

VORLÄUFIGE VERSION – NICHT FERTIGGESTELLT

Robointerface – Basic

Ein Grundkurs zur Programmierung des Robointerface mit dem Basic-Compiler

Einleitung

Basic - ("Beginners All purpose Symbolic Instruction Code" = „symbolische Allzweck-Programmiersprache für Anfänger“) wurde ursprünglich als einfache einsteigs-Programmiersprache für Anfänger entwickelt. In den 80er Jahren war Basic sehr populär, wurde es doch mit den damals neu auf den Markt kommenden Homecomputern i.A. mitgeliefert. In dieser Zeit wurden auch die ersten Interfaces von fischertechnik mit Hilfe von Homecomputern in Basic programmiert. Um an diese „Tradition“ anzuknüpfen, wurde das RoboInterface-Basic entwickelt.

Das „EVA“ - Prinzip

Jedem Computerprogramm liegt ein Prinzip zu Grunde: Das „EVA“ - Prinzip.

Eingabe – **V**erarbeitung – **A**usgabe.

Es werden also Daten eingelesen, verarbeitet und ausgegeben. Eine Verarbeitung ohne Eingabe macht wenig Sinn, genauso wenig, wie Daten, die der Computer nach der Verarbeitung für sich behält ;-)

Das Robointerface unterscheidet sich allerdings in einem entscheidenden Punkt von einem handelsüblichen Computer: Die Ein- und Ausgabemedien sind nicht Tastatur und Bildschirm, sondern Taster-Eingänge (bzw. analoge Eingänge) und Motor-Ausgänge (bzw. Lampen-Ausgänge). Zwar gibt es die Möglichkeit, mit dem Robointerface-Basic auch Strings (Texte) zu verarbeiten und auch über die serielle Schnittstelle (RS232) zu kommunizieren, aber dies ist nicht die Hauptaufgabe eines Interface.

Die Basic-Programme können in jedem beliebigen Editor geschrieben werden und werden anschließend mit dem Compiler (= Übersetzer) in ein für das Interface ausführbares Programm umgewandelt. Eine ausführliche Anleitung, wie man Programme compiliert und zum Interface überträgt, befindet sich in der Bedienungsanleitung.

Auch zu beachten ist, dass es zur Zeit nur möglich ist, die Basic-Programme direkt in das Interface zu laden. Eine Simulation bzw. Ein Debugger mit PC-Unterstützung ist derzeit in Arbeit, ob und wann dieses Tool fertiggestellt wird, ist nicht sicher.

Ein erstes Programm

Üblicherweise wird bei Programmiersprachen als erstes Beispiel ein sog. „Hello world“ (= „Hallo Welt“) Programm vorgestellt. In Ermangelung des Bildschirms am Robointerface sieht das etwas anders aus, wir wollen einen Motor ein- und wieder abschalten. Hierzu schließen wir an O1 und O2 einen beliebigen Motor an.

```
main:
  motor 1,8
  waitms 500
  motor 1,0
```

Der Befehl motor benötigt zwei Parameter, die mit einem Komma getrennt werden: Zunächst die Nummer des Motors, der angesprochen werden soll und zuletzt die Geschwindigkeit und Drehrichtung. Es gibt 8 Geschwindigkeiten. Somit ist 8 die maximale Geschwindigkeit, 0 schaltet den Motor aus und negative Werte lassen den Motor in die andere Richtung drehen.

Gleich zu Beginn lernen wir auch den relativ wichtigen Befehl waitms kennen. Er wartet einfach ab, bis die

angegebene Zeit abgelaufen ist. Die Wartezeit wird in ms (Millisekunden = 1/1000 Sekunden) angegeben. In obigem Beispiel wartet das Interface also eine halbe Sekunde.

Also wird unser „Hello world“ Programm folgendes tun: Den Motor 1 mit maximaler Drehzahl einschalten und nach einer halben Sekunde wieder abschalten. Danach ist das Programm beendet.

Wichtig: Jedes Programm muss eine Sprungmarke (label) namens „main:“ enthalten.

Drehrichtung des Motors

```
main:
    motor 1,8
    waitms 500
    motor 1,0
    waitms 500
    motor 1,-8
    waitms 500
    motor 1,0
```

Das obige Programm ist nun um einige Zeilen vergrößert worden und wird nun folgendes tun: Den Motor 1 mit maximaler Drehzahl einschalten, nach einer halben Sekunde ausschalten. Nach einer weiteren halben Sekunde wird der Motor wieder eingeschaltet, allerdings mit umgekehrter Drehrichtung. Nach einer weiteren halben Sekunde wird der Motor abgeschaltet und das Programm beendet.

