John Billingsley

Commodore Sachbuchreihe Band 7

AUTOMATEN UND SENSOREN ZUM SELBERBAUEN

für Commodore Computer

John Billingsley

AUTOMATEN UND SENSOREN ZUM SELBERBAUEN

für Commodore-Computer

Commodore Sachbuchreihe Band 7

John Billingsley

AUTOMATEN UND SENSOREN ZUM SELBERBAUEN

für Commodore-Computer

Commodore

Titel der Originalausgabe: Billingsley, John – DIY robotics and sensors on the Commodore computer.

First published 1984 by:
Sunshine Books (an imprint of Scot Press Ltd.)
12–13 Little Newport Street, London WC2R 3LD

Aus dem Englischen übertragen von Dr. Wolfgang Wefelmeier und Brigitte Pohl.

Artikel-Nr. 580005/11.84 Änderungen vorbehalten ISBN-Nr. 3-89133-007-3

Komponenten

8 9-pin D-type male plug Farnell 140–820 oder RS 466–179
11 Potentiometer RS 161–830
13, 116, 118: Fotozelle OP 500
13 npn-Transistor 2N 3705
34 Lüsterklemmenleiste Cinch 251.12.90.160 oder Amp 530657–3 oder Teka TP3–121–E04
39 Triac RS 348–431
49 Operationsverstärker 747
49, 118: Komparator LM 339
55, 73, 90: Schrittmotor Philips ID35
57 pnp-Transistor 2N 3703 (RS 294–334)
65 Darlington-Verstärkerchip RS 307–109
101 Differenzverstärkerchip TL081
101 Motor Skyleader SRC 4BB
101 Operationsverstärker 759 (RS 303–258)

INHALTSÜBERSICHT

KAPITEL 1

VORBEREITUNGEN

Als stolzer Besitzer eines Commodore Mikrocomputers haben Sie mehr Rechenkapazität zur Verfügung als die gesamte Universität Cambridge im Jahr 1953. Sicher haben Sie endlos Spiele gespielt, eigene Programme geschrieben und viele Geheimnisse der Maschine selbst erforscht. Und nun?

Bis jetzt war Ihr Gerät auf eingetippte Eingaben angewiesen – auf Zahlen, die in Berechnungen eingehen sollten, oder auf einen Tastendruck, um ein Spielmanöver zu steuern. Warum lassen Sie es nicht selbst Daten finden? Warum geben Sie ihm nicht ein paar Muskeln in Form von Motoren und Relais, damit es auf die Außenwelt reagieren kann? Eine ganze Welt von Möglichkeiten öffnet sich auf diese Weise, angefangen bei Turtles und Robotern bis hin zu den Grenzen Ihrer Vorstellungskraft.

Der einfachste Sensorkanal benutzt einen der Analog-Digital-Wandler-Ports des C 64. Selbst auf dem PET* ist es jedoch möglich, eine Eingangsspannung mit nicht mehr als einem Widerstand, einem Kondensator und einem Bit des User-Ports – bei geschicktem Einsatz von etwas Maschinencode – zu messen. Schalter sind leicht zu lesen, und Festkörperrelais sind problemlos. Nur wenn Sie eine variable Ausgangsspannung wollen, brauchen Sie mehr dazu als ein oder zwei Widerstände – und selbst dann kann man 'mogeln' und 256 Ausgangspegel mit zwei weiteren Widerständen und einem zusätzlichen Kondensator erhalten.

Man kann auch ohne tieferes technisches Verständnis der Computerarchitektur ziemlich weit kommen, doch ich hoffe, daß Sie einiges von den Grundlagen beim Durcharbeiten des Buches kennenlernen. Jede neu eingeführte Bezeichnung ist in Anführungsstriche gesetzt und erklärt. Gelegentlich werden Anführungsstriche auch für ein 'Stichwort' verwendet, für das keine Erläuterung nötig ist.

Die ersten Kapitel sehen vielleicht sehr einfach und einleuchtend aus. Oft ist es jedoch gerade die banalste Einzelheit, zum Beispiel 'wie herum gehört der Stecker', die einen zu Fall bringt. Das Kapitel über Roboter ist scheinbar auf einen ganz speziellen Robotertyp zugeschnitten; tatsächlich befaßt es sich aber mit dem allgemeinen Problem, eine Vielzahl von Kanälen über einen beschränkten User-Port zu steuern.

Beim Schreiben jedes Kapitels sind die Konstruktionen und Programme ausprobiert und getestet worden, und ich bin meinem Sohn Richard und Timothy Dadd sehr dankbar für ihre Mitarbeit dabei. Alles, was schließlich in das Buch aufgenommen worden ist, hat wohl zumindest einmal funktioniert! Viel Glück.

---

* PET ist die amerikanische Bezeichnung für die großen Commodore-System-Rechner der 2000er- bis 8000er-Serien.

GRUNDAUSSTATTUNG

Die für die Konstruktion von Schnittstellen und Systemen erforderlichen Komponenten werden in jedem Kapitel beschrieben. Zusätzlich brauchen Sie einen kleinen Lötkolben und etwas Lötdraht mit Flußmittel. Verläßliches Löten ist eine Kunst, zu deren Perfektionierung man Jahre brauchen kann. Wesentlich ist das 'Verzinnen' der beiden Drähte, die verbunden werden sollen, wobei man jeden mit frischem Lot benetzt. Bringen Sie nie einen Tropfen geschmolzenes Lot auf das Metall auf – in wenigen Sekunden kann es eine Kruste bilden, die eine 'kalte' Lötstelle verbirgt und stundenlange Fehlersuche nach sich zieht.
Außerdem brauchen Sie ein Prüfmeßgerät. Kaufen Sie sich ein einfaches Drehspulen-Multimeter, kein digitales Meßgerät. Sie werden eher am groben Wert eines Signals interessiert sein, etwa ob es Masse oder logisch High ist, als an seinem Wert bis auf drei Dezimalstellen. Außerdem ist es überzeugender, wenn sich eine Nadel bewegt, die dazu einige zehn Mikroampere benötigt, als wenn Ziffern flimmern, die durch kleine statische Ladungen beeinflußt werden können. Das Meßgerät soll Unsicherheiten beseitigen, wenn etwas Unerwartetes eintritt, und es ist wenig hilfreich, wenn Sie sich erst fragen müssen, ob das Meßgerät etwas Reales anzeigt.
Obwohl ich versucht habe, die Komponenten bis zum letzten Stück Draht aufzuzählen, ist etwas zusätzlicher Draht sicher nützlich – ein oder zwei Meter, von welcher Farbe auch immer. Einadriger 0,6-mm-Schaltdraht läßt sich wohl am besten verwenden.

KONSTRUKTIONSVERFAHREN

Beim Durchblättern des Buches wird Ihnen sicher auffallen, daß nirgends eine gedruckte Schaltung zu sehen ist. Wenn Sie ein System entwickelt und erprobt haben, sei es eine Turtle oder ein einfacher Glättungskreis, möchten Sie vielleicht eine dauerhafte und sauber verlegte Version der Schaltungsanordnung herstellen. Dieses Buch beschäftigt sich jedoch nicht mit Strickmustern, sondern versucht ohne viel Aufhebens die Grundlagen der Schnittstellen zwischen Apparaten und Computern zu vermitteln. Es mag unordentlich aussehen, wenn ein dreidimensionales Rattennest von Komponenten an einer Anschlußleiste hängt, aber bei sorgfältigem Löten ist die Fehlerwahrscheinlichkeit gering. Fangen Sie mit einem Provisorium an, das funktioniert, und übertragen Sie es erst danach in eine elegantere Form. Wenn Sie dann merken, daß die Schaltung nicht mehr funktioniert, suchen Sie nach kalten Lötstellen, Haarrissen in den gedruckten Leiterbahnen oder nach Kupferfäden, die Bahnen überbrücken, die Sie Ihrer Meinung nach getrennt hatten.

NETZGERÄT

Für einige der späteren Konstruktionen ist eine Stromversorgung erforderlich, mit der man kleine Motoren betreiben kann. Es ist nicht ratsam, dem Computer mehr als 100 Milliampere zu entnehmen, und deshalb brauchen Sie ein eigenes Netzgerät. Die Komponenten bestehen aus einem Transformator mit zwei 6V-Ausgängen, einer Diodenbrücke und zwei Speicherkondensatoren. Damit erhält man Ausgänge von +7V und −7V, die alternativ als 14V-Stromzuführung verwendet werden können. Außerdem brauchen Sie Netzkabel, mindestens eine und vorzugsweise drei Sicherungen, drei Schraubklemmen oder eine dreiwegige Lüsterklemmenleiste und eine passende Kiste zum Einmontieren.

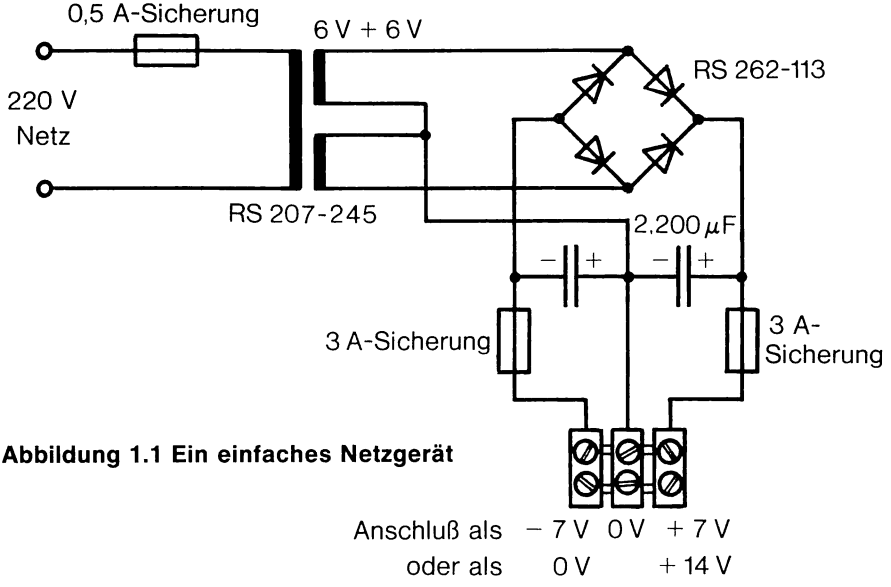


Abbildung 1.1 Ein einfaches Netzgerät

UND ZUM VC 20?

Das Buch ist hauptsächlich für den Commodore 64 und die neuere PET-Serie geschrieben, und die Programme berücksichtigen sorgfältig die Unterschiede zwischen diesen Geräten. Viele Programme laufen ebenfalls auf dem VC 20 und auch auf dem alten PET 2001. Es wäre lästig, jede Version der Programme vollständig aufzulisten, aber sie lassen sich ohne großen Aufwand durch Ersetzen von Zahlen entsprechend den untenstehenden Tabellen modifizieren.

Ausdrücke wie 'Port-Adresse' und 'Datenrichtungsregister' werden ziemlich ausführlich in Kapitel 4 behandelt. Im Augenblick brauchen Sie sich nur zu merken, daß diese Informationen hier stehen, damit Sie bei Bedarf nachschlagen können.

User-Port-Adresse:

Alle PETs: 59471=$E84F C 64: 56577=$DD01 VC 20=37136=$9110

Datenrichtungsregister:

Alle PETs: 59459=$E843 C 64: 56579=$DD03 VC 20: 37138=$9112

Zeiger auf 'BASIC-Anfang' (s. o. BASIC) und 'Variablen-Anfang' (s. o. vars):

PET 30xx, 40xx,80xx:	s.o. BASIC: 40, 41=$28, $29	s.o. vars: 42, 43=$2A, $2B
PET 2001:	s.o. BASIC: 122,123=$7A, $7B	s.o. vars: 124,125=$7C, $7D
C 64:	s.o. BASIC: 43, 44=$2B, $2C	s.o. vars: 45, 46=$2D, $2E
VIC 20:	s.o. BASIC: 43, 44=$2B, $2C	s.o. vars: 45, 46=$2D, $2E

Der BASIC-Programmbereich beginnt bei:

Alle PETs: 1025=$401 C 64: 2049=$0801 VC 20: 4097=$1001

IRQ-Interruptvektor:

PET 30xx,40xx,80xx:	144,	145	=$ 90,	$ 91
PET 2001:	537,	538	=$219,	$21A
C 64:	788,	789	=$314,	$315
VIC 20:	788,	789	=$314,	$315

PROGRAMMLISTINGS

Die am Ende der meisten Kapitel angegebenen Programmlistings fassen alle im Verlauf jedes Kapitels aufgeführten Zeilen und Routinen zusammen. Sie sind als Checkliste gedacht; die REM-Anweisungen sind weggelassen.

