

Roboter im Informatikunterricht

Matthias Stolt

3. Mai 2001

Zusammenfassung

Seit den späten 1970'er Jahren wird am Massachusetts Institute of Technology (MIT) in der 'Epistemology and Learning Group' über die Zusammenhänge zwischen der Lernumgebung und den erlernten Fähigkeiten geforscht. Die ersten Schritte in die Richtung des Einsatzes von Computern in der Lernwelt unternahm Seymour Papert in seinem Buch „Mindstorms: Children Computers and Powerful Ideas in dem er unter anderem die Programmierumgebung Logo vorstellt. Aus dieser Forschungsarbeit ist ein Roboterlabor hervorgegangen und in der 'Lifelong Kindergarten group' am 'MIT Media Lab' wurde der 'programmable brick' entwickelt, ein Baustein zur Steuerung von Autonomen Mobilien Robotern. Den vorläufigen Abschluß dieser Entwicklungslinie bildet das kommerzielle Produkt LEGO Mindstorm der Firma LEGO.

Das Ziel dieser Arbeit ist es, den grundlegenden Aufbau eines Roboterlabors zu beschreiben. Dies umfaßt die notwendige Hardware, die Software und Hinweise für die Durchführung des Labors. Es werden aber auch Ansätze zur Begründung für den Einsatz eines Roboterlabors gegeben.

Inhaltsverzeichnis

1	Einleitung	5
2	Motivation	5
2.1	pädagogisch/didaktisch	5
2.1.1	Soziale Fähigkeiten	6
2.1.2	Alternative zu ‘telling and testing’	6
2.1.3	‘single answer questions’	6
2.1.4	Komplexität der Welt	6
2.1.5	learn how to design not about design	7
2.2	behandelte Informatikthemen	7
2.2.1	Imperative Programmierung	7
2.2.2	Strukturierte Analyse/Strukturierte Programmierung	8
2.2.3	Parallele Abläufe	8
2.2.4	Meß-, Steuer- und Regelungstechnik	9
2.2.5	Echtzeitbedingungen	9
2.2.6	behavior based / subsumption architecture	9
2.2.7	reaktive Systeme	10
2.2.8	Künstliche Intelligenz und Bildverarbeitung	10
2.2.9	Braitenberg Vehicle	10
2.2.10	Multiagenten Systeme	11
2.3	Zusammenfassung	11
3	Altersgruppen	12
3.1	Sekundarstufe I (5.–10. Schuljahr)	12
3.2	Sekundarstufe II (11.–13. Schuljahr)	12
3.3	Studium	12
4	Laborumgebung	14
4.1	Steuerrechner	14
4.2	Chassis	16
4.3	Zusammenfassung	17
5	Programmierung	18
5.1	Visuelle Programmierung	18
5.1.1	Mindstorm RIS	19
5.1.2	Robolab	20
5.2	Textuelle Programmierung	22
5.2.1	NQC	22
5.2.2	LegOS	24
5.3	Tabellarischer Vergleich	25
5.4	Zusammenfassung	25

6	Durchführung	28
6.1	generelle Hinweise	28
6.1.1	Spielregeln	28
6.1.2	Zwischenwettbewerbe	29
6.1.3	Knowledgebase	30
6.2	exemplarischer Wettbewerb	30
6.2.1	einfaches Rennspiel auf Linienfolgerbasis	31
6.2.2	Variation mit Kollision	33
6.2.3	Variation mit Kooperation	34
6.3	Informatikkonzepte in den exemplarischen Wettbewerben	35
6.3.1	Inhalte der einzelnen Wettbewerbe	35
6.3.2	Inhalte des Ensembles der Wettbewerbe	36
7	Schlußbemerkung	38
A	Vergleich der Steuerrechner	39
B	Überblick Programmierkonstrukte	40
C	Fallen und Fallstricke	41
D	Codebeispiele	42
E	Standard Softwarearchitektur des LEGO Mindstorm	45
F	Installationshinweise NQC/RCXCC	47
G	Internetquellen	48
	Literatur	49

Tabellenverzeichnis

1	Vergleich der Programmierumgebungen	27
2	Anschlußvergleich	39
3	Überblick Programmierkonstrukte, Anwendungsbereiche	40

Abbildungsverzeichnis

1	Elemente der Robotprogrammierung	14
2	Handyboard	15
3	Fischertechnik, Intelligent Interface	15
4	LEGO Mindstorm RCX	16
5	Arbeitsoberfläche vom RIS	19
6	Beispielregelseite eines Wettbewerbs	29

7	Spielfeld für den Linienfolgerwettbewerb	31
8	Endpunktmarkierung mit Lichtsensor	32
9	Aufsicht auf die Endpunktmarkierung	32
10	Zwei Roboter bei der Kollision	33
11	Zwei Roboter bei der Übergabe	34
12	Softwarearchitektur des RCX mit Standardfirmware	45

1 Einleitung

Diese Arbeit untersucht die Möglichkeiten, mit einem Roboterlabor eine alternative Informatikausbildung an Schulen zu bieten. Mit dem Hintergrund des zur Zeit herrschenden Fachkräftemangels in der Informatik bekommt sie einen sehr aktuellen Bezug. Obgleich die Grundlagen für die Begründung eines Roboterlabors schon in den späten 1970`er Jahren gelegt wurden und auch ohne die aktuelle Problematik der Förderung zur Informatikausbildung sehr starke Argumente für diese Form der Lernumgebung vorliegen [Papert1980].

In dieser Arbeit wird zunächst eine Einführung in die Motivationen zum Einsatz von Roboterlaboren und die möglichen Informatikthemen gegeben. Anschliessend wird die Laborumgebung an sich erläutert. Dies umfaßt sowohl die Hardware des Roboters, als auch die Entwicklungsumgebungen. Abschliessend werden einige Hinweise zur Durchführung des Roboterlabores und eine Beispielaufgabe mit Variationen gegeben. Hierbei wird auch immer ein Blick auf die umsetzbaren Informatikkonzepte geworfen.

Noch eine Anmerkung zu den in dieser Arbeit zitierten Internetquellen. Das WorldWideWeb ist ein sehr lebendiges Medium und gerade zu aktuellen Themen finden sich dort sehr interessante Quellen. Leider gibt es keine Garantie dafür, daß ein heute existente Quelle auch später unter der gleichen Adresse auffindbar ist. Insofern stellt sich das Problem, ob solche Quellen überhaupt angegeben werden sollten. Ich habe mich in dieser Arbeit entschieden die reichhaltigen Quellen des Internet anzugeben, da dort sehr viel gutes Material zu finden ist. Alle Verweise ins WorldWideWeb waren zum Zeitpunkt des Druckens dieser Arbeit aktuell und erreichbar¹.

2 Motivation

Die Motivationen für die Beschäftigung mit Robotern im Unterricht lassen sich in zwei Gruppen gliedern. Zum Einen die pädagogisch/didaktischen [Martin1994] für die der Roboterbau eigentlich nur ein Mittel zum Zweck ist und zum Anderen die Möglichkeiten im Roboterbau und vor allem in der Roboterprogrammierung einige Informatikthemen zu behandeln, die in normalen Aufgabenstellungen so nicht behandelt werden.

2.1 pädagogisch/didaktisch

Ein großer Punkt, der für das Roboterlabor spricht, ist das starke Motivationspotential für die Teilnehmer sich mit Informatikthemenstellungen zu befassen. Die Entwicklung einer eigenen Lösung einer Aufgabe, mit anschliessendem Wettbewerb besitzt trotz - oder vielleicht gerade wegen - der

¹Unter der Adresse <http://www.stolt.de/lego> befindet sich eine aktuelle Linkliste der Verweise aus dieser Arbeit.

möglichen Komplexität einen sehr großen Unterhaltungswert und stellt eine spannende Herausforderung dar. Pädagogisch ist das Roboterlabor durch seine teamorientierte Arbeitsweise und die Möglichkeit zum explorativen Lernen interessant. Didaktisch steht hier das Erlernen und Erfahren von Programmier- und Designmethoden im Vordergrund.

2.1.1 Soziale Fähigkeiten

Eine der offensichtlichsten Motivationen für den Einsatz von Roboterbau im Unterricht ist die Teamarbeit [Jadud]. In unserer arbeitsteiligen Welt ist die Fähigkeit im Team zu arbeiten in den meisten Berufen unabdingbar. Im herkömmlichen Unterricht kommt diese Arbeit aber meist zu kurz. Dies liegt zum Teil in der Art der Aufgabenstellung bzw. der mangelnden Komplexität — und damit nur schlechten Teilbarkeit — der typischen Aufgaben begründet.

2.1.2 Alternative zu ‘telling and testing’

Die meistgenutzte Methode des Unterrichtens ist heute ‘telling and testing’. Es wird eine Menge Stoff vermittelt und an Hand einfacher kurz zu beantwortender Fragen überprüft ob der Stoff gelernt wurde. Auf diese Weise gelangt allerdings das selbständige Ermitteln von Wissen durch eigenständige Untersuchungen und Tests (‘exploration and construction’) ins Hintertreffen. In den Robotkursen wird gerade auch dieses selbständige Ermitteln des gerade notwendigen Wissens gefördert. Man muß hier aber zu bedenken geben, daß gerade für die jüngsten Jahrgangsstufen eine weitergehendere Betreuung notwendig ist, als dies für die Sekundarstufe II bzw. Studierende gilt.

2.1.3 ‘single answer questions’

Ein gewichtiger pädagogischer Punkt ist, daß die Lernenden in der bisherigen Ausbildung meist ‘single answer questions’ gestellt bekamen. Dies fördert nicht die kreative und kritische Auseinandersetzung mit möglichen Lösungen und ist auch nicht typisch für die Probleme, die in der normalen Arbeits- und Umwelt auftauchen. In der Robotik gibt es typischerweise mehrere Wege, die zum Ziel führen. Jeder dieser Wege hat Vor- und Nachteile und die Lernenden müssen selbst Entscheidungen für oder gegen einen Weg treffen.

2.1.4 Komplexität der Welt

In den Natur- und Ingenieurwissenschaften wird mit Modellen der realen Welt gearbeitet. Es sind immer Vereinfachungen, gleich den Schatten des platonischen Höhlengleichnisses². In der Robotik werden diese Modelle nun

²An einem Höhleneingang sind Betrachter so fixiert, daß sie einzig an der Höhlenwand die Schatten (Projektionen) von Gegenständen, die hinter ihnen vorbeigetragen werden

sehr deutlich wieder mit der realen Welt in Kontakt — oder auch in Konflikt — gebracht. Die Lernenden erfahren, daß die Welt der Theorien sich nicht unbedingt einfach auf die realen Bedingungen übertragen läßt.

2.1.5 learn how to design not about design

Dieser Aspekt der Robotlabs reiht sich in die Philosophie des ‘learning by doing’ ein. Das Design eines Robots kann je nach Aufgabenstellung eine recht vielfältige Aufgabe sein, ganz so wie die Designaufgaben eines Ingenieurs in der realen Welt. Die Studierenden können im Robotlab sehr gut verschiedene Designansätze ausprobieren und so Designerfahrungen an Hand eines nicht trivialen Problems sammeln. Die beiden wichtigsten Varianten für Designs sind das vollständige Durchplanen und die evolutionäre Annäherung. Beide Varianten können prinzipiell im Roboterlabor genutzt werden, allerdings ist die evolutionäre Annäherung die am häufigsten genutzte Variante.

Ebenfalls kann das Roboterlabor als eine Lernumgebung für neue Software-Engineering Methoden dienen, wie z.B. dem *Extreme Programming* [Beck2000] einer stark teamorientierten Methode.

2.2 behandelte Informatikthemen

Bei der Programmierung von Robotern kann ein sehr breites Spektrum an Informatikthemen eingesetzt und ausprobiert werden. Angefangen bei einfachsten imperativen Programmen, über die strukturierte Programmierung bis hin zu aufwendigen Themenstellungen aus dem Bereich der künstlichen Intelligenz oder der Bildverarbeitung. Einige Themen werden im folgenden herausgehoben, da sie im Umfeld der Programmierung von Robotern besondere Bedeutung besitzen oder für den Unterricht interessant sind.

2.2.1 Imperative Programmierung

Die meisten der heute eingesetzten Programmiersprachen beruhen auf der imperativen Programmierung. Dies sind Programme, die auf den Elementen Anweisung, Variable und bedingte Verzweigung beruhen³. Mit diesen Grundlagen können schon einfache Robotermodelle und Aufgabenstellungen programmiert werden. Dies bedeutet, daß die Roboterprogrammierung auch schon für Einstiegsurse in die Programmierung genutzt werden kann.

sehen können aber nicht den eigentlichen Gegenstand. Die Betrachter glauben nun die Schatten seien die *wahre* Welt.

³Tatsächlich bilden diese drei Elemente den einzig notwendigen Kern einer Programmiersprache, die die Mächtigkeit einer Turing-Maschine besitzt, denn mit diesen Elementen läßt sich eine Turing-Maschine nachbilden (abgesehen von dem notwendigen, unendlichen Speicherplatz). Turing-Maschinen können jedes berechenbare Problem lösen. Alle weiteren Programmierkonstrukte dienen *nur* der Vereinfachung der Programmierung, liefern aber prinzipiell keine Erweiterung der Klasse der lösbaren Probleme.

Alle Konzepte der imperativen Programmierung können so in einem stark motivierenden Umfeld erlernt werden.

