganz nebenbei sollen die vorgestellten Methoden es ermöglichen, bereits gelernte Fähigkeiten des Agenten wiederzuverwenden und diese Fähigkeiten zu verbessern.

[2]

1.2 Komplexe Aufgaben lösen durch *Skill Chaining*

Durch *Skill Chaining* wird es dem Agenten ermöglicht weitere Aktionen (*Skills*) seinem Repertoir hinzuzufügen. Ein solcher *Skill* ist definiert durch

- eine eigene Strategie π_o, die die konkrete Aktion definiert
- eine *Initiationsmenge* $I_o \subset S$ die die Zustände beinhaltet, in welchen die neue Aktion ausgeführt werden kann,
- eine *Endbedingung* β_o welche für einen Zustand die Wahrscheinlichkeit repräsentiert, dass die neue Aktion abgeschlossen wurde und
- eine eigene *Belohnungsfunktion*

Der Skill an sich stellt wieder ein eigenes aber einfaches RL-Problem dar. Um nun ein komplexes Problem mithilfe von Skills zu lösen wird nach folgendem Schema vorgegangen:

1. Exploriere bis das Ziel entdeckt wurde[2]
2. Sobald das Ziel entdeckt wurde, erzeuge einen neuen Skill wobei die Endbedingung (β_o) dem gefundenen Ziel entspricht
3. Die Initiationsmenge I_o wird im weiteren Verlauf nach dem Prinzip von Versuch & Irrtum mithilfe eines Klassifikators gelernt.

Das *Chaining* geschieht dann, wenn der Agent sich in die Initiationmenge eines vorhandenen Skills begibt. Genau wie in Schritt 1 wird dann ein neuer Skill erzeugt. Es entsteht also im Idealfall eine Kette von Skills, die den Agenten vom Start zum Ziel führt. Dieser Vorgang ist in Abbildung 1 veranschaulicht.[3]

Zu beachten ist, dass sich dieser Vorgang über eine vielzahl von Episoden erstreckt, in denen der Agent immer wieder versucht vorhandene Skills auszuführen und erfolgreich zuende zu bringen. Die vorgeschlagene Lösung für dieses Problem wird in Abschnitt 1.4 erläutert.

[2] Für komplexe Probleme führt dieser Schritt meinem Verständnis nach ins Leere. Das Prinzip des *Skill Chaining* lässt sich aber erklären und wie in Abschnitt 1.4 erwähnt, wird der Agent ohnehin von einem Lehrer zum Ziel geführt.

[3] Bringen den Agenten mehrere Skills in die Initiationsmenge eines weiteren Skills, führt dies zu einem *Skill Tree*.

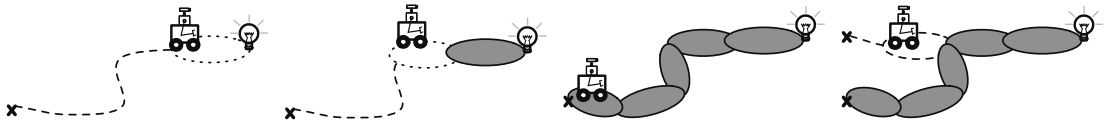

Abbildung 1: Beim Finden des Ziels (Lampe) wird ein Skill erstellt. Sukzessive wird vom Start bis zum Ziel eine Kette von Skills erstellt. Gerät der Agent über einen neuen Pfad in die Initiationsmenge eines Vorhandenen Skills, entsteht ein Skill Tree. Aus [2].

Skill Chains oder allgemeiner *Skill Trees* eignen sich also um Lösungsstrategien zu repräsentieren, die sich schwer durch eine einzige Strategie mit einfachen Aktionen darstellen lassen. [2]

1.3 Dimensionalität reduzieren durch *Abstraction Selection*

Ein Agent der in einer »natürlichen« Umgebung agieren soll benötigt eine Vielzahl von Aktionen. Zur einfachen Bewegung (die sich auch schon sehr als sehr komplex erweisen kann) können etwa noch ein oder mehrere Endeffektoren hinzukommen. Jedes zusätzliche Mittel erweitert den Aktions- und Zustandsraum des Agenten. Für bestimmte Aufgaben werden jedoch nicht alle Zustände bzw. Aktionen benötigt. Um dies zu vereinfachen werden Abstraktionen eingeführt. Eine Abstraktion M ist ein Paar von Abbildungen (σ_M, τ_M):