KAPITEL 2

SIGNALEINGÄNGE

Der Computer ernährt sich von einer Diät aus Zahlen, die im Speicher als Binärstellen oder 'Bits' abgelegt sind und vom Prozessor zu Ergebnissen verarbeitet werden, die wiederum Zahlen sind. Im Computer sind elektrische Signale entweder nahe 5V und stellen eine logische 'Eins' dar, oder nahe Masse und stehen für 'Null'. Aus Kombinationen solcher Signale sind numerische Werte in Zweierschritten aufgebaut; ein Byte aus acht Bits kann Zahlen von 0 bis 255 repräsentieren, zwei zusammen können als Zahlen von 0 bis 65535 oder auch als −32768 bis +32767 interpretiert werden.
Die Außenwelt sieht häufig anders aus. Zwar nehmen Tasten auf der Tastatur binäre Werte an, entweder 'gedrückt' oder 'nicht gedrückt', aber andere Größen wie Roboterpositionen, Motorgeschwindigkeiten oder Spannungen an Drehreglern können stetig über einen bestimmten Bereich variieren. Irgendwie müssen diese 'analogen' Werte in Zahlen umgewandelt werden, damit der Computer sie verdauen kann.
In diesem Kapitel wird Ihnen das schon im Commodore 64 eingebaute Interface begegnen, an dem sich bis zu vier Analogsignale anschließen lassen. Die Signale sollen die Form veränderlicher Widerstände haben, aber es ist nicht allzu schwierig, auch Spannungen zu lesen. Sie lernen hier, ein einfaches Kabel mit einer Anschlußleiste herzustellen, was sich immer wieder als nützlich erweisen wird, um Entwürfe für Eingänge mit geringstmöglichem Aufwand auszuprobieren; und sie erfahren, wie sich ein Joystick machen läßt, mit dem man das Grafikprogramm aus dem nächsten Kapitel steuern kann. Sie werden auch entdecken, wie man einen 'Lichtgriffel' an den Computer anschließt, mit dem man ein Objekt auf dem Bildschirm auswählen kann, indem man einfach darauf zeigt.

ANSCHLÜSSE FÜR DIE ANALOGEN PORTS (NUR C 64)

Der C 64 besitzt zwei Anschlüsse für Steuerknüppel und Drehregler. Neben den Eingängen für Logiksignale hat jeder Port zwei analoge Eingänge zur Versorgung eines Analog-Joysticks, und ein einfacher POKE-Befehl schaltet die Leistungen von den Pins eines Anschlusses auf die des anderen. Einer der beiden Eingänge hat auch einen Lichtgriffelanschluß, der es zweifellos verdient, später genauer untersucht zu werden.

Die Konstruktion und der Gebrauch eines selbstgebastelten Joysticks wird Ihr Selbstvertrauen steigern. Einen solchen Joystick kann man später als einfaches Prüfgerät zur Fehlerbeseitigung bei beliebigen Analogeingängen gebrauchen. Selbst bei einem so einfachen Gerät wie diesem, das nur zwei Potentiometer enthält, sind die Fehlermöglichkeiten zahllos. Es ist wichtig, Schritt für Schritt vorzugehen und jedesmal Unklarheiten zu beseitigen.
Verbinden Sie zunächst eine Anschlußleiste mit einem der Spiele-Eingänge, dem Control Port 1. Sie brauchen dazu einen Stecker vom Typ D für 9 Pins, z. B. Farnell 140-820 oder RS 466-179, eine mindestens 9wegige Lüsterklemmenleiste und genug Draht, um sie anzuschließen – entweder etwa 30 cm Flachkabel oder ein entsprechendes Sortiment farbiger Drähte. Auf diese Weise bringen Sie die Anschlüsse zu einer bequemen Anschlußleiste neben der Tastatur, so daß Sie Hardware- und Softwarefehler gleichzeitig beseitigen können.
Es lohnt sich, die Pins zu 'entwirren', damit die Signale in vernünftiger Reihenfolge auf der Anschlußleiste liegen. Die Verdrahtungsreihenfolge wird dann:

Anschlußleiste:

Signal:	PotX	PotY	+5V	0V	Joy0	Joy1	Joy2	Joy3	But/LPen

Stecker:

Pin:	9	5	7	8	1	2	3	4	6

Nachdem die Anschlußleiste verdrahtet ist, schließen Sie sie an und prüfen sie. Das Geheimnis der erfolgreichen Entwicklung elektronischer Schaltungen besteht darin, allem zu mißtrauen. Wenn Sie überzeugt sind, daß ein Signal auf einem Ende eines Drahtes liegt, prüfen Sie nach, daß es wirklich am anderen Ende ankommt – sonst kann Ihnen Ihr Vertrauen in Draht und Verlötung stundenlange Fehlersuche einbringen.
Benutzen Sie zunächst ein einfaches Multimeter (Bereich 6V Gleichstrom, negatives Ende an die 0V-Leitung), um die +5V– und 0V-Anschlüsse zu prüfen. Testen Sie auch Joy0, Joy1, Joy2 und Joy3, die zwischen 4V und 5V liegen werden. Ein schlichtes Meßgerät mit Nadel und Skala ist viel vertrauenerweckender als die flackernde Anzeige eines Digitalmeßgeräts, besonders wenn veränderliche Signale untersucht werden.
Als Nächstes sind die Anschlüsse an die analogen Eingänge zu testen. Geben Sie folgendes Programm ein:

```
10 PRINT PEEK(54297),PEEK(54298)
20 GOTO 10
```

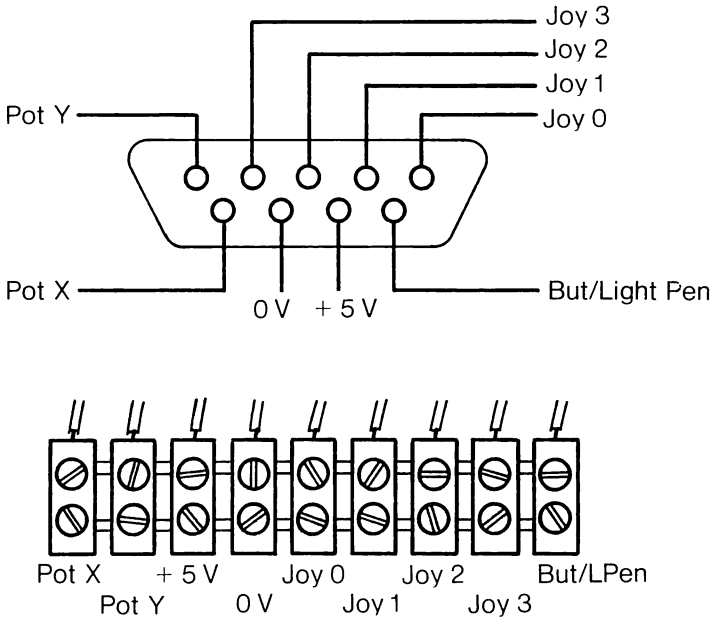


Abbildung 2.1 Anschlußleiste für einen Analog-/Spiele-Port

und lassen es laufen. Der Bildschirm müßte sich mit zwei Spalten füllen, die den Wert 255 enthalten. Schließen Sie jetzt über einen 1000 Ohm-Widerstand einen Draht an +5V an, und berühren Sie nacheinander die Analogeingänge mit dem anderen Ende. Sie sehen dann die entsprechende Spalte auf dem Bildschirm einen Wert nahe Null anzeigen, solange der Kanal berührt wird – und Ihre Zuversicht wächst.
Überprüfen Sie zum Schluß die Steuerknüppel-Eingangsbits Joy0 bis Joy3, und das 'Feuerknopf'-Bit but/LPen. Geben Sie dieses Programm ein:

```
10 PRINT PEEK(56321) AND 31
20 GOTO 10
```

Entfernen Sie den Prüfdraht von +5V und verbinden ihn stattdessen mit 0V (ohne den Widerstand). Während das Programm läuft, berühren Sie nacheinander die Joy-Pins und den But-Pin. Der auf dem Bildschirm angezeigte Wert sollte sich dabei vom normalen Wert 31 in 30, 29, 27, 23 und 15 ändern. In anderen Worten: jedes Bit liegt auf '1', bis es an Masse angeschlossen wird.

WIE ARBEITET DER ANALOGEINGANG?

Das Operationsprinzip ähnelt sehr dem PET-Eingang, der in Kapitel 5 beschrieben wird, aber im C 64 ist es der 'SID'-Musikchip, der die ganze harte Arbeit tut. Er hat zwei Eingangsbits, die zum Lesen zweier Drehregler gedacht sind, während der C 64 mit einem zusätzlichen Schalter die Leitungen von einem Drehreglerpaar auf das andere legen kann. Jeder Eingangskanal arbeitet wie folgt:
Das Eingangsbit wird zuerst vom Chip mit Masse verbunden und entlädt dabei einen Kondensator. Der Kondensator lädt sich dann über den variablen Widerstand des Drehreglers auf, während der Chip die Zeit zählt, in der die Kondensatorspannung den Triggerpegel erreicht. Dieser Zählwert wird dann in ein anderes Register innerhalb des Chips geschrieben, aus dem es mit einem einfachen PEEK gelesen werden kann. Beim Einschalten müßte automatisch Port 1 gewählt werden. Um sicherzugehen, können Sie mit

```
POKE 56322,PEEK(56322) OR 192
```

das Datenrichtungsregister initialisieren (zur Erläuterung siehe Kapitel 4). Mit

```
POKE 56320,64
```

wählen Sie dann die Drehregleranschlüsse von Port 1. Wenn Sie stattdessen Port 2 wollen, müssen Sie poken:

```
POKE 56320,128
```

– allerdings ist das nicht so unkompliziert, wie es aussieht. Der Chip prüft den Eingang mehrmals pro Millisekunde. Dennoch tritt im allgemeinen eine leichte Verzögerung vor dem Lesen des Wertes auf, wenn Sie innerhalb eines Programms umschalten. Die Werte der X- und Y-Umwandlungen ergeben sich als:

```
X=PEEK(54297)
Y=PEEK(54298)
```

Die Adresse, die die Schalterstellung wählt, ist auch am Lesen der Steuerknüppel-Bits beteiligt. Leider geht sie auch in das System ein, das die Tastatur liest. Das bedeutet, daß etwa fünfzig Mal in der Sekunde eine 'Interruptroutine' durch alles hindurchfegt, was sonst gerade passiert, worauf der Schalter auf Port 1 zeigt, selbst wenn Sie eigentlich Port 2 wollten. Wenn Sie nur Port 1 lesen wollen, können Sie ruhig BASIC verwenden, doch wenn Sie beide Ports (d. h. alle vier Kanäle) lesen wollen, müssen Sie eine Routine in Maschinensprache schreiben, die Interrupts im entscheidenden Augenblick unterdrückt.

Sie können jetzt die Anschlüsse von Port 2 prüfen, indem Sie Ihre Anschlußleiste von dem einen Port trennen und mit dem anderen verbinden. Die Joy-Bits erscheinen nun in einer anderen Adresse, 56320. Das Testprogramm lautet:

```
10 PRINT PEEK(56320) AND 31
20 GOTO 10
```

Zum Prüfen der Analogsignale verwenden Sie das nächste Programm. Die Zeilen 10 und 40 sind schlichte Maschinensprache-Programme aus je zwei Anweisungen. Die Anweisung SEI wird in 1024 gesetzt und erscheint links oben auf dem Bildschirm. Sie unterdrückt Interrupts – was verhängnisvoll werden könnte, wenn es nicht in Zeile 40 durch CLI korrigiert würde. Beide Zeilen setzen den Code für die Rückkehr vom Unterprogramm in 1025, um die Kontrolle wieder an das BASIC-Programm zu übergeben:

```
10 POKE 1024,7*16+8:POKE1025,6*16:SYS 1024:REM SEI RTS
20 POKE 56320,128
30 PRINT PEEK(54297),PEEK(54298)
40 POKE 1024,5*16+8:POKE1025,6*16:SYS 1024:REM CLI RTS
50 GOTO 10
```

KONSTRUKTION EINES JOYSTICKS

Nun zum Joystick. Dafür brauchen Sie zwei Potentiometer mit einem Wert von 500 kOhm. Geeignet wäre RS 161-830, aber fast alles andere geht auch. Selbst der Widerstandswert ist nicht besonders kritisch; ein zu niedriger Wert beschränkt den Bereich der erzeugten Zahlen, während bei einem zu hohen Wert der brauchbare Abschnitt zu einem Ende der Drehung hin zusammengedrängt ist. Fangen Sie einfach an. Das Potentiometer besteht aus einem Widerstandsdraht, der mit den äußeren Lötfahnen verbunden ist. Die mittlere Lötfahne ist an den 'Schleifer' angeschlossen, der am Draht entlanggeleitet, wenn die Achse gedreht wird, und einen Widerstandswert entsprechend seiner Position aufnimmt. Verbinden Sie eine der äußeren Fahnen eines der Potentiometer mit +5V. Schließen Sie die mittlere Fahne an PotX an; geben Sie dann das erste der obigen Programme ein und starten es. Wenn Sie die Achse hin- und herdrehen, sehen Sie die Zahlen in der ersten Spalte von nahe 0 bis 255 variieren.

Das haben Sie nicht getan? Dann prüfen Sie mit dem Multimeter die Spannung am Schleifer des Potentiometers. Keine da? Dann bauen Sie das Potentiometer aus und messen seinen Widerstand und den Widerstand zwischen dem Schleifer und jedem Ende, während Sie am Knopf drehen. Sieht in Ordnung aus? Dann schließen Sie es wieder an und versuchen es wieder; prüfen Sie den Wert von +5V. Alles in Ordnung, aber noch immer keine wechselnden Zahlen? Dann testen Sie wieder PotX wie eben; gehen zu Bett und versuchen es morgens noch einmal.
Verbinden Sie jetzt eine der äußeren Fahnen des zweiten Potentiometers ebenfalls mit +5V und seinen Schleifer mit PotY. Wenn das Testprogramm läuft, müßten Sie die Zahlen in beiden Spalten steuern können. Ihre elektronischen Probleme sind gelöst, und Sie stehen nun der Aufgabe gegenüber, die Drehbewegungen in die Bewegungen eines Joysticks umzusetzen. Eine Möglichkeit ist in **Abbildung 2.2** skizziert.
Bis jetzt war dieses Kapitel sicher äußerst frustrierend für PET-Besitzer. Obwohl der PET keinen Analogeingang hat, läßt sich dennoch ein Joystick anbringen – indem man Gebrauch von den in Kapitel 5 beschriebenen gemeinen Tricks macht. Dazu ist es jedoch zunächst nötig, die Machenschaften des User-Ports und die Subtilitäten des Vielseitigen Interface-Adapters VIA zu verstehen.

