Künstliche Intelligenz manuscript No.
(will be inserted by the editor)

Integration von Programmieren und Lernen in eine Steuerungssprache für autonome Roboter

Alexandra Kirsch

the date of receipt and acceptance should be inserted later

Zusammenfassung Durch den zunehmenden Einsatz von Robotern in Alltagsumgebungen wird die Fähigkeit sich durch Lernen an dynamische Umgebungen anzupassen immer wichtiger. Die *Robot Learning Language (RoLL)* schafft die Grundlage für eine nahtlose Integration von Lernen in Robotersteuerungsprogramme. Sie enthält Konstrukte zum Definieren und automatischen Sammeln von Erfahrungsdaten sowie zum Definieren und Ausführen von Lernproblemen. Durch ihre Modularität kann RoLL mit verschiedenen Lernalgorithmen verwendet werden.

1 Motivation

Ein interessantes Einsatzgebiet von Robotern sind Anwendungen im Haushalt, wo ein Roboter alltägliche Arbeiten verrichtet, oder zumindest eine Person bei solchen Aufgaben unterstützt. Ein schwieriger Aspekt dabei ist, dass Haushalte verschieden sind und damit nicht von vornherein jede Situation vorhersehbar ist, in die solch ein Roboter kommen kann. Selbst scheinbar einfache Entscheidungen können von der Umgebung abhängen und müssen daher autonom entschieden werden.

Der simulierte Haushaltsroboter in Abbildung 1 muss sich beispielsweise entscheiden, wo er sich hinstellen sollte, um Objekte auf dem Tisch zu greifen oder abzustellen. Schon das Vorhandensein des Stuhls vor dem Tisch stellt eine Hürde dar, die nicht in jeder Umgebung gleich ist. Ebenso muss je nach Situation entschieden werden, welchen Arm

Dr. Alexandra Kirsch
Institut für Informatik, Technische Universität München
Boltzmannstr. 3
D-85748 Garching
Tel.: +49 89 289 17779
Fax: +49 89 289 17757
E-Mail: kirsch@in.tum.de

Abb. 1 Simulation eines Haushaltsroboters.

der Roboter zum Greifen benutzt oder wo er Türen anfassen muss um sie zu öffnen.

Abgesehen davon, dass es sehr aufwendig wäre, spezielle Strategien für jede dieser Entscheidungen zu entwickeln, sollte sich ein Roboter eigenständig an seine Umgebung anpassen, wofür sich maschinelles Lernen anbietet. Statistische Lernverfahren stellen gut erforschte Methoden zur Verfügung. Egal ob man neuronale Netze, Support Vector Machines oder Entscheidungsbäume verwendet, die Ergebnisse liefern in meist zuverlässige Vorhersagen, sofern genügend Daten in angemessener Qualität vorliegen.

Was bisher weniger betrachtet wurde, ist das Einbinden von Lernen in das Gesamtverhalten eines Roboters. In der vorgestellten Arbeit wird eine Robotersteuerungssprache so erweitert, dass ein Roboter während seines normalen Betriebes Erfahrungen sammelt, die er später in einem automatischen Prozess lernt und als fertige Funktion wieder in sein Steuerungsprogramm einbettet, um sie für Entscheidungen zu benutzen.

Diese Einbettung von Lernen in das Roboterverhalten hat mehrere Vorteile. Zum Einen kann sich die Umgebung eines Roboters oder der Roboter selbst ändern, zum Beispiel wenn sich Gelenke des Roboters abnutzen und mit der Zeit anders reagieren. In diesem Fall kann der komplette Lernprozess automatisch wiederholt werden, inklusive dem

Beobachten neuer Daten und dem Ausführen des Lernvorgangs. Die Funktionen, die für die Entscheidungen verwendet werden, werden dabei durch die neu gelernten ersetzt.

Weiterhin ist es vorteilhaft, wenn ein Roboter während seines normalen Betriebs Erfahrungen sammeln kann, anstatt spezielle Skripte zum Datensammeln auszuführen. Damit bekommt man die Daten nicht nur effizienter, da keine Zeit mit nicht zielführenden Aktionen vergeudet wird, sondern man sammelt die Erfahrungen auch genau in den Situationen, in die der Roboter in seinem normalen Betrieb überhaupt kommt.

