Hauptseminar: Verbindungen RCX – PC

Frank Zimmermann
Matrikelnummer: 27596

1. Vorstellung des Robotik Invention System und der Hardware

1.1 Das Robotik Invention System (RIS) allgemein

Das Robotik Invention System (RIS) ist die Grundlage für das bauen und programmieren der Lego Roboter.
Das RIS wurde von Lego und dem MIT entwickelt um „intelligente“ Legosteine zu erzeugen.
Das System gibt es nun in der Version 2, in welcher die Software ein update erfahren hat.
Das RIS besteht aus 718 Lego - Teilen zum bauen des Roboters, 2 Motoren (mit integriertem Getriebe), 2 Berührungssensoren, 1 Lichtsensor und natürlich den RCX selbst, das Herzstück des ganzen.
Der RCX hat 3 Eingänge (Lichtsensoren, Temperatursensoren, Berührungssensoren, Drehsensoren usw.) und 3 Ausgänge für Motoren oder Lichter und dergleichen. Weiterhin hat der RCX einen IR Ein- bzw. Ausgang sowie ein Display und 4 Tasten zum starten der Programme oder dergleichen.
Das RIS 2.0 kostet zurzeit im Handel 239 €.

Abbildung 1 RCX mit Standard Peripherie

1.2. Die Hardware des RCX

Im RCX selbst befindet sich ein Hitachi H8 Prozessor, Serie H8/3297 Produkt Name H8/3292. Der Prozessor selbst hat 512 Byte RAM und 16KByte ROM welche beim ersten einschalten die Firmware per IR holt (auch 16KByte).
Der Prozessor ist mit 16 Mhz getaktet und läuft mit 5 Volt.

Der RCX besitzt weiterhin 2 Timer mit 8bit Auflösung und 1 16bit Timer, 8 8bit A/D Konverter und 43 I/O Pins.
Er besitzt weiterhin einen 32K Byte fassenden Speicher für seine Programme und Daten (Datalog).