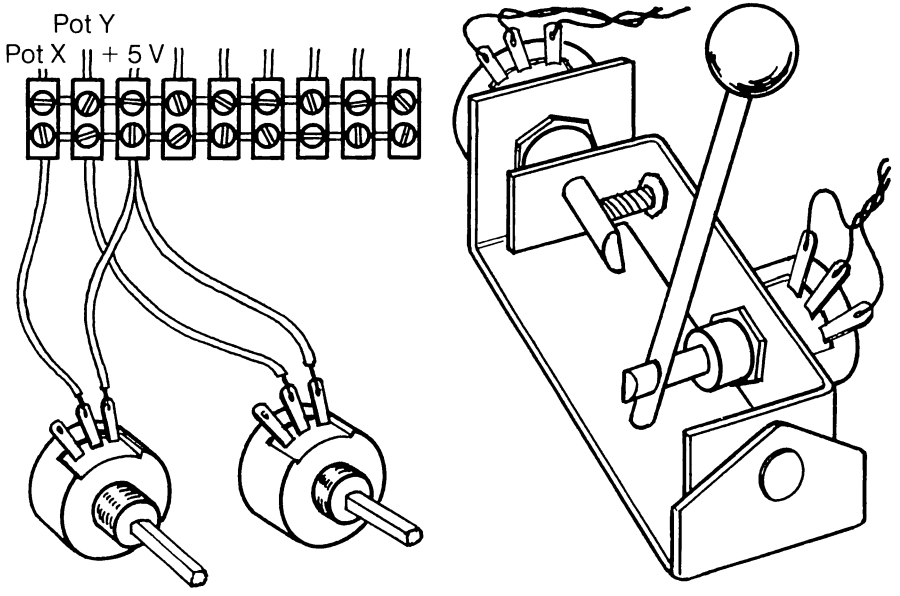


Abbildung 2.2 Zwei Potentiometer – ein einfacher Joystick

ANSCHLIESSEN EINES LICHTGRIFFELS

Bevor wir weitermachen: Erinnern Sie sich an den verführerischen Lichtgriffelanschluß, der zugleich als Feuerknopf von Port 1 fungierte? Was ist ein Lichtgriffel, und was kann er? Bekanntlich wird das Fernsehbild aus einem einzigen Punkt aufgebaut, der 15000mal in der Sekunde den Bildschirm abtastet und dabei 50mal in der Sekunde von oben nach unten wandert. Wenn ein Griffel mit einem Fototransistor darin gegen den Bildschirm gehalten wird, gibt der Fototransistor einen Ausgangsstromimpuls ab, sobald der Punkt unter ihm vorbeiläuft. Aus dem Zeitpunkt des Impulses läßt sich die gewählte Position auf dem Bildschirm erschließen. Mit der richtigen Schnittstelle müßte es möglich sein, einen Satz von Alternativen auf dem Bildschirm anzuzeigen – etwa 'auf', 'ab', 'links', 'rechts' für Roboterbefehle – und seine Wahl einfach dadurch zu treffen, daß man mit einem an den Computer angeschlossenen Griffel auf das Wort auf dem Bildschirm deutet.

Das 'offizielle' Lichtgriffel-Interface kostet ungefähr 80 Mark. Kann man auch billiger davonkommen? Alles, was man braucht, sind ein Schaltkreis aus einem Fototransistor OP500, drei Widerstände und ein npn-Transistor 2N3705 – Gesamtkosten unter 4 Mark! Wenn Sie den TV-Helligkeitsregler aufdrehen, kommen Sie sogar mit dem Fototransistor und einem einzigen Widerstand aus.

Der Lichtgriffelanschluß LPen führt zum Video Interface Chip VIC. Das ist eine vielbeschäftigte Einrichtung, die Zahlenfolgen im Speicher in ein passendes Bild auf dem Schirm umsetzt. Nehmen wir an, daß als erste Textzeile auf dem Bildschirm 'the quick brown fox' erscheinen soll. Der Chip sieht sich zunächst den zu Beginn der Zeile darzustellenden Speicherplatz an – hier enthält er den Code für den Buchstaben 't'. Aus einem anderen Teil des Speichers muß er dann das Muster des Buchstabens 't' lesen, oder genauer das Muster für die erste Abtastzeile des 't'.

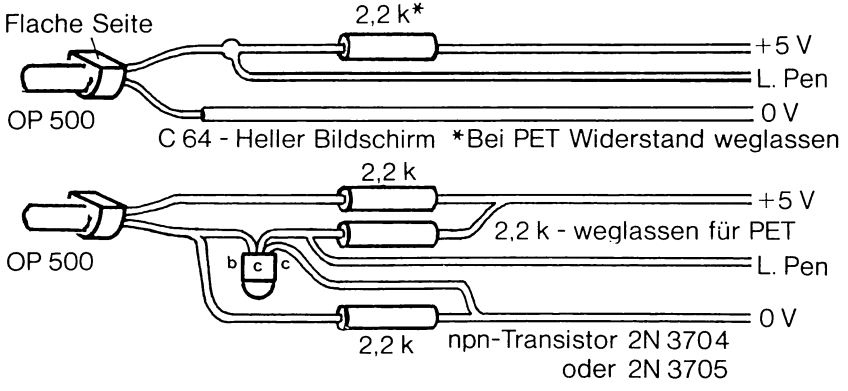


Abbildung 2.3 Lichtgriffel für hellen und dunklen Bildschirm

Während damit der Elektronenstrahl der Röhre moduliert wird, sucht der Chip den Code für 'h' auf und liest dessen Muster für die oberste Zeile, und so weiter bis zum Ende der Buchstabenzeile. Wenn die Abtastzeile vollständig ist, liest der Chip 't' zum zweitenmal und sucht nun das Muster für die zweite Zeile, und so fort, bis acht Abtastzeilen angezeigt sind. Die Adresse des jeweils verarbeiteten Buchstaben wird in einem Sechzehnbit-Register innerhalb des Chips gehalten und automatisch so inkrementiert, daß sie dem Text folgt.
Bis hierher paßt die Beschreibung des Chips sowohl auf den VIC-Chip als auch auf den CRT-Controller-Chip in der 8000er Serie und im CBM 4032 mit großem Bildschirm ('Fat 40' PET). Der CRT-Controller hat zwar ebenfalls einen Lichtgriffelanschluß, er ist jedoch mit Pin 21 des Steckplatzes J 4 über einen Wandler 7404 mit 1-kOhm-Pullup verbunden. Dieser Steckplatz ist der Wald von Pins zur Speichererweiterung auf der rechten Seite des Computers – Sie müssen ziemlich abenteuerlustig sein, um ihn zu benutzen. Jedesmal, wenn der LPen-Pin des CRT-Controllers von Logikpegel 0 auf 1 geht, wird die Adresse des gerade angezeigten Zeichens von zwei Achtbit-Registern innerhalb des Chips erfaßt. Aus deren Wert läßt sich erschließen, wo das Zeichen auf dem Bildschirm erschienen war, und die entsprechende Reaktion kann erfolgen. Der Chip ist einer von den lästigen, die Adressenkapazität zu erhalten versuchen (siehe Kapitel 4), indem sie in einer Adresse mit der Registernummer angestachelt werden müssen und mit der Antwort in einer anderen herausrücken. Machen Sie sich keine großen Sorgen, wenn Ihnen das nächste Programm wie Hokuspokus vorkommt, es ist kurz, und es funktioniert:

```
10 REM SIMPLE LIGHT-PEN DEMO FOR 8000 OR FAT40 PET
20 REM FILL SCREEN WITH WHITE (GREEN)
30 FOR I=32768 TO 34768: POKE I,160:NEXT
40 AO =32768 : REM OLD HIT
```

Und jetzt zum Hokuspokus:

```
 50 CR=14*16↑3+8*16↑2+8*16:REM ADDRESS OF CHIP IS $E880
100 POKE CR,17:A=PEEK(CR + 1):REM READ REGISTER 17
110 POKE CR,16:A=PEEK(CR + 1)+256*A:REM REGISTER 16
120 B=A-32768: REM B IS NOW CHARACTER NUMBER ON
SCREEN
130 IF B< 0 OR B> 1999 THEN 100:REM MISSED SCREEN!(999 FOR
    FAT 40)
```

Mit dem Ergebnis wollen wir jetzt einen Fleck auf dem Bildschirm bewegen:

```
140 POKE AO,160 :   REM RUB OUT OLD BLOB
150 POKE A,32 :     REM PLANT NEW BLACK BLOB
160 AO=A :          REM REMEMBER WHERE IT IS
170 GOTO 100:       REM AND DO IT AGAIN
```

Nun zurück zum C 64 und seinem VIC-Chip. Dieser Chip gibt sich nicht mit dem Verarbeiten von Farben, Bit-Map-Grafik und acht Spites zufrieden, er besteht auch noch darauf, den Lichtgriffeltreffer bis auf ein Pixelpaar genau anzugeben. Es gibt jetzt zwei Register in 53267 und 53268, die den X- und Y-Wert des Lichtgriffeltreffers enthalten. Der Y-Wert gibt die genaue Nummer der Abtastzeile an, während der X-Wert ein Paar Pixelpositionen anzeigt. Da diese vom Rand des Bildes aus, hinter der Einfassung, gemessen werden, muß 46 von jeder Zahl subtrahiert werden, um den Anzeigebereich zu treffen. Weil diese Werte nur einmal pro Abtastzeile ein Strobe-Signal erhalten, sollten Sie den Griffel nicht zu schnell bewegen.
Das folgende Programm soll nur grob anzeigen, daß das Verfahren funktioniert. Es gehört nicht viel Phantasie dazu, es auf das Zeichnen einer genau zwei Pixel breiten Linie zu erweitern, aber im Augenblick wollen wir uns damit zufrieden geben, einen Fleck zu bewegen.

```
 10 REM SIMPLE LIGHT-PEN DEMO FOR COMMODORE 64
 20 REM FILL SCREEN WITH WHITE
 30 SC=1024:CO=55296 : REM SCREEN START, COLOUR
 40 FOR I=0 TO 999 : POKE I+SC,160
 50 POKE I+CO,1:NEXT : REM COLOUR WHITE
 60 AO=0 : REM OLD HIT
 70 PX=53267:PY=53268: REM ADDRESSES OF HIT REGISTERS
100 X=PEEK(PX)-46:Y=PEEK(PY)-46:REM READ REGISTERS
110 A=INT(X/4)+40*INT(Y/8):REM WHICH CHARACTER?
120 IF A< 0 OR A> 999 THEN 100 : REM MISSED THE SCREEN!
```

Nun wollen wir einen roten Fleck anzeigen:

```
140 POKE CO+AO,1 : REM RUB OUT OLD BLOB - MAKE WHITE
150 POKE CO+A,2 :  REM PLANT NEW RED BLOB
160 AO=A :         REM REMEMBER WHERE IT IS
170 GOTO 100:      REM AND DO IT AGAIN
```

Das ist ein gutes und einfaches Programm, doch Tim und Richard hatten das Gefühl, Sie würden etwas Unterhaltsameres vorziehen. Es folgt ihre Version eines Xylophons. Auf dem Bildschirm erscheinen Do, Re, Mi usw., und links davon eine Spalte mit Flecken. Richten Sie den Lichtgriffel auf eine Note, so wird diese Note gespielt; die Spalte links davon gibt himmlische Ruhe. Haben Sie Mitleid mit Ihren Nachbarn.

