Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik

Jan Lukoschus
Martin Steffen

Programmieren von Lego-Mindstorms

Tag der offenen Tür

16. Mai 2003

Der Zettel soll ein sehr einfaches Beispiel vorstellen, und zwar den *Bumper*, der am Stand vorhanden ist, und ermutigen (mit Hilfe des "Standpersonals"), sich an ein selbstgeschriebenes Programm zu wagen um zu versuchen ihn selbst zu realisieren.

Daneben versucht der Zettel die Frage zu beantworten (oder zumindest zu stellen, siehe Abschnitt 2):

> Der Roboter, der auf der Fläche hin und her fährt und Klötzchen schiebt, sieht ja ganz nett aus, aber
>
> - was ist daran komplex?
> - Was muß man bei der Realisierung bedenken

1 Ein einfaches Beispiel: der "Bumper"

In der 2ten Hälfte das Vortrags wurde ein Beispielprogramm entwickelt, das das Verhalten eines sehr einfachen mobilen Roboters beschreibt, den *"Bumper"*. Sein Verhalten ist wie folgt:

> Er fährt (1) *geradeaus* bis er auf ein Hindernis trifft. Je nachdem, ob er das Hindernis *links* oder *rechts* touchiert, weicht er (2) *rechts* bzw. (3) *links* aus und fährt danach wieder *vorwärts.* Das Ausweichen besteht aus *Rücksetzen* und *Drehen.*

In dem *Esterel/Estudio*-Werkzeug, welches wir verwenden, sieht das Verhalten so aus:

Abgesehen vom Anfangszustand (`I` für initial) den man immer braucht, damit man weiß wo es losgeht, erkennt man in Abbildung 1 die drei oben erwähnten Zustände.

Das *Ausweichen* ist aus mehreren Einzelaktionen zusammengesetzt, was man in Abbildung 1 daran erkennt, das `Ausweichen-L` und `Ausweichen-R` als große, viereckige Kästen dargestellt sind. Anders der Zustand fürs Vorwärtsfahren `V`, der nicht zusammengesetzt ist.

Das Nach-Links-Ausweichen (also das "Innere" des Kastens `Ausweichen-L`) ist in Abbildung 2 gezeigt. Es wird die Motoren der Reihe nach auf Stopp, Rückwärts, und Linksdrehen geschaltet. Beachte die Zeitverzögerungen zwischen den Umschaltungen!

Das gesamte Programm ist ist Abbildung 3 dargestellt, wobei anders als in Abbildung 1 das Innere von der Ausweich-Zustände gezeigt ist.

Man hätte das gleiche Verhalten auch ohne *Hierarchisierung*, d.h. ohne zusammengesetzte Zustände, programmieren können. Eine mögliche Realisierung zeigt Abbildung 4. Für sehr große Systeme mit Tausenden von Zuständen wird eine derartige Darstellung unüberschaubar.

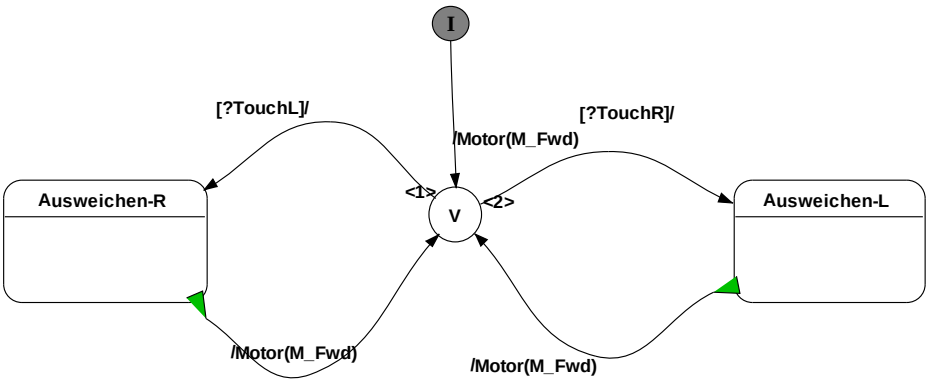

Abbildung 1: Bumper: Grobsicht

2 Der Klötzchensammler

Der *Klötzchensammler,* der auf unserem Stand seine Runden dreht, ist bereits bedeutend komplexer und zwar aus mehrerlei Gründen. Auf Details seiner Umsetzung einzugehen, sprengt den Rahmen dieser Vorführung, jedoch sind die *Gründe, warum* er schwieriger als der *Bumper* interessant, denn man kann an ihnen exemplarisch einige Probleme von Systemen die mit realer Umwelt interagieren, erkennen.