$$\sigma_M : S \to S_M$$

$$\tau_M : A \to A_M$$

Sie transformieren Zustands- und Aktionsraum in niedrigdimensionale Räume. Bei diesen kann es sich entweder um eine einfache Submenge der Ursprungsvariablen handeln (etwa nur die, die einen Endeffektor steuern) oder es können komplexe Transformationen involviert sein (etwa die Berechnung einer Distanz unter Zuhilfenahme der Sensorik oder die Transformation eines Differenzialantriebs in Vorwärts- und Rotationsgeschwindigkeit). Die Abstraktionen werden im vorgestellten Ansatz vorher definiert und nicht vom Agenten selbst gelernt. Ihm fällt die Aufgabe zu, für ein Ziel die richtige Abstraktion zu wählen.

Diese Idee wird mit den Skills aus dem vorherigen Abschnitt kombiniert, so dass jeder Skill seine eigene Abstraktion hat. Ferner hat jede Abstraktion bedingt durch die Einschränkung der Zuständen eigene Basisfunktionen Φ_M zur Approximation der Bewertungsfunktion. Die Auswahl der Abstraktion erfolgt dann über die Güte der jeweiligen Approximation der Bewertungsfunktion. [2]

1.4 *LfD* durch *Constructing Skill Trees*

Constructing Skill Trees beschreibt nun die Idee, aus Beispieltrajektorien die in Abschnitt 1.2 beschriebenen Skill Trees oder Chains direkt zu berechnen. Die Aufwändige Exploration und häufige Wiederholung von Interaktion mit der Umgebung bleibt dadurch erspart. Zwei Schritte sind nötig, um aus einer Reihe Beispieltrajektorien einen Skill Tree zu erstellen:

1. Jede Trajektorie wird in eine Reihe von Skills (Skill Chain) segmentiert
2. Die gewonnen Skill Chains werden zu einem Skill Tree zusammengefügt

1.4.1 *Changepoint Detection* zur Segmentierung der Beispieltrajektorien

Ein *Changepoint* ist ein Punkt der zwei verschiedene Aktionen innerhalb einer Trajektorie trennt. *Changepoint Detection* ist der Mechanismus der diese Punkte anhand einer gegebenen Trajektorie automatisch identifiziert. Der vorgestellte Ansatz benutzt hierfür ein Hidden Markov Modell. Gegeben ist eine Menge an linearen Modellen, die die einzelnen Abschnitte der Trajektorien repräsentieren können. Diese entspricht den vorher definierten Abstraktionen mit linearen Basisfunktionen über den gegebenen Zustandsräumen.

Die verborgenen Zustände q_i des HMM entsprechen eben diesen Modellen. Die Zustandsübergangswahrscheinlichkeiten hängen ab von der Länge des betrachteten Segments und der A-priori-Wahrscheinlichkeit $p(q_i)$ des Modells:

$$T(q_i, q_j) = g(j - i - 1)p(q_j)$$

Die Werte i und j $(j > i)$ beschreiben verschiedene Zeitpunkte der Trajektorie, q_j dementsprechend das verwendete Modell zum Zeitpunkt j. Die Ausgabewahrscheinlichkeiten werden über die Güte der Approximation des Segments mit dem entsprechenden Modell und ebenfalls über die Länge des Segments beschrieben:

$$P(y_{i+1} : y_j | q_i) = P(i, j, q_i)(1 - G(j - i - 1))$$

Der Wert $P(i, j, q_i)$ basiert auf der angesprochenen Approximationsgüte[4]. Die zu approximierende Wertfolge $y_{i+1} : y_j$ entspricht dem erhaltenen Return beim abfahren der Trajektorie. Die Wahrscheinlichkeit, dass ein Segment exakt die Länge l hat ist $g(l)$. Die, dass ein Segment eine Länge $> l$ hat $(1 - G(l))$. Diese werden über eine geometrische Verteilung definiert für die vorher eine erwartete Segmentlänge gewählt wird.

Basierend auf diesem Modell wird unter Verwendung „eines online Viterbi Algorithmus" zu jedem Zeitpunkt t die Wahrscheinlichkeit berechnet, dass zu einem vorhergehenden

[4] Auf die Berechnung dieses Wertes möchte ich mangels Verständnis und Zeit nicht näher eingehen.