2.2.2 Strukturierte Analyse/Strukturierte Programmierung

Um die Komplexität von großen Aufgabenstellungen zu bewältigen, wurde die Strukturierte Analyse entwickelt. Hierbei werden Probleme in Teilaufgaben und deren Beziehungen zerlegt und diese Teilaufgaben dann weiter bearbeitet. Dabei sollen die Teilaufgaben möglichst unabhängig bearbeitbar sein. Bei den Aufgabenstellungen der Robotik lassen sich oft Teilprobleme identifizieren, z.B. Bewegungsapparat, Zielfindung, Wegplanung und Manipulatorsteuerung. Die Prinzipien der Strukturierten Analyse lassen sich also einsetzen.

In Zusammenhang mit der Strukturierten Analyse steht der Begriff der Strukturierten Programmierung, in der alle Programmkonstrukte sauber geschachtelt zusammengefügt werden. Ein absoluter Sprung irgendwo in das Programm durchbricht diese saubere Schachtelung (ein typischer Vertreter für diese Konstrukte ist das BASIC-Sprachelement `goto`). In der Strukturierten Programmierung wird auf solche Sprünge verzichtet. Der Grund für die Einführung dieser Beschränkung liegt in dem möglichen formalen analytischen Korrektheitsbeweis von Strukturierten Programmen. Mit der Existenz von absoluten Sprüngen ist der Beweis von Korrektheit nur sehr schwer zu führen, bzw. unmöglich. Auch sind Strukturierte Programme mit nicht formalen Methoden leichter auf Korrektheit überprüfbar.

Eine andere Ebene der Strukturierung ist die der Datenstrukturen (besonders in Pascal bzw. Modula eingeführt) bzw. noch weitergehend die Ebene der abstrakten Datentypen, die letztlich in der Objektorientierung mündet (Simula, C++, Java). In den elementaren Themen der Robotik spielt die Datenstrukturierung nur eine untergeordnete Rolle. Typischerweise werden aber auf den oberen Ebenen — z.B. in Planungsalgorithmen — Datenstrukturen wichtiger und umfangreicher.

2.2.3 Parallele Abläufe

Viele Aufgaben können oder sollten in einem Roboter parallel verarbeitet werden. So gibt es beispielsweise eine Architektur für Programmierung in der ein Task Sensordaten ermittelt und vorverarbeitet, ein anderer Task die Reaktion auf die vorverarbeiteten Daten veranlaßt und ein weiterer Task die übergeordnete Planung übernimmt. In anderen Architekturen (z.B. behavior based oder subsumption, siehe Abschnitt 2.2.6) laufen parallel verschiedene Verhaltensweisen ab, die sich unabhängig um den Zugriff auf Ressourcen bewerben.

Die Kommunikation zwischen diesen Tasks, der Ausschluß von Blockaden und Kontrolle über Ressourcen sind hier Thema der Programmierung

und eine stark erweiterte Herausforderung gegenüber der Programmierung ‘einfacher’ einzeln ablaufender Programme (single threaded programs).

2.2.4 Meß-, Steuer- und Regelungstechnik

Da die Roboter natürlicherweise als Teil eines Regelkreises agieren — jeder ernstzunehmende Roboter reagiert auf seine Umwelt und manipuliert sie, ist also ein Glied eines Regelkreises —, kann man die Steuer- und Regelungstechnik hier sehr praxisnah einsetzen. Allerdings ist es nicht notwendig zunächst einen theoretischen Unterbau zu schaffen, denn die absolut notwendigen Regelungen⁴ und deren Parameter sind auch durch Experimente ermittelbar, bzw. sie sind mit Hilfe des ‘gesunden’ Menschenverstandes herleitbar. Wenn es aber gewünscht ist, so kann die ganze Tiefe der Steuer- und Regelungstechnik ausgelotet werden. Angefangen bei einfachen — intuitiv zu verstehenden — Zweipunktreglern über proportionalen Regler bis hin zu tatsächlich berechneten Regelkreisen läßt sich in der Robotik alles erforschen und anwenden. Ebenso kann man sich intensiv mit Sensorik [Everett1995] auseinandersetzen. Über die Filtertechniken und differentielle Messungen hin zu elektronischen Schaltungen für besondere Vorverarbeitung und Aufbereitung der Signale. Ebenfalls ein Thema der Regelungstechnik, das mit Robotern bearbeitet werden kann ist die Fuzzy-Logic-Programmierung.

2.2.5 Echtzeitbedingungen

Roboter können auch sehr gut als Echtzeitprobleme verstanden und behandelt werden. Typischerweise muß ein Roboter in Echtzeit auf ein aufgetretenes Signal reagieren. Als Beispiel sei ein Roboter angeführt, der sich frei auf einer Tischplatte bewegen soll, ohne vom Tisch herunterzufallen. Die Signale der Sensorik, die die Tischkante erkennt müssen in Echtzeit verarbeitet werden — d.h. es muß innerhalb einer harten Zeitgrenze eine Reaktion der Steuerung der Motoren erfolgen —, sonst fällt der Roboter von Tisch und die Aufgabe ist nicht gelöst.

2.2.6 behavior based / subsumption architecture

Speziell für die Bedürfnisse der Programmierung von Robotern, die sich in einer realen Umwelt bewegen sollen, wurde das Modell des ‘behavior based programming’ bzw. die ‘subsumption architecture’ entwickelt⁵. Hier werden unterschiedliche Verhaltensweisen entworfen und programmiert und diese dann konkurrierend parallel betrieben. Über Prioritäten und die Anknüpfung von Verhalten an externe Signale wird nun das Verhalten ausgewählt, das den Roboter kontrollieren soll. Diese Art der Programmie-

⁴Zwei-Punkt-Regler und Proportional-Regler

⁵Die ‘subsumption architecture’ wurde von Rodney Brooks am MIT in den späten 1980er Jahren entwickelt [Brooks1990].

rung/Systembetrachtung ist dem Behaviorismus⁶ entlehnt. Sehr wichtig ist hier, daß nicht ein Modell von Verhalten im Ganzen entworfen wird, sondern nur kleine einfache Verhaltensweisen und die jeweilige Notwendigkeit für das Auftreten einer dieser Verhaltensweisen. Dies dient ebenfalls der Reduzierung von Komplexität eines Gesamtverhaltens auf kleine beherrschbare Verhaltensweisen. Es muß hierbei eine Regelung des Zugriffes auf beschränkte Ressourcen (z.B. den Motorantrieb) eingeführt werden, damit sich konkurrierende Verhaltensweisen (Angriff und Fluch) nicht gegenseitig blockieren (was zur Kaninchenstarre führen kann).

2.2.7 reaktive Systeme

Die formaleren Modelle für reaktive Systeme nach der 'subsumption architecture' sind Zustandsautomaten nach Mealy und Moore. Hier werden spezielle Zustände definiert und Übergänge zwischen diesen Zuständen, die auf Grund von (externen oder internen) Signalen ausgelöst werden. Außerdem werden Aktionen definiert, die bei Betreten oder Verlassen eines Zustandes oder bei Durchführen eines Überganges ausgeführt werden. Die Automatentheorie der theoretischen Informatik beschreibt die Zustandsautomaten vollständig und gibt Methoden an die Hand, um in solchen Automaten z.B. Verklemmungen zu entdecken oder die Automaten zu optimieren. Von Harel stammt eine Erweiterung von hierarchischen Zustandsautomaten die besonders zur Darstellung (und Implementierung) von reaktiven Systemen geeignet ist [Harel1987]. Die Harel-Automaten sind — leicht modifiziert — mittlerweile auch Bestandteil der Unified Modelling Language (UML) [OMG1999].

2.2.8 Künstliche Intelligenz und Bildverarbeitung

Die Künstliche Intelligenz hält in Robotern im Zusammenhang mit höheren Funktionen Einzug. Hierzu zählen vor allem Planungsstrategien für das Erreichen von Zielen. Ebenfalls spielt in der Robotik das Thema Bildverarbeitung und -erkennung sowie der Bereich der kognitiven Systeme eine große Rolle. Weitere Schlagwörter sind Neuronale Netze, Selbstlernende Systeme, Evolutionäre Programme und Genetische Algorithmen.

2.2.9 Braitenberg Vehicle

Ein interessantes Experiment aus der experimentellen Psychologie kann mit den Robotern ebenfalls durchgeführt werden. Es handelt sich um die sogenannten Braitenberg Vehikel [Braitenberg1993]. Dies sind einfache Wagen

⁶Behaviorismus, [aus englisch behavio(u)r »Verhalten«], 1912 von dem Amerikaner J. B. Watson begründete psychologische Richtung, die nur objektiv beobachtbares Verhalten als Gegenstand wissenschaftlicher Forschung zuläßt. Aus diesem Verhalten leitet der Behaviorismus Regeln für die Erziehung und das menschliche Zusammenleben ab.

mit einem Differenzantrieb und einfachen Sensoren, die mit einfachen internen Verschaltungen (Programmierungen) sehr interessante Verhaltensweisen an den Tag legen. Die Analyse bzw. die Synthetisierung der gewünschten Verhaltensweisen ist sehr spannend, da aus sehr einfachen inneren Strukturen ein sehr komplexes Verhalten entstehen kann. Die Experimente mit den Braitenberg Vehikeln eignen sich auch als Einführung in dynamische Systeme.

2.2.10 Multiagenten Systeme

Aus dem kooperativen Zusammenspiel mehrerer Roboter ergeben sich Szenarien von Multiagenten Systemen [DudekJenkinMiliotWilkes1996]. Hier ist die Ebene der Kommunikationsprotokolle interessant, ebenso wie verteilte Planungsstrategien oder die Robustheit gegen Kommunikationsfehler (Problem der Byzantinischen Generäle).

2.3 Zusammenfassung

In der Robotik gibt es eine große Vielzahl an Motivationen und Möglichkeiten⁷. Die Faszination, die von den künstlichen Schöpfungen ausgeht ist ein starker Antrieb für die Beschäftigung mit vielen beigeordneten Themen und Methoden. Ebenso ist es ein Weg sich selbst klar zu machen, zu welchen enormen Leistungen einfachste Lebewesen in der Lage sind oder was für beeindruckende Werkzeuge die menschlichen Hände sind.

⁷Im vorhergehenden Abschnitt sind nur ein Teil der Möglichkeiten genannt worden. Weitere Themen finden Sie z.B. in [DudekJenkin2000].

3 Altersgruppen

Für den Aufbau eines Roboterlabores muß man 3 Altersgruppen unterscheiden.

3.1 Sekundarstufe I (5.–10. Schuljahr)

Für die jüngste Gruppe an Teilnehmern muß das Labor relativ stark vordstrukturiert werden. Eine völlig freie Lernumgebung führt hier eher zur Überlastung der Kinder, als zu funktionsfähigen Robotern. Zur eigenständigen Informationsbeschaffung ist diese Altersgruppe noch nicht ausreichend in der Lage, um eine komplexe Aufgabenstellung selbständig zu lösen. Es ist wichtig für die Kinder einzelne Schritte zur Aufgabenlösung hin vorzubereiten. Der Reihe nach werden so kleine Probleme gemeinsam erarbeitet und gelöst.

Mit dem Material des Roboterlabors — wenn das Labor aus Baukästen wie Lego oder Fischertechnik besteht — lassen sich auch sehr gut Themen aus dem allgemeinen Technikunterricht untersuchen. Angefangen bei Hebelgesetzen, über Zahnrad und Getriebebau, bis hin zu Themen wie einem Differentialgetriebe oder einer Fahrzeuglenkung. Erste elektrische Antriebe können zum Beispiel mit der Anfängerversion der Software Robolab (siehe Abschnitt 5.1.2) programmiert werden⁸.

3.2 Sekundarstufe II (11.–13. Schuljahr)

Mit den Schülern der höheren Jahrgangsstufen können schon anspruchsvollere Labore durchgeführt werden. Hier können deutlich mehr Freiräume genutzt werden und auch komplexere Aufgabenstellungen angegangen werden. In einem zeitlich längeren Labor sollte zunächst eine einfache Aufgabe wie die des Linienfolgers (siehe Abschnitt 6.2.1) gelöst werden, um dann mit dem erworbenen Wissen an die komplexeren Aufgaben zu gehen (Abschnitt 6.2.2 bzw. 6.2.3).

3.3 Studium

Für Studierende ist das Roboterlabor in allen Studienstufen interessant. Es bietet die Möglichkeit verschiedene Prinzipien direkt anzuwenden und so das theoretisch erworbene Wissen aus vielen Fächern auf dem Boden der Tatsachen zu erproben und miteinander zu kombinieren. Zu den angesprochenen Fächern gehören unter anderem: Software Engineering, Algorithmik,

⁸Im St. Agnes LEGO Workshop an der Tufts University, Massachusetts wird seit drei Jahren ein entsprechendes Curriculum entwickelt. Es gibt dort auch Ansätze zum Einsatz der Legoelemente im Mathematikunterricht.

Siehe <http://www.ceeo.tufts.edu/graphics/StAgnes/index.html> bzw. <http://www.tufts.com>.

Automatentheorie, Systemtheorie, Meßtechnik, Elektronik, Steuer- und Regelungstechnik, Kognitive Systeme und Konstruktionslehre.

4 Laborumgebung

Im Bezug auf die einzusetzende Hardware muß man im Roboterlabor vier Hauptelemente unterscheiden. Zwei davon gehören direkt zu den zu bauenden Robotern, es sind der Steuerrechner (mit Aktoren und Sensoren) und das Chassis. Als drittes gesellt sich der notwendige Computer als Host für die Entwicklungsumgebung dazu und nicht zu letzt ist die zu lösende Aufgabenstellung ein wichtiger Teil des Labors.