Zunächst zu seiner *Aufgabe:*

> Eine willkürlich auf einer umgrenzten Fläche verteilte Ansammlung von *Klötzchen* soll ins *Tor* befördert werden. Eine *Leuchte* zeigt die Richtung zum Tor an. Während einer vorgegebenen Zeitspanne sollen soviele Tore wie möglich erziehlt werden. Danebenschießen ergibt Strafpunkte.

Dies entspricht grob einer Aufgabe eines Praktikums im Hauptstudium. Die Realisierung (Strategie, Aufbau des Roboters, Auswahl der Sensoren . . .) war den Teilnehmern freigestellt.

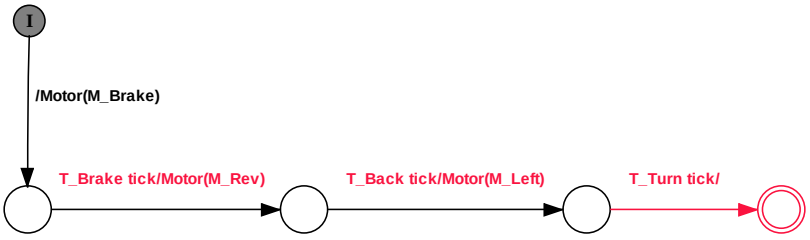

Abbildung 2: Ausweichen (links)

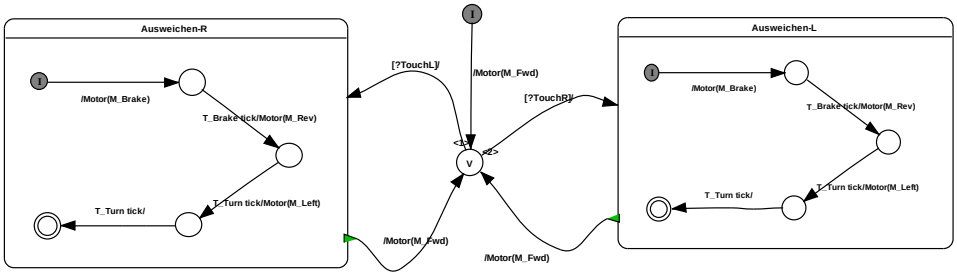

Abbildung 3: Bumper (gesamt)

Der Roboter hier ist eine Lösung aus einem breiteren Spektrum. Nun zu den *Gründen,* warum die Aufgabe bzw. der gezeigte Roboter und seine Software komplexer als der Bumper ist.

Mechanik Der Bumper tut seine Aufgabe selbst wenn er schlampig gebaut ist die Motoren ungleich laufen, die Zeiten nicht ganz stimmen, etc. Der Klötzchensammler schießt neben das Tor wenn Software, seine Zeiten und Hardware (Größe der Räder ...), die Zeiten nicht aufeinander abgestimmt werden.

Frage: wie geht man damit um, daß die Batterie im Laufe des Spiels schwächer wird und dann eventuell der Roboter sich langsamer dreht?

Sensorik Die "Sinne" der Lego-Roboter sind sehr eingeschränkt! Zum einen hat er nur *drei* Sinne (= drei Eingänge) mit denen man auskommen muß, zum anderen ist der Lichtsensor alles andere als ein Auge, er kann ein wenig hell und dunkel unterscheiden. Die Unterscheidung ist unpräzise (wie "hell" ist hell?) und schwankend. Die Drucksensoren des Bumpers sind im Vergleich dazu ziemlich einfach (es gibt nur "gedrückt" oder "nicht gedrückt") und das auch einigermaßen zuverlässig.