Lichtgriffeldemonstration

```
100 INPUT"SPEED (100)ISH";SP
110 GOTO10000
200 N=INT((PEEK(VY)-26)/16)
210 IF N<0 OR PEEK(VX)< 80 THEN N=0:GOTO200
220 POKEPH,PH(N):POKEPL,PL(N)
230 POKEW,0:POKEW,WW
240 FOR I=1 TO 2000 STEP SP:NEXT
250 GOTO 200
10000 RV$=CHR$(18):RO$=CHR$(146):REM REVERSE,OFF
10010 NN$="DO  RE  MIFA  SO  LA  TIDO"
10020 PRINTCHR$(147)CHR$(5);
10030 FOR I=1 TO 13: PRINT : PRINT"  "RV$"  "
10040 PRINT"  "RV$"  "RO$"                    ";
10050 IFI=2ORI=4ORI=7ORI=9ORI=11THENPRINT"    ";
10060 PRINTRV$"  "MID$(NN$,2*I-1,2);"     ";
10070 NEXT
10100 DIMPH(16),PL(16)
10110 P=2^(1/12):PO=4000
10120 FOR I=1 TO 13
10130 PH(I)=INT(PO/256):PL(I)=PO AND 255
10140 PO=PO*P:NEXT
10200 POKE 54296,15:POKE54277,7:W=54276:WW=17
10210 PH=54273:PL=54272
10220 VX=13*16^3+19:VY=VX+1
10230 POKE53281,0:GOTO200
```

KAPITEL 3

GRAFIKDESIGN MIT EINEM JOYSTICK

Ich bin mit einer Grafikdesignerin verheiratet. Eines Tages wollte meine Frau Ros mehr über Computergrafik erfahren und ein paar Bildschirmdisplays entwerfen.
Auf dem Commodore existieren so gut wie keine Grafikunterprogramme, und obwohl der C 64 eine sehr leistungsstarke Hardware für Grafiken hat, ist es keine leichte Aufgabe, ein vollständiges Farbgrafikpaket für ihn zu schreiben. Falls Sie nicht ein Paket in Maschinensprache kaufen (oder schreiben), sind die Routinen sehr langsam in der Ausführung. Der Hauptverdienst des hier entwickelten Programms liegt in den Hinweisen, die es für das Schreiben Ihrer eigenen Routinen enthält. Trotzdem stimmen Sie mir hoffentlich zu, daß es eine nützliche Sammlung von Funktionen bereitstellt und Spaß macht. Es liefert einen guten Vorwand, einen Joystick zu benutzen, und enthält sogar einen Hauch digitales Filtern, um das Zittern des Joysticks zu beseitigen. Alternativ können Sie es mit dem im vorhergehenden Kapitel entwickelten Lichtgriffel steuern. Leider wird Grafik auf dem PET nur durch eine Zusatzplatine getragen, und deshalb bezieht sich dieses Kapitel ausschließlich auf den C 64.

GRAFIK UND SPRITES

Der VIC-Chip, der das Display des C 64 steuert, zeigt im Normalmodus 1000 Speicherplätze in Form von Zeichen an, die einer Tabelle von Mustern entnommen werden. Alternativ ermöglicht er ein hochauflösendes Display von 320 mal 200 Punkten aus 4000 Speicherplätzen. In beiden Fällen ist in einem getrennten Speicherbereich von 1000 Bytes die Vorder- und Hintergrundfarbe jeder einem Zeichen entsprechenden Anzeige enthalten. Diese Zeichen sind in 40 Zellen nebeneinander und 25 untereinander angeordnet.
Zusätzlich kann der Chip acht 'Sprites' verwalten. Das sind Bit-Map-Muster, die über Text oder Grafik gelegt werden können und sich als Ganzes durch Ändern von jeweils zwei Speicherplätzen verschieben lassen. Jedes Sprite ist durch eine Folge von 63 Speicherplätzen definiert und hat eine Auflösung von 24 Bits waagerecht und 21 senkrecht. Die Sprites sind durch Zeiger mit der Festlegung des Musters verbunden, und mehrere oder alle Sprites können mit demselben Muster verbunden sein. Das nächste Programm ist so einfach wie nur möglich und erlaubt Ihnen, ein Sprite mit Hilfe des Joysticks über den Bildschirm zu bewegen. Es soll ein paar Grundsätze verdeutlichen, muß aber noch stark ausgeschmückt werden, um interessanter zu werden.

```
10 VV=13*16↑3:          REM VIC CHIP IS AT $D000
20 POKE VV+21,1:        REM TURN ON SPRITE 0 (BIT 0)
30 POKE 2040,13:        REM POINT SPRITE 0 TO
                        ADDRESS 64*13
40 POKE VV,160:         REM X OF SPRITE 0
50 POKE VV +1,100:      REM Y OF SPRITE 0
60 FOR I=832 TO 832+62: POKE I,255:NEXT
70 REM FILL SHAPE WITH SOLID COLOUR
```

Wenn Sie das Programm bis hierher eingeben und laufen lassen, sehen Sie das Sprite erscheinen und sich mit Farbe füllen. Jetzt möchten Sie es sicher über den Bildschirm bewegen:

```
100 POKE VV,PEEK(54297):        REM JOYSTICK X
110 POKE VV + 1,PEEK(54298):    REM JOYSTICK Y
120 GOTO 100
```

Und das ist alles! Weitere Sprites lassen sich mit den übrigen Bits von VV+21 anschalten, und der Sprite-Zeiger für den Sprite N liegt in 2040+N. Die Farbe des Sprites N läßt sich durch Poken von VV+39+N mit einer Zahl von 0 bis 15 ändern, während die X- und Y-Werte in VV+2*N und VV+2*N+1 sind, wobei Bit N von VV+16 die X-Koordinate um 256 erhöht.
Das Entwerfen eines Sprites ist eine mühselige Angelegenheit. Am Ende des Kapitels ist ein 'Sprite-Editor' aufgelistet, mit dem man einen Pixelcursor über das Sprite fahren und dabei beliebig Bits setzen oder löschen kann.
Wir wollen nun die kühnere Aufgabe angehen, ein Grafikpaket zusammenzustellen, das das Entwerfen eines ganzen Bildschirms voller farbiger Linien und Flächen vereinfacht.

PROGRAMMBESCHREIBUNG

Die Funktionen, die das Programm bereitstellt, sind Punkt, Linie und Fläche, wobei Tinten- und Papierfarbe durch Anschlagen von C gewählt werden können. Wird der Joystick bewegt, läuft ein Punkt über den Bildschirm. Durch Anschlagen von P wird ein fester Punkt auf dem Bildschirm markiert, und es werden die Koordinaten des Punktes gespeichert.
Wird der Joystick bewegt und L angeschlagen, so wird eine Gerade vom zuletzt registrierten Punkt gezogen. Eine weitere Bewegung und ein weiteres L ziehen eine zweite Gerade vom Ende der ersten, und so weiter. Wird die Taste L festgehalten, werden Geradenstücke in schneller Folge gezeichnet und ziehen die Joystickbewegung in einer glatten Kurve nach.

Flächen werden durch horizontales Schraffieren gefüllt. A setzt einen Umriß in derselben Weise, in der L eine Linie zieht. Falls bei der vertikalen Koordinate schon ein Umriß besteht, wird die Zeile zwischen dem alten und dem neuen Umriß mit Farbe angefüllt. Wenn Sie A festhalten und ein 'U' mit dem Joystick ziehen, wird der Grundstrich als Kurve auf den Bildschirm gezeichnet. Beim Aufstrich 'füllt' sich dann die Kurve wie ein Weinglas. Sobald P angeschlagen wird, wird die alte Umrißlinie vergessen.
Um ein versehentliches Löschen dieses Kunstwerkes zu vermeiden, ist die 'Clear'-Anweisung ein Ausrufungszeichen, das das gleichzeitige Festhalten von 'Shift' erfordert.
Durch Anschlagen von C kehrt der Bildschirm in den Normalmodus zurück, und Sie werden nach den neuen Vordergrund- und Hintergrundfarben gefragt. Nach der Eingabe wechselt der Bildschirm wieder in den Grafikmodus mit Ihrem unbeschädigten Gemälde.

EINSCHALTEN DES GRAFIKMODUS

Es ist schon einmal ein Anfang, die Anforderungen an das Programm festzulegen, aber wie kommen wir auf dem C 64 in den hochauflösenden Grafikmodus? (PET-Besitzer werden in diesem Kapitel leider völlig ingnoriert.) Wir müssen einige Datenbytes an den VIC-Chip senden, der die Ausgabe steuert, und außerdem müssen wir ein Byte im CIA2 ändern. Das nächste Kapitel versucht, die Magie etwas zu erhellen, aber für jetzt nehmen Sie sie einfach hin.

Den nächsten Abschnitt können Sie beim ersten Lesen des Buches überschlagen und später darauf zurückkommen.
Der Bit-Map-Modus wird durch Poken von Register 17 des VIC-Chips mit einem Wert eingeschaltet, der durch OR mit 32 verknüpft ist. Der CIA2 in Adresse $DD00 bestimmt die 'Bankadresse' für den Bildschirm- und Farbspeicher des VIC-Chips, unter Verwendung des wertniedrigsten Bits. Die Banken sind spiegelbildlich angeordnet, so daß bei einem Wert von 3 die Bank bei $0000 beginnt, für 2 bei $4000, für 1 bei $8000 und für 0 bei $C000. Register 24 des VIC-Chips definiert den Rest der Adresse durch den Wert 16*(Farbrelativadresse)+(Bildschirmrelativadresse). Diese Relativadressen werden mit 1024 multipliziert, um die tatsächliche Erhöhung der Adressen zu ergeben. Also legen wir durch Wahl von Bank 1 und Poken des VIC-Registers 24 mit dem Wert 16*7+8 den Bildschirmspeicher auf den Bereich von $6000 bis $7FFF und den Farbspeicher auf $5C00 bis $5FFF fest. (Das beschränkt natürlich das für Programme zugängliche RAM. Die Maschine könnte ebenso gut das 'RAM unter dem ROM' benutzen, mit dem Bildschirm bei $E000 und der Farbe bei $C000: Wählen Sie Bank 3 und poken Register 24 mit Wert 8.

Das geht gut, solange Sie auf den Bildschirm schreiben, aber sobald Sie ihn peeken wollen, sehen Sie statt dessen das ROM. In Maschinensprache läßt sich das Problem leicht umgehen, aber nicht in BASIC – deshalb benutzen wir Bank 1.)

Lesen Sie hier weiter. Bauen Sie das Programm von Anfang an in austestbaren Modulen auf. Geben Sie das folgende Programm ein und lassen es laufen. Die 'Verwaltung' ist bei 10000 geparkt, wo sie nicht im Weg ist. Der Grafikmodus wird mit den Unterprogrammen bei 9000 bzw. 9100 gesetzt und rückgesetzt. Wenn alles klappt, erscheint die Abfrage 'Colour, Pattern'. Versuchen Sie es zunächst mit 33, 15. Der Bildschirm springt dann in den hochauflösenden Modus, und die Farben werden rasch auf schwarz-weiß gesetzt. Das Programm knabbert daraufhin ein Streifenmuster; es braucht mehrere Sekunden, den ganzen Bildschirm zu bedekken und schließlich in den Normalmodus zurückzuspringen. Falls Sie das Programm unterbrechen, können Sie es vielleicht durch Eingeben von GOSUB 9100 retten – andernfalls versuchen Sie es mit RESTORE.

```
    5 POKE 56,64:CLR: REM PROTECT GRAPHIC MEMORY AREA
   10 GOTO 10000

10000 REM
10020 SC=1*16384+8*1024: REM SCREEN STARTS BANK 1,
      SECTOR 8
10025 CC=1*16384+7*1024: REM COLOUR MAP=BANK 1,
      SECTOR 7
10030 VV=13*4096:CI=VV + 13*256: REM VIDEO CHIP=$D000,
      CIA2=$DD00
10040 GF=VV+17:CM=PEEK(GF):GM=CM OR 32
10045 REM GRAPHICS FLAG, CHAR. MODE, GRAPHICS MODE.
10200 GOTO 100

  100 INPUT"COLOUR,PATTERN";CL,CH
  110 GOSUB 9000: REM SET GRAPHICS MODE
  120 FOR I=CC TO CC+1000:POKE I,CL:NEXT:REM SET COLOUR
  130 FOR I=SC TO SC+8000:POKE I,CH:NEXT:REM PATTERN
  140 GOSUB 9100: REM RESET TO CHARACTER MODE
  190 STOP

 9000 POKE CI,PEEK(CI)AND(255-1):REM SET BANK 1
 9010 POKE VV+24,8+16*7:REM SCREEN SECTOR=8,COLOUR =7
 9020 POKE GF,GM:REM GRAPHICS MODE
 9030 RETURN
```

```
9100 POKE CI,PEEK(CI)OR 3:REM BANK 0
9110 POKE VV+24,20:REM NORMAL SCREEN
9120 POKE GF,CM:REM CHARACTER MODE
9130 RETURN
```

Sie haben sicher erraten, daß die seltsame Zeilennumerierung auf spätere Zusätze zugeschnitten ist.