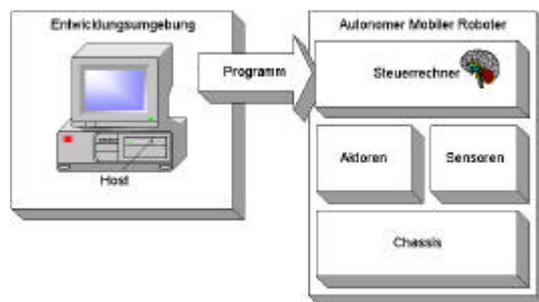


Abbildung 1: Elemente der Robotprogrammierung

Da für die Entwicklungsumgebung als Host in allen Fällen ein handelsüblicher PC mit Windows Betriebssystem ausreichend⁹ ist und dieser auch für alle relevanten Varianten benötigt wird, wird hier nicht weiter darauf eingegangen. Es wird ein Vergleich der möglichen Varianten für den Steuerrechner und das Chassis durchgeführt und im Bezug auf den Host Rechner werden einige exemplarische Programmierumgebungen besprochen.

4.1 Steuerrechner

Für den Steuerrechner stehen viele verschiedene Möglichkeiten bereit. Vier davon mochte ich hier näher betrachten. Die erste — wohl indiskutable — ist die eigene Entwicklung von Hard- und Software für den Steuerrechner, dies ist zwar am anspruchvollsten und flexibelsten, jedoch so komplex, das die Aufgabe den Betrieb eines einfachen Roboterlabores weit überbeansprucht¹⁰.

⁹Teilweise wird in den besprochenen Systemen auch MacOS bzw. Linux unterstützt, siehe auch Tabelle 1 auf Seite 27.

¹⁰An der FH-Hamburg wurde eine Eigenentwicklung in Angriff genommen und das Projekt dauert noch an. Der Initialarbeit zum KRABAT [WitthöftHoffmann1997] folgte die Entwicklung eines eigenen Betriebssystems [DrewsLehmann1998] oder auch die Portierung von Linux auf das KRABAT-Board [GeßnerNeumann1999] und weitere Arbeiten z.B. eine Bildverarbeitungserweiterung [Thiele1999].

Die zweite Möglichkeit ist der Einsatz des originalen 6.270-Boards¹¹ oder dessen Nachfolger Handyboard¹² vom MIT. Diese Kombination von Hard- und Software (Interactive C) ist sehr leistungsfähig. Es gibt eine Reihe von analogen (8 bit breiten) und digitalen Ein- und Ausgängen. Ebenso spezielle Leistungstreiber für den bidirektionalen pulsweitenmodulierten Anschluß von Motoren und spezielle Anschlußmöglichkeiten von Servomotoren.



Abbildung 2: Handyboard

Auf der Eingabeseite stehen spezielle Eingänge zur Erkennung von modulierten Infrarotsendern zur Verfügung. Für Rückmeldungen während des Betriebes steht ein zweizeiliges Display zur Verfügung. Die Kommunikation der Roboter mit dem Host der Entwicklungsumgebung erfolgt mittels einer seriellen Kabelverbindung (RS 232). Mit diesen Möglichkeiten stellen diese Boards die Leistungsspitze der hier besprochenen Varianten dar. Leider haben die Boards auch deutliche Nachteile. Zum einen sind die Boards mit den zugehörigen Batterien (9 Volt Blockbatterie für den Computer und 6 Volt Bleiakku für die Motoren) recht groß und schwer, zum anderen sind die Anschlüsse sehr diffizil und empfindlich. Die im 2,52 mm Rasterabstand angeordneten Pfostenstecker sind für Kinderhände der SEC I nicht geeignet¹³ und die Eignung im rauen Schulalltag der SEC II ist ebenfalls fraglich. Selbst beim Einsatz im Studium sind — ebenfalls wegen der Kontaktproblematik — in den abschließenden Wettbewerben schon einige Favoriten gescheitert.

Die dritte Variante des Steuerrechners ist der Einsatz des Rechnermoduls ‘Intelligent Interface’ von Fischertechnik¹⁴. Es ist mit 4 Motorausgängen, 2 analogen und 8 digitalen Eingängen etwas schlechter ausgestattet als die Boards des MIT. Mit Hilfe einer Erweiterungsbox können zusätzliche 4 Motorausgänge und 8 digitale Eingänge genutzt werden, allerdings ist die Erweiterungsbox ebenso groß wie das eigentliche ‘Intelligent Interface’ und benötigt ebenfalls eine weitere Stromversorgung. Die Kommunikation mit dem Host der Entwicklungsumgebung erfolgt mittels einer seriellen Kabelverbindung (RS 323). Die Anschlüsse der Sensorik und Aktorik an das Board sind mittels kleiner Rundsteckkontakte realisiert und nicht verpolungssicher.



Abbildung 3: Fischertechnik, Intelligent Interface

¹¹<http://www.mit.edu/courses/6.270/home.html>

¹²<http://el.www.media.mit.edu/groups/el/projects/handy-board/>

¹³siehe auch Anhang A

¹⁴Herstellerinformationen unter <http://www.fischertechnik.de>

Die Alternative zu den 6.270 bzw. Handyboards und Fischertechnik stellt das LEGO Mindstorm Set¹⁵ dar. Der Steuerrechner ist mitsamt der notwendigen Batterien für Rechner und Motoren in einem ‘größeren’ Legosteine, dem RCX, untergebracht. Es sind drei Ausgänge für bidirektionale pulswitengesteuerte Gleichstrommotoren und drei analoge — 10 bit breite — Eingänge vorhanden. Die Anzahl der Ein- und Ausgänge erscheint zunächst



Abbildung 4: LEGO Mindstorm RCX

sehr beschränkt, aber durch kleine Softwaretricks lassen sich an einem Eingang ein Lichtsensor und ein Schalter parallel betreiben oder mittels einer einfachen Widerstandsschaltung mehrere Schalter, hierzu gibt es neben Literatur [BaumGasperiHempelVilla2000] auch im Internet etliche Anleitungen¹⁶. Die Kommunikation mit dem Host der Entwicklungsumgebung erfolgt hier mittels eines in den RCX eingebauten Infrarotsenders und -empfängers. Über diese Infrarotschnittstelle können auch mehrere RCX in einem Roboter oder auch mehrere Roboter mit jeweils einem RCX zusammenarbeiten. Diese kabellose Kommunikation während des laufenden Betriebes ermöglicht ganz neue Szenarien für Aufgabenstellungen und Untersuchungen (Multiagentensysteme, siehe auch Abschnitt 2.2.10). Die Verbindung der Motoren und Sensoren mit dem RCX erfolgt über einen speziellen Steckkontakt in Form eines einfachen Legosteines¹⁷. Diese Verbindung ist verpolungs- und kurzschlußsicher und sehr robust, damit ist der RCX auch für Kinderhände geeignet.

4.2 Chassis

Für das Chassis von Mobilen Robotern stehen im wesentlichen drei Varianten zur Verfügung.

Der Eigenbau stellt eine Herausforderung dar, allerdings sind durch das Ausnutzen von fertigen oder halbfertigen Bausätzen aus dem Modellbau interessante Möglichkeiten zum Chassisbau vorhanden. Das Spektrum reicht von der Ackermannsteuerung (Prinzip des Autos) über Raupen bis hin zu Differenzantrieben. Eine gute Einführung in das Thema bietet das Buch „*Mobile Robots. Inspiration to Implementation*“ [JonesFlynn1993].

Gerade bei wechselnden Aufgabenstellungen ist eine einfache Anpaßbarkeit des Chassis notwendig. Hier liegen LEGO und Fischertechnik auf Grund ihres Bausteinprinzips deutlich vor der Eigenkonstruktion. LEGO ist vielen Kindern und Studierenden gut bekannt und die Konstruktionsprinzipien sind einfach zu erlernen. Im Vergleich mit Fischertechnik wird

¹⁵ Herstellerinformationen unter <http://mindstorms.lego.com/>

¹⁶ Als Ausgangspunkt für eine Recherche eignet sich <http://www.lugnet.com> recht gut.

¹⁷ siehe auch Anhang A

oft eine geringere Stabilität ins Feld geführt, die aber durch konstruktive Maßnahmen ausgeglichen werden kann. Im Bezug auf die evolutionäre Entwicklung von Robotern erscheint LEGO etwas flexibler als Fischertechnik. Wie die bisherigen Erfahrungen sowohl in den 6.270 Kursen, als auch an der FH-Hamburg zeigen, lassen sich mit LEGO auch aufwendige Aufbauten wie z.B. Greifer konstruieren. Im Bereich der Antriebe sind auch hier alle Varianten vom einfachen Dreirad mit Differenzantrieb oder einem Synchrodrive, über Raupenantriebe bis hin zur Ackermannsteuerung möglich.

4.3 Zusammenfassung

Die Schwierigkeiten, die bei Eigenkonstruktionen zu erwarten sind lassen als sinnvolle Variante nur die kommerziellen Produkte von Fischertechnik oder LEGO zu. Die LEGO Mindstorm Produkte erscheinen sowohl in der Konstruktion, als auch in der Programmierung (siehe Kapitel 5) flexibler zu sein als die Fischertechnik Produkte, ebenso sind die Anschlüsse (siehe Anhang A) robuster und sicherer. Nicht zuletzt wurde mit dem Einsatz der LEGO Technik sowohl im 6.270 Kurs des MIT, als auch im Roboterlabor der FH-Hamburg gute Erfahrungen gemacht. Aus diesem Grund werde ich mich in den folgenden Ausführungen auf den Einsatz von LEGO Mindstorm beschränken.

Die Bausteine von LEGO sind ebenfalls für die Erarbeitung von nicht Informatikinhalt geeignet. Hierzu zählen Konstruktionen im Technikunterricht (Statik, Verstreben, Hebel, Getriebebau, Differential, Lenkgestänge usw.) ebenso wie der Einsatz der Bauteile im Mathematikunterricht. Doch dieses Thema ist nicht Inhalt dieser Untersuchung¹⁸.

¹⁸Im St. Agnes LEGO Workshop an der Tufts University, Massachusetts wird seit drei Jahren ein entsprechendes Curriculum entwickelt. Es gibt dort auch Ansätze zum Einsatz der Legoelemente im Mathematikunterricht. Siehe <http://www.ceeo.tufts.edu/graphics/StAgnes/index.html> bzw. <http://www.tufts.com>.

5 Programmierung

Für LEGO Mindstorm gibt es eine große Zahl verschiedener Programmierumgebungen und eine sehr aktive Fangemeinde im Internet¹⁹ die ständig an Erweiterungen, neuen Programmierumgebungen und weiteren Sprachen arbeitet. Beispiele dafür sind der fertige Forth-Interpreter²⁰ oder die Projekte zur Entwicklung von virtuellen Maschinen für Java auf dem RCX²¹. Ebenso existieren Projekte zu Smalltalk, Logo, Scheme und Varianten in denen auf der RCX Hardware nur ein Kommunikationsprogramm läuft, das via Infrarotschnittstelle mit den eigentlichen Steuerprogramm auf dem PC Daten und Steuerinformationen austauscht. Die eigentliche Programmierung erfolgt dann z.B. in Java auf dem PC (In einer solchen Umgebungen sollten allerdings keine harten Echtzeitanforderungen gestellt werden).

Trotz der Vielfalt der zur Verfügung stehenden Varianten möchte ich mich im folgenden auf vier Produkte beschränken (LEGO Mindstorms RIS, Robolab, NQC und LegOS). Die Auswahl der besprochenen Programmierumgebungen erfolgte nach Verfügbarkeit, Stabilität, Einfachheit und allgemeiner Akzeptanz der Sprache oder des Werkzeuges. Des Weiteren deckt die Auswahl ein großes Spektrum der Möglichkeiten der Programmierung des RCX ab. Es wird im folgenden auch kurz dargestellt welche Konzepte aus der Informatik mit der jeweiligen Umgebung realisierbar sind.

Generell muß man bei der Software für LEGO Mindstorm zwei Typen unterscheiden. Der eine Typus nutzt die Standardfirmware von LEGO (siehe Anhang E) und der andere Typus stellt eine komplette eigene Laufzeitumgebung auf dem Roboter bereit. Zum ersten Typ zählen die im nachfolgenden beschriebenen Umgebungen ‘LEGO Mindstorm RIS’, ‘Robolab’ und ‘Not Quite C’ (NQC), zum zweiten Typus gehört ‘LegOS’ und auch ‘pbForth’. Auf Grund der Einschränkungen der Standardfirmware sind die Umgebungen, die nicht auf der Standardfirmware beruhen üblicherweise deutlich leistungsfähiger und mächtiger.

5.1 Visuelle Programmierung

Die visuelle Programmierung, also die Programmierung mittels der grafischen Anordnung von Steuerelementen ist eine relativ intuitive Programmierweise. Es müssen im Gegensatz zur textuellen Programmierung meist keine Syntaxregeln erlernt werden, denn die grafischen Entwicklungsumgebungen sollten syntaktisch fehlerhafte Verknüpfungen der Elemente gar nicht zu lassen. Zudem liegt bei der grafischen Darstellung die Struktur des

¹⁹Siehe Anhang G

²⁰pbForth unter <http://www.hempeldesigngroup.com/lego/pbFORTH/>

²¹RCXJVM unter <http://misc.traveller.com/rcxjvm/> und TinyVM unter <http://sourceforge.net/projects/tinyvm>

Programmes direkt sichtbar vor (meist in einer Form, die Programmablaufplänen, Schaltplänen oder Struktogrammen ähnelt).