Programm Die Aufgabe als solches und dieses Programm ist schwieriger und zwar in mehrfacher Hinsicht:

- Strategie: Der Roboter macht zu verschiedenen Zeiten verschiedene Dinge.
- Gleichzeitigkeit: der Roboter macht mehrere Dinge gleichzeitig ("parallel")
- Umwelt"wissen": das Programm merkt sich (in, zugegebenermaßen, einfachster Form) gewisse Dinge über seine Umwelt, zum Beispiel, wo er den letzten Klotz fand sodaß er dorthin nach einem "Tor" (oder Torversuch ...) zurückkehren und mit dem Abgrasen fortfahren kann. Positionen werden über die Zeit geschätzt. Das ist nicht viel, vor allem: Wenn er sich einmal *verfährt,* kommt er wieder "in die Spur"? Merkt er es überhaupt daß er sich verfahren hat oder spult er sein Programm stupide runter?

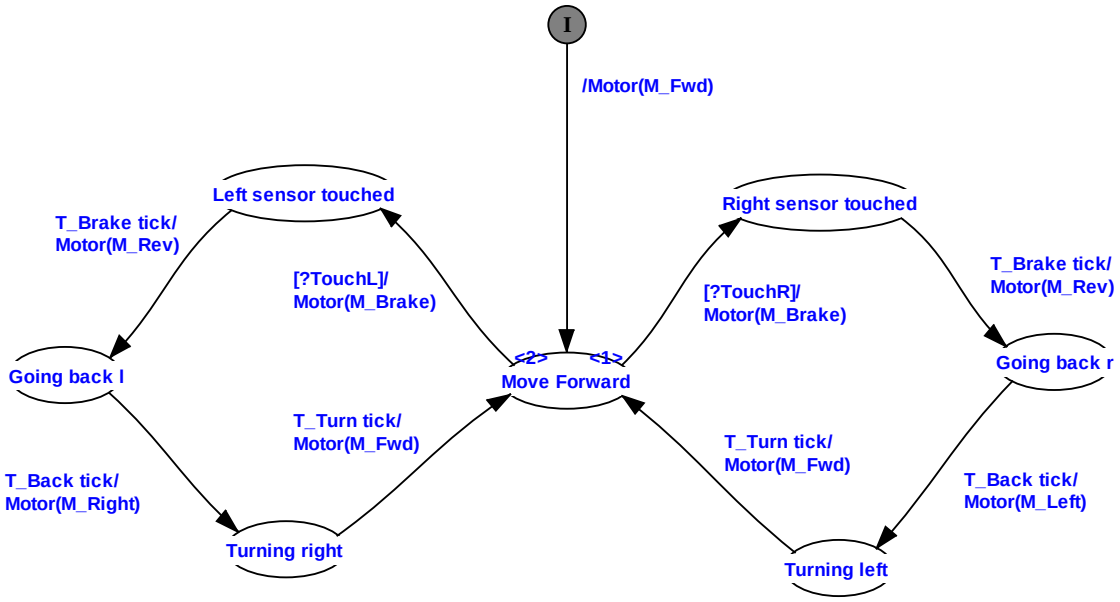

Abbildung 4: Bumper (flach)

Zu komplex darf das Programm auch nicht werden, da die Speicherkapazität des Legochips recht begrentzt ist. Der vorgeführte Roboter ist bereits am Limit[1]

Zeit *Zeit* spielt eine kritischere Rolle. Der Bumper ist wesentlich *robuster* auf ein paar Prozente schneller oder langsamerer Reaktion. Beim Klötzchensammler machen die Prozente den Unterschied zwischen Tor und kein-Tor (Minuspunkte) aus.

Umwelt Es ist beim Klötzchensammler schwieriger, alle *Umwelteinflüsse* zu kontrollieren und gegebenenfalls zu kompensieren.