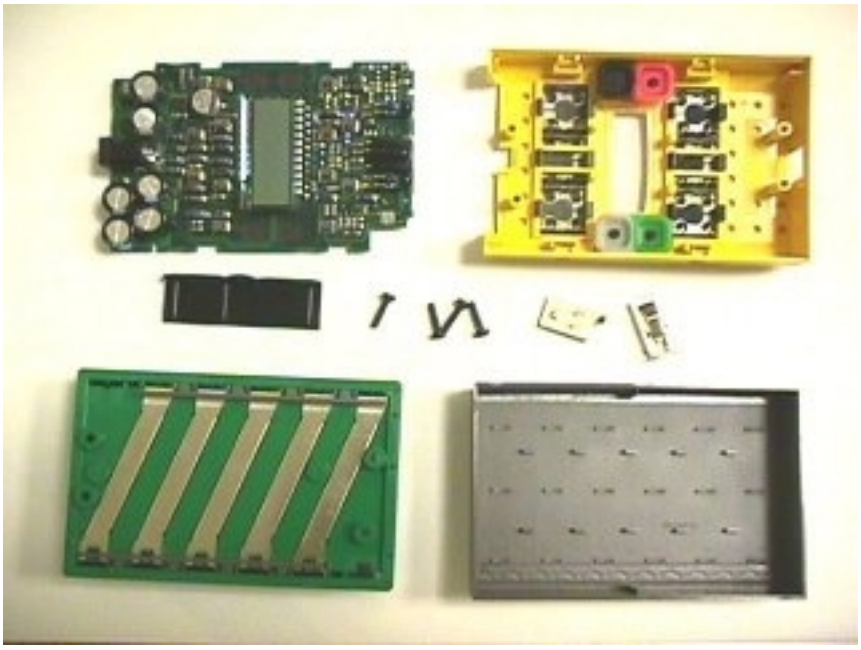

Abbildung 2 RCX mal aufgeschraubt

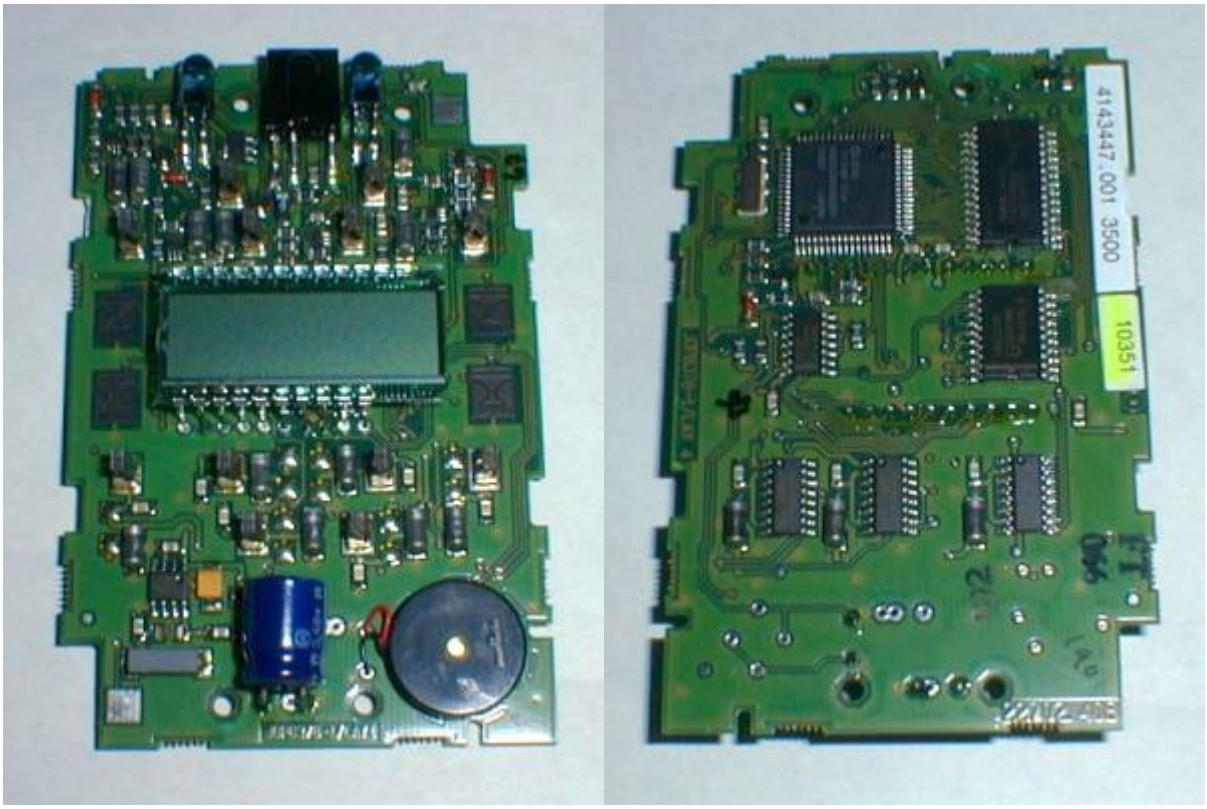

Abbildung 3 Platine im Detail

Die IR Kommunikation erfolgt bei 38khz und überträgt 2400 Baud.
Es ist die gleiche Technik die auch bei den normalen Fernsehfernbedienungen zum Einsatz kommt und so ist es möglich mit einer lernbaren Fernbedienung den RCX ebenfalls zu steuern.

1.3. Tricks um die Eingänge zu „vermehren“

Wen die 3 Eingänge nicht ausreichen kann man durch Tricks diese Eingänge auch „vermehren“ indem man unterschiedliche Sensoren auf einen Eingang schaltet und die Ergebnisse im Programm dann trennt. Z.B. wird der Wert des Lichtsensors immer in Prozenten angegeben wobei 100 das hellste ist und 0 eben kein Licht. In der normalen Umgebung wird es aber sehr selten bis nie geschehen, dass eine Lichtquelle 100 ist.
Also kann man den Berührungssensor drauf stecken.
Dieser liefert bei einer Berührung den Wert 100. Nun ist es also klar wenn im Programm eine 100 empfangen wird, kann es eigentlich nur der Berührungssensor sein.
Auch ein Trick ist es zwei Berührungssensoren auf einen Eingang zu legen und zwischen Sensor und Kabelende einen Widerstand einzubauen. Wenn man nun den Eingang so konfiguriert das er wieder Prozente angibt so kann man die 2 Sensoren unterscheiden.
Da sie ja jetzt unterschiedliche Widerstände haben, erzeugen sie auch unterschiedliche Werte. (also eine Art Frequenzmultiplexing).

2. Programmiermethoden im Überblick

2.1. Überblick über die Programmiersprachen

Für den RCX gibt es viele verschiedene Programmiersprachen. Doch nicht alle Programme, die man damit erzeugen kann, laufen auch wirklich sofort auf den RCX, denn nicht alle Sprachen erzeugen Code für das Lego Betriebssystem im RCX.
Man kann die Sprachen also einteilen in diejenigen, die man mit der Standard Firmware verwenden kann und in die, für die man ein neues Betriebsystem für den RCX benötigt. Sprachen bzw. Entwurfsysteme die die Standard Firmware verwenden sind z.B. RCX Code vom RIS, NQC und Sprachen die unter anderen Betriebssystemen laufen sind z.B. LegOS oder PBForth.

2.2. Programmiersprachen die auf eine andere Firmware aufsetzen

Das LegOS Betriebssystem bzw. die Programmiersprache gestatten ein besseres Speichermanagement und man hat mehr Möglichkeiten zum programmieren (wie unter C).
So unterstützt LegOS auch den Einsatz von Zeigern.
Der Compiler und der download zum RCX funktionieren per Textkommando. Also ein Einfaches klicken und das Programm ist drüben ist hier nicht möglich. (sicherlich wird es irgendwann auch bequemere Tools dafür geben)
Bei PBForth wird nicht ein kompiliertes Programm den RCX übergeben sondern man schickt die Befehle vom PC Terminal hin zum RCX und dieser interpretiert den Befehl und führt in dann aus. Das heißt, dass auf dem RCX ein Interpreter läuft, der diese Sprache interpretiert.
Bei interpretierbaren Sprachen wird sicher der ein oder andere auch an Java denken, und tatsächlich gibt es einen Java Interpreter der auf den RCX läuft (nur eine Teilmenge der Befehle wird unterstützt).