SETZEN EINES PUNKTES

Die Anordnung des Bildschirmspeichers ist nicht ganz unkompliziert, und es erfordert sorgfältige Planung, eine X,Y-Koordinate zum Aufzeichnen in eine Byteadresse und ein Muster umzusetzen. Die ersten acht Bytes bilden die Zeilen der ersten Zeichenposition auf dem Bildschirm – sie werden eine unter der anderen dargestellt. Das nächste Byte ist wieder oben auf dem Bildschirm in der nächsten Zeichenposition, und so weiter. Am Ende der obersten Textzeile liegen die Bytes 39*8 bis 39*8+7, gefolgt von Byte 40*8 oben links in der zweiten Textzeile.
Zuerst wollen wir die Zeichenposition auf dem Bildschirm berechnen. Immer, wenn Y um 8 erhöht wird (von oben nach unten gemessen), bewegen wir uns um eine Zeile mit 40 Zeichen nach unten, d. h. wir beginnen mit LL*INT(Y/8), wobei die Zeilenlänge LL auf 40 gesetzt ist. Entsprechend gehen wir jedesmal, wenn X um 8 erhöht wird, um ein Zeichen weiter, also um INT(X/8). Das zugehörige Byte des Farbspeichers muß eventuell mit der Farbe gepoket werden, und ein Vergleich der Zeichenposition mit 0 und dem Maximalwert 999 verrät uns, ob der Punkt außerhalb des Bildschirms liegt. Wir wollen die Routine 'Setzen eines Punktes' bei 8500 unterbringen:

```
8500 CS=LL*INT(Y/8)+INT(X/8):                REM CHARACTER
                                             POSITION
8510 IF CS< 0 OR CS> MX THEN RETURN:         REM MISSED THE
                                             SCREEN
```

Die Byteadresse erhält man jetzt, indem man 8*CS plus (Y AND 7) zur Anfangsadresse SC des Bildschirms addiert, also:

```
8520 SS=SC+8*CS+(Y AND 7)
```

Jetzt müssen wir das Byte in SS mit einem Wert poken, der durch OR mit dem Bit verknüpft ist, das angezeigt werden soll. Dieses Bit wiederum ist durch das Bit (X AND 7) gegeben und kann als 2↑(7–(X AND 7)) berechnet werden. Um Rechenzeit zu sparen, werden die Werte in einem Feld BP(7) gehalten, und wir bekommen:

```
8530 POKE SS,PEEK(SS) OR BP(X AND 7)
```

Wir prüfen, ob das zu schreibende Zeichen neu ist; falls das der Fall ist, setzen wir die Farbe:

```
8540 IF CS<> OS THEN POKE CC+CS,CL:OS=CS
```

dann

```
8550 RETURN
```

Es bleibt noch etwas Verwaltungsarbeit zu erledigen:

```
10000 Y=0:X=0:CS=0:SS=0:DIM BP(7)
10010 LL=40:MX=999
```

und

```
10050 FOR I=0 TO 7:BP(I)=2↑(7-I):NEXT
```

Die Routine läßt sich bis hierher ausprobieren, wenn Zeile 140 wie folgt geändert wird:

```
140 FOR A=0 TO 6.3 STEP .01
```

mit

```
150 X=INT(100+90*COS(A)):Y = INT(100+90*SIN(A))
160 GOSUB 8500:NEXT
170 GOSUB 9100 :REM SET NORMAL SCREEN
100 CL=33:CH=0
```

und nach dem Löschen des Bildschirms müßte ein Kreis erscheinen.

ZIEHEN VON GERADEN

Ein wesentlicher Teil jedes Grafikpakets ist die Fähigkeit, eine geneigte Gerade von XH, YH nach XT, YT zu ziehen (H für Here und T für There). Später werden uns ähnliche Routinen begegnen, wenn wir Roboter programmieren, sich schräg zu bewegen. Diese Routine wollen wir bei 8000 unterbringen.

```
8000 DX=XT-XH:DY=YT-YH:                REM CALCULATE SIZE OF
                                       MOVE
8010 IF ABS(DY)> ABS(DX) THEN 8100:    REM Y MOVE IS THE
                                       GREATER
8020 IF ABS(DX)< 1 THEN RETURN:        REM NO LINE, GO HOME
8030 Y=YH:RA=DY/ABS(DX):               REM SLOPE OF LINE
8040 FOR X=XH TO XT STEP SGN(DX):      REM FROM HERE TO
                                       THERE
8050 GOSUB 8500:Y=Y+RA:NEXT
8060 XH=XT:YH=YT:RETURN:               REM HERE IS NOW THERE

8100 IF ABS(DY)< 1THEN RETURN:         REM NO LINE
8110 X=XH:RA=DX/ABS(DY)
8120 FOR Y=YH TO YT STEP SIGN (DY)
8130 GOSUB 8500:X=X+RA:NEXT
8140 XH=XT:YH=YT:RETURN
```

Um diesen Teil zu testen, müssen die Zeilen 140 bis 160 wie folgt geändert werden:

```
140 XH=190:YH=100:FOR A=0 TO 50 STEP 2.2
150 XT=INT(100+90*COS(A)):YT=INT(100+90*SIN(A))
160 GOSUB 8000:NEXT
```

DER UMRISSFÜLLER

Wir müssen Umrisse mit Farbe füllen können. Dazu bringen wir ein Unterprogramm bei 9500 unter, das aufgerufen wird, nachdem die Variablen X und Y gesetzt sind. Wenn der Umriß-Speicher SH(Y) belegt ist, wird der Umriß von SH(Y) bis X gefüllt. Wenn nicht, wird das durch SH(Y) = −1 angezeigt, und SH(Y) wird gleich X gesetzt, worauf das Programm zurückkehrt.
Zusätzliche Verwaltung in Form der nächsten Zeilen ist erforderlich:

```
10060 M8=255:XM=319:YM=199:SH=0: REM MAX X,Y SHAPE FLAG
10070 JX=54297:JY=54298:REM ANALOGUE ADDRESSES (FOR
      LATER)
10080 DIM SH(200),FL(7),FR(7):REM SHAPE,ROWS OF BITS
10090 FOR I=0 TO 7:FL(I)=2*BP(I)-1:FR(I)=BP(I)-1:NEXT
```

und die Routine erhält folgende Gestalt:

```
8600 IF Y< 0 OR Y> YM OR X< 0 OR X> XM THEN RETURN
8610 X1=X:X2=SH(Y):SH(Y)=X:REM SET SHAPE TO NEW POINT
8620 IF X2< 0 THEN 8500:REM SHAPE WAS NOT SET, JUST DRAW
     EDGE
8630 IF X1> X2 THEN X1=X2:X2=X:REM SWAP
8640 CS=LL*INT(Y/8)+INT(X1/8):POKE CC+CS,CL:REM COLOUR
8650 SS=SC+8*CS+(Y AND 7)
8660 I=(INT(X2/8)-INT(X1/8))*8:REM I=0 SAME BYTE, 8 IF NEIGH-
     BOURS
8670 IF I=0 THEN POKE SS,PEEK(SS)OR(FL(X AND 7)-FR(X2 AND
     7)):RETURN
8680 POKE SS,PEEK(SS)OR FL(X1 AND 7):REM LH END OF LINE
8690 POKE SS+I,PEEK(SS+I)OR (M8-FR(X2 AND 7)):REM RH END
8700 POKE CC+CS+I/8,CL
8710 IF I=8 THEN RETURN:REM NO MIDDLE TO FILL
8720 X1=SS+8:X2=SS+I-8:CS=CC+CS+1:FOR SS=X1 TO X2
     STEP 8
8730 POKE SS, M8:POKE CS,CL:CS=CS+1
8740 NEXT:RETURN
```

Wir brauchen außerdem noch eine Routine, um das Umriß-Feld auf −1 rückzusetzen und ein Statusbit (Flag) rückzusetzen:

```
9600 IF SH> 0 THEN FOR I=0 TO YM:SH(I)= -1:NEXT
9610 SH=0:RETURN
```

Jetzt zur Routine, die den Umrißfüller aufruft: Sie ähnelt der Routine zum Zeichnen von Geraden, arbeitet aber nur ein einziges Mal für jeden Wert von Y:

```
8200 DX=XT-XH:X=XH
8210 IF ABS(YT-YH)< 1 THEN XH=XT:X=XT:Y=YH:GOSUB
     8600:RETURN
8220 R=DX/ABS(YT-YH)
8230 FOR Y=YH TO YT STEP SGN(YT-YH)
8240 GOSUB 8600:X=X+R:NEXT
8250 XH=XT:YH=YT:RETURN
```

Ändern sie 160 in

```
160 GOSUB 8200:NEXT
```

um den neuen Abschnitt auszuprobieren.

DIE JOYSTICK-ROUTINE

Bis jetzt haben wir Routinen entwickelt, die innerhalb eines Programms aufgerufen werden können, um Grafik auf dem Bildschirm zu erzeugen. Wie verbinden wir sie mit Joystick-Bewegungen? Wir benötigen eine Joystick-Routine, die zwei analoge Werte liest und dann in Bewegungen auf dem Bildschirm im Bereich 0 bis 320 für X und 0 bis 200 für Y übersetzt. Da der analoge Wert eine ganze Zahl im Bereich 0 bis 255 ist, haben wir die Wahl, an Auflösung in der X-Richtung zu verlieren oder das rechte Fünftel des Bildschirms zu verschenken – letzteres ist wohl das geringere Übel. In der gewählten Position muß dann ein Punkt erscheinen, der sich bewegt, wenn der Joystick in eine neue Position gebracht wird. Ein solcher Punkt ist am besten zu erkennen, wenn er blinkt. Deshalb müssen wir zunächst das Byte des Bildschirmspeichers in der alten Joystick-Position wiederherstellen, das in der Variablen OB (old byte) abgelegt ist, und ein neues Byte in die neue Position setzen, bei dem das entsprechende Bit den umgekehrten Wert hat. Dann fangen wir wieder von vorn an.

```
5000 POKE SJ,OB:REM PUT BACK OLD BYTE
5010 XT=PEEK(JX):REM READ ANALOGUES
5020 YT=PEEK(JY)
5030 CS=LL*INT(YT/8)+INT(XT/8)
5040 SJ=SC+8*CS+(YT AND 7):REM ADDRESS OF SCREEN BYTE
5050 OB=PEEK(SJ)
5060 I=BP(XT AND 7):POKE SJ,(OB OR I)-(OB AND I):REM FLIP BIT
5070 RETURN
```

Wenn Sie das Programm einmal ausprobieren, fällt Ihnen möglicherweise auf, daß der gewählte Punkt ein wenig zittert. Durch digitales Filtern innerhalb des Programms läßt sich das ohne weiteres abschwächen. Betrachten Sie zunächst die einfache Anweisung:

```
X=PEEK(JX)
```

Sobald der analoge Wert sich ändert, ändert sich der Wert von X entsprechend. Betrachten Sie statt dessen:

```
X=X+(PEEK(JX)-X)/2
```

Wenn der PEEK-Wert eine Zeitlang Null gewesen ist und sich dann plötzlich auf 100 ändert, ist der nächste Wert von X gleich 0+(100-0)/2=50. Der folgende Wert ist dann 50+(100-50)/2=75, und so weiter. Jedesmal wird im Programm die Differenz zwischen X und dem PEEK-Wert halbiert, so daß schließlich PEEK von X

eingeholt wird, aber die Wirkung plötzlicher Änderungen geglättet ist. Wenn in der Programmzeile eine Zahl größer als 2 verwendet wird, ist die Glättung stärker, aber X braucht auch länger, um PEEK einzuholen. Dieses Glättungsverfahren nennt man 'Tiefpaßfilter'. Es erzielt den gleichen Effekt wie der Einbau eines Serienwiderstandes und eines Shunts in den analogen Schaltkreis. Schreibt man die Programmzeile als

```
X=X+(PEEK(JX)-X)/F
```

läßt sich der Wert von F so wählen, daß ganz unterschiedliche Zeitkonstanten erzielt werden. Je größer die Zeitkonstante ist, um so größer ist die Wirkung des Zitterrauschens, um so langsamer reagiert aber auch der Joystick.
Wenn Sie also filtern wollen, müssen die Zeilen 5010 und 5020 ersetzt werden durch

```
5010 XT=XT+(PEEK(JX)-XT)/F
5020 YT=YT+(PEEK(JY)-YT)/F
```

DER REST DES PROGRAMMS

Wir wollen jetzt erledigen, was noch an Verwaltung übrig ist. Zunächst müssen wir den Bildschirm löschen und die Werte von OB und SJ initialisieren. Wir erleichtern uns das Leben, wenn wir die Routine, die den Bildschirm löscht, von der 'Versuchs'-Position bei 110 nach 9700 bringen:

```
9700 FOR I=CC TO CC+999:POKE I,CL:NEXT:REM CLEAR COLOUR
9710 FOR I=SC TO SC+7999:POKE I,0:NEXT:REM CLEAR DISPLAY
9720 FOR I=SC TO SC+3POKE I,M8:NEXT
9730 RETURN:REM SET TOP LEFT CHAR TO SHOW SET COLOUR
```

Dann können wir ergänzen:

```
10100 CL=22:GOSUB 9000:GOSUB 9700:REM WHITE ON BLUE
10110 F=4:GOSUB 5010:REM JOYSTICK SET-UP
10120 GOTO 100
```

Jetzt ist alles für die Hauptschleife des Programms vorbereitet.
Erst ist der Joystick zu lesen und seine Position als Punkt anzuzeigen:

```
100 GOSUB 5000
```

Dann prüfen wir, ob eine Taste angeschlagen ist. Wenn nicht, durchlaufen wir die Joystick-Testschleife. Der Tastendruck-Test ist etwas ungewöhnlich, denn wir möchten erreichen, daß der Benutzer eine Taste festhalten kann, um eine Operation zu wiederholen. Für diese Art Anweisung stellt die automatische Repeat-Funktion eine Bedrohung dar, und wir sehen uns besser das Echo des Tastencodes in der Adresse 197 an; es hat den Wert 64, wenn keine Taste gedrückt ist. Falls eine Taste festgehalten wird, liefert GET B$ beim ersten Mal den Stringwert der Taste und danach " ". Die nächsten Zeilen sollen diesen Zweck erfüllen:

```
110 GET B$
120 IF B$<>"" OR PEEK (197)=64 THEN A$=B$
```

Falls keine Taste angeschlagen ist, wird die Schleife neu begonnen:

```
130 IF A$="" THEN 100
```

Ist die Taste ein 'P'? Wenn ja, wird der Punkt gesetzt, der Bildschirm gelöscht (wenn nötig), und die Schleife neu begonnen:

```
140 IF A$<>"P" THEN 170
150 OB=OB OR BP(XT AND 7):GOSUB 9600
160 XH=XT:YH=YT:A$="":GOTO 100
```

Ist die Anweisung 'L'? Wenn ja, wird die Linie gezogen und die Schleife neu begonnen:

```
170 IF A$="L" THEN GOSUB 8000:GOTO 100
```

Ist die Anweisung 'A', wird der Umrißfüller aufgerufen, ist sie '!', wird der Bildschirm gelöscht:

```
180 IF A$="A"THEN SH=1:GOSUB 8200:GOTO 100
190 IF A$="!" THEN GOSUB 9700:GOTO 100
```

Sonst ist die Anweisung 'C' – oder sie wird nicht erkannt. In beiden Fällen kehren wir zum normalen Bildschirm zurück:

```
200 GOSUB 9100: IF A$<>"C" THEN 240
210 PRINT "FOREGROUND COLOUR (0–15), BACKGROUND":
    INPUT I,CL
220 GOSUB 9000:CL=(CL+16*I) AND M8
230 POKE CC,CL:GOTO 100:REM SHOW AT TOP LEFT
```

Bei einer nicht identifizierten Anweisung wird eine Nachricht auf den Bildschirm gegeben und auf einen weiteren Tastendruck gewartet:

```
240 PRINT CHR$(147)"P-POINT     L-LINE"
250 PRINT "A-AREA     C-COLOUR"
260 PRINT "!-CLEAR SCREEN"
270 GET A$:IF A$=""THEN 270
280 GOSUB 9000:A$="":GOTO 100
```

Jetzt können Sie Ihrer künstlerischen Begabung freien Lauf lassen. Sie brauchen eine sehr ruhige Hand, um den Joystick zu führen und gleichzeitig eine Taste zum ununterbrochenen Schreiben festzuhalten. Das Resultat kann sehr beeindruckend sein – besonders wenn Sie mit einem Grafikdesigner verheiratet sind.