Ein weiterer Vorteil ist die explizite Darstellung der verwendeten Zustandsraumabstraktion und der Parameter eines Lernalgorithmus. Damit sind alle Parameter explizit im Programm repräsentiert und sie können entsprechend konfiguriert werden. Man kann für jedes Problem sehr bequem ausprobieren, welcher Lernalgorithmus am geeignetsten ist. Im Prinzip ließe sich dieses Verfahren auch automatisieren.

Im Folgenden gebe ich eine kurze Zusammenfassung der Robot Learning Language (RoLL), die ich in meiner Dissertation [2] entwickelt habe. Die Arbeit beschreibt die theoretischen Grundlagen zur Repräsentation von Erfahrungen in Form von hybriden Automaten und die praktische Umsetzung in Entwurf und Implementierung der Programmiersprache RoLL.

2 Robot Learning Language

Die Robot Learning Language (RoLL) ist eine Erweiterung der Plansprache RPL (Reactive Plan Language) [5], die auf Lisp basiert. Ein besonderes Augenmerk lag auf der Lösung von folgenden Problemen:

1. Wie können Erfahrungen deklarativ beschrieben werden, sodass ein unabhängiger Beobachtungsprozess die Daten sammeln kann ohne dass der Code für die eigentliche Funktionalität des Roboters verändert werden muss?
2. Wie kann man gelernte Funktionen automatisch in das Steuerungsprogramm einbauen?
3. Wie können verschiedene Modi zum Lernen verwendet werden? Modi umfassen online und offline Lernen ebenso wie das aktive und passive Datensammeln.

Bei dem gewählten Lösungsansatz ist der Lernprozess in zwei grobe Schritte geteilt: das Sammeln von Erfahrungen und das Lernen (s. Abb. 2). Die Erfahrungen werden vom Programmierer deklarativ beschrieben. Daraus wird Code generiert, der in einem parallelen Prozess zu irgendeinem Steuerungsprogramm laufen kann und automatisch die gewünschten Daten herausschreibt. Auch für den Lernvorgang an sich gibt es ein deklaratives Konstrukt, das die zu lernende Funktion und die Lernparameter beschreibt. Daraus wird eine Funktion generiert, die ebenfalls an einer beliebigen Stelle des Programms aufgerufen werden kann.

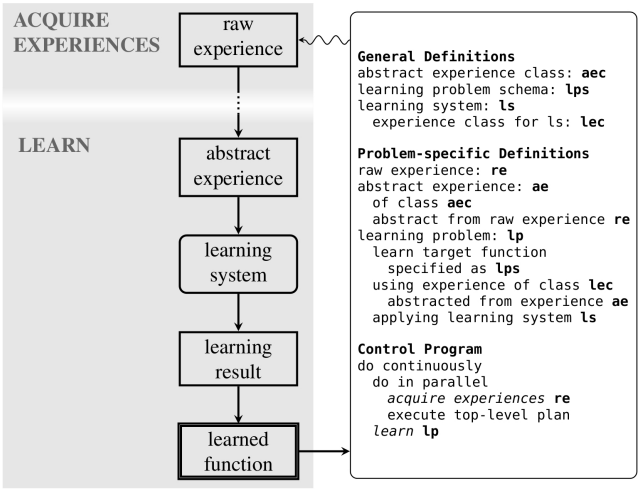

Abb. 2 Allgemeiner Lernvorgang mit RoLL.

Die deklarative Beschreibung führt zu einer kompakten Darstellung aller Parameter des Lernvorgangs, inklusive der beobachteten Daten. Damit lässt sich genau rekonstruieren, wie eine Funktion gelernt wurde. Außerdem bietet die deklarative Beschreibung eine Grundlage für eine weitere Automatisierung des Lernprozesses.

Durch die Einteilung in Datensammlung und Lernen lassen sich verschiedene Lernmodi umsetzen. Man kann beispielsweise einen Roboter so programmieren, dass er während des Tages seine normalen Tätigkeiten ausführt und nebenbei Erfahrungen sammelt und aus diesen Erfahrungen nachts seine Vorhersagemodelle und Entscheidungsfunktionen neu lernt. Wenn man die Erfahrungssammlung und das Lernen im Code näher zusammenrückt, bekommt man einen stärkeren online Charakter, was auch für Reinforcement Learning genutzt werden kann. Oder sollte ein Roboter feststellen, dass eines seiner Modelle gar nicht mehr funktioniert, könnte er statt seines normalen Programmes ein spezielles Skript zum Datensammeln ablaufen lassen, um in kurzer Zeit eine größere Menge von Lerndaten zu erzeugen.