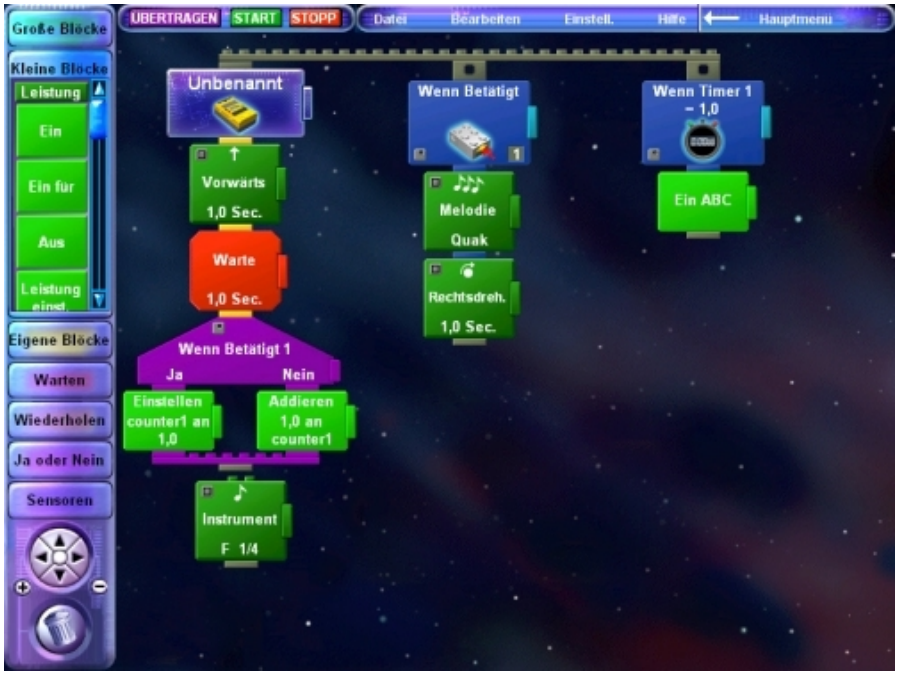

Abbildung 4 Robotik Invention System 2

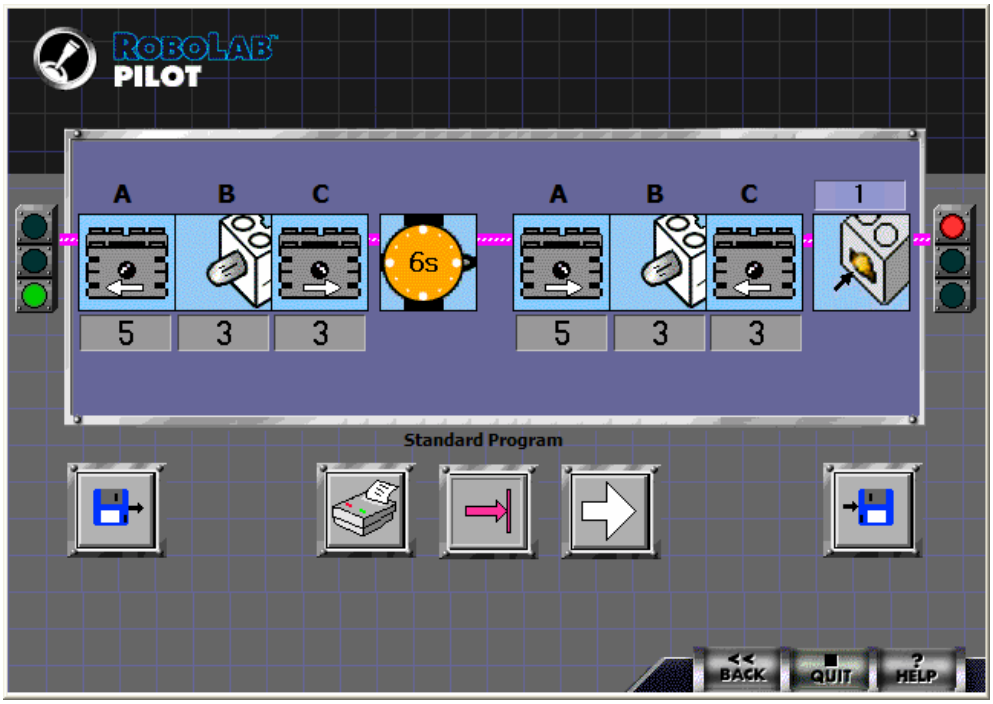

Abbildung 5 auch eine Entwicklungsumgebung für den RCX von Lego die aber hauptsächlich in Schulen während des Unterrichts verwendet wird (Grundlagen der Programmierung beizubringen)