Grafik mit dem Joystick

```
5 POKE 56,64:CLR:REM PROTECT GRAPHIC MEMORY AREA
10 GOTO 10000
100 GOSUB 5000
110 GET B$
120 IF B$<>"" OR PEEK(KB)=KO THEN A$=B$
130 IF A$="" THEN 100
140 IF A$<>"P"THEN 170
150 OB=OB OR BP(XT AND 7): GOSUB 9600
160 XH=XT:YH=YT:A$="":GOTO 100
170 IF A$="L" THEN GOSUB 8000: GOTO 100
180 IF A$="A" THEN GOSUB 8200: GOTO 100
190 IF A$="!" THEN GOSUB 9700: GOTO 100
200 GOSUB 9100: IF A$<>"C" THEN 240
210 PRINT "FOREGROUND COLOUR (0-15), BACKGROUND"
215 INPUT I,CL
220 GOSUB 9000: CL=(CL+16*I)AND M8
230 POKE CC,CL: GOTO 100:   REM SHOW AT TOP LEFT
240 PRINT CHR$(147)"P-POINT     L-LINE"
250 PRINT "A-AREA       C-COLOUR"
260 PRINT "!-CLEAR SCREEN"
270 GETA$: IFA$="" THEN 270
280 GOSUB 9000:A$="":GOTO 100
5000 POKE SJ,OB:            REM PUT BACK OLD BYTE
5010 XT=XT+(PEEK(JX)-XT)/F
5020 YT=YT+(PEEK(JY)-YT)/F
5030 CS=LL*INT(YT/8)+INT(XT/8)
5040 SJ=SC+8*CS+(YT AND 7)
5050 OB=PEEK(SJ)
```

```
5060 I=BP(XT AND 7):POKE SJ,(OB OR I)-(OB AND I)
5070 RETURN
8000 DX=XT-XH: DY=YT-YH
8010 IF ABS(DY) > ABS(DX) THEN 8100
8020 IF ABS(DX)<1 THEN RETURN
8030 Y=YH: RA=DY/ABS(DX)
8040 FOR X=XH TO XT STEP SGN(DX)
8050 GOSUB 8500: Y=Y+RA: NEXT
8060 XH=XT: YH=YT: RETURN
8100 IF ABS(DY)<1 THEN RETURN
8110 X=XH: RA=DX/ABS(DY)
8120 FOR Y=YH TO YT STEP SGN(DY)
8130 GOSUB 8500: X=X+RA: NEXT
8140 XH=XT: YH=YT: RETURN
8200 DX=XT-XH:X=XH:XH=XT
8210 IF YT=YH THEN X=XT:Y=YH: GOSUB 8600: RETURN
8220 R=DX/ABS(YT-YH)
8230 FOR Y=YH TO YT STEP SGN(YT-YH)
8240 GOSUB 8600: X=X+R: NEXT
8250 XH=XT: YH=YT: RETURN
8500 CS=LL*INT(Y/8) + INT(X/8)
8510 IF CS<0 OR CS>MX THEN RETURN
8520 SS=SC+8*CS+(Y AND 7)
8530 POKE SS,PEEK(SS) OR BP(X AND 7)
8540 IF CS<>OS THEN POKECC+CS,CL: OS=CS
8550 RETURN
8600 IF Y<0 OR Y>YM OR X<0 OR X>XM THEN RETURN
8610 X1=X: X2=SH(Y): SH(Y)=X
8620 IF X2<0 THEN 8500
8630 IF X1>X2 THEN X1=X2:X2=X
8640 CS=LL*INT(Y/8)+INT(X1/8):POKE CC+CS,CL
8650 SS=SC+8*CS+(Y AND 7)
8660 I=(INT(X2/8)-INT(X1/8))*8:IF I=0 THEN 8750
8680 POKE SS,PEEK(SS)ORFL(X1 AND 7)
8690 POKE SS+I,PEEK(SS+I)OR(M8-FR(X2 AND 7))
8700 POKE CC+CS+I/8,CL
8710 IF I=8 THEN RETURN
8720 X1=SS+8:X2=SS+I-8:CS=CC+CS+1
8725 FORSS=X1 TO X2 STEP 8
8730 POKE SS,M8: POKE CS,CL:CS=CC+1
8740 NEXT: RETURN
8750 POKE SS,PEEK(SS)OR(FL(X1 AND7)-FR(X2 AND7))
8760 RETURN
9000 POKE CI,PEEK(CI) AND (255-1):REM SET BANK 1
9010 POKE VV+24,8+16*7
```

```
9020 POKE GF,GM:                    REM GRAPHICS MODE
9030 RETURN
9100 POKE CI,PEEK(CI) OR 3:         REM SET BANK 0
9110 POKE VV+24,20
9120 POKE GF,CM:                    REM CHARACTER MODE
9130 RETURN
9600 IF SH>0 THEN FOR I=0 TO YM: SH(I)=-1: NEXT
9610 SH=0: RETURN
9700 FOR I=CC TO CC+999: POKE I,CL: NEXT
9710 FOR I=SC TO SC+7999:POKE I,0: NEXT
9720 FOR I=SC TO SC+3: POKE I,M8: NEXT
9730 RETURN: REM TOP LEFT CHARACTER SHOWS COLOUR
10000 Y=0: X=0: CS=0: SS=0: DIM BP(7)
10010 LL=40: MX=999:KB=197:KO=64
10020 SC=1*16384+8*1024:REM SCREEN=BANK1 SECTOR8
10025 CC=1*16384+7*1024:REM COLOUR=BANK1 SECTOR7
10030 VV=13*4096:CI=VV+13*256:REM VIDEO CHIP,CIA
10040 GF=VV+17: CM=PEEK(GF): GM=CM OR 32
10045 REM GRAPHICS FLAG,CHAR. MODE,GRAPHICS MODE
10050 FOR I=0 TO 7: BP(I)=2^(7-I): NEXT
10060 M8=255:XM=319:YM=199:SH=0:      REM MAX X,Y
10070 JX=54297:JY=54298:     REM ANALOGUE ADDRESS
10080 DIM SH(200),FL(7),FR(7)
10090 FOR I=0 TO  7
10095 FL(I)=2*BP(I)-1: FR(I)=BP(I)-1:NEXT
10100 CL=22:GOSUB 9000:GOSUB 9700:REM WHITE/BLUE
10110 F=8: GOSUB 5010
10120 SH=1:GOSUB9600
10200 GOTO 100
```

Sprite-Editor

```
10 GOTO2000:     REM SPRITE EDITOR STARTS AT 2000
200 X=160:Y=100:DX=0:DY=0:X2=0:Y2=0:REM FUN DEMO
210 A=20:B=300:AY=40:BY=230
220 M=255:V4=V+4:V5=V+5:VH=V+16
300 X2=.1*(RND(1)-.5):Y2=.4*(RND(1)-.3)
310 X1=X1+X2:Y1=Y1+Y2
320 IF X<A THEN X1=ABS(X1)
325 IF X>B THEN X1=-ABS(X1)
330 IF Y<AY THEN Y1=ABS(Y1)
335 IF Y>BY THEN Y1=-ABS(Y1)
340 X=X+X1:Y=Y+Y1
```

```
350 POKE V5,Y:POKE V4,X AND M:POKE VH,-4*(X>M)
360 GOTO 300
2000 V=53248:POKEV+21,4:POKE2042,13
2005 SB=832:POKEV+4,100:POKEV+5,100
2010 POKEV+23,4:POKEV+29,4:PRINT CHR$(147);
2020 E=0:B=7
2030 U$=CHR$(145):D$=CHR$(17):  REM CURSOR CHARS
2040 L$=CHR$(157):R$=CHR$(29)
2050 PRINT"SPACE TO CLEAR, . TO SET"
2060 PRINT"CURSOR TO MOVE, X TO END"
2100 P=PEEK(SB+E):C=2^B:Q=(P OR C)-C
2200 POKE SB+E,Q:FORI=1TO10:NEXT:POKE SB+E,Q ORC
2210 GET A$:IFA$=""THEN:FORI=1TO10:NEXT:GOTO2200
2220 POKE SB+E,P
2230 IFA$=" "THEN P=Q:A$=R$:POKE SB+E,P
2240 IFA$="."THEN P=Q OR C:A$=R$:POKE SB+E,P
2250 IFA$=R$ THEN B=B-1:IFB>=0 THEN 2100
2260 IFA$=R$ THEN B=7:E=E+1:IFE>62 THEN E=0
2270 IFA$=L$ THEN B=B+1:IF B<8 THEN 2100
2280 IFA$=L$ THEN B=0:E=E-1:IF E<0 THEN E=63
2290 IFA$=D$ THEN E=E+3:IF E>62 THEN E=E-63
2300 IFA$=U$ THEN E=E-3:IF E<0 THEN E=E+63
2310 IFA$="X"THEN PRINT CHR$(147):GOTO 200
2400 GOTO 2100
```

KAPITEL 4

LOGISCHE EIN- UND AUSGABE

In Kapitel 2 wurde schon ein wenig über die internen Vorgänge im Computer und den Unterschied zwischen analogen und digitalen Signalen gesprochen. Selbst wenn die externen Signale digital sind, ist es nicht ganz einfach, die Aufmerksamkeit des Computers darauf zu lenken.

DIGITALE SCHNITTSTELLEN

Das Rückgrat des Computers besteht aus zwei 'Bussen', Signalbündeln, die die meisten Komponenten miteinander verbinden. Der einfachste davon ist der Datenbus. Wenn der Prozessor (ein anderer Name für Mikroprozessor-Chip) ein Datenbyte im Speicher ablegen möchte, legt er die zugehörigen Logikspannungen an den Achtbit-Datenbus, erteilt eine Anweisung, und der entsprechende Speicherplatz merkt sich die Information. Wenn er ein Byte zurückholen will, sei es ein Datenbyte oder die nächste Anweisung in seinem Programm, sendet er einen entsprechenden Befehl aus; der Speicher sucht das Byte heraus und legt einen entsprechenden Satz von Logikspannungen an den Datenbus, die der Prozessor dann liest. Wie Sie sehen werden, ist der Datenbus außerordentlich beschäftigt, und der Versuch, ihm externe Signale zuzuführen, kann der Prozessor abtrudeln lassen.

Um den Speicher herumzukommandieren, muß der Prozessor eine Adresse bestimmen können. Hier begegnen wir dem zweiten Bus, dem Sechzehnbit-Adreßbus. 65 536 verschiedene Adressen kann der Bus unmittelbar spezifizieren, aber mit etwas Mogeln läßt er sich so erweitern, daß er einen Speicher jeder Größe adressiert, die Sie sich leisten können. Über eine weitere wichtige Leitung kann der Prozessor dem Speicher mitteilen, ob er Daten lesen oder schreiben will.

Was hat das alles mit Schnittstellen zu tun? Offensichtlich muß irgend etwas zwischen jede Logikeingabeleitung und den Datenbus geschaltet werden, damit die Daten nur für den kurzen Augenblick auf den Bus gegeben werden, in dem der Prozessor sie lesen möchte. Das ist die Aufgabe des Interface-Chips. Das bedeutet zum Beispiel, daß acht Pins des Chips zur Bequemlichkeit des Benutzers mit einem Steckplatz am Gerät verbunden werden können. Wenn Sie einen Stecker auf diesen Steckverbinder schieben, können Sie zusätzliche Tasten, Sensorkontakte oder irgendein anderes logisches Signal anschließen und mit einer oder zwei Programmanweisungen in den Computer lesen. Das also ist der User-Port.