RoLL macht keine Vorgaben zum verwendeten Lernalgorithmus. Lernalgorithmen können nah Bedarf modular als `learning system` ergänzt werden, wofür man vorhandene Bibliotheken verwenden kann. In meiner Dissertation habe ich mit neuronalen Netzen (mit dem Stuttgart Neural Network Simulator, SNNS) und Entscheidungsbäumen (mit WEKA) gearbeitet. Mittlerweile ist auch eine Anbindung für Support Vector Machines (mit libSVM) implementiert.

2.1 Erfahrungssammlung

Wenn ein Roboter viele verschiedene Dinge lernen soll, wäre es extrem unübersichtlich, den Code des Steuerungsprogramms jedes Mal entsprechend umzuschreiben um an der passenden Stelle im Programm die gewünschten Daten herauszuschreiben. Andererseits könnte man alle Daten im Programm

speichern, um sie später zu analysieren. Abgesehen von den großen Datenmengen, die man damit erzeugen würde, wäre es auch aufwendig den Kontext wiederherzustellen, in dem die Daten beobachtet wurden.

In RoLL werden Erfahrungen zusammen mit dem Kontext definiert, in dem sie beobachtet werden sollen, sodass genau die Daten gespeichert werden, die auch gebraucht werden. Die Erfahrungsdefinition (`raw experience`in Abbildung 2) basiert auf dem Konzept von hybriden Automaten, wobei hybrid hier für die Kombination aus kontinuierlichen Vorgängen und diskreten Zustandsänderungen steht. Das Robotersteuerungsprogramm kann als großer hierarchischer hybrider Automat betrachtet werden, in dem jeder Plan (in RPL ist jedes Programm ein Plan) einem Zustand entspricht, in dem sich die Welt kontinuierlich ändert. Durch hierarchische Schachtelung von Plänen bzw. entsprechend Automatenzuständen kann man das komplette Programm in verschiedenen Granularitäten darstellen.

Außer dem Roboterprogramm kann man auch Vorgänge in der Umgebung als hybride Automaten betrachten. Ein Zustand könnte sein, dass ein Mensch in der Küche ist. Während dieser Zustand aktiv ist, ändert sich die Welt kontinuierlich oder wenn man an Details interessiert ist, kann man Unterzustände definieren, wie beispielsweise ob die Person gerade ein Objekt in der Hand hat.

Zum Definieren einer Erfahrung gibt man an, welcher Teil des imaginären Programm- bzw. Weltzustandsautomaten von Interesse ist, z.B. alle Situationen, in der der Roboter seinen Navigationsplan aktiviert und sich eine Person in der Küche befindet. Zusätzlich spezifiziert man die Daten, die bei so einem Ereignis beobachtet werden sollen. Dabei wird unterschieden in Daten, die man einmalig am Anfang des interessanten Zustands oder an dessen Ende aufzeichnet und Daten, die kontinuierlich aufgezeichnet werden, solange die Bedingung erfüllt ist. Die Daten können sowohl aus globalen als auch aus lokalen Programmvariablen stammen.

Durch das allgemeine Konzept von hierarchischen hybriden Automaten können sehr komplexe Erfahrungen spezifiziert werden, die auch Unterziele mit einbeziehen. Beispielsweise könnte man zählen, wie oft der Roboter seine Greifer schließt, bis es ihm gelingt eine Tasse zu greifen (d.h. wie viele Fehlversuche er braucht).

Doch mit dem Beobachten von Variablenwerten zum richtigen Zeitpunkt ist die Arbeit noch nicht getan. Um erfolgreich zu Lernen müssen die Daten in den meisten Fällen in einen abstrakten Zustandsraum überführt werden. Beispielsweise sollte man relative Koordinaten verwenden, um den Weg eines Roboters von einem Punkt zu einem anderen zu beschreiben anstatt mit absoluten Positionen zu lernen. Dadurch bekommt man schneller die benötigte Anzahl an Datenpunkten und der Lernalgorithmus kann besser verallgemeinern.