2.3. Programmiersprachen die auf die Standardfirmware aufsetzen

Doch nun wieder zu den Programmiersprachen die die Lego Firmware benutzen.
Dies ist zum einen, wie schon oben erwähnt NQC und eben die ganze Sparte von Lego selbst.
Das RIS selbst wandelt das grafische Programm (RCX Code) in Lego P-Script um.
Dies ist eine Sprache die fast so aussieht wie C und die man auch per Hand programmieren kann. Wenn man einen Einblick bekommen will, wie das RIS die Sprache umwandelt, so

kann man einfach ein abgespeichertes Programm vom RIS per Editor öffnen und man hat den Lego P – Script Code.
Zum übertragen wird die Sprache umgewandelt zu LASM (Lego ASseMbler).
LASM ist ein Byte Code welche von einer virtuellen Maschine im RCX übersetzt wird.
Zu Lego P – Script Code oder zu LASM gibt es bei Lego im SDK2 eine sehr gute Dokumentation und ein Tool in dem man sein Programm laden und es komfortabel übertragen kann.
Weiterhin besteht die Möglichkeit auch per Active-X den RCX zu programmieren, doch dazu später mehr.
Wie schon erwähnt gibt es noch die NQC (Not Quite C) Sprache die Dave Baum entwickelt hat und sehr beliebt geworden ist um den RCX zu programmieren. Gestützt wird sich ebenfalls auf die Standard Firmware von Lego, das heißt man braucht kein Extra Betriebsystem wie es bei LegOS der Fall wäre.
NQC erzeugt ebenfalls Byte Code kann aber den RCX effizienter nutzen als die Lego Sprachen es tun.
Wie der Name schon sagt ist die Sprache an C angelehnt so dass C Programmierer schnell zurecht kommen werden. Natürlich steht nicht der gesamte C Umfang zur Verfügung, denn man muss ja die Möglichkeiten des RCX beachten.
Das Betriebssystem ist ja gleich geblieben.
Um nicht nur im Textmodus zu arbeiten gibt es verschiedene Entwicklungsumgebungen die ein komfortableres Arbeiten mit NQC ermöglichen.
Eine bekannte Umgebung ist z.B. RCX Command Center oder Brick Command Center.

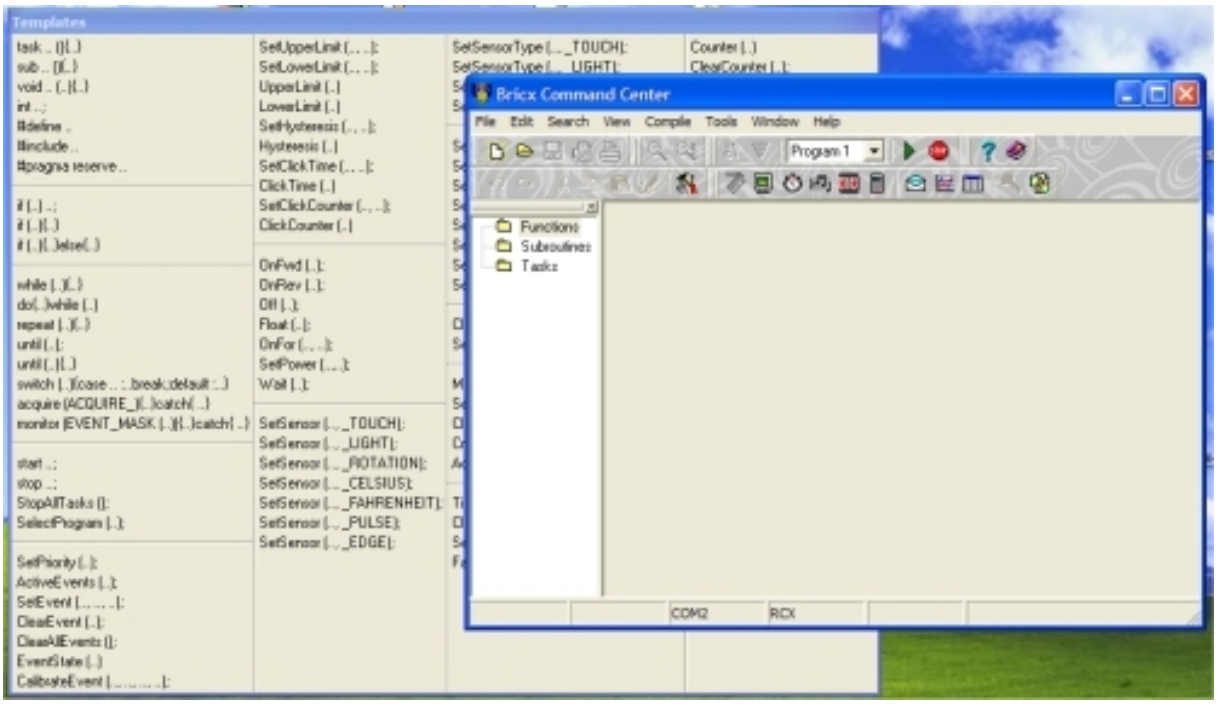

Abbildung 6 Brick Command Center

Natürlich ist es nicht möglich, auf alle oben genannten Verfahren genau einzugehen, besonders zu NCQ gibt es eine Unmenge von Material und Entwicklungsumgebungen im Netz.