Die letzte Zeile `motor 1,0` könnte man in diesen beiden ersten Programmen auch weglassen, da bei Beenden des Programms automatisch alle Motoren bzw. Ausgänge abgeschaltet werden.

Eingänge

Nun genügt es ja nicht, Motoren zeitgesteuert ein- und auszuschalten, in den seltensten Fällen wird dies eine sinnvolle Steuerung ergeben. Ein einfacher Anwendungsfall wäre beispielsweise das Abschalten des Motors an einer Endstellung. Wir schließen für die weiteren Experimente einen Taster an I1 (Eingang 1) des Interface an (der Motor bleibt natürlich ebenfalls angeschlossen).

Um den Zustand eines Eingangs zu ermitteln, gibt es die `in()` Funktion. Funktion, das ist ein neues Wort – bislang haben wir nur mit Befehlen zu tun gehabt. Der Unterschied zwischen einer Funktion und einem Befehl besteht darin, dass eine Funktion ein Ergebnis zurückgibt – dies tut der Befehl nicht. Dies ist ja auch notwendig, denn wie sollte man vom Zustand des Eingangs erfahren? Eine Funktion erkennt man auch daran, dass die evtl. notwendigen Parameter in Klammern angegeben werden.

Die `in()` Funktion ergibt 0, wenn der dazugehörige Eingang nicht aktiv ist (z.B. Taster nicht gedrückt) und 1, wenn der Eingang aktiv ist (Taster gedrückt).

Um dies auf einen Motor zu übertragen, müsste man also das Ergebnis von `in()` mit 8 multiplizieren, um einen Motor mit maximaler Geschwindigkeit abhängig von einem Taster ein- und abzuschalten.

```
main:
    motor 1,in(1)*8
```

Würden man dieses Programm compilieren und ausführen lassen, würde folgendes passieren: Das Programm wird gestartet und ist gleich darauf beendet. Was ist passiert?

Ganz einfach: Der Programmablauf ist so schnell, dass man das Ergebnis am Motor gar nicht sehen kann. Man muss also dem Programm Gelegenheit geben, lange genug den Motor anzusteuern, dass man etwas sehen kann.

Programmiertechnisch macht man dies mit einer Schleife. Dies bedeutet, dass ein bestimmter Programmteil mehrfach ausgeführt wird. Eine Schleife kann endlos sein (wird also ewig wiederholt) oder wird durch eine entsprechende Abfrage beendet.

Beschäftigen wir uns zunächst mit einer Endlosschleife:

```
main:
    while 1
        motor 1,in(1)*8
    end
```

Wir haben nun zwei neue Befehle kennengelernt: „while“ und „end“. Im obigen Programm wurde der motor-Befehl auch zusätzlich eingerückt, dies macht die Zugehörigkeit des „end“ zum „while“. Der motor-Befehl ist der Inhalt der Schleife, der „while“-Befehl markiert den Anfang, während „end“ das Ende der Schleife markiert.

Zu „while“ gehört auch immer eine „Abbruchbedingung“. Was dies genau ist, dazu kommen wir später. Nur soviel: Durch die Angabe der „1“ ist diese Abbruchbedingung niemals gegeben und die Schleife wird endlos durchgeführt (bis das Programm im Interface gestoppt wird oder das Interface von der Spannungsversorgung getrennt wird).

Mit diesem einfachen Programm kann man nun den Motor 1 über den Taster 1 steuern – solange der Taster gedrückt ist, läuft der Motor, ist der Taster nicht gedrückt, wird der Motor gestoppt.

Da dieses Programm endlos läuft, muss es manuell gestoppt werden (über die Programm-Tasten am Interface oder über das Übertragungsprogramm) bevor wir weitermachen können.

Wir haben ja nun das Ziel, den Motor an einer Endstellung abzuschalten, nicht ganz erreicht. Schauen wir uns einmal an, wie man mit den neu kennengelernten Befehlen diese Funktion hinbekommt:

```
main:
    motor 1,8
    while in(1)=0
    end
    motor 1,0
```

Dieses Programm schaltet zunächst einmal den Motor ein. Anschließend haben wir die „while – end“ Schleife in einer bislang nicht gesehenen Form: In der Abbruchbedingung hinter while steht eine Gleichung. Der „while“ - Befehl reagiert nun so, dass er die Schleife so lange ausführt, wie die Gleichung „wahr“ ist – also stimmt. Dies ist in diesem Fall so lange so, wie der Taster an Eingang 1 nicht gedrückt ist. Weiterhin sehen wir, dass der end-Befehl direkt nach dem while-Befehl erfolgt. In diesem Fall wird also innerhalb der Schleife nichts gemacht – es wird also im Endeffekt nur gewartet, bis der Eingang 1 aktiv, d.h. der zugehörige Taster gedrückt wurde.