5.1.1 Mindstorm RIS

Beim Kauf eines normalen ‘Robotics Invention System 1.5’ von LEGO Mindstorm wird eine grafische Programmierumgebung mitgeliefert, die ebenfalls den Namen ‘Robotics Invention System’ (RIS) trägt. Diese Umgebung beruht auf dem Konzept, daß per Drag-and-Drop einzelne Befehle auf einer Arbeitsfläche angeordnet und verknüpft werden.

Es gibt drei Hauptarten von Blöcken, grüne Befehlsblöcke mit einfachen Anweisungen, blaue Sensorblöcke und rote Stackcontroller mit Schleifen und Sensorabfragen mit bedingten Ausgängen. Zudem gibt es noch die Möglichkeit Zusammenstellungen von Blöcken zu eigenen gelben Makroblöcken zusammenzufassen. Ein Programm besteht immer aus einem Hauptzweig (ausgehend von einem grünen Startblock)

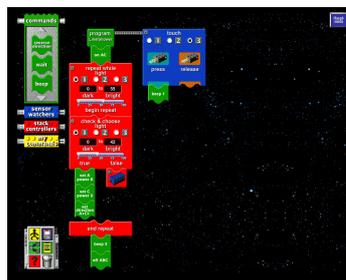


Abbildung 5: Arbeitsoberfläche vom RIS

und maximal 8 Nebenzweigen (blaue Sensorblöcke) die pseudoparallel ausgeführt werden. Hierbei wird ein Nebenzweig in Abhängigkeit der gerade aktuellen Sensorwerte und der Einstellung des Blocks zyklisch ausgeführt oder nicht. In den Zweigen können mit Hilfe der roten Stackcontroller in Abhängigkeit von Sensorwerten bedingte Verzweigungen und auch bedingte Wiederholungen durchlaufen werden.

Es gibt keine Datenstrukturen im Sinne von Datenspeichern, einzig die Sensorwerte, Timer und der Zähler stellen eine Art Datenstruktur dar. Da keine Variablenspeicher vorhanden sind, sind auch keine Berechnungen möglich. Damit sind die Möglichkeiten der Programmierung doch sehr beschränkt. Schon einfache Zeitmessungen sind nicht möglich, geschweige den Arten von quasikontinuierlichen Regelungen oder gar Gedächtnismodelle. Ebenso ist eine Synchronisierung der parallel ablaufenden (von den blauen Sensorblöcken ausgehenden) Tasks nur unter extremen Klimmzügen und sehr eingeschränkt mit Hilfe des einzigen Zählers möglich.

Ebenso ist keine Möglichkeit zur Introspektion (Beobachtung des Zustandes der Ein- und Ausgänge oder der Abarbeitung des Programmes) des RCX vorhanden.

Die Nachteile des RIS stellen aber auch einen Vorteil dar. Durch die bewußte Beschränkung auf die elementarsten Teile der Programmierung ist das RIS sehr leicht verständlich und für die ersten Schritte auf dem Gebiet der Programmierung durchaus geeignet. Allerdings sind die Beschränkungen doch so stark, das schon einfache Alternativen in Bewegungsmustern (Linie folgen oder Hindernis ausweichen) zu sehr komplexen und schwer beherrschbaren und wartbaren Darstellungen führen. Es sollte daher schon

relativ früh auf eine andere Plattform gewechselt werden, hier bietet sich als Alternative in visueller Programmierung nur die Software Robolab an oder aber der Einsatz einer textuellen Programmiersprache.

In das RIS ist eine kleine Verwaltung für Programme eingebaut. In dieser Verwaltung können eine beliebige Anzahl von Benutzern angemeldet werden, für die dann ein eigenes Verzeichnis für die Programme angelegt wird. Allerdings gibts es keinerlei Schutz vor Zugriff oder Veränderung der Programme durch andere Nutzer. Zu den Programmen kann eine kurze Beschreibung des Roboters für den das Programm geschrieben wurde und eine Beschreibung des Zwecks des Programmes hinterlegt werden.

Die Installation des RIS wird einfach mittels eines Standard-Windows-Installations-Programmes vorgenommen und stellt sich relativ problemlos dar. Im RIS gibt es einen eingebauten Trouble-Shooter, der bei Auftreten von Kommunikationsproblemen zwischen Entwicklungsrechner und RCX eine Schritt für Schritt Anleitung zur Problembeseitigung liefert. Ebenfalls ist eine deutschsprachige Online-Hilfe zu den Programmblöcken und der allgemeinen Bedienung des RIS vorhanden. Zusätzlich zur reinen Programmierumgebung gibt es geführte Touren zu einfachen Programmieraufgaben und Robotern (die sogenannte Programmopedia). In diesen Touren werden die ersten Schritte zur Programmierung mittels Video und Ton passend zu den, in der zugehörigen gedruckten Anleitung (Constructopedia) vorgestellten, Modellen durchgeführt. Das Programm und die Erklärungsvideos liegen vollständig in deutscher Sprache vor.

Es wird zur Zeit an einer Weiterentwicklung des RIS gearbeitet. In dieser Weiterentwicklung soll eine neue Firmware eingesetzt werden, in der Mechanismen zur Programmierung von ‘subsumption architectures’ eingebaut sind, in wie fern diese Mechanismen in der grafischen Programmierung genutzt werden können ist zur Zeit unklar. Weiter soll es möglich sein zwischen der grafischen Darstellung und einer textuellen Darstellung zu wechseln.

5.1.2 Robolab

Das Robolab 1.5²² ist ein Abkömmling der Umgebung ‘Labview’ von National Instruments²³. Die Software wurde speziell für die Bedürfnisse eines Roboterlabors an der Tufts University College of Engineering²⁴ entwickelt.

Auch diese Umgebung beruht wie das RIS auf dem Konzept, daß per Drag-and-Drop einzelne Befehle auf einer Arbeitsfläche angeordnet und verknüpft werden. Die Darstellung ist allerdings wesentlich kompakter und die Möglichkeiten der Sprache deutlich umfangreicher.

Im Robolab gibt es 2 verschiedene Modi der Programmierung einen Einführungsmodus und einen Fortgeschrittenen Modus. Innerhalb der bei-

²²Zu Beziehen von LEGO Dacta, <http://www.lego.com/dacta/robolab/>

²³<http://www.ni.com/>

²⁴<http://www.tufts.edu/> bzw. <http://www.ceeo.tufts.edu/>

den Modi gibt es 4 Level, in denen jeweils von Level zu Level mehr Varianten und Möglichkeiten zur Programmierung bereitgestellt werden. Den unterschiedlichen Leveln können in einer Administratorsicht Aufgabenstellungen in Form von Programmfragmenten mit Kommentierungen zugewiesen werden. Auf diese Weise sollen die Schüler durch die Programmierung in den verschiedenen Leveln geführt werden.

Die Programmierung erfolgt dadurch, daß verschiedene Arten von Blöcken mit Verknüpfungen zu Strängen von Anweisungsfolgen zusammengesetzt werden. Im höchsten Level des Fortgeschrittenenmodus stehen Blöcke für alle Funktionen des RCX zur Verfügung. Dies sind Ansteuerung der Motoren, Tonausgaben, Abfrage der Sensorik und Timer. Weiterhin sind sogenannte Warteblocks vorhanden, mit denen der Programmfluß bis zum Eintreten eines bestimmten Ereignisses (Sensorik, Timer, Wert eines Containers oder Eintreffen einer Infrarot-Nachricht) angehalten wird.

Als Programmierkonstrukte stehen Blöcke für bedingte Verzweigungen, Schleifen und absolute Sprünge²⁵ (Vergleichbar mit dem Basic-Konstrukt `goto`) zur Verfügung. Die Anzahl der absoluten Sprünge ist auf 5 begrenzt.

Als Variablen und einzige Datenstruktur stehen sogenannte Container bereit, in denen Integerwerte gespeichert werden können. Die Quellen für die Inhalte sind Sensorenwerte. Mit den Inhalten der Container können Rechnungen mit den Grundrechenarten durchgeführt werden. Hierzu werden Rechenblöcke in den Programmfluß eingesetzt. Das Handling ist recht gewöhnungsbedürftig. Leider ist die Anzahl der Container in einem Programm auf 3 begrenzt.

Es ist ebenso möglich durch Verzweigungen nebenläufige Stränge zu erzeugen, allerdings sind keine direkten Synchronisationsmechanismen vorhanden. Es kann aber über die Container und die Warteblocks eine einfache Synchronisation erreicht werden.

Leider läßt die Umgebung syntaktisch fehlerhafte Verknüpfungen der Funktionsblöcke zu, so daß erst zum Zeitpunkt des Überspielens der Programme auf den RCX Fehler angezeigt und repariert werden müssen. Ebenso sind die Editiermöglichkeiten relativ eingeschränkt und ein Umbau oder die Erweiterung eines Programmes ist recht langwierig.

Die Installation wird mittels eines Standard-Windows-Installationsprogrammes durchgeführt. Die deutschsprachige Version des Robolab besitzt leider nur deutschsprachige Menüleisten. Alle weiteren Texte, vor allem die Hilfestellungen und Erklärungen der Programmsymbole sind nur in englischer Sprache vorhanden.

²⁵Die Existenz von absoluten Sprüngen kann dazu führen, das die Programme nicht mehr der Strukturierten Programmierung (siehe Abschnitt 2.2.2) — der sorgfältigen Schachtelung von Programmierkonstrukten/-blöcken — entsprechen und sogenannter Spagetticode, also unentwirrbarer unlesbarer Code entsteht.

5.2 Textuelle Programmierung

Die textuelle Programmierung ist die klassische Form der Programmierung. Die Quellen der Programme sind in der Darstellung deutlich kompakter als die der visuellen Programmierung. Aus diesem Grunde ist es bei einer strukturierten Vorgehensweise mit Hilfe der textuellen Programmierung deutlich einfacher komplexe Aufgaben zu lösen — dies bedeutet allerdings nicht, das man während des Entwurfs und der Planung eines Programmes auf grafische Darstellungen verzichten kann.

Im folgenden werden zwei Programmierumgebungen für Dialekte der Sprache C auf dem RCX vorgestellt. NQC beruht auf der gleichen Firmware wie schon das RIS und das Robolab und LegOS stellt ein komplettes eigenständiges Betriebssystem für den RCX bereit.

5.2.1 NQC

‘Not Quite C’ oder kurz NQC²⁶ ist ein kommandozeilenorientierter Compiler der einen Dialekt der Sprache C in den Pseudocode der Standardfirmware verwandelt. Dieser Pseudocode kann nun über den IR-Tower auf den RCX übertragen werden und dort von der Firmware ausgeführt werden. NQC ist die leistungsfähigste Sprache der auf der Firmware basierenden vorgestellten Sprachen.

NQC stellt eine ganze Reihe der Standardkontrollstrukturen von C zur Verfügung. Dies sind die Konstrukte der bedingten Verzweigung und Wiederholung, lokale und globale Variablen, arithmetische Ausdrücke, Bibliotheksbefehle zur Steuerung und Einstellung der RCX-Hardware, Bibliotheksbefehle zur Steuerung der Taskverwaltung der Firmware und eine Art von erweiterten Makros (als Subroutine mit dem Schlüsselwort `function` bezeichnet). Im Zusammenhang mit der neuen Firmware²⁷, die als Entwicklungsversion von LEGO verteilt wird, stehen auch Konzepte des prioritätsgesteuerten Hardwarezugriffs und der ereignisgesteuerten Programmierung bereit.

Was NQC fehlt ist die Möglichkeit rekursiver Funktionsaufrufe, denn das leistet die Firmware nicht²⁸. Ebenso ist die zur Verfügung stehende Anzahl an Variablenspeicherplätzen recht begrenzt²⁹, so daß umfangreiche interne Landkarten und ähnliche Experimente nicht möglich sind.

²⁶NQC ist eine Entwicklung von Dave Baum und unter <http://www.enteract.com/~dbaum/nqc/index.html> zu finden. Die Software ist frei unter der Mozilla Public License (MPL) und wird aktiv weiterentwickelt.

²⁷Zur Zeit in der Version 3.28 vorliegend und von der offiziellen LEGO Mindstorm Website <http://mindstorms.lego.com> unter dem Stichwort „Hacker Info - RCX 2.0 BETA SDK“ zu beziehen.

²⁸In der Firmware ist kein Stack für das Ablegen von Rücksprungadressen oder lokalen Variablen vorhanden.

²⁹32 globale Variable, 16 lokale Variable

Einzige Datenstruktur ist die Variable. Dies sind 16 bit Intergerzahlen. Mit der neuen Firmware sind auch Arrays fester Länge als Datenstruktur möglich.

Mit den vorhandenen Möglichkeiten ist man aber in der Lage mit NQC durchaus komplexe Steuer- und Regelungen, subsumption architectures und einfache Automaten zu realisieren. Ebenso sind parallel laufende Task richtig implementiert und es kann eine Kommunikation zwischen diesen Task eingerichtet werden, so daß Themen aus dem Bereich der Parallelverarbeitung behandelt werden können.

Die Handhabung von NQC wird durch integrierte Entwicklungsumgebungen (IDE's) wie z.B. das RCX Command Center (RCXCC)³⁰ sehr komfortabel. Ebenso wird die Fehlersuche durch die Werkzeuge des RCXCC zur Variableninspektion stark vereinfacht. Es können hier im laufenden Betrieb via Infrarotschnittstelle sowohl die Ein- und Ausgänge des RCX überprüft werden, als auch die Variableninhalte gelesen und beschrieben werden. Ebenso können einzelne Task gestartet und gestopt werden.