3. Programmieren via ActiveX Control

3.1. Allgemeines zum ActiveX Control und Entwicklung

Ich möchte in diesem Kapitel darauf eingehen wie man mit Hilfe von Visual Basic Programme für den RCX schreiben kann bzw. wie man auf ihn zugreifen kann.
Natürlich ist Visual Basic nicht die einzige Sprache, bei der dies geht, man kann auch Visual C++ oder Delphi dafür benutzen. (Bsp. und Abbildungen stammen von Visual Basic 6 SP 5)
Von der RIS Version 1.0 bis 1.5 lag der CD-ROM eine Datei bei, die spirit.ocx hieß.
Dies war eine ActiveX Datei die man einfach in die Entwicklungsumgebung einbinden konnte und schon konnte man programmieren.
ActiveX ist eine 32 Bit Schnittstelle, ein Microsoft Standard, für Anwendungen, Internetwerkzeuge und Betriebssysteme. Ein ActiveX Control erkennt man an der Endung ocx. Diese Datei kann man in seine Programmierumgebung einbinden und sofort steht das Objekt bereit. (Standard Öffnen oder Speicher Dialoge sind ActiveX Komponenten usw.)
Also sind ActiveX Steuerelemente eigentlich kleiner Programme (Objekte) dessen Funktionalität man in seine Programme einbauen kann ohne das Problem selbst lösen zu müssen (wie man z.B. AVI Dateien abspielt usw.)
Ab Version 2 liegt die Datei nicht bei, doch das programmieren geht auch hier mit Basic und dergleichen.
Nur die Einbindung hat sich etwas geändert.
In Version 2 hat es sich deshalb geändert weil der RCX auch eine neue Firmware bekommen hat und daher der „Programmierschnittstelle“ mehr Befehle zur Verfügung stehen und ein neuer IR Tower (USB statt seriell) hinzugekommen ist, der ja auch korrekt angesprochen werden muss.
Das Programmieren mit der spirit.ocx funktioniert aber auch hier noch, nur man kann die neuen Befehle nicht nutzen.
Da aber die meisten Befehle ebenfalls in spirit.ocx enthalten sind, zeige ich alle Beispiele mit diesem ActiveX Modul. (weil man keine neue Firmware benötigt)
Doch nun erstmal zum einbinden der ocx Datei unter Basic.

3.2. Arbeiten mit der spirit.ocx

3.2.1. Installieren des ActiveX Controls

Zuerst legt man unter Visual Basic 6 ein neues Projekt an (Standard EXE), danach klickt man in der Menuleiste auf Projekt, Komponente und dann auf Lego PBrickControl.

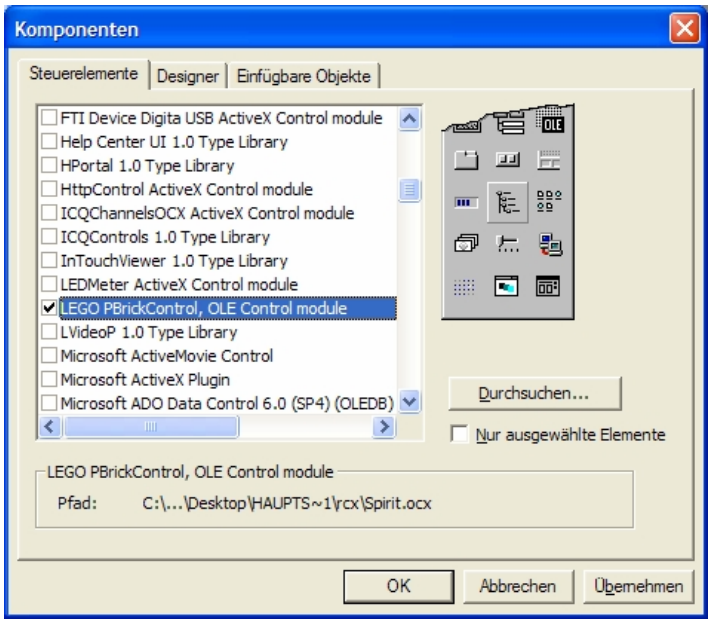

Abbildung 7 Komponente einfügen

Ist es korrekt eingebunden erscheint in der Toolbar ein neues Symbol mit einem Lego Logo. Dieses Steuerelement zieht man einfach auf das Formular.

3.2.2. Initialisieren des Com Port und schließen

Nun erstellen wir z.B. ein kleines Programm, womit man den RCX (Roverbot) per Mausklick fahren lassen kann. Bevor man mit den Ansteuerungen der Motoren oder dergleichen überhaupt beginnen kann, muss man den korrekten Com-Port ermitteln, an dem der Infrarot Sender angeschlossen ist und diesen Port initialisieren.
Am bequemsten löst man dies in der Form_Load Methode des Fensters.

```
Private Sub Form_Load()
Dim i As Integer
Dim port As Integer

For i = 1 To 4
    rcx.ComPortNo = i
    If rcx.InitComm = True Then
        port = i
        opt(i - 1).Value = True
    Else
        opt(i - 1).Enabled = False
        opt(i - 1).Value = False
    End If

Next i
rcx.ComPortNo = port
```

```
rcx.InitComm
End Sub
```

Natürlich muss man beim beenden des Programms des Com-Port auch wieder freigeben. Dies macht man mit einem CloseComm.
Doch nun zum eigentlichen Programm.