Zeitpunkt j ein *Changepoint* mit einem Modell q geschah: $P_t(j, q)$. Da dieser Wert häufig sehr gering sein sollte (die Abstraktionen sollten so gewählt werden das im Idealfall nur eine zur Repräsentation genügt). Wird ein Partikelfilter verwendet um das Verfahren zusätzlich zu beschleunigen. [2]

1.4.2 Skill Chains zu Skill Trees zusammenfügen

Die erhaltenen Skill Chains werden im Anschluss zu Bäumen zusammengefügt. Es wird zunächst der letzte Skill zweier Ketten betrachtet. Falls diese die selbe Abstraktion verwenden, sich im Zustandsraum ausreichend überlappen und ähnliche Parameter zur Approximation verwenden, werden sie zu einem einzigen Skill zusammengefügt. Diese Ähnlichkeit wird über die Größe $P(i, j, q_i)$ aus dem vorherigen Abschnitt und einer Überprüfung des Wertebereichs der Trajektorie im Zustandsraum bestimmt.

Erweisen sich zwei Skills als ähnlich genug werden die nächsten beiden Skills der Kette betrachtet. Erfüllen diese nicht die Vorraussetzungen um zusammengefügt zu werden, entsteht an dieser Stelle ein Skill Tree. Abbildung 2 veranschaulicht den Vorgang des Zusammenfügens. [2]

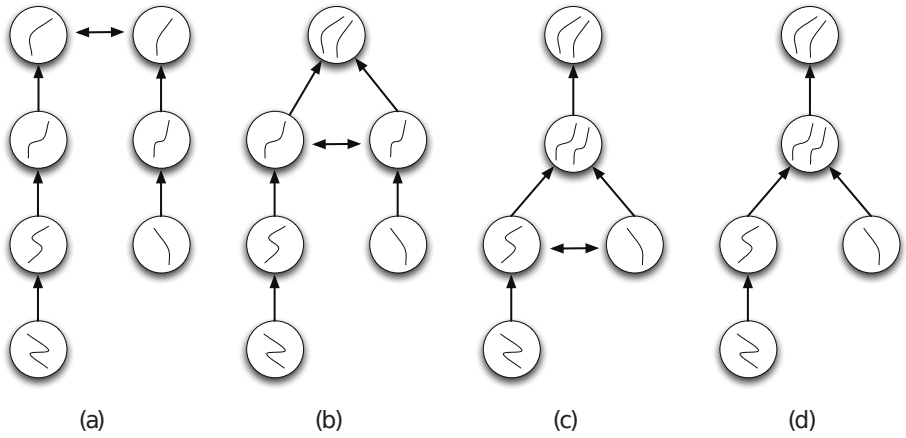

Abbildung 2: Das Zusammenfügen zweier Skill Chains zu einem Skill Tree. Beginnend beim letzten Skill (a) werden der Reihe nach die Skills der Kette zu einem vereint, bis zwei zu unterschiedlich sind um zusamemengefügt zu werden (c). Aus [2].

2 Einschätzung

Der Vorgestellte Ansatz beschreibt eine praktische Lösung um konkreten Systemen oder Agenten »intelligentes« Verhalten zu verleihen. Der Schwerpunkt liegt dabei nicht in der Imitation menschlichen Verhaltens sondern um die Ingenieursmäßige Lösung von Problemstellungen in diesem Bereich.

Die Idee des Reinforcement Learning wird dabei aus der nach meinem Kenntnissstand eher theoretischen Umgebung geholt und mithilfe einer Mehrzahl von neuen Ideen und Weiterentwicklungen soweit vorangetrieben, dass sich ein mobiler Roboter in einer verhältnismäßig komplexen Umgebung zurechtfinden kann. Der größte Kniff daran ist die Verwendung von Beispieltrajektorien oder Demonstrationen um den Agenten überhaupt erst auf den richtigen Pfad zu schicken.

Kritik an der vorgestellten Veröffentlichung lässt sich vielleicht in zweierlei Punkten finden[5]: Ersterer hat mit dem Aufbau des Aufsatzes zu tun und liegt wohl darin begründet, dass die letztendlich vorgestellte Technik eben die (komplexe) Synthese einer Vielzahl neuer Konzepte darstellt: Die Erweiterung von RL, die einführung von Abstraktionen, die Segmentierung von Beispieltrajektorien. Während erste beiden direkt im Kontext von »Robot Learning« eingeführt wurden, wurde die *Changepoint Detection* erst allgemein erklärt und dann auf den Kontext übertragen, was der Übersicht nicht zuträgich war. Weiterhin wurde ein Großteil der Techniken deutlich erklärt, wobei meines Erachtens nach wichtige Details in Nebensätzen unter den Tisch fielen („Note that it is possible...", „Abstraction Selection finds...", „This is a natural way..")