Da das System aus NQC und einer Entwicklungsumgebung nicht als geschlossene Lösung daher kommt, ist die Installation mit vielen Freiheitsgraden ausgestattet. Man kann und muß sich seine individuelle Entwicklungsumgebung zusammensetzen; aus dem Compiler NQC in einer seiner Versionen, einer passenden Firmwareversion von LEGO Mindstorm und einer IDE oder auch eines einfachen Editors mit entsprechenden Batchprogrammen. Die Installation von z.B. RCXCC mit der alten Firmware erfolgt weitgehend mit einem Standard-Windows-Installationsprogramm. Erst wenn die neue Firmware eingesetzt werden soll muß ein erhöhter Aufwand getrieben werden (siehe auch Anhang F), der aber auch durch deutlich mehr Möglichkeiten zur Programmierung belohnt wird. Zur Verwaltung der Quelltexte lassen sich beliebige Verwaltungsstrukturen einsetzen von einfachen Ablagesystem in (geschützten) Verzeichnisse bis hin zu Versionsverwaltungen. Da das System aus einfachen Komponenten besteht, die weitgehend austauschbar sind besteht ein hoher Grad an Anpassungsfähigkeit auf die lokalen Begebenheiten (z.B. Netzwerkbetrieb oder nicht, Wahl des Editors).

³⁰RCXCC ist eine Entwicklung von Mark Overmars und unter <http://www.cs.uu.nl/markov/lego/> zu finden. Leider wurde die Weiterentwicklung von Mark Overmars eingestellt, jedoch ist der Sourcecode unter der Mozilla Public License (MPL) veröffentlicht worden. Eine Nachfolgesoftware stellt Visual NQC 2001 von Ronald Strijbosch dar, die sich zur Zeit in der Betaphase befindet (<http://home.hetnet.nl/~myweb1>).

5.2.2 LegOS

Da LegOS³¹ nicht auf der Standardfirmware beruht fallen viele Beschränkungen der vorherigen Programmierumgebungen fort. LegOS ist eine Sammlung von Bibliotheken und ein multitaskingfähiger preemptiver Betriebssystemkern, der direkt auf der RCX-Hardware läuft.

Als Entwicklungsumgebung wird der Crosscompiler GNU C eingesetzt. Dies bedeutet, daß der volle Sprachumfang eines ANSI C Compilers zur Verfügung steht, mit allen guten Seiten und allen Stolperfallen. LegOS benutzt die Speicherverwaltung von C, man hat also die Möglichkeiten mit Pointern, verketteten Listen und echten Datenstrukturen wie Verbänden (**struct** bzw. Rekords) und Vektoren (Arrays) zu arbeiten. Es gibt auch für die Anzahl der Variablen nur die Einschränkung der Größe des RAM im RCX. Da LegOS einen multitaskingfähigen preemptiven Betriebssystemkern bereitstellt, gibt es natürlich auch Tasks und eine Interprozesskommunikation mittels echter Semaphore. Auch Ereignissteuerung — das Warten auf das Erreichen eines bestimmten Sensorwertes, oder das Ablaufen eines Timers — ist vorhanden. Ebenso existiert ein einfaches infrarotbasiertes Netzwerkprotokoll, daß es ermöglicht mehrere RCX miteinander kommunizieren zu lassen³².

Die Möglichkeiten LegOS umfassen damit alle im Abschnitt 2.2 angesprochen Informatikthemen und durchaus auch Teile der KI, wie kognitive Landkarten, evolutionäre/genetische Algorithmen³³, usw. Nach oben sind nur durch die Hardware Grenzen gesetzt. Unter LegOS hat man die vollständige Kontrolle über den RCX. Es ist z.B. ohne weiteres möglich für eigene Sensorik auch eigene Treiber nahtlos in das Betriebssystem einzubinden und in die Sprache zu integrieren.

Im Vergleich zu NQC (mit der neuen Firmware) fehlt bei LegOS der priorisierte Zugriff auf Motoren bzw. Ressourcen. Es fehlen auch Tools zur Introspektion des laufenden RCX, so daß eine Fehlersuche auf Ausgaben auf das RCX-Display (oder Nutzung des Infrarot-Netzwerk-Protokolls) beschränkt bleibt.

Da LegOS streng genommen nur ein System von Bibliotheken für einen Cross-Compiler darstellt, ist zur Installation ein erhöhter Aufwand nötig. Zunächst muß der GNU-C-Compiler als Cross-Compiler installiert und dann die entsprechenden Bibliotheken eingespielt werden. Dann ist es notwendig eine Entwicklungsumgebung (mindestens einen Editor und einige Batchda-

³¹LegOS ist ein OpenSource-Projekt das von Markus L. Noga (www.noga.de/legOS/) initiiert wurde. Mittlerweile ist die Weiterentwicklung und Pflege an SourceForge gegangen und unter <http://legos.sourceforge.net> zu finden.

³²In der Praxis stellt die Überlagerung der Infrarotsignale der einzelnen RCX allerdings ein Problem dar, so daß die Kommunikationshäufigkeit stark eingeschränkt sein sollte, da sonst zu viele Fehler auftreten.

³³Siehe hierzu z.B. die Master Thesis von Jesper Rasmussen unter <http://www.daimi.au.dk/~repsej/thesis.html>

teien) zu installieren und zu konfigurieren. Für die Installation des Compilers und der Bibliotheken gibt es ein Projekt in dem ein Standard-Windows-Installationprogramm bereitgestellt wird, ebenso sind im Internet Anleitungen zur Installation vorhanden³⁴.

Für LegOS liegt zur Zeit noch keine IDE vor, daher ist der Umgang mit dem System von Haus aus zunächst nicht sehr bequem, aber durch konfigurierbare Editoren/IDE's sind sehr komfortable Umgebungen erstellbar.

5.3 Tabellarischer Vergleich

Dem tabellarischen Vergleich in Tabelle 1 auf Seite 27 sei vorangestellt, daß es keinesfalls das einzige Kriterium für oder gegen eine Programmierumgebung darstellt wieviele der angegebenen Punkte erfüllt wurden, sondern das es viel wichtiger ist, daß die Programmierumgebung in das Umfeld in dem sie eingesetzt werden soll auch hineinpasst.

Der Abschnitt *Bewertung*³⁵ in der Tabelle berücksichtigt Wertungen von gut (++) über neutral (0) bis schlecht (--). Mit dem Kriterium *Installation* ist vor allem die Einfachheit der Installation selbst und die Einfachheit der Informationsbeschaffung zur Installation gemeint. Unter *Komfort beim Betrieb* ist sowohl das Erstellen und Verändern von Programmen, als auch das Handling der Programme und des Übertragens auf den RCX zusammengefaßt. Die *Dokumentation Umgebung* bezieht sich auf die Bedienung der Entwicklungsumgebung. Die *Dokumentation Sprache* bezieht sich auf die Darstellung der Syntax und Semantik der Sprache und auf Beispielprogramme, hierbei ist für NQC und LegOS die Sekundärliteratur [Knudsen1999, Baum2000, BaumGasperiHempelVilla2000] berücksichtigt worden. Unter *Quelltextverwaltung* verstehe ich die Ablage der erstellten Programme und deren Dokumentation. Hier ist für Schulen besonders die Verwaltung mehrerer Teilnehmer oder verschiedener Aufgabenstellungen, die Bewertung *frei* bedeutet das ein beliebiges System verwendet werden kann, da die Umgebungen einerseits kein System enthalten aber auch nicht die Benutzung eines Systems verhindern.

5.4 Zusammenfassung

Die Möglichkeiten der Programmierung des RCX sind sehr umfassend. Ausgehend von den beiden visuellen Programmierumgebungen RIS und RoboLab als Einstiegsumgebungen, weitergehend mit NQC als einer 'richtige' textuellen Programmiersprache mit den sehr spannenden Erweiterungen der Ereignissteuerung und dem prioritätsgesteuerten Hardwarezugriff bis hin zum vollständigen Sprachumfang von ANSI C mit LegOS bietet sich ein breites

³⁴Z.B. die Seite von Rossz Vmos-Wentworth unter <http://www.jps.net/rossw/legos/>

³⁵Java als Referenzsprache wird hier nicht bewertet, da die Ausprägungen in der verschiedensten Umgebungen zu unterschiedlich sind.

Portfolio an Möglichkeiten an. Während RIS schnell an seine Grenzen stößt stellt der Einsatz von LegOS allerdings schon recht hohe Anforderungen an die Fähigkeiten der Lernenden. Es ist wohl sinnvoll hauptsächlich die beiden Umgebungen Robolab und NQC einzusetzen, wobei NQC hier deutlich mächtigere Konzepte bietet. LegOS ist allerdings die wohl mächtigste und flexibelste Programmiersprache für LEGO Mindstorms.

	RIS	Robolab	NQC	LegOS	Java
Art der Umgebung					
textuell			*	*	*
visuell	*	*			
Datenstrukturen					
globale Variable		*	*	*	*
lokale Variable			*	*	*
Integerwerte		*	*	*	*
eindimensionale Arrays			*	*	*
mehrdimensionale Arrays				*	*
zusammengesetzte Strukturen				*	*
dynamische Datenstrukturen				*	*
Pointer				*	
Objektorientierung					*
Programmierkonstrukte					
Anweisungsfolge	*	*	*	*	*
absoluter Sprung		*		*	*
bedingter Sprung	*	*	*	*	*
Wiederholungen (Zählschleife)	*	*	*	*	*
bedingte Wiederholung	*		*	*	*
endabweisend	*		*	*	*
eingangsabweisend	*		*	*	*
Funktionen als Codeblöcke	*		*	*	*
Funktionen mit Parametern				*	*
Rekursion				*	*
Parallele Ausführung	*	*	*	*	*
Taskkommunikation			*	*	*
per globale Variable			*	*	*
per Semaphore				*	
gegenseitiger Ausschluß				*	*
Prioritätsverteilung			*	*	
bei Tasks			*	*	
bei Zugriff auf Ressourcen			*		
Ereignisbehandlung			*	*	
Fehlersuche					
Introspektion			*		*
Breakpoints					*
Single Step					*
Bewertung					
Installation	++	++	+	--	/
Komfort beim Betrieb	+	+	+	0	/
Dokumentation Umgebung	++	+	0	0	/
Dokumentation Sprache	0	+	++	+	++
Quelltextverwaltung	0	0	frei	frei	/
Lizenzmodell	kommerziell	kommerziell	freie Lizenz	freie Lizenz	/
Betriebssystem	Windows	Windows MacOS	Windows MacOS Linux	Windows Linux	/

Tabelle 1: Vergleich der Programmierumgebungen

6 Durchführung

Nach der Hard- und Software für das Roboterlabor stellt die Durchführung eines Kurses den dritten wichtigen Teil dar. Basierend auf den 6.270 Kursen am MIT wird das Labor als ein Wettbewerb durchgeführt. Die Teilnehmer arbeiten in Gruppen zusammen an jeweils einem Roboter und bekommen zu Beginn des Kurses eine Aufgabe für einen Wettbewerb gestellt. Diese Aufgabe soll nun gemeinschaftlich im Verlauf des Kurses gelöst werden und am Ende des Kurses treten die Gruppen mit ihren Lösungen im Wettbewerb gegeneinander an.

Ein idealer Wettbewerb läßt unterschiedliche Lösungen des Problems zu, denn so wird vermieden, daß man wieder bei einer ‘single answer’-Aufgabe angelangt³⁶ und es kann eine ganze Vielfalt an Robotern entstehen und das Roboterlabor und der Wettbewerb werden zu einem interessanten Ereignis auch für andere nicht direkt am Roboterlabor beteiligte³⁷.

6.1 generelle Hinweise

Für den Erfolg des Labors gibt es einige Hinweise, mit deren Hilfe Stolpersteine überwunden werden können³⁸.

6.1.1 Spielregeln

Die Spielregeln für den Abschlußwettbewerb sollten einfach und übersichtlich sein. Es sollten nicht zu viele Spezialpunkte enthalten sein, sondern nur die elementaren Regeln. Alles weitere sollte durch eine Schiedsperson im Bedarfsfall entschieden werden. Ein Beispiel für ein sehr knappes aber ausreichendes Regelwerk³⁹ ist in der Abbildung 6 zu sehen.

³⁶Siehe auch Abschnitt 2.1.3

³⁷Die Abschlußwettbewerbe des Roboterlabors der FH-Hamburg finden beispielsweise immer in einem öffentlichen Raum statt und es werden Zuschauer eingeladen dem Wettbewerb beizuwohnen. Dies ist eine gute Möglichkeit Öffentlichkeitsarbeit zu leisten.