Abstrahierte Daten können in verschiedenen Formen gespeichert werden. RoLL bietet die Möglichkeit eigene Speichervarianten einzubauen. Standardmäßig wird eine Datenbank für die permanente Speicherung der Daten verwendet.

2.2 Lernen

Die Abstraktion der Daten kann in mehreren Schritten erfolgen, sodass man Daten auf verschiedenen Abstraktionsstufen permanent speichern und gegebenenfalls für verschiedene Lernprobleme verwenden kann. Am Ende dieser Folge muss jedoch immer eine abstrakte Erfahrung (`abstract experience` in Abbildung 2) definiert sein, die genau die Daten für das zu lernende Problem in einer sinnvollen abstrakten Beschreibung enthält.

Der Lernvorgang wird ebenfalls deklarativ beschrieben. Ein Lernproblem besteht aus der zu verwendenden Erfahrung, dem zu verwendenden Lernsystem (d.h. dem Lernalgorithmus und seiner Parameter) und einer Beschreibung der Funktion, die gelernt werden soll. RoLL lässt verschiedene Funktionstypen zu: Modelle, die speziell abgespeichert werden um beispielsweise Fehler bei der Planausführung automatisiert zu finden, (Lisp-)Funktionen, (Lisp-)Methoden und Routinen, bei denen die gelernte Funktion in eine Schleife eingebaut wird, um sie zur Steuerung zu verwenden.

Zu einer gelernten Funktion wird eine Quelldatei erzeugt, die entweder eine eigenständige Funktion enthält oder eine Wrapper-Funktion, die beispielsweise mit einem “Foreign Function Call” die gelernte Funktionalität herbeiführt. Diese Datei wird sofort nach dem Lernen geladen, sodass sie gleich verwendet werden kann. Andererseits dient sie als permanente Speicherung der gelernten Funktion, die jedes Mal mit dem Gesamtprogramm geladen werden kann.

2.3 Sprachumfang

Die Kernfunktionalität von RoLL wird durch drei deklarative Konstrukte zur Verfügung gestellt: `define-raw-experience` und `define-abstract-experience` zum Sammeln und Abstrahieren von Lerndaten und `define-learning-problem` für die Spezifikation des Lernproblems. Aus diesen deklarativen Angaben werden zwei prozedurale Konstrukte generiert, die vom Programmierer flexibel im Programm eingebaut werden können: `acquire-experiences`, parametrisiert mit einer Erfahrungsklasse, kann in einem parallelen Prozess zu einem beliebigen Programmteil eingebaut werden und zeichnet automatisch die spezifizierten Daten auf. Die Funktion `learn`, parametrisiert mit dem Namen des Lernproblems, führt den Lernvorgang aus, wobei Datenabstraktion, Aufruf des Lernalgorithmus und Einbettung der

gelernten Funktion in das Steuerungsprogramm automatisch durchgeführt werden.

Außerdem bietet RoLL Hilfestellung für das gezielte Datensammeln an. Das Konstrukt `with-problem-parameters` erlaubt es eine Menge von Variablen zu definieren und nach vorgegebenen Regeln mit verschiedenen Werten zu belegen. Will man beispielsweise lernen, an welchen Positionen auf einem Tisch sich eine Tasse greifen lässt, kann man die Fläche auf dem Tisch diskretisieren und den Roboter die Tasse an jeder so definierten Gridposition greifen lassen. Alternativ kann man zufällige Samples aus dem Zustandsraum ziehen oder systematische mit zufälliger Datensammlung kombinieren.

Schließlich bietet RoLL noch Möglichkeiten zur Anpassung und Erweiterung:

- mit `define-abstract-experience-class` können verschiedene Möglichkeiten der Speicherung von Erfahrungen, z.B. in verschiedenen Dateiformaten oder einer Datenbank definiert werden;
- `define-learning-problem-class` bietet die Möglichkeit weitere Arten von Funktionen zu definieren, die gelernt werden sollen;
- zum Definieren eines neuen Lernsystems muss eine entsprechende abstrakte Erfahrungsklasse definiert werden (entsprechend dem Datenformat, das der Lernalgorithmus verwendet) und zwei Methoden zum Ausführen des Lernen und zum Generieren der gelernten Funktion spezifiziert werden.