ANSCHLÜSSE FÜR DEN USER-PORT

Bevor wir uns mit dem User-Port befassen, ist es ratsam, ihn in Reichweite zu bringen. Dazu stellen wir ein Verbindungskabel ähnlich dem in Kapitel 2 beschriebenen her, um die Signale zu einer Anschlußleiste neben der Tastatur zu führen. Der Anschluß ist in diesem Fall eine 12wegige doppelseitige Lüsterklemmenleiste mit 24 Kontakten und 0,156 Zoll Abstand. Geeignet sind Cinch 251.12.90.160, Amp 530657-3 und Teka TP3-121-E04. Die Anschlüsse der Logikdatenleitung sind für den C 64 und den PET gleich, aber die +5V-Leitung auf dem Steckverbinder des C 64 fehlt beim PET. PET-Besitzer müssen einen zweiten Kontakt eines der beiden Kassetten-Ports benutzen (gegen den Rat der PET-Handbücher – aber in Ordnung bis 50 mA).
Die Kontaktbelegung ist (von außen betrachtet):

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	E	F	H	J	K	L	M	N

Strip:	1	2	3	4	5	6	7	8	9	10	11	12
C 64:	2	B	C	D	E	F	H	J	K	L	M	N
Signal:	+5V	Flg	PB0	PB1	PB2	PB3	PB4	PB5	PB6	PB7	PA2	Gnd
Pet:	*	B	C	D	E	F	H	J	K	L	M	N
Signal:	+5V	CA1	PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7	CB2	Gnd

*(von Pin B oder 2 eines PET-Kassetten-Ports)

Nachdem Sie die Anschlußleiste mit dem Computer verbunden haben, prüfen Sie die Signale mit dem Multimeter. Legen Sie die negative Prüfschnur an Position 12. Prüfen Sie auf +5 V an Position 1 – falls nichts da ist, ist der Steckverbinder vielleicht falsch herum. Die Signale PB0 bis PB7 (oder PA0 bis 7) – wir wollen sie von nun an P0 und P7 nennen – prüft man auf eine etwas mysteriöse Art, die später in diesem Kapitel klar werden wird. Geben Sie dieses Programm ein:

```
10 PO=56577 : REM : *** CBM 64
```

oder

```
10 PO=59471 : REM : *** PET
20 PRINT 255-PEEK(PO)
30 GOTO 20
```

Lassen Sie das Programm laufen, und verbinden Sie einen Draht von 0 V nacheinander mit P0, P1 bis P7. Auf dem Bildschirm müßten entsprechend die Werte 1, 2, 4, 8, 16, 32, 64 und 128 erscheinen.

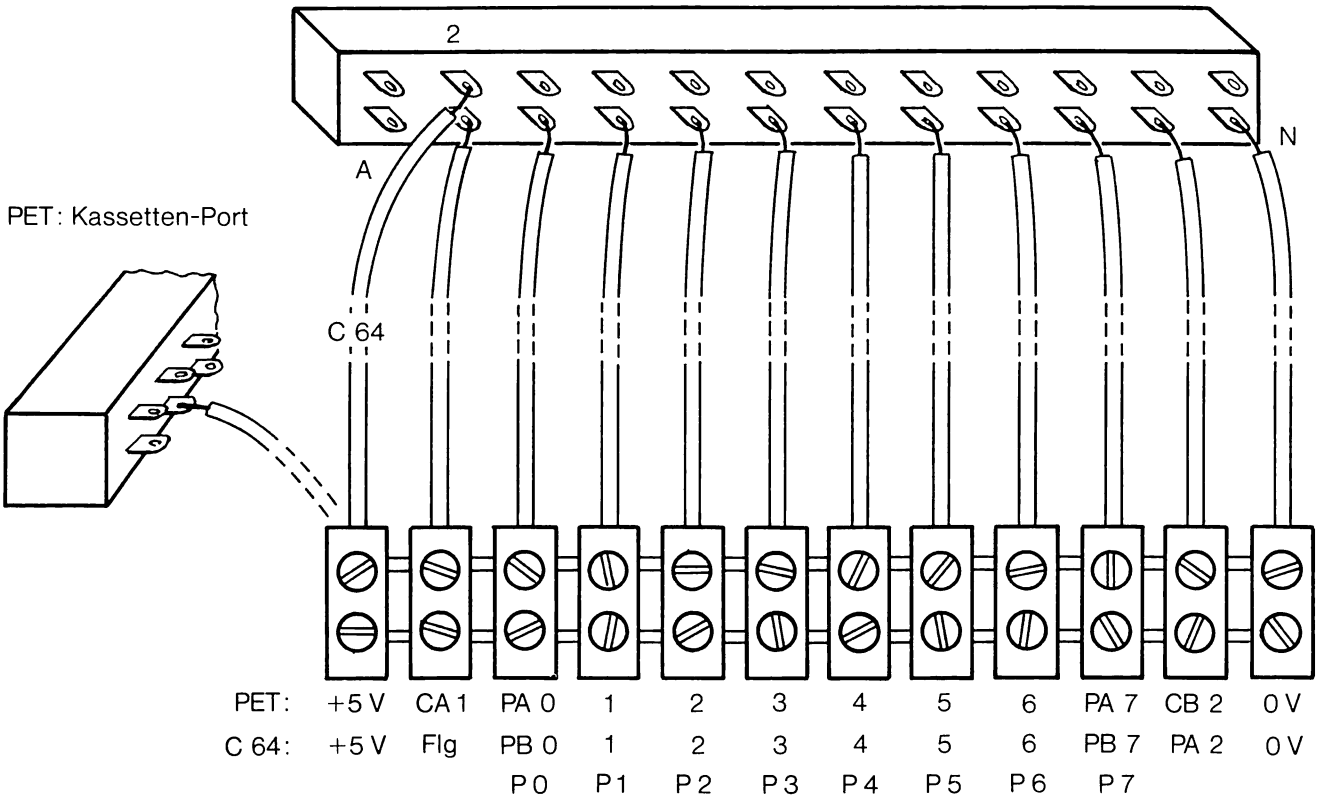


Abbildung 4.1 Anschlußleiste für den User-Port

WIE ARBEITET DIE SCHNITTSTELLE?

Jetzt sind wir endlich soweit, uns die Arbeitsweise des User-Ports anzusehen. Die Signale, die uns zunächst interessieren, sind P0 bis P7, die beim C 64 vom B-Port eines Complex Interface Adapters herrühren, und beim PET vom A-Port eines Versatile ('vielseitigen') Interface Adapters. Der VIA des PET ist so vielseitig, daß er in kürzester Zeit verwirren kann. Er ist ein einzelner Chip, ein 6522, und memory-mapped bei Adresse $E840 (das $ steht für hexadezimal). Wenn Ihnen das etwas sagt, überschlagen Sie die nächsten Abschnitte. Dem CIA 6526 reicht es nicht, vielseitig zu sein; er ist dazu auch noch komplex. Er ist memory-mapped bei $DD00.

Frühere Computersysteme verwendeten ein spezielles Bussystem zur Ein-Ausgabe-Steuerung, und viele Mikrocomputer haben immer noch besondere Ein-Ausgabe-Anweisungen. Man hat jedoch rasch bemerkt, daß Eingänge und Ausgänge behandelt werden können, als ob es Speicherplätze wären. Wenn eine Zahl im Speicher abgelegt wird, sagen wir in Adresse $1234, werden Spannungen innerhalb der Schaltungsanordnung eines Speicherchips verändert. Wenn diese verstärkt und mit der Außenwelt verbunden würden, könnten sie acht Ausgangsleitungen steuern, indem beim Speichern des Wertes Null alle Leitungen auf Low gesetzt würden, bei 255 hingegen auf High. Wenn andererseits der Inhalt eines Speicherplatzes geladen wird, werden die Logikwerte der in einem Chip gespeicherten Spannungen in den Akkumulator des Mikrocomputers kopiert. Falls diese Signale statt von gespeicherten Spannungen von Drähten kämen, die an acht Spannungen in der Außenwelt angeschlossen sind, hätten wir acht Eingänge. Ein Interface-Chip kann also so angelegt sein, daß er große Ähnlichkeit mit einem Speicherchip hat – und einige Firmen stellen Chips her, die beide Funktionen vereinigen.

ADRESSDECODIERUNG

Die sechzehn Adreßleitungen des 6502-Mikrocomputers können 65 536 getrennte Speicherbytes direkt adressieren – viel mehr, als in einen durchschnittlichen Speicherchip passen. Die oberen paar Adreßbits werden deshalb so decodiert, daß sie einzelne Speicherbits adressieren, während die übrigen Bits parallel auf alle Chips geschaltet werden und die Adresse innerhalb des ausgewählten Chips bestimmen. (Das stimmt nicht ganz für einige RAM-Arten, aber lassen Sie das im Augenblick außer acht.) Es brauchen nicht alle diese Speicherchips vorhanden zu sein, damit das Verfahren läuft; es können also leere Stellen in der Memory-Map der Maschine sein.

Wenn die oberen vier Leitungen decodiert sind, existieren sechzehn 'Chip-Select'-Leitungen, wobei die erste auf die Adressen von $0000 bis $0FFF anspricht, die

nächste von $1000 bis $1FFF, und so weiter. Eine davon, zum Beispiel $D, könnte einen weiteren Decodierer zum Decodieren der nächsten vier Leitungen aktivieren, und es ergäben sich sechzehn weitere Signale, die auf die Adressen $D0XX,$D1XX, . . .,$DFXX ansprächen, wobei XX zwei beliebige Hexadezimalziffern sind. Eine dieser Leitungen, sagen wir die zu $DDXX, kann wiederum einen weiteren Decoder aktivieren, worauf sich weitere sechzehn Leitungen ergeben, die auf $DD0X, $DD1X und so weiter ansprechen. Eine dieser Leitungen schließlich, sagen wir Leitung $DD0X, könnte einen Chip mit genau 16 Speicheradressen, $DD00 bis $DD0F, aktivieren. Nehmen wir jetzt an, daß der Chip kein echter Speicherchip ist, sondern mit der Außenwelt verbunden werden kann. Wenn wir dann den Wert 7 (binär 00000111) in Adresse $DD01 ablegen, können wir acht Ausgangsleitungen steuern, wobei drei Pins auf High und die anderen fünf auf Low gehen. Das ist, in aller Kürze, das Prinzip der memory-mapped Ein-Ausgabe. Das Problem ist, daß wir diese Nuß jetzt knacken müssen – und der 6522 und der 6526 sind harte Nüsse!

PORTS UND DATENRICHTUNGSREGISTER

Sechzehn Bytes haben 128 Bits. Wenn wir getrennte Leitungen für Eingabe und Ausgabe hätten, müßte der Chip entsetzlich viele Pins besitzen. Nachdem die zur Einbindung des Chips in das Computersystem erforderlichen Signale bereitgestellt sind (18 Leitungen), plus zwei Leitungen für die Stromversorgung, bleiben von 40 Pins nicht viele übrig. Die restlichen 20 Pins sind zu zwei Ports angeordnet, jeder mit acht Datenleitungen und zwei Steuer- oder 'Handshake'-Leitungen. Einer dieser Ports ist es, der inzwischen mit der Anschlußleiste neben Ihrer Tastatur verbunden sein sollte. Für Sie sieht sie vielleicht wie eine Anschlußleiste aus, aber der Computer ist überzeugt, daß es sich um das Speicherbyte in der Adresse $DD01 (C 64) oder $E84F (PET) handelt. Jeder Port ist bidirektional, das heißt, jedes einzelne Bit kann ein Eingang oder ein Ausgang sein. Die Richtung jedes Bits ist in einem Register innerhalb des Chips hinterlegt, das (Sie ahnen es) Datenrichtungsregister heißt. Für den User-Port sieht der C 64 DDR-B in $DD03, während der PET DDR-A mit $E843 adressiert. Jedes Bit, das in dieser Adresse auf 1 liegt, wird als Ausgangsbit gewählt, während die Nullen Eingänge wählen.

Was machen wir jetzt, wenn wir uns eine bestimmte Adresse ansehen wollen? Der Ausdruck PEEK(. . .) steht für 'Adresseninhalt'. Also druckt die Anweisung PRINT 5 den Wert 5, während PRINT PEEK(5) den Inhalt von Speicherplatz 5 druckt. Ähnlich legt die Anweisung POKE 5,6 den Wert 6 im Speicherplatz 5 ab – und kann gleichzeitig das System zum Absturz bringen!