Zusammengefasst: Motor 1 wird eingeschaltet, sobald der Taster an E1 gedrückt wird, wird der Motor abgeschaltet. Hiermit ist die Aufgabe erfüllt.

Ansteuern von einzelnen Ausgängen

Bei der Ansteuerung von Motoren werden die Ausgänge immer paarweise benutzt, d.h. Motor 1 wird an O1 und O2 angeschlossen, Motor 2 an O3 und O4 usw.

Bei der Ansteuerung von Lampen, oder wenn Motoren nur in einer Drehrichtung benötigt werden, kann man Ausgänge sparen, indem man auf der einen Seite den Minuspol der Spannungsversorgung anschließt, und auf der anderen Seite einen einzelnen Ausgang.

Um dies zu testen, schließen wir einen Motor entsprechend an: Eine Seite an den Minuspol der Spannungsversorgung und die andere Seite an O8.

Um den Motor nun einschalten zu können, ohne auch gleichzeitig den Ausgang O7 zu beeinflussen, benötigen wir einen anderen Befehl: out.

```
main:
    out 8,8
    waitms 500
    out 8,0
```

Dieses Programm schaltet den Motor an Ausgang 8 ein (mit höchster Geschwindigkeit), wartet dann eine halbe Sekunde und schaltet dann den Motor wieder aus.

Impulse zählen

Um bei einem mobilen Roboter den zurückgelegten Weg oder bei einem Industrieroboter die Position einer Achse feststellen zu können, wird bei den fischertechnik-Robotern ein Impulsrad zusammen mit einem Taster verwendet. Die Anzahl der gezählten Impulse repräsentiert den zurückgelegten Weg bzw. die Position der Achse.

Für diesen Abschnitt bauen wir an einen S-Motor mit Getriebe ein Impulsrad und einen Taster an, der vom Impulsrad beim Drehen gedrückt wird. Der S-Motor wird an O1 und O2 angeschlossen, der Taster an I2. Ein weiterer Taster, der die Referenzposition anzeigen soll, wird an I1 angeschlossen.

Um weiter machen zu können, brauchen wir einen kleinen Exkurs in das Thema **Variablen**. Im Computerbereich kann man in Variablen Werte speichern und diese Werte jederzeit ändern. Natürlich kann man die zuvor gespeicherten Werte auch wieder auslesen.

Man kann sich das ungefähr so vorstellen, wie ein Regal mit vielen Schubladen. An jeder Schublade steht vorne der Name der Variablen drauf, in den Schubladen liegt jeweils ein Zettel, auf dem der aktuelle Wert der Variablen steht. Nehmen wir einmal an, wir hätten eine Variable „Position“ definiert. Dementsprechend gibt es nun eine Schublade, auf der vorn „Position“ draufsteht. Weist man nun der Variablen „Position“ den Wert

10 zu, wird in die Schublade „Position“ ein Zettel hineingelegt, auf dem „10“ draufsteht. Dementsprechend kann man jederzeit in die Schublade hineinschauen, um zu sehen, was auf dem Zettel steht.

Wie benutzt man nun Variablen?

Zunächst einmal ist es wichtig, dem Basic-Compiler mitzuteilen, dass man eine Variable nutzen möchte. Dies ist in Basic allgemein nicht üblich, normalerweise kann man in Basic jederzeit einfach eine Variable nutzen, sie wird automatisch definiert. Um aber den begrenzten Speicher des Interface optimal zu nutzen und auch die Abarbeitungsgeschwindigkeit entscheidend zu erhöhen, ist es günstiger, dem Compiler exakt mitzuteilen, was man mit einer Variablen vor hat.

Entscheidend für die Definition einer Variablen sind zwei Dinge: Der Name und der Typ. Der Name ist in weiten Grenzen frei wählbar, es muss lediglich darauf geachtet werden, dass eine Variable immer mit einem Buchstaben beginnt. Im weiteren Verlauf dürfen auch Zahlen vorhanden sein. Sonderzeichen dürfen nicht enthalten sein (außer dem Unterstrich `_`).

Zu den unterschiedlichen Variablentypen kommen wir später, zunächst benutzen wir den Typ „int“ = Integer (Ganzzahl). Auf diesem „Zettel“ ist Platz für Werte von -32768 bis +32767, das reicht für die meisten Projekte voll und ganz aus.

(Beispiel: Würde man die Position eines mobilen Roboters auf einen Millimeter genau bestimmen wollen, so würde dieser Zahlenbereich für mehr als 32 Meter ausreichen).