3.2.3. Ansteuerung der Ausgänge

Der RCX soll, wenn man die Taste gedrückt hält, eine bestimmte Bewegung machen und wenn man sie los lässt soll er stehen bleiben.
Für diese Art von Befehlen eignet sich gut das Mouse down bzw. Mouse up Ereignis in Basic.
Weiterhin muss man die Richtung der Motoren bestimmen in die sie sich drehen sollen. Hier nun der Quellcode für das rechts fahren des RCX.

```
Private Sub cmdrechts_MouseDown(Button As Integer, Shift As Integer, X As
Single, Y As Single)
rcx.SetFwd "0"
rcx.SetRwd "2"
rcx.On "02"
End Sub
```

Zuerst werden die Richtungen deklariert und die Liste der Motoren übergeben die diese Richtung fahren sollen. Genauso funktioniert das aktivieren der Motoren selbst, man übergibt einfach die Liste der Anschlüsse, die diesen Befehl ausführen sollen.
Die Ausgänge des RCX werden dabei von links nach rechts gezählt und beginnen bei 0.
Also ist der am weitesten links liegende Anschluss eine 0 und der rechte ist die 2.
Mit der Anweisung

```
Rcx.off „02“
```

werden die Motoren dann wieder deaktiviert.
Weiterhin kann man natürlich auch bestimmen, mit welcher Leistung die Motoren angesteuert werden sollen.
Doch ich lasse diesen Befehl aus Zeitgründen einfach weg, da sie auf dieselbe Weise wie oben genannte Befehle funktionieren.

3.2.4. Zugriff auf die Eingänge

Die Ansteuerung der Ausgänge ist nun klar.
Nun muss man natürlich die Eingänge überwachen um auf irgendein Ereignis reagieren zu können. Man muss am Anfang aber erst festlegen um was für eine Art von Sensor es sich handelt. Damit der RCX diesen auch richtig ansteuern kann.
Dies geschieht mit dem Kommando „SetSensorType“.
Einzelheiten des Befehls können sie in der Dokumentation nachlesen. Es würde jetzt zulange dauern jede Variable zu erklären und wäre sicherlich auch nicht das Ziel dieses Hauptseminars.
Danach muss man die Skala angeben auf der gemessen werden soll.
Die Rohdaten liegen in einem Bereich von 0 – 1023 vor.
Diese Rohdaten werden im RIS eigentlich nie genutzt, sondern immer umgewandelt in z.B. Boolean, Flankenzähler, Prozent, Fahrenheit, Celsius und Winkel.
Der Lichtsensor wird immer mit Prozent angegeben.
Das setzen dieser Skalen geschieht mit „SetSensorMode“.

Ist der Modus und der Sensor bestimmt, kann es losgehen. Das Abrufen geschieht mit einem „Poll“ Befehl.
Man kann damit nicht nur Sensoren abrufen sondern auch interne Variablen, Timer oder den Motor Status.
Je nach übergebenen Variablen bestimmt man, was man abrufen will bzw. an welchem Port der Sensor sitzt. Das abrufen der Sensoren habe ich im Bsp. Programm in einen Timer verpackt, der alle 100msec diese Aktionen ausführt. Zuvor überprüft er aber natürlich ob der RCX überhaupt noch da ist.

```
If rcx.PBAliveOrNot = True Then

    lbltext.Caption = rcx.Poll(9, 1)

    If rcx.Poll(9, 0) = 1 Then
        lbllinks(0).BackColor = &HFF&
    Else
        lbllinks(0).BackColor = &H8000&
    End If

    If rcx.Poll(9, 2) = 1 Then
        lbllinks(1).BackColor = &HFF&
    Else
        lbllinks(1).BackColor = &H8000&
    End If
End If
```

Je nach Input kann man nun die unterschiedlichsten Aktionen ausführen.
Hier habe ich mich nur darauf beschränkt, die Werte optisch im Programm wiederzugeben.
Schwierigkeiten bei Sensoren bestehen natürlich darin, den genauen Schwellenwert z.B. des Lichtsensors zu finden. Nimmt man nur die einfache Standardaufgabe von Lego den Roverbot auf einer schwarzen Linie fahren zu lassen.
Doch wie viel Prozent ist schwarz.
Und was ist, wenn man es unter Tageslicht machen will? Stimmen die Werte noch?
(meist nein…).

3.2.5. Erzeugen, speichern und lesen des Datalog

Natürlich hat der RCX mehr als nur Ein- und Ausgänge. Wie schon oben erwähnt besitzt er auch ein Datalog. Dies ist eine Art kleiner Speicher, in dem er Variablen oder Sensorwerte speichern kann.
Zuerst muss man ein Datalog erzeugen.
In meinem Beispiel erzeuge ich ein Datalog mit 2 Feldern und gebe anschließend per Klick auf eine Schaltfläche den aktuellen Lichtsensorwert in das Datalog.
Da zwei Felder deklariert sind, kann das Datalog auch nur 2 Werte aufnehmen. Ist man am Ende, so geschieht kein automatischer Wrap around sondern der „Zeiger“ bleibt stehen. Beim auslesen gibt man an, von wo man mit dem auslesen starten will und wie viele Felder man auslesen will.
Daraufhin bekommt man ein Array zurück, welches aus 3-mal x Felder besteht. X ist hier die Anzahl der Felder die man ausgelesen hat.
Im Feld 0, x steht, woher der Wert kam, also z.B. von einem Sensor von einer Variable, Timer usw. Im Feld 1, x steht, welche Variable es z.B. ist oder welcher Sensoreingang. In dem Feld 2, x steht nun der eigentliche Wert.
Im ersten Feld des Datalog (also 2,0) findet man die Größe des Datalogs.
Doch nun zu dem Programmtext, den ich im Beispielprogramm programmiert habe.