³⁸Siehe auch Anhang C

³⁹Entnommen aus der Website zum Software Engineering Wettbewerb der Firma iXL unter <http://www.bussetech.com/lego/mindstorms/>

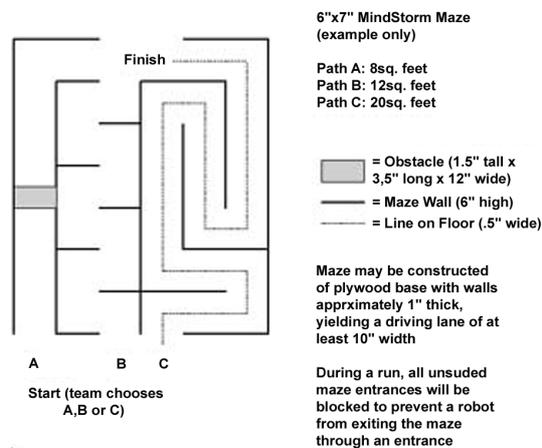


Abbildung 6: Beispielregelseite eines Wettbewerbs

Wichtig für den Wettbewerb ist die rechtzeitige Festlegung der Regeln, da sonst die Konstruktionen den sich ändernden Anforderungen hinterherlaufen und so viel Zeit verloren geht⁴⁰. Ebenso wichtig ist die Überprüfung der Regeln durch einen Zwischenwettbewerb. Im Fall von Schwierigkeiten mit den Regeln, sollten gegebenenfalls auch Anpassungen möglich sein. Dazu kann eine Regelgruppe aus Teammitgliedern verschiedener Teams gebildet werden, die dann die Änderungen beschließt. Der Einsatz einer Regelgruppe aus Teammitgliedern birgt allerdings auch Gefahren in sich, die Gruppenzusammensetzung des Roboterlabores muß hierfür geeignet sein. Als Alternative bietet sich ein Schiedsrichter an, der die Regeländerungen nach bestem Wissen und Gewissen verkündet und somit bindende Entscheidungen trifft. Zu guter Letzt ist es sinnvoll vor der Ankündigung der Regeln, selbst eine Konstruktion eines Robots für den Wettbewerb durchgeführt zu haben. Dies muß nicht in aller Vollkommenheit geschehen, sondern kann auch nur prototypisch erfolgen.

6.1.2 Zwischenwettbewerbe

Die Wahrscheinlichkeit, daß am Ende eines Kurses funktionsfähige Roboter entstanden sind, die die gestellte Aufgabe im Wettbewerb lösen können wird durch geeignete Zwischenwettbewerbe deutlich erhöht. Die Zwischenwettbewerbe sollten die Funktionsfähigkeit der Roboter für eine Teillösung der späteren Wettbewerbsaufgabe überprüfen. Dies führt dazu, daß das Ziel des Wettbewerbs nicht aus den Augen verloren wird und das eventuelle Probleme frühzeitig erkannt und beseitigt werden können. Ebenso wird mit der

⁴⁰Die sich ändernden Anforderungen stellen allerdings ein Standardproblem in der Softwareentwicklung dar, insofern ist es für Studenten eine interessante Übung ihre Architektur auf unvorhergesehene Änderungen auszurichten und flexibel zu halten.

Durchführung der Zwischenwettbewerbe die Notwendigkeit deutlich, rechtzeitig vom Bau der Roboter zur Programmierung der Roboter zu wechseln.

6.1.3 Knowledgebase

In einem Laborbuch können gut die gewonnenen Erkenntnisse der laufenden Labore eingetragen werden. So lernen die Schüler von den Schülern der vorhergehenden Labore und die Aufgaben können im Schwierigkeitsgrad variiert werden. Die Existenz eines geführten Laborbuches fördert auch sehr das explorative Lernen und gibt eine weitere Möglichkeit eigenständig Wissen zu sammeln. Zudem vertieft das Eintragen von Erkenntnissen in das Laborbuch das erworbene Wissen und schult die Fähigkeit das Wissen anderen mitzuteilen.

Als Mittel für das Laborbuch bietet sich ein einfacher Ringordner an, in den die Beiträge einsortiert werden können (gruppiert nach verschiedenen Themen wie Konstruktion, Programmierung, Sensorik, Aufgabenstellungen, weitere Quellenangaben). Als Hightech-Alternative könnte auf dem Webserver der Institution ein interaktives Web⁴¹ entstehen, in dem die Erkenntnisse gesammelt werden⁴². Allerdings birgt ein solches Hightech-Laborbuch die Gefahr, das es ob des erhöhten Aufwandes nicht gepflegt und so nutzlos wird.

6.2 exemplarischer Wettbewerb

Als Verdeutlichung der Prinzipien eines Roboterlabors folgt nun ein exemplarischer Wettbewerb. Dieser Wettbewerb ist in seiner Form auf Einsteiger in das Thema ausgerichtet. Er bietet aber durchaus genügend Herausforderungen und Erweiterungsmöglichkeiten (z.B. in der Variation mit Kollision Abschnitt 6.2.2) um auch für Fortgeschrittene interessant zu sein. Der Wettbewerb ist so ausgelegt, das er mit dem Material⁴³ eines LEGO Mindstorm Robotics Invention System pro Team erfolgreich durchgeführt werden kann. Als Programmierumgebung ist NQC (Abschnitt 5.2.1) zu empfehlen.

⁴¹Interaktiv sollte das Web-System insofern sein, als das es auf einfache Weise möglich sein sollte Beiträge im System zu verfassen und beispielsweise mit Fotos einer Digitalkamera aufzubereiten. Auch die Kommentierung bestehender Beiträge ist ein sehr wertvolles Mittel zur aktiven Beschäftigung mit den Inhalten. Eine technische Lösung wäre der Einsatz eines WikiWebs [LeufCunningham2001]. Eine freie Implementierung eines WikiWebs ist z.B. unter <http://phpwiki.sourceforge.net/> zu finden.

⁴²Ein schönes Beispiel für ein solches „Laborbuch“ stellt die Website des Convict Episcopal de Luxembourg <http://www.convict.lu/Jeunes/RoboticsIntro.htm> dar.

⁴³Für die zweite Variante, der Kooperation ist ein weiterer Motor notwendig, um die Übergabe des Gegenstandes durchzuführen. Hierzu ist z.B. der Mikromotor von LEGO geeignet.

6.2.1 einfaches Rennspiel auf Linienfolgerbasis

Ziel des Wettbewerbs ist, so schnell wie möglich vom Startpunkt des Spielfeldes (siehe Abbildung 7) an der schwarzen Linie entlang bis zum Endpunkt zu gelangen.

Spielregeln Im folgenden die Grundregeln für das Spiel:

- Fahre auf dem Spielfeld (Abbildung 7) vom Startpunkt zum Endpunkt (Abbildung 9).
- Nutze die schwarze Linie als Leitweg. Es darf nicht blind über das Spielfeld auf die andere Seite gefahren werden⁴⁴.
- Der Start erfolgt durch Betätigung der Starttaste am Roboter.
- Der Roboter muß am Endpunkt selbsttätig die Motoren abstellen und die benötigte Zeit anzeigen.
- Die Zeitnahme erfolgt mittels der eingebauten Uhr der Roboter durch eine im Rahmenprogramm vorgegebene Anweisungsfolge (siehe Anhang D).
- Nach dem Start ist kein Eingreifen durch einen Menschen erlaubt (die Hände müssen von Spielfeld entfernt werden).
- Alles weitere regelt die Schiedsperson. Ihr Wort ist bindend.

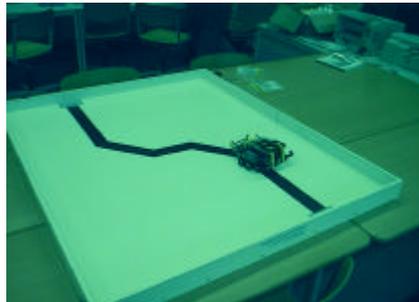


Abbildung 7: Spielfeld für den Linienfolgerwettbewerb

Spielfeld Das Spielfeld besteht aus einer einfachen beschichteten Spanplatte. Es ist ca. 1 m x 1,20 m groß und umrandet mit einer etwa 5 cm hohen Bande. Die Linie besteht aus aufgeklebtem (schwarzen) Klebeband. Als Endpunktmarkierung wird ein Stück Aluminiumfolie aufgeklebt (zum Beispiel mit einem Fotokleber). Diese Art der Markierung (Klebeband und

⁴⁴Anmerkung: Diese Regel existiert nur, um die Programmierung von 'festen Wegen' (siehe Abschnitt C) zu verhindern.

Fotokleber) läßt sich vom Spielfeld wieder entfernen und erlaubt so die einfache Wiederbenutzung des Spielfeldes für andere Aufgabenstellungen. Gleichzeitig ist die Markierung belastbar genug um ein Labor und den Wettbewerb unbeschadet überstehen zu können⁴⁵, zu dem ist die Unterscheidbarkeit zwischen Linie (schwarz), nicht Linie (weis) und Endmarke (silber) für die Sensorik der Roboter (nach unten gerichteter aktiver Lichtsensor, Abbildung 8) sehr gut.



Abbildung 8: Endpunktmarkierung mit Lichtsensor

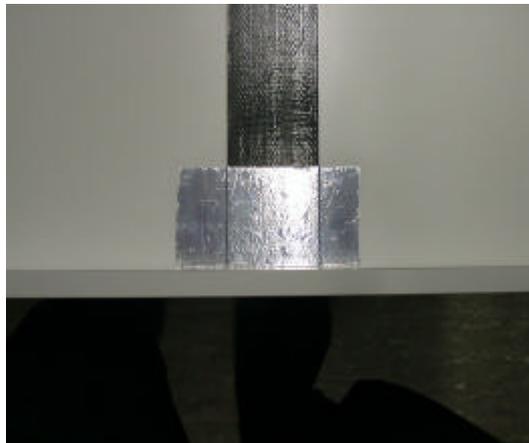


Abbildung 9: Aufsicht auf die Endpunktmarkierung

⁴⁵Experimente mit mittels Folienstiften oder Wandmarkern aufgemalten Linien haben gezeigt, daß diese Linien von den Robotern schnell verkratzt und verwischt werden.

6.2.2 Variation mit Kollision

Eine Variation des Linienfolger Rennspiels ist es, wenn zwei Teilnehmer gleichzeitig an den Start gehen. Der eine startet vom Endpunkt des jeweils anderen. Beim Verfolgen der Linie müssen sich die beiden Roboter zwangsweise irgendwo an der Linie begegnen. An dieser Stelle müssen sie die Kollision erkennen und einander umfahren.

Damit die Roboter sich umfahren können, muß zu den Spielregeln eine Ergänzung erfolgen:

- Die Roboter fahren (wie im Straßenverkehr) auf der rechten Seite der Linie entlang.
- Es muß eine Absprache über die Lage der Stoßstangen (Bumper) erfolgen, so daß sich die Roboter nicht ineinander verhaken.



Abbildung 10: Zwei Roboter bei der Kollision

Anmerkungen Um diese Aufgabe zu lösen, muß die Sensorik der Roboter gegenüber dem einfachen Rennspiel des vorherigen Abschnitts erweitert werden. Mindestens um einen Frontsensor (Bumper) der die Kollision erkennt. Wenn nach der Kollision kein fest programmierter Weg⁴⁶ durchfahren werden soll, dann müssen auch an der Seite der Roboter geeignete Sensoren zum Erkennen eines anderen Robots vorhanden sein. Zudem ist es notwendig zwei unterschiedliche Verhalten — das Linienfolgen und das Ausweichen — in dem Roboter zu programmieren und zu koordinieren.

In Testläufen mit diesem Wettbewerb hat sich herausgestellt, daß viele Einsteigerteams an der Aufgabe scheitern oder das, wenn die Roboter es bis zur Endmarke schaffen viel Glück dabei war. Insofern ist dieses Spiel trotz der nur scheinbar kleinen Änderung der Regeln eher für Fortgeschrittene geeignet⁴⁷.

Besonders schwierig an dieser Aufgabe ist der Zeitraum zwischen dem Verlassen der Linie und dem Wiederaufsetzen auf die Linie zur Fortsetzung

⁴⁶Feste Wege stellen durchaus ein Problem dar, siehe auch Anhang C.

⁴⁷Aufgrund dieser Erfahrung ist die Variation mit Kooperation entstanden.

der Linienverfolgung. Während dieser Zeit ist der Roboter ohne die feste Führung der Linie und bewegt sich durch unsicheres Gebiet.

Auch das Wiederaufsetzen auf die Linie an sich ist ein hoher Schwierigkeitsgrad. Häufig stossen die Roboter nach dem Ausweichen senkrecht auf die Linie und der Linienfolgeralgorithmus muß eine solche Situation erkennen oder aber man muß ein Übergangsverhalten definieren zwischen dem Ausweichen und dem Linienverfolgen.

6.2.3 Variation mit Kooperation

Aufbauend auf den Robotern der Variation mit Kollision des vorherigen Abschnittes wurde in einem Testlauf des Wettbewerbs die Variation mit Kooperation entwickelt. Hier sollen jeweils zwei Roboter zusammenarbeiten, um einen Gegenstand vom Anfangspunkt der Linie bis zum Endpunkt zu transportieren. Dazu müssen die Roboter mit Übergabemechanismen und Auffangbehältern ausgestattet werden⁴⁸.

Der Ablauf des Spieles ist wie folgt. Auf dem einen Startpunkt steht ein Roboter beladen mit einem Gegenstand (z.B. einem Zylinder oder einem Ball). Auf dem gegenüberliegenden Startpunkt steht ein weiterer Roboter, dieser ist unbeladen. Beide Roboter werden gleichzeitig gestartet und bewegen sich an der Linie entlang aufeinander zu. Sobald sich die Roboter treffen kommunizieren sie via Infrarotschnittstelle miteinander und koordinieren die Übergabe des Gegenstandes. Nach erfolgter Übergabe verabschieden sich die Roboter voneinander und fahren wieder zurück zu ihren jeweiligen Ausgangspunkten.

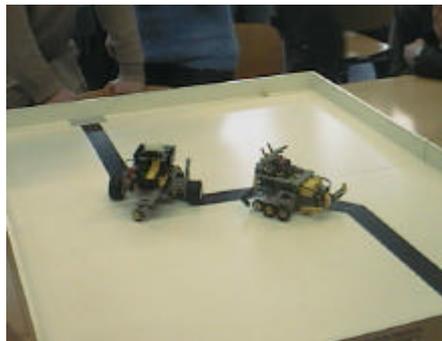


Abbildung 11: Zwei Roboter bei der Übergabe

⁴⁸Zum Betreiben des Übergabemechanismus muß zusätzliche Hardware beschafft werden. Ein LEGO Mikromotor ist für solche Mechanismen gut geeignet.