Die Definition von Variablen muss am Anfang des Programms vorgenommen werden. Im Folgenden ein Beispiel für die Definition einer Variablen und Zuweisung eines Wertes.

```
var position int

main:
    position = 10
```

Doch nun zurück zum eigentlichen Anliegen: Das Zählen von Impulsen.

Schauen wir uns einmal einen ersten Entwurf an. Das Programm soll den Motor einschalten und nach 10 Impulsen wieder abschalten. Ein solches Programm könnte so aussehen:

```
var position int

main:
    position = 0
    motor 1,8
    while position <> 10
        while in(2) = 0
            end
        position = position + 1
    end
    motor 1,0
```

Um es vorweg zu nehmen: Dieses Programm wird nicht wie gewünscht funktionieren. Woran könnte das liegen?

Zunächst wird die Variable position definiert und mit 0 definiert (diese Zuweisung könnte man auch weglassen, da alle Variablen automatisch mit 0 vordefiniert sind – sicherer ist es aber, diese Zuweisung zu machen). Als nächstes wird der Motor 1 eingeschaltet. Nun wird abgefragt, ob der Wert der Variablen position ungleich 10 ist – ist dies der Fall, wird die while-end Schleife ausgeführt. Beim ersten Durchlauf ist dies der Fall, denn gerade eben wurde position ja der Wert 0 zugewiesen.

In der nächsten while-end Schleife soll gewartet werden, bis der Taster 2 gedrückt wird – dies ist auch so weit in Ordnung. Ist der Taster gedrückt, gehts weiter und durch die Zuweisung position = position + 1 wird der Wert der Variablen position um 1 erhöht (Anders, als in der Mathematik üblich, ist es hier erlaubt, eine offensichtlich nicht korrekte Gleichung anzugeben). Als nächstes wird position also den Wert 1 haben. Anschließend wird die while-end Schleife fortgesetzt, die die Variable position mit 10 vergleicht. Aber was passiert nun? Da das Programm wesentlich schneller ausgeführt wird, als der Motor sich weiterdrehen kann, wird der Taster beim nächsten Durchlauf immer noch gedrückt sein und somit die Variable position weiter hochgezählt werden. Dies geschieht so schnell, dass der Motor nur kurz zuckt, anstatt erst nach 10 Impulsen stehen zu bleiben.

Also muss man nochmals warten, bis der Taster wieder losgelassen wurde, damit tatsächlich ein Impuls gezählt wird. Das tatsächlich funktionierende Programm sieht nun so aus:

```
var position int

main:
    position = 0
```

```

motor 1,8
while position <> 10
    while in(2) = 0
        end
    position = position + 1
    while in(2) = 1
        end
    end
end
motor 1,0

```

Dies sieht nun schon besser aus, allerdings auch sehr aufwändig – und da bei vielen Modellideen das Zählen von Impulsen eine wichtige Funktion ist, wurde der Befehl „edge“ = Flanke in das Basic eingebaut. Dieser Befehl führt intern alle Funktionen selbsttätig aus, zusätzlich bietet er noch weitere Funktionen.

```

var position int

main:
    position = 0
    motor 1,8
    while position <> 10
        edge #up,2
        position = position + 1
    end
    motor 1,0

```

Wir sehen, dass der edge-Befehl zwei Parameter hat: Zuerst die Art und Weise, wie der Edge-Befehl auf Veränderungen des Eingangs reagiert, anschließend die Nummer des Eingangs, der abgefragt werden soll. Für die Art der Flanke gibt es folgende Möglichkeiten: #up, #down und #any. Wie aus der englischen Übersetzung zu entnehmen ist, reagiert der Befehl auf steigende, fallende und beliebige (beide) Flanken. Eigentlich verbergen sich hinter #up, #down und #any Zahlen: Steigende Flanken werden bei 1 erkannt, fallende bei -1 und beliebige bei 0. Um sich diese Zahlen nicht merken zu müssen, sind im Compiler Konstanten eingebaut, die durch die entsprechende Zahl ersetzt werden. Die Raute # bezeichnet grundsätzlich eine dieser vordefinierten Konstanten. Man könnte also im obigen Programm durch einfaches ändern von #up auf #any die Auflösung verdoppeln, weil nicht nur die Flanke erkannt wird, die den Taster drückt, sondern auch die, die den Taster loslässt. Man braucht also für den gleichen Weg doppelt so viele Impulse und kann deshalb doppelt so genau positionieren.

Anhang

Das Robointerface Basic unterstützt die folgenden Variablentypen:

Typ	Speicherbedarf	Wertebereich		
– byte	1	-128	bis	+ 127
– int	2	-32768	bis	+32767
– long	4	2147483648	bis	+2147483647
– float	4	$(+/-)3,4*(+/-)10^{38}$		
– double	8	$(+/-)1,7*(+/-)10^{308}$		