Setzen des Datalog

```
rcx.SetDatalog (2)
```

Werte setzen in das Datalog

```
rcx.DatalogNext 9, 1
```

Werte lesen aus Datalog

```
Dim arr As Variant

arr = rcx.UploadDatalog(0, 3)
If IsArray(arr) = True Then
    For i = LBound(arr, 2) To UBound(arr, 2)
        List1.AddItem Str(arr(0, i)) + Str(arr(1, i)) + Str(arr(2, i))

    Next i
End If
```

3.2.6. Programme komplett übergeben

Alle Befehle, die ich aufgeführt habe, bewirken eine unmittelbare Aktion. Dies ist, wenn man mit dem RIS programmiert, nicht der Fall.
Dort klickt man sich ein Programm zusammen, überträgt es auf dem RCX und startet es dann. Doch dies ist auch mit Visual Basic möglich.
Alle Programmausdrücke wie Schleifen, wenn-dann-Bedingungen sind auch hier möglich. Man kann ein Programm im Basic mit Hilfe dieser speziellen Befehle schreiben und dieses auf den RCX übertragen. Wenn man dann irgendein bestimmtes Ereignis hat z.B. der Lichtsensor hat einen bestimmten Wert so kann man dieses Programm ausführen. Man braucht also nur einen Befehl zu übermitteln, um eine ganze Reihe anderer Befehle anzustoßen. Dies ist besonders für Aufgaben interessant, die etwas zeitkritischer sind, da nur ein Befehl gesendet wird zur Ausführung mehrerer Tätigkeiten.
Auch das abbrechen des Programms ist wieder möglich.
Da diese Befehle keine Aufregenden Neuerungen bringen, werde ich sie an dieser Stelle weglassen. Wie geschrieben kann man Task bzw. Unterprogramme erstellen und diese dann ausführen (wie RIS).

3.2.7. Was es noch alles gibt

Nun noch schnell zu den anderen Befehlen, die ich einfach übersprungen habe, die aber auch interessant und Erwähnenswert sind, damit man sieht, wie viele Möglichkeiten der RCX mit den Standard Programmiersprachen bietet.
Man kann die Reichweiten des Towers verändern. Von short (schneller) auf long (langsamer bessere Fehlerkorrektur). Man kann sich Statistiken über die Verbindung anzeigen lassen, die Firmware neu übertragen, checken ob der Tower und das Kabel eingesteckt sind, die interne Softwareuhr des RCX stellen, Töne abspielen, Nachrichten an einen anderen RCX senden (man könnte eine Art Fußballspiel programmieren wo sich die RCX (Spieler) verständigen könnten), Motoren eine bestimmte Zeit laufen lassen, Task starten, beenden, löschen,

arithmetische und logische Operationen ausführen mit Variablen oder Sensorwerten, den Speicherstatus des RCX abrufen lassen usw.
Man sieht schnell, dass das Beispielprogramm nur einen Bruchteil der Befehle abdeckt.
Das Tolle an den ActiveX Control ist eben, dass man es perfekt in vorhandene Programmiersprachen einbinden kann und so alle Möglichkeiten von richtigen Programmen nutzen kann.
Ein interessantes Projekt, welches ich einmal gebaut habe, war ein RCX Roboterarm welcher von einem Visual Basic Programm gesteuert wurde. Das allein ist ja nichts neues, doch ich habe dazu eine Spracherkennungssoftware genutzt, um das Visual Basic Programm zu steuern (Tastenklicks im Programm auslösen). Das heißt also, ich konnte den RCX Roboterarm per Sprache steuern. Das bedeutet also, dass man durch geschicktes Verknüpfen von Programmen Konstruktionen erzeugen kann, die es im realen Leben noch gar nicht gibt bzw. die noch in der Forschung sind. (Oder gibt es schon einen Sprachgesteuerten Roboterarm zu kaufen??)
Hier, denke ich, wird das wahre Ausmaß der Flexibilität des RCX deutlich.

4. Vision Command

4.1. Vision Command vorgestellt

Das Vision Command Set kam einige Zeit nach dem RIS auf dem Markt.
Es beinhaltet eine „Lego Cam“, Lego Teile und Software. (99€)
Die Lego Cam ist eigentlich eine Logitech Cam. Welches Modell aber lässt sich schlecht sagen denn es ist sicherlich eine Spezialanfertigung für Lego.
Die Cam hat eine Auflösung von 352x288 Pixel (30 Frames pro Sekunde) und wird über ein 5 Meter langes USB Kabel mit dem PC verbunden.
Sie lässt sie auch prima als einfache Webcam einsetzen.
Zu dem Paket gehören noch 145 Legosteine und Software. Die Software läuft leider nur unter Windows 9x. Unter Windows 2000 bzw. unter Windows XP ist sie nicht zum laufen zu bekommen. Auch eine Anfrage an Lego selbst brachte keine Lösung.
Ein Update gibt es auch nicht. Zwar gibt es im Internet ein paar Tricks, aber dazu braucht man ein parallel Installiertes Windows 98 System usw.
Ein Teil der Software ist Standardsoftware für Webcams.
Sie umfasst das einfache Aufnehmen von avi Filmchen oder Bildern.
Auch Zeitrafferaufnahmen und dergleichen sind möglich.