6.3 Informatikkonzepte in den exemplarischen Wettbewerben

Die exemplarischen Wettbewerbe wurden so ausgewählt, um unterschiedliche Informatikkonzepte aufeinander aufbauend nutzen zu können.

6.3.1 Inhalte der einzelnen Wettbewerbe

Zunächst werden die Konzepte, die in den einzelnen Wettbewerben auftauchen besprochen.

Zwei-Punkt-Regler Der Linienfolger ist mittels eines einfachen Zwei-Punkt-Reglers ausführbar. Das Grundprinzip des Zwei-Punkt-Reglers besteht darin, zwei Grenzwerte festzulegen bei deren Überschreitung eine korrigierende Aktion stattfindet. Im Fall der Linienverfolgung sind die Grenzwerte das Messen von Weiß (ganz neben der Linie) bzw. das Messen von Schwarz (ganz auf der Linie). Wenn man rechts neben der Linie fahren möchte, so steuert man auf der Linie (Schwarz) nach rechts und neben der Linie (Weiß) nach links.

Um einen Zwei-Punkt-Regler zu programmieren bedarf es lediglich der Konzepte des Befehls, der bedingten Verzweigung und dem der Wiederholung (siehe auch das zweite Codebeispiel im Anhang D). Dies ist einfache imperative Programmierung und somit gut als Einstieg geeignet.

Proportionalregler Um die Roboter schneller an der schwarzen Linie entlang zu führen, ist der Einsatz eines Proportionalreglers geeignet. Dieser sorgt für kleinere 'Überschwinger' über die Ideallinie und somit für eine größere Geschwindigkeit in die gewünschte Richtung entlang der Linie.

Beim Proportionalregler wird nicht wie beim Zwei-Punkt-Regler zwischen zwei diskreten Punkte operiert und mit zwei diskreten Antworten reagiert, sondern es wird versucht eine Korrektur in linearer Abhängigkeit von der Abweichung durchzuführen. Es wird hierzu die integrierende Eigenschaft des Lichtsensors ausgenutzt. Der Lichtsensor mißt, wenn er sich über der Kante der schwarzen Linie befindet einen Mittelwert zwischen Schwarz und Weiß (Grau). Genau auf diesen Mittelwert wird die Regelung abgestellt. Es wird die Differenz zwischen dem gewünschten Sollwert (Grau) und dem gemessenen Istwert ermittelt und proportional zu dieser Differenz eine Stellgröße festgelegt und auf die Motorausgänge gelegt (siehe auch das Codefragment für einen Proportionalregler im Anhang D auf Seite 44). Der Proportionalregler kann auch ohne die theoretischen Hintergründe von Steuer- und Regelungstechnik programmiert werden, die notwendigen Parameter (Proportionalfaktor bzw. Verstärkung, Clipping und Offset) werden dann mittels Experiment ermittelt.

Um einen Proportionalregler zu programmieren benötigt man zusätzlich zu *Befehl*, *bedingter Verzweigung* und *Wiederholung* noch das Konzept der *Variablen*, denn mit der Variablen werden Berechnungen durchgeführt und die Ergebnisse später im Programm weiterverarbeitet.

Kollision und Ausweichen Bei der Aufgabenstellung der Kollision mit Ausweichen werden an die Reaktionen des Roboter zwei unterschiedliche Aufgaben gestellt. Einseits muß eine Linie verfolgt werden, andererseits muß auf eine Kollision mit dem Gegenverkehr geachtet werden. Hierbei hat die Kollision mit dem Gegenverkehr die höhere Behandlungspriorität. Ebenso muß zwischen mindestens zwei Verhalten umgeschaltet werden. Dem des Linienfolgens und dem des Ausweichens.

Zur Lösung dieses Problems ist es sinnvoll, die verschiedenen Aufgaben von unterschiedlichen parallel laufenden Tasks behandeln zu lassen. Die Tasks an sich und die Synchronisation der Tasks stellen hier die neuen Konzepte dar.

Es ist aber auch möglich die einfache Struktur des Linienfolgers als Zwei-Punkt-Regler so umzubauen, daß als weiterer Zweig neben der Erkennung von Schwarz oder Weiß, die Erkennung der Kollision eingesetzt wird.

Kooperation In der Kooperationsaufgabe ist die Kommunikation das zentrale Thema. Die notwendigen Protokolle zwischen den Robotern müssen vereinbart werden und auf diesen Protokollen müssen Schwierigkeiten wie die Robustheit gegenüber Fehlern und die Fehlerkorrektur beherrscht werden.

6.3.2 Inhalte des Ensembles der Wettbewerbe

Im Ensemble der drei Wettbewerbe stecken die Prinzipien der Modularität und der Wiederverwertung von Material. Dies gilt sowohl für das Roboterchassis, als auch für die Programmierung des Roboter. Ebenfalls ist die Notwendigkeit für Gruppeninteraktion steigend.

So wird das Chassis des ersten Wettbewerbes in den beiden folgenden weiterverwertet und mit weiteren Fähigkeiten ausgerüstet. Zunächst wird im zweiten Wettbewerb der Frontsensor und im dritten Wettbewerb der Übergabe- und/oder der Auffangmechanismus angesetzt.

Auf Seiten der Software wird der Linienfolger-Algorithmus des ersten Wettbewerbs durchgehend weiterverwandt. Im zweiten Wettbewerb kann die Struktur des Linienfolger-Algorithmus entweder erweitert werden, um die Kollision zu behandeln oder aber der Linienfolger-Algorithmus wird als Subroutine oder Task wiederverwandt. Im dritten Wettbewerb kann auch die Kollisionserkennung weiterbenutzt werden, es muß nur eine andere Reaktion auf die Kollision eingefügt werden (nicht Ausweichen, sondern Übergabeprotokoll ausführen).

Die Gruppeninteraktion erfolgt dreistufig. Zunächst muß garnicht interagiert werden. Das Rennspiel kann jede Gruppe für sich entwerfen. Bei der Kollision ist es notwendig sich auf mechanische Eigenschaften der Roboter zu einigen und bei der Kooperation muß das Protokoll zwischen den Robotern festgelegt werden.

7 Schlußbemerkung

Anhand der Roboterprogrammierung lassen sich sehr weitgehend die verschiedenen Konzepte der Informatik untersuchen und erlernen. Angefangen von der einfachen imperativen Programmierung, bis hin zu Parallelprogrammierung und Themen der künstlichen Intelligenz. Ebenso wird der Bereich des Software Engineering abgedeckt und die Soziabilität der Teilnehmer gefordert. In Kombination mit dem Roboterbau und weiteren Experimenten im Bereich der Sensorik und Aktorik lassen sich gleichzeitig Themen wie die Grundprinzipien der Konstruktion und Meßtechnik erforschen. All die Themen des Roboterbaus und der Programmierung werden als Mechatronik bezeichnet und finden in unserer automatisierten Gesellschaft immer stärkere Verbreitung.

Trotz der Freiheiten, die ein Roboterlabor zum explorativen Lernen bietet ist es doch ratsam gerade für jüngere Teilnehmer eine gut strukturierte Lernwelt zu schaffen, in der die Teilnehmer geführt werden, ohne an einem zu eng gestecktem Plan vom selbständigen Erkunden des Stoffes abgehalten zu werden. Diese Gradwanderung zwischen Freiheit und Führung stellt durchaus Ansprüche an die begleitenden Lehrkräfte. Mit zunehmendem Alter der Teilnehmer tritt im Roboterlabor das freie explorative Lernen und Arbeiten in den Vordergrund.

Ich hoffe, mit dieser Arbeit den Einstieg in das interessante Thema eines Roboterlabores erleichtert zu haben. Ein solches Labor bietet in einer spannenden Lernumgebung vielfältige Möglichkeiten, über die informationstechnische Grundbildung hinausgehende Konzepte der Informatik zu vermitteln.

A Vergleich der Steuerrechner

Zur Übersicht folgt hier in kurzer Vergleich der verschiedenen Steuerrechner.

HANDYBOARD	FISCHERTECHNIK	LEGO MINDSTORMS
PROZESSOREN UND SPEICHER		
Motorola 68HC11 2 MHz - 32 kByte RAM 2x16 Zeichen LCD	Intel 80C32 24 MHz 64 kByte EPROM 32 kByte RAM -	Hitachi H8 16 MHz 16 kByte ROM 32 kByte RAM kleines LCD
EIN- UND AUSGÄNGE		
4 pulswellenmodulierte Motorausgänge 1 Servomotorausgang 7 Analogeingänge (8 Bit A/D-Wandler) 9 Digitaleingänge 1 Eingang für modulierten IR-Empfänger 1 Ausgang für modulierten IR-Sender - freier Erweiterungsbus	4 bipolare digitale Motorausgänge - 2 Analogeingänge (10 Bit A/D-Wandler) 8 Digitaleingänge - - - -	3 pulswellenmodulierte Motorausgänge - 3 Analogeingänge (10 Bit A/D-Wandler) - - - Infrarotkommunikation zwischen zwei RCX -
AUSSCHLUSSVERGLEICH		
 <p>Die Anschlüsse sind sehr kleinteilig und empfindlich. Durch die enge Anordnung sind Kurzschlüsse möglich.</p>	 <p>Die Anschlüsse sind nicht verpolungssicher.</p>	 <p>Durch das IBM Patent sind die Anschlüsse sehr robust, verpolungs- und kurzschlußsicher.</p>

Tabelle 2: Anschlußvergleich

C Fallen und Fallstricke

LEGO-Getriebemotoren können Ihnen eine Regelung zerstören.

Durch das hohe Drehmoment bei niedrigen Leistungen sorgen diese Motoren dafür, das ein Leistungswechsel an den Motoren (einzige Möglichkeit der Steuerung) bei Fahrzeugen ohne genügende Reibungen keine nennenswerten Drehzahländerungen hervorrufen. Damit ist es nicht möglich einen genügend großen Aussteuerbereich zu erlangen und ein Regeln hat keinen Effekt. Bei einer solchen Konstruktion können nur noch Zweipunktregelungen eingesetzt werden. Also Ein-/Ausschalten der Motoren. Teilweise ist es sogar erforderlich eine leichte gegenläufige Ansteuerung einzusetzen.

Ebenfalls ist in der Standardfirmware⁴⁹ das Problem vorhanden, daß ein Motor nicht auf eine bestimmte Geschwindigkeit gestellt wird, sondern nur mit einer bestimmten Menge Energie versorgt wird. Wenn also der Motor z.B. auf die Motorleistung 0 gesetzt wird, so kann er sich durchaus noch drehen, da er frei läuft. Nur im `output_mode off` wird der Motor tatsächlich gebremst⁵⁰.

Programmierte Wege gehen schief. Einige Schüler/Studierende versuchen eine Lösung aufgrund von in den Roboter einprogrammierten Wegen anstatt mit echten Regelungen zu erstellen. Dies geht in aller Regel schief, denn durch die wechselnden Batteriespannungen und Umweltbedingungen ist die Wiederholgenauigkeit der Roboter nicht hoch genug um feste Wege immer gleich auszuführen. Die entstehende Frustration bei den Teilnehmern ist sehr kontraproduktiv.

Infrarot Störungen Bei der Kommunikation von mehreren Robotern über Infrarotschnittstellen kann es durch die Überlagerung der Signale zu Störungen kommen. Die Signalhäufigkeit sollte so klein wie nötig sein, so daß die Wahrscheinlichkeit einer Überlagerung so gering wie möglich ist.

Ebenfalls können die meist als Infrarotsensoren ausgeführten automatischen Schärfereinstellungen moderner Foto- oder Videokameras oder eine tief stehende Sonne Störungen verursachen. Es ist ratsam während eine Wettbewerbes das Publikum auf die Problematik hinzuweisen und darum zu bitten keine Aufnahmen zu machen, sondern einen getrennten Aufnahmelauf einzuplanen.

Einfache berührungslose Sensoren beruhen ebenfalls oft auf Infrarotmessungen. Diese Sensorik ist natürlich ebenso von der Problematik der Störungen betroffen.

⁴⁹zur Zeit Version 3.09

⁵⁰Durch Kurzschliessen der Motoranschlüsse und damit unter Ausnutzung der Lenzschen Regel durch den selbsterregten Strom.