- Kann der so gut wie blinde Lichtsensor das Tor noch finden wenn die Sonne scheint (z.B. am Tag der offenen Tür `:-)`)? Oder wenn man eine Leuchtstoffröhre nimmt? Wenn jemand mit einem Photoapparat blitzt?
- arbeitet der Roboter noch akzeptabel wenn er nicht auf einem Teppich fährt sondern auf Linoleum? (er hat weniger Griff, also beschleunigt er weniger? aber dafür rutschen die Klötzchen leichter, er fährt damit weiter bei gleicher Zeit? vielleicht fallen Klötzchen aber auch weniger leicht um und rollen deswegen nicht? Und hat man überhaupt eine Chance, den Roboter den Unterschied merken zu lassen, vor allem wenn der Speicherplatz bereits sogut wie mit den anderen Aufgaben aufgefüllt ist. Man könnte andere Programmteile streichen (welche?), oder eventuell ein ganz anderes Design wählen uswuswusw.

Vorgehen im Sinne von gutem Design und systematischen, methodischen Vorgehen: welche Chance hat man, sich über derartige Dinge *vor der Realisation* ein klares Bild zu verschaffen und nicht solange "rumzuprobieren" bis es geht?

[1]Das liegt aber nur zum Teil an der Komplexität der Strategie. Platz braucht auch der Programmierluxus eines eigenen *Betriebssystems* und der fortschrittlichen Art der Programmierung.

3 Verwendete Software

Die Plattform, auf der die Programme laufen, ist BrickOS (auch bekannt als LegOS[2] ("Lego operating system")), ein kostenloses und freies Betriebssystem für RCX-e, welches in in Forschung und Lehre eingesetzt wird. Es ist unter

`http://brickos.sourceforge.net`

erhältlich und kann unter Linux, Unix und Windows verwendet werden.

Bei Lego mitgeliefert wird *RCX-Code*, welches eher Laien als Zielgruppe hat und mit dem man einfache Programme leicht realisieren kann, aber kaum anspruchsvollere Aufgaben. Es kann unter Windows und MacOS verwendet werden.

Das graphische Design-Werkzeug, mit Hilfe dessen wir auch die Abbildungen dieses Zettels erzeugt haben, ist *Estudio/Esterel*, eine Entwicklungsumgebung und eine Sprache, die in Forschung, Industrie, und Lehre Anwendung findet. Es ist *nicht* speziell für Lego konzipiert sondern für professionelle Systeme. Es ist für Forschung und Lehre kostenfrei. Was den Umfang dieses (oder vergleichbarer) Werkzeuge betrifft, konnten wir nur einen verschwindend kleinen Ausschnitt vorführen.

Mögliche Befehle

Die Tabelle stellt Befehle zur Steuerung der Motoren und Ablesen der Berührungssensoren zusammen.

Motorsteuerung	
`Motor(M_Fwd)`	Vorwärtsgang
`Motor(M_Rev)`	Rückwärtsgang
`Motor(M_Brake)`	Stopp
`Motor(M_Stop)`	Motor aus
`Motor(M_Left)`	rechts drehen
`Motor(M_Right)`	links drehen
Berührungssensoren	
`?TouchR`	rechter Sensor gedrückt?
`?TouchL`	linker Sensor gedrückt?

Literatur

[1] Gerard Berry. *The Esterel v5 Language Primer (Version v5_.91).* Centre de Mathématiques Appliquées Ecole Mineur and INRIA, July 2000.

[2] Esterel Technologies. *Esterel Studio V3.1, Reference Manual*, October 2001.

[3] Jonathan B. Knudsen. *The Unofficial Guide to LEGO Mindstorms Robots.* O'Reilly, first edition, October 1999.

[4] Hermann Kopetz. *Design Principles for Distributed Embedded Application.* Kluwer Academic Publishers, 1997.

[5] Marcus L. Noga, 2002. http://brickos.sourceforge.net/HOWTO.

[6] Reinhard von Hanxleden. Real-time systems programming. Lecture, Summer Term 2002, 2002. available at `www.informatik.uni-kiel.de/inf/von-Hanxleden/teaching`.

[2] Wegen Lizenzquerelen wurde der urspüngliche Name "LegOS" nicht mehr verwendet.