4.2. RCX programmieren mit Vision Command

Der zweite Teil der Software ist der interessantere Teil. Mit ihm ist es möglich Programme für den RCX zu erstellen. Das Programm ist ähnlich aufgebaut wie die Standard RIS Software, so dass es keine Umstellungsprobleme gibt.
Als zentrales Element beim Programmieren dient natürlich die Kamera.
Sie ist der Auslöser für die Befehle die der RCX ausführen soll.

Es ist möglich, auf bestimmte Farben und Bewegungen zu reagieren. Dabei kann man angeben, in welchen Teil des Bildes nach der genannten Eigenschaft „gescannt“ werden soll.
Das Kamerabild lässt sich in verschiedene Bereiche einteilen.
In jedem Bereich kann die Kamera auf andere Farben, Lichter bzw. Bewegungen reagieren.
Lego hat verschiedene Raster vorgegeben, doch man kann durch einen Trick selbst Raster entwerfen, wie man das Kamerabild aufteilen will. (genaue Beschreibung siehe im Netz http://home.arcor.de/mindrobots/de/MindStorms/TippsTricks/VCBereiche/moin.htm)
Klickt man einen Bereich in der Software an, wird gefragt, auf was die Kamera reagieren soll (Bewegung oder Farbe bzw. Licht oder Dunkel).
Danach kann man den Bereich angeben, in welcher Spanne dieser Eigenschaft die Kamera reagieren soll (man hält zum Bsp. einen farbigen Ball vor die Kamera, damit man die Farbe bestimmen kann).
Danach kann man die Befehle wie gewohnt zusammenklicken (wie bei RIS).
Die Bildauswertung geschieht auf dem PC und so muss der RCX, wenn er die Kamera huckepack trägt, natürlich eine Schnur hinter sich herziehen. Erkennt der PC ein bestimmtes Ereignis, so übermittelt er die entsprechenden Befehle den RCX bzw. startet ein zuvor übertragenes Unterprogramm.

4.3. Probleme des Systems

Das Problem dabei ist, dass der ganze Vorgang sehr zeitaufwendig und rechenlastig ist. Denn die Kamera ist erst einmal per USB angebunden.
Das Bild muss erst einmal geholt und aufbereitet werden und dann durch die Erkennungssoftware von Vision Command „geschleust“ werden.
Danach wird der Befehl per Infrarot an den RCX übermittelt.
Jeder dieser Schritte, sei es nur die Übertragung zum RCX, dauert in der Regel relativ lang. Daher kommt es oft zu unerwünschten Ergebnissen bei dem Experimentieren. Z.B. wenn er sich auf einen Ball zu bewegen soll so ist der Ball meist schon woanders und der RCX fährt in die falsche Richtung, weil er noch die alten Befehle ausführt.
Das erkennen von Bewegung hat auch einige Tücken.
Wenn der RCX auf eine bestimmte Bewegung, im Bild, sich selbst bewegen soll, so bewegen sich ja relativ gesehen die Objekte vor der Kamera bei der Bewegung ebenfalls mit.
Dies erkennt der RCX dann wiederum als Bewegung und bewegt sich wieder.
Ebenfalls eine große Schwierigkeit ist die Erkennung der Farbe zum Beispiel.
Ändern sich die Lichtverhältnisse nur geringfügig, so erkennt die Kamera die Farbe meist nicht mehr.

Das Vision Command Set ist ein guter Ansatz, doch reagiert das ganze System nach meiner Auffassung nicht schnell genug und ist zu fehleranfällig.
Auch das Hinterherziehen des Kabels bei fahrenden RCX (Roverbot) ist problematisch, denn bei einer Wende kann sich der Roverbot leicht verheddern und man kann die ganze Navigation vergessen.

Das benutzen der Lego Cam unter Visual Basic 6 ist nicht ohne weiteres möglich. Man muss sich dazu ein SDK von Logitech runterladen welches einiges an Dokumentationen enthält und einige Beispiele. Natürlich beschränkt sich alles auf die reine Webcam.
Ein erkennen von Bewegungen und Farben wie in der Lego Software ist hier nicht möglich, weshalb ich auch nicht näher auf das SDK eingehen werde.

5. Quellenangabe

MINDSTORMS
Roboter Programmieren Lernen
Markt & Technik Verlag ISBN 3-8272-9081-3

NQC Homepage
http://www.enteract.com/~dbaum/nqc/index.html

Lugnet (größtes Forum im Internet zu RCX und Vision Command)
http://news.lugnet.com/robotics/

MindRobotic (kleineres Forum im Netz)
http://pub32.ezboard.com/bmindrobots

Roboter programmieren – Vorlesungsreihe von Axel-Tobias Schreiner
Fachbereich Mathematik-Informatik
Universität Osnabrück
http://www.vorlesungen.uni-osnabrueck.de/informatik/robot00/html/skript.html

The Lego RCX Design Page (Hardware von RCX)
http://www.rcx.ic24.net

RCX Programming (Survey)
http://prelude.psy.umontreal.ca/~cousined/lego/4-RCX/Survey/index.html

Lego Mindstorms Seite
http://www.legomindstorms.com

Controlling
LEGO® Programmable Bricks
Technical Reference (pdf Datei war im früheren SDK enthalten und beschreibt spirit.ocx)
http://mindstorms.lego.com/sdk/SDK.asp

SDK2 von Lego mit Tools und Dokumentation für LASM, Mindscript usw.
http://mindstorms.lego.com/sdk2