D Codebeispiele

Die hier angeführten Codebeispiele wurden mit NQC 2.2 und der Firmware 3.28 programmiert und getestet. Rahmenprogramm mit Zeitmessung:

```
#define LEFT  OUT_A // Definition der Motorenanschluesse
#define RIGHT OUT_C // Definition der Motorenanschluesse

int time; // globale Variable zur Zeitmessung

task main()
{
  // Vorbereiten der Zeitmessung
  time=0;
  ClearTimer(0);

  // Hauptteil des Programms
  //
  // An dieser Stelle befindet sich der Programmcode zur Bewaeltigung der Aufgabe
  //
  // Ende des Hauptteils des Programms

  // Ziel erreicht, Motoren abschalten
  Off(RIGHT+LEFT);

  // Zeitmessung vornehmen und die benoetigte Zeit auf dem Display ausgeben.
  time=Timer(0);
  SelectDisplay(DISPLAY_USER);
  SetUserDisplay(time,0);

  // Endlos warten, damit die Zeit auch abgelesen werden kann.
  while(1);
}
```

Codebeispiel für einfache Zweipunktregelung mit Zeitmessung:

```
#define LIGHT_SENSOR_1 // Definition einiger Konstanten
#define LEFT OUT_A
#define RIGHT OUT_C

// Definition von Schwellwerten fuer die Steuerung
#define SCHWARZ 47 // gemessen 42
#define WEISS 49 // gemessen 52
#define SILBER 55 // gemessen 60

int time; // globale Variable zur Zeitmessung

task main()
{
    // Lichtsensor aktivieren
    SetSensor(LIGHT,SENSOR_LIGHT);

    // Motorrichtung und Motorkraft einstellen
    SetPower(LEFT+RIGHT,4);
    OnFwd(RIGHT+LEFT);

    // Vorbereiten der Zeitmessung
    time=0;
    ClearTimer(0);

    // Hauptteil des Programms

    // Hauptschleife des Programms, wird solange durchlaufen bis der Lichtsensor
    // auf die silberne Endmarke trifft
    while(LIGHT<SILBER){
        // Zweipunktregelung
        if(LIGHT<SCHWARZ){ // erster Punkt
            // Auf Schwarz; nach links drehen
            Off(LEFT);
            OnFwd(RIGHT);
        }else{
            if(LIGHT>WEISS){ // zweiter Punkt
                // Auf Weiss; nach rechts drehen
                Off(RIGHT);
                OnFwd(LEFT);
            }else{ // zwischen erstem und zweitem Punkt
                // Auf "Grau"; geradeaus
                OnFwd(LEFT);
                OnFwd(RIGHT);
            }
        }
    }
    // Ende Zweipunktregelung
}
// Ziel erreicht, Motoren abschalten
Off(RIGHT+LEFT);

// Ende des Hauptteils des Programms

// Zeitmessung vornehmen und die benoetigte Zeit auf dem Display ausgeben.
time=Timer(0);
SelectDisplay(DISPLAY_USER);
SetUserDisplay(time,0);

// Endlos warten, damit die Zeit auch abgelesen werden kann.
while(1);
}
```

Codefragment für einen Proportionalregler

```
#define LIGHT_SENSOR_1 // Definition einiger Konstanten
#define LEFT OUT_A
#define RIGHT OUT_C

// Definition von Schwellwerten fuer die Steuerung
// gemessen Schwarz 42
// gemessen Weiss 52
#define GRAY 47

// Definition von Konstanten der Regelung
#define PROPORTIONALFAKTOR 1
#define CLIPPING 8
#define OFFSET 8

//
// ... Teile geloescht
//

int diff;
// Codefragment fuer Proportionalregler
while(1){
    diff=(GRAY-LIGHT) * PROPORTIONALFAKTOR;    // Abweichung ermitteln

    if(diff < -CLIPPING){diff = -CLIPPING;} // Clipping negativ
    if(diff > CLIPPING){diff = CLIPPING;}    // Clipping positiv

    // Proportionalregelung
    if(diff<0){
        // zu weit im Wei"sen
        SetPower(RIGHT,OFFSET + diff);
        SetPower(LEFT,OFFSET);
    }else{
        // zu weit im Schwarzen
        SetPower(RIGHT,OFFSET);
        SetPower(LEFT,OFFSET - diff);
    }
}

//
// ... Teile geloescht
//
```

E Standard Softwarearchitektur des LEGO Mindstorm

Die RCX Software Architektur des LEGO Mindstorm besteht aus einer Reihe von Layern (siehe Abbildung 12 auf der nächsten Seite), die miteinander die Entwicklung, die Übersetzung und das Ausführen von Programmen auf dem RCX realisieren. Viele Programmierumgebungen nutzen Teile dieser Architektur und klinken sich an unterschiedlichen Stellen ein. Die Elemente dieser Architektur sind auf dem RCX die folgenden:

- Ein ROM mit hardwarenahen Routinen,
- die sich im RAM befindliche Firmware; ein Interpreter, der die Pseudocodeprogramme ausführt.
- Die sich ebenfalls in RAM befindlichen Pseudocodeprogramme (User Programme). Das sind die Programme, die wir schreiben, um den RCX dazu zu bringen, daß zu tun was wir wollen.

Auf der Seite des PC's gibt es die folgenden Elemente:

- Den Infrarot Tower. Das Stück Hardware das die Kommunikation mit dem RCX ermöglicht.
- Die Serielle Schnittstelle, um den IR Tower anzusprechen
- Die spirit.ocx, eine spezielle Bibliothek, die die softwareseitige Kommunikation mit dem RCX erledigt und die über die Firmware auf dem RCX direkte Steuerbefehle an die RCX Hardware senden kann.
- Die Pseudocode-Compiler. Die Software, die aus der für den Menschen lesbaren Darstellung der Programme (graphisch oder textuell) eine für die Firmware des RCX lesbare Darstellung (Pseudocodeprogramme) erzeugt.
- Nicht zu letzt, die Quelltexte zur Programmierung des RCX.

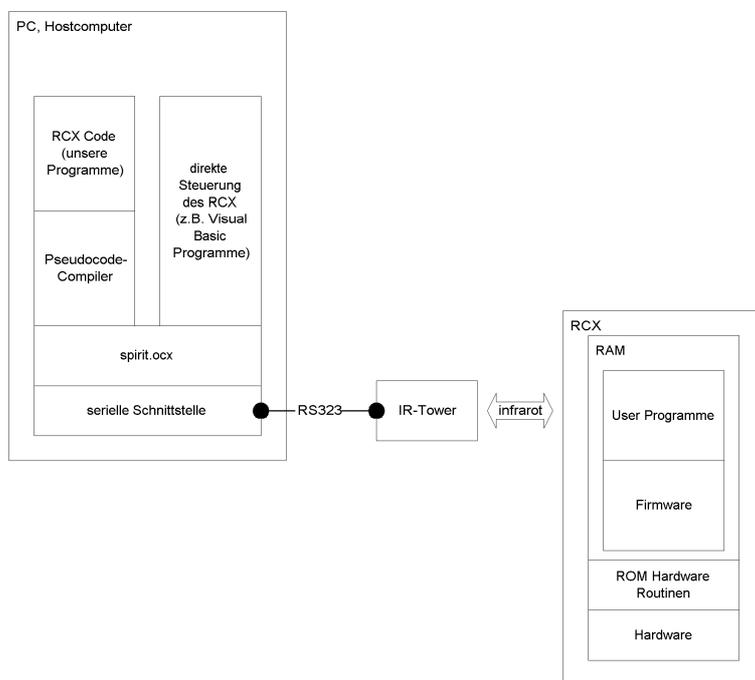


Abbildung 12: Softwarearchitektur des RCX mit Standardfirmware

F Installationshinweise NQC/RCXCC

Wenn das RCX Command Center in Zusammenhang mit der Firmware des RCX 2.0 Beta SDK eingesetzt werden soll, dann kann der Firmware Download nicht mit dem RCX Command Center stattfinden. Die Firmware muß mittels NQC geladen werden. Das notwendige Kommando ist:

```
nqc.exe -firmware D:\Programme\lego\rcc3\firm0328.lgo
```

Damit NQC beim Aufruf aus dem RCX Command Center auch Code für die neue Firmware erzeugt, muß eine Umgebungsvariable geeignet gesetzt werden. Dies kann mittels einer kleinen Batchdatei geschehen über die das RCX Command Center gestartet wird.

```
set NQC\_OPTIONS=-Trcx2  
start /w D:\Programme\lego\rcc3\RcxCC.exe
```

Zu beachten ist dabei, daß auch genügend Umgebungsspeicherplatz für die Variable bereitsteht.

G Internetquellen

Da die Verfügbarkeit von Informationen im Internet leider nicht immer gewährleistet ist und die Autoren der Seiten durchaus ihr Angebot auch manchmal entfernen, haben die Adressen und Informationen nur zum Zeitpunkt der Erstellung der Arbeit Gültigkeit. Für alle zitierten Internetquellen gilt, das sie zu diesem Zeitpunkt der Erstellung der Arbeit 3. Mai 2001 unter den angegebenen Adressen erreichbar waren ⁵¹.

Häufig ändern sich auch nur die Positionen innerhalb einer Domain, so daß zur Wiederauffindung der Quelle über statt der vollständigen Adresse wie <http://www.meinedomain.de/robot/index.html> nur der Domainname also <http://www.meinedomain.de> eingegeben werden sollte.

- LEGO Webseiten
 - Die Offizielle LEGO Mindstorms Webseite
<http://mindstorms.lego.com>
englischsprachig
 - LEGO Cybermaster + Mindstorms ROBOTICS-Forum
<http://f17.parsimony.net/forum30312/index.htm>
deutschsprachig
 - LUGNET - international LEGO Users Group Network
<http://www.lugnet.com>
englischsprachig

- Andere Anbieter
 - 6.270 - MIT's Autonomous Robot Design Competition
<http://www.mit.edu/courses/6.270/home.html>
englischsprachig
 - The Handy Board
<http://el.www.media.mit.edu/groups/el/projects/handy-board/>
englischsprachig
 - Fischertechnik
<http://www.fischertechnik.de>
deutschsprachig

⁵¹Unter der Adresse <http://www.stolt.de/lego> befindet sich eine aktuelle Linkliste mit den Verweisen dieser Arbeit.

Literatur

- [BaumGasperiHempelVilla2000] DAVE BAUM, MICHAEL GASPERI, RALPH HEMPEL UND LUIS VILLA. *Extreme Mindstorms: An Advanced Guide to LEGO Mindstorms*. Apress, 2000.
- [Baum2000] DAVE BAUM. *Definitive Guide to LEGO Mindstorms*. Apress, 2000.
- [Beck2000] KENT BECK. *Extreme Programming Explained, embrace change*. Addison Wesley, 2000. In deutscher Sprache [Beck2000ger]
- [Beck2000ger] KENT BECK. *Extreme Programming, Revolutionäre Methode für Softwareentwicklung in kleinen Teams*. Addison Wesley, 2000.
- [Braitenberg1984] VALENTINE BRAITENBERG. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, 1984.
- [Braitenberg1993] VALENTINE BRAITENBERG. *Vehikel: Experimente mit kybernetischen Wesen*. Rowohlt, Juli 1993. Im amerikanischen Original [Braitenberg1984].
- [Brooks1990] R.A.BROOKS. *A robust layered control system for a mobile robot*. in P.H. Winston und S.A.Shellard (hrsg.), "Artificial Intelligence at MIT Expanding Frontiers", Vol.2, MIT Press, 1990.
- [DrewsLehmann1998] CLEMENS DREWS, HOLGER LEHMANN. *KRABAT++*, *Ein Echtzeitbetriebssystem*. Diplomarbeit, Fachhochschule Hamburg, 1998.
- [DudekJenkinMiliosWilkes1996] G.DUDEK, M.R.M.JENKIN, E.MILIOS AND D.WILKES. *A taxonomy for multi-agent robotics*. Autonomous Robots, 1996.
- [DudekJenkin2000] G. DUDEK UND M. JENKIN. *Computational Principles of Mobile Robotics*. Cambridge University Press, 2000.
- [Everett1995] H. R. EVERETT. *Sensors for mobile Robots. Theory and Application*. A. K. Peters Ltd. 1995.
- [GeßnerNeumann1999] LENNART GESSNER, OLIVER NEUMANN. *Portierung eines embedded Linux-Systems und Erstellung einer IDE für das Krabat-Board*. Diplomarbeit, Fachhochschule Hamburg, 1999.
- [Harel1987] DAVID HAREL. *Statecharts: A Visual Formalism for Complex Systems*. Science of Computer Programming, Vol. 8, 1987.
- [Jadud] MATT JADUD. *TeamStorms as a Theory of Instruction*. 1999. <http://www.indiana.edu/~legobots/paperspres.html>

- [JonesFlynn1993] JOSEF J. JONES, ANITA M. FLYNN. *Mobile Robots. Inspiration to Implementation*. A. K. Peters Ltd. 1993.
- [Knudsen1999] JONATHAN B. KNUDSEN. *The Unofficial Guide to LEGO Mindstorms Robots*. O'Reilly, 1999. In deutscher Sprache [Knudsen2000].
- [Knudsen2000] JONATHAN B. KNUDSEN. *Das Inoffizielle Handbuch für LEGO MINDSTORMS Roboter*. O'Reilly, 2000.
- [LeufCunningham2001] BO LEUF, WARD CUNNINGHAM. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley 2001. Auch <http://www.wiki.org>
- [Martin1994] FRED GARTH MARTIN. *Circuits to Control: Learning Engineering by Designing LEGO Robots*. Arbeit zur Erlangung des PhD, MIT, Juni 1994. Auch <http://fredm.www.media.mit.edu/people/fredm/>
- [OMG1999] OBJECT MANAGEMENT GROUP OMG. *Unified Modelling Language Specification*. Version 1.3, 1999. Auch <http://www.omg.org/technology/uml/index.htm>
- [Papert1980] SEYMOUR A. PAPERT. *Mindstorms: Children Computers and Powerful Ideas*. Basic Books, 1980. Zweite erweiterte Auflage, 1993.
- [Thiele1999] KAI-MICHAEL THIELE. *KRABAT V*. Diplomarbeit, Fachhochschule Hamburg, 1999.
- [WitthöftHoffmann1997] KATJA WITTHFT, FRANK HOFFMANN. *KRABAT, Entwicklung eines erweiterbaren Mikrocontrollersystems als Grundlage für komplexe Robotikexperimente*. Diplomarbeit, Fachhochschule Hamburg, 1997.