3 Anwendungen und Ausblick

RoLL wurde an mehreren Lernproblemen getestet [3], zum Beispiel zum Lernen einer Navigationsroutine, für die Vorhersage der Zeitdauer für eine Navigationsaktion und für die Entscheidung welche Hand der Roboter in einer Situation zum Greifen benutzen sollte. Das volle Potential von RoLL liegt allerdings in komplexeren Systemen mit vielen weiteren Lernproblemen. Wir sind derzeit dabei weitere Lernprobleme zu implementieren, sodass ein Roboter von Grund auf Modelle für die Armsteuerung und die Navigation lernt und diese für verschiedene Entscheidungsprobleme verwendet.

Bei den Experimenten mit RoLL hat sich gezeigt, dass es äußerst schwierig ist, Steuerungsentscheidungen direkt zu Lernen, z.B. welcher Navigationsbefehl in einer Situation gegeben werden muss, um an einen gegebenen Zielpunkt zu fahren [4]. Ein Problem liegt darin, dass man zur Lernzeit alle relevanten Informationen des Zustands kennen und das Performanzmaß festlegen muss. Dies bedeutet, dass man für die Navigation Wände und andere Hindernisse im Lernproblem modellieren müsste, was allerdings den Zustandsraum so groß macht, dass mit einer Datenmenge in realistischem Umfang keine Verallgemeinerung möglich ist. Und es wäre durch das Lernen festgelegt, ob ein Roboter sein Ziel zum Beispiel schnell oder sehr genau erreichen soll, was jedoch oft eher durch die Anwendungssituation bestimmt wird. Außerdem ist die Auswahl der Lerndaten extrem schwierig, da man viele Daten sammelt, die das Ziel zwar erreichen, aber nach dem festgelegten Performanzmaß nicht gut sind. Zum Beispiel bei der Navigation gibt es unendlich viele Möglichkeiten einen Zielpunkt zu erreichen, aber nur wenige davon sind effizient.

Daher konzentrieren wir uns aktuell darauf Vorhersagemodelle zu lernen und diese zur Laufzeit für eine einfache Suche zu verwenden. Bei der Entscheidung, welche Hand ein Roboter verwenden soll, ist dies noch sehr einfach, solange er nur zwei Arme hat. Die Entscheidung, wo sich der Roboter positioniert um ein Objekt zu greifen, ist dagegen mit größerem Suchaufwand verbunden.

RoLL wurde komplett überarbeitet und steht jetzt als open source ROS Stack zur Verfügung[1]. Dabei wurde das zugrundeliegende RPL durch die neue CRAM Plan Language[2] [1] ersetzt.

Literatur

1. Michael Beetz, Lorenz Mösenlechner, and Moritz Tenorth. CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments. In *IEEE/RSJ International Conference on Intelligent RObots and Systems.*, 2010.
2. Alexandra Kirsch. *Integration of Programming and Learning in a Control Language for Autonomous Robots Performing Everyday Activities*. PhD thesis, Technische Universität München, 2008.
3. Alexandra Kirsch. Robot learning language — integrating programming and learning for cognitive systems. *Robotics and Autonomous Systems Journal*, 57(9):943–954, 2009.
4. Alexandra Kirsch, Michael Schweitzer, and Michael Beetz. Making robot learning controllable: A case study in robot navigation. In *Proceedings of the ICAPS Workshop on Plan Execution: A Reality Check*, 2005.
5. Drew McDermott. A reactive plan language. Technical report, Yale University, Computer Science Dept., 1993.

Alexandra Kirsch hat 2008 ihre Promotion abgeschlossen und leitet jetzt die Nachwuchsgruppe "Human-Centered Artificial Intelligence" an der TU München. Forschungsschwerpunkte sind planbasierte Aktionsausführung, Roboterlernen, Mensch-Roboter Kooperation, autonome Entscheidungsfindung und wissensbasierte Fehlererkennung.

[1] `http://www.ros.org/wiki/roll`
[2] `http://www.ros.org/wiki/cram_pl`