Zudem ist mir aus dem Text nicht klar geworden was nun die wirkliche Neuerung ist. Nur die Segmentierung der Beispieltrajektorien und das Zusammenfügen zu einem Skill Tree? Dies würde auch dem Titel der Publikation entsprechen. Nur halte ich dann den „Background"-Teil für zu Umfangreich.

Ein anderer Kritikpunkt richtet sich konkret an den Inhalt: Die vorgestellte Technik wird *Constructing Skill Trees* genannt. und beinhaltet auch explizit die Methode um solche Skill Trees zu erstellen. In keinem der vorgestellten Experimente jedoch zeigt sich ein solcher Skill Tree. Es handelt sich stets um Skill Chains. Auch ähnliche und wiederkehrende Skills (betätigen zweier unterschiedlicher Knöpfe durch den Endeffektor eines mobilen Roboters oder *Drive to wall* und *Drive to panel*) werden im Experiment durch zwei unterschiedliche Skills abgebildet. Ich kann nicht ausschließen, dass mir die Ursache für diesen Umstand mangels Verständnis entgeht, dennoch stört dieser Punkt. Zudem behandelt der Teil im „Discussion"-Abschnitt der Publikation der diese Thematik tangiert diese Frage nicht.

Auch stellt sich mir die Frage, wie genau der Agent die einzelnen Skills im Verlauf verbessert. Die Fähigkeit wurde einleitends angesprochen taucht jedoch später nicht mehr

[5] An dieser Stelle soll erwähnt sein, dass diese Ausarbeitung (aufgrund von schlechtem Zeitmanagement und mangels sinnvoller Organisation) unter erheblichem Zeitdruck entstanden ist. Einige der Kritikpunkte könnten gründlicher Überprüfung wohlmöglich nicht standhalten.

auf. Es wird nicht klar inwiefern der Agent nach der verarbeitung der Beispieltrajektorien noch selbst exploriert. Der Schwerpunkt schien darin zu legen, den Roboter die Beispieltrajektorien wiederholen zu lassen.

Diese Kritikpunkte vermindern nicht die bemerkenswerten Fähigkeiten, die ein mit dieser Technik versehener Roboter an den Tag legt. Auf der Webseite des Autors ([1]) sind einige (meines Erachtens nach) faszinierende Videos zur Verfügung gestellt, die einen mit dem CST-Algorithmus programmierten Roboter in Aktion zeigen.

3 Vorträge im Seminar mit ähnlichem Gebiet

Von den übrigen im Seminar vorgestellten Ansätzen lassen sich diejenigen in Bezug setzen, die die Thematik von Reinforcement Learning oder mobiler Robotik aufgreifen. In Frage kommen demnach die Vorträge von

- Ercan Kücükkaraca – *Where do Rewards come from?*,
- Peter Hirschfeld – *Grundlagen zu Inversem Reinforcement Learning* und
- Stephan Zeisberg – *Knowledge Processing for Autonomous Robot Control.*

Während die ersten beiden Vorträge Teilprobleme oder Abwandlungen von RL betrachten, besteht der Bezug zum letzten Vortrag dahingehend, dass dieser eine Grundproblematik von autonomer Robotik thematisiert: Wie kann ein Agent autonom seine Umwelt interpretieren. Im folgenden sollen die Berührungspunkte dieser drei Vorträge zu dem hier vorgestellten Ansatz aufgezeigt werden.

3.1 Vortrag 1: Where do Rewards come from?

Dieser Vortrag behandelt eine eher theoretische Fragestellung die sich aus der Idee des RL ergibt. Der *Reward* oder *Return* ist der zentrale Bestandteil eines RL-Problems, ist es doch das Element, dass dem Agenten Rückmeldung über seine Aktionen in der Umgebung liefert.

In der vorgestellten Publikation wurde der Return von vornherein für bestimmte Zustände und Aktionen definiert und über dem Zustandsraum Approximiert. In diesem Vortrag wurde eine Untersuchung vorgestellt, die die allgemeine Natur dieser Returns genauer beleuchtet.

Für den CST-Algorithmus konkret liefern die Ergebnisse wenig relevante Information, da sie mehr RL als Modell für intelligentes Verhalten zu verbessern versuchen.

3.2 Vortrag 2: Grundlagen zu Inversem Reinforcement Learning

Dieser Vortrag betrachtet eine Abwandlung von RL, Inverses Reinforcement Learning. Inverses Reinforcement Learning hat mit LfD dahingehend einiges gemein, dass es Beispieltrajektorien zur erlernung von intelligentem Verhalten verwertet.

Während RL darin besteht anhand der Returns optimale Strategien für einen Agenten zu finden, liegt das Ziel von IRL darin, anhand von Beispielen die zugrundeliegende Wertefunktion zu »rekonstruieren«. Zu CST bestehen durchaus Ähnlichkeiten: Dieses verwendet Beispiele um die Bewertungsfunktion (durch die Hinzunahme und interpretation von Skills) zu verfeinern. IRL verwendet Beispiele um die Bewertungsfunktion zu erstellen.

3.3 Vortrag 3: Knowledge Processing for Autonomous Robot Control

Zu diesem Vortrag sehe ich Anknüpfungspunkte auf einer höheren Ebene. Er betrachtet die Fragestellung nach der selbstständigen Interpretation der Umgebung durch den Agenten. Dieser Bereich fällt nicht unter die Betrachtung von LfD oder dem vorgestellten CST-Algorithmus, spielt jedoch eine Rolle bei der Implementierung von Autonomen Robotern allgemein.

In den Experimenten zu CST wurden relevante Objekte in der Umwelt durch Farbmarkierungen oder Tags gekennzeichnet. Die in diesem Vortrag vorgestellte Technik arbeitet daraufhin, diese semantische Hintergrundinformation vom Agenten selbst zu identifizieren.

In meiner Vorstellung eines Autonomen Roboters könnten also beide Techniken wichtige Bausteine eines Gesamtsystems sein. CST steuert das Verhalten des Agenten (nach der Demonstration durch einen Lehrer) und das KNOWROB-System ermöglicht ihm die Aufgabe(n) ohne Labels oder Tags durchzuführen, und, möglicherweise, seine Fähigkeiten und Aktionen eher zu verallgemeinern.

4 Eigene Gedanken

Es wurde die Frage gestellt, nach „eigenen Gedanken zu zukünftiger Forschung auf dem Gebiet". Meine Gedanken beschränken sich auf das Gebiet RL in der Anwendung für mobile Robotik und mobile Robotik allgemein.

Aus der vorgestellten Publikation direkt ergeben sich zwei Bereiche die auch im Laufe dieser Ausarbeitung schon tangiert wurden. Zum einen verwendet das vorgestellte System zur Analyse seiner Umwelt Labels und Tags. Sollten autonome Roboter irgendwann in einer realen Umgebung eingesetzt werden ist das natürlich nicht praktikabel. Das Gesamtsystem müsste also um eine, die Umwelt »verstehende« Komponente erweitert werden. Ob hier nun wirklich die im vorherigen Abschnitt genannte Technik verwendet

werden kann wäre vielleicht eine Frage. Generell halte ich diesen Bereich aber für einen wichtigen Punkt.

Der andere Punkt betrifft die in 1.3 erläuterten Abstraktionen. Diese werden bisher von vornherein von einem Experten definiert und unveränderbar dem Agenten zur verfügung gestellt (Stichwort *Abstraction Library*). Diesen Umstand zu beenden, und dem Agenten auch zu ermöglichen eigene Abstraktionen zu erlernen halte ich für die zweite Anknüpfungsmöglichkeit für „eigene Forschung". Schließlich ist das ultimative Ziel der Autoren der Publikation „general-purpose robot learning from demonstration". Verwendet man eine »hart-kodierte« Abstraktionsbibliothek ist diese allgemeine Verwendbarkeit nur in Grenzen gegeben. Interessant wäre möglicherweise herauszufinden, wie existierende Feature-Selection Algorithmen mit CST kombiniert werden können, um solche Abstraktionen zu bestimmen.

Literatur

[1] G. Konidaris. George konidaris - autonomous robot skill acquisition. http://people.csail.mit.edu/gdk/arsa.html, 2012.

[2] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, December 2011.

[3] G. Konidaris, S. Osentoski, and P. S. Thomas. Value function approximation in reinforcement learning using the fourier basis. *Computer Science Department Faculty Publication Series*, page 101, 2008.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning : an introduction*. MIT Press, Cambridge, Mass. [u.a.], 2004.

[5] Marc Toussaint. Lectures on machine learning. June 2012.