Machen Sie sich jetzt wieder mit Ihrem Multimeter bereit. Definieren Sie alle User-Port-Bits als Ausgänge, indem Sie tippen:

PO=56577:DD=56579 für den C 64

oder

PO=59471: DD=59459 für den PET,

dann POKE DD,255

Tippen Sie außerdem:

POKE PO,255

um alle Bits auf High zu setzen. Wenn Sie jetzt mit Ihrem Meßgerät nachprüfen, müßten auf P0, P1 usw. jeweils ungefähr +5 V liegen. Geben Sie nun ein:

POKE PO,0

und Sie sehen, daß P0, P1 usw. alle auf 0 V gefallen sind. Verwenden Sie jetzt die Werte 1, 2, 4, 8, 16, 32, 64 und 128, um sicherzustellen, daß Sie die Leitungen einzeln auf High setzen können. Probieren Sie dann verschiedene Kombinationen aus – sehen Sie, warum es einfacher ist, hexadezimal zu arbeiten?
Versuchen Sie nun den Port als Eingang zu schalten. Tippen Sie einfach:

POKE DD,0

und jede Leitung wird zu einem Eingang. Geben Sie folgendes Programm ein:

```
10 PRINT PEEK(56577) : REM 64
```

oder

```
10 PRINT PEEK(59471) : REM PET

20 GOTO 100
```

und lassen es laufen. Wenn Sie jetzt jeden Pin P0, P1 usw. mit einem 0 V-Draht berühren, sehen Sie die Zahl von 255 abweichen. Nach kurzem Überlegen möchten Sie vielleicht das Programm wie folgt modifizieren:

```
10 PRINT 255 – PEEK(56577) : REM OR 59471
20 GOTO 10
```

SENKEN UND QUELLEN

Sie haben natürlich bemerkt, daß Sie den Eingang mit 0 V, nicht mit 5 V, kitzeln müssen, um ihn zu ändern. Jeder Pin hat einen internen Pullup-Widerstand, der 'im Leerlauf' den Eingang bei 5 V hält, was der Computer als '1' liest. Um den Eingang auf '0' zu ziehen, muß das Eingangssignal ungefähr 1 mA Strom 'versenken' können. Dieser Strom entspricht einer 'TTL-Ladung' und beschränkt die Zahl der Logikeingänge, die ein TTL-Gatterausgang betreiben kann. Um eine gute 'Ausgangsfächerung' zu erhalten, muß ein TTL-Gatter mehrere Milliampere versenken können, braucht aber nicht wirklich 'Quelle' eines Stroms zu sein, um zu arbeiten. Die Pullup-Fähigkeit der Logikausgänge ist deshalb wesentlich höher als die Pulldown-Fähigkeit. (Obwohl der 6522 und der 6526 MOS-Bauelemente sind, sind sie TTL-kompatibel gehalten.)

NOCH VIELSEITIGER!

Damit ist der 6522 natürlich noch nicht vollständig beschrieben. Vier seiner sechzehn Adressen befassen sich mit Daten und Datenrichtungen – bleiben zwölf weitere, auf die er den Ruf der Vielseitigkeit stützen kann. Zwei Adressenpaare betreffen zwei Sechzehnbit-Zähler, die für eine Vielfalt von Zeitgeberfunktionen verwendet werden können. Eine andere steuert ein Schieberegister, mit dessen Hilfe Daten über CB2 ein- und ausgegeben werden. Ein weiteres Register, das Statusregister, zeigt die Zustände an, die einen Interrupt verursacht haben könnten, wie etwa Timer-expired, Data-Handshake usw. Schließlich ergänzen zwei weitere Register, das Peripheral Control Register und das Auxiliary Control Register, das ganze Orchester. Selbst der einfach aussehende Port B hat einige Überraschungen in petto, denn unter der Kontrolle von Zeitgeber 1 kann PB7 einen Impuls variabler Länge oder eine Impulsfolge abgeben, während PB6 als Eingang für Impulse benutzt werden kann, die dann von Zeitgeber 2 gezählt werden.

Der 6526 ist ein naher Verwandter des 6522, versucht ihn aber noch zu übertreffen. Er hat eine eingebaute Tageszeituhr, die Netzperioden zu 50 oder 60 Schwingungen zählt. Sonst ist er ihm ganz ähnlich, abgesehen von einer Verfeinerung des seriellen Augangs, die das Verfahren der pseudo-analogen Augabe völlig ruiniert, das in Kaptiel 9 für den PET mit seinem 6522 beschrieben wird. Mit zwei zusätzlichen Widerständen und einem Kondensator kann der PET ein Signal aussenden, das zur Steuerung eines analogen Servomotors ideal ist. Der C 64 muß sich andere Methoden suchen.

Von größerem unmittelbarem Interesse für PET-Besitzer ist, wie das analoge Signal eines Joysticks eingegeben werden kann, und das rechtfertigt ein eigenes kurzes Kapitel.

NETZSPANNUNGSSCHALTER

Nachdem wir uns mit dem Port auseinandergesetzt haben, wird es jetzt Zeit, ihn zu benutzen. Wenn man einen Netzschalter braucht, ist ein Halbleiterrelais ein besonders nützlicher Baustein. Das ist nichts anderes als eine optoisolierte Zweirichtungs-Thyristortriode, die eine Wechselstrom-Netzlast von mehreren Ampere nach Belieben an- und abschaltet. Vorausgesetzt, alle nötigen Vorsichtsmaßregeln sind getroffen, damit nicht vagabundierende Leiter (oder insbesondere Finger) das Signalende mit dem 'heißen' Ende der Vorrichtung kurzschließen, läßt sie sich wegen der Optoisolierung gefahrlos an den User-Port anschließen. Der Anschluß könnte kaum einfacher sein; der positive Stift wird mit +5 V verbunden und der negative mit dem Bit P, das die Einheit steuern soll. Immer dann, wenn dieses Bit als Ausgang konfiguriert ist, und wenn das entsprechende Ausgabedatenbit Null ist, ist der Schalter eingeschaltet.
Die RS-Nummer des 2,5A-Relais ist RS 348-431. Bei einem Preis von fast 40 Mark werden Sie nicht allzuviele Kanäle fahren wollen. Mit dem untenstehenden simplen Programm könnte man eine Leselampe in unregelmäßigen Abständen ein- und ausschalten. Mit einigen einfachen Veränderungen ließen sich die Abstände weniger zufällig machen und so der Eindruck erwecken, als arbeite jemand lange und gehe dann zu Bett. Nimmt man eine Fotozelle und ein oder zwei PEEK hinzu, kann das System auf das Einsetzen der Dämmerung reagieren. Passen Sie aber auf, daß ein vorbeigehender Einbrecher aus dem Aufleuchten der Lampe nicht auf das Vorhandensein eines Computers im Haus schließt!

```
 10 PRINT CHR$(147)"TIMES IN MINUTES:"
 20 INPUT "MAX ON-TIME";OM
 30 INPUT "MAX OFF-TIME";FM
 40 INPUT "MIN ON-TIME";OL
 50 INPUT "MIN OFF-TIME";FL
 60 IF (OL-OM>0)OR(FL-FM>0) THEN GOTO 10

100 PO=56577: DD =56579 : REM *** CBM 64
```

oder

```
100 PO=59471: DD=59459 : REM *** PET

110 POKE DD,1: REM MAKE BIT 0 AN OUTPUT

200 POKE PO,0: REM TURN ON LIGHT
210 T=OL+(OM-OL)*RND(1):REM BETWEEN OL, OM
220 GOSUB 1000
```

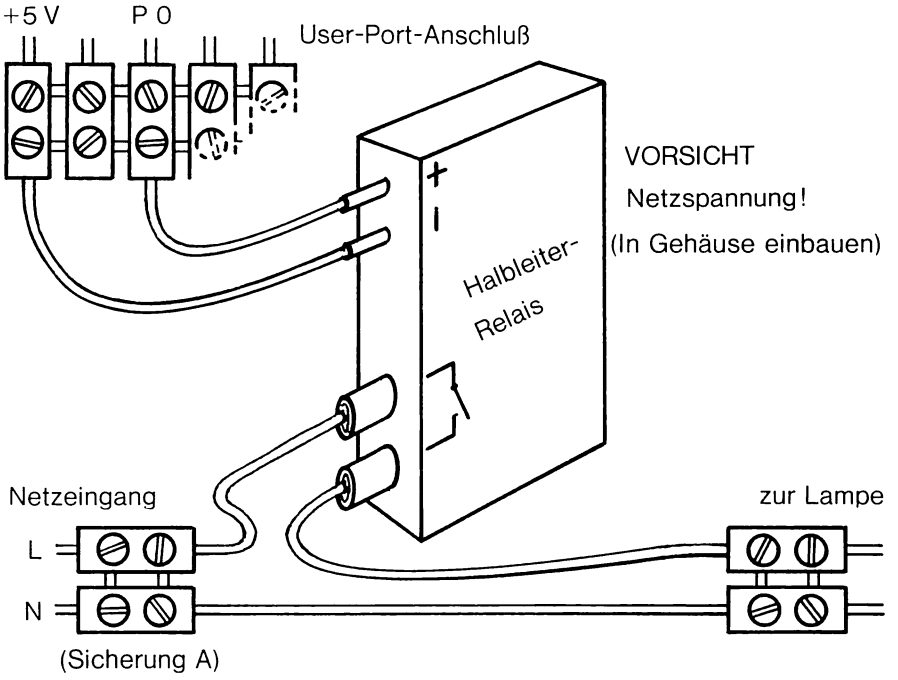


Abbildung 4.2 Festkörperrelais zur Lichtsteuerung

```
 300 POKE PO,1: REM TURN OFF LIGHT
 310 T=FL+(FM-FL)*RND(1)
 320 GOSUB 1000
 330 GOTO 200

1000 REM WAIT T MINUTES
1010 T=T*60 : REM MAKE SECONDS
1020 FOR I= 1 TO T
1030 FOR J= 1 TO 1000:NEXT REM ONE SECOND DELAY
1040 NEXT
1050 RETURN
```

Das Programm ist noch etwas primitiv, aber Sie können es gewiß um die Besonderheiten ergänzen, die Sie brauchen.

KAPITEL 5

ANALOGE EINGABE FÜR PET UND C 64

Die PET-Besitzer sind sicher inzwischen schon ungeduldig zu erfahren, wie man ein Joystick-Signal ohne zusätzlichen Analog-Digital-Chip eingibt. Die hier beschriebene Methode benötigt nicht mehr als ein einziges Bit des User-Ports. Die Wurzeln des Verfahrens reichen in die Tiefen der Antike zurück – sie sind mindestens fünf Jahre alt. Obwohl dasselbe Prinzip hinter dem im C 64 eingebauten Wandler steckt, kann es zu einer Umwandlung hoher Qualität ausgeweitet werden, die sich auch zur Instrumentierung eignet.

WIE ARBEITET DER EINGANG?

Die alten TV-Tennisspiele mit einem einzigen Chip mußten die Joystick-Signale mit den geringsten Mitteln codieren. Für Analog-Digital-Umwandlung war kein Platz – der Chip wäre ohnehin beim Verarbeiten des digitalen Wertes zu Display-Geschwindigkeiten in Bedrängnis geraten. Statt dessen war die Joystick-Variable mit einem Kondensator verbunden, was eine variable Zeitkonstante zur Folge hatte. Sehen wir uns eine horizontale Schlägerbewegung an. Zu Beginn jeder Bildschirmzeile wurde der Kondensator entladen. Während des Abtastens der Zeile lud sich dann der Kondensator über den Joystick-Widerstand auf. Wenn der Kondensator die Schwellenspannung des Eingangsanschlusses überschritt, erhellte sich der Bildschirmpunkt und schrieb das Bild des Schlägers. In anderen Worten: der Joystick-Widerstand wurde in eine Verzögerung umgesetzt, die von einem einzelnen Eingangsbit gelesen werden konnte. Können wir nicht denselben Trick mit einem Bit des User-Ports anwenden?
Versuchen Sie es zu Demonstrationszwecken zuerst mit einem großen Kondensatorwert, damit eine Schleife eines BASIC-Programms als Zeitgeber dienen kann. Reduzieren Sie danach den Kondensator, um eine schnelle Antwort in Maschinensprache zu erhalten. Beginnen Sie mit 1000 Mikrofarad (ein 6-V-Elektrolytkondensator ist tatsächlich recht klein). Schalten Sie ihn zwischen 0 V (das negative Ende) und P0. Geben Sie jetzt das folgende Programm ein und lassen es laufen:

```
10 PO=59471:DD=59459 :REM *** PET
```

oder

```
10 PO=57577:DD=56579 :REM *** C 64
```

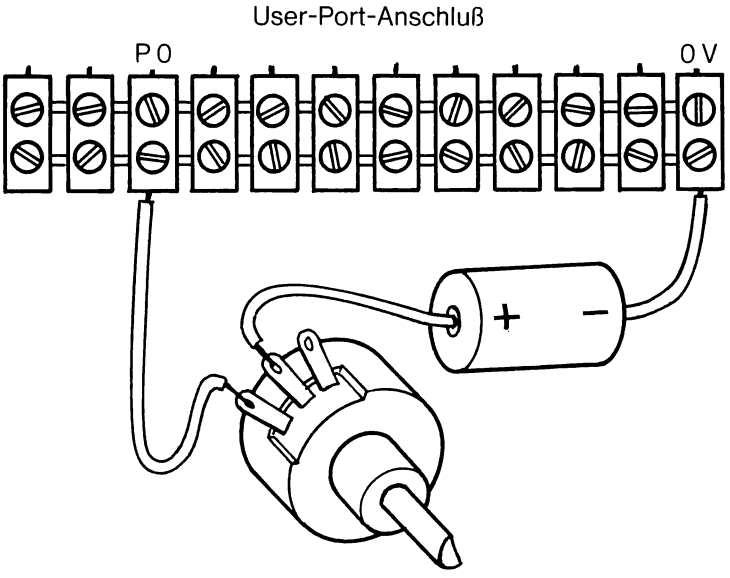
```
 20 POKE DD,1:    REM configure PB0 as an output
 30 POKE PO,0:    REM zero output to discharge capacitor
 40 GOSUB 1000:   REM brief delay

100 C=0:          REM set count to zero
110 POKE DD,0:    REM PB0 becomes an input, capacitor released
120 IF (PEEK(PO) AND 1) > 0 THEN 200: REM got to threshold?
130 C=C+1:            REM keep counting
140 GOTO 120:         REM round again
200 POKE DD,1:        REM discharge the capacitor again
210 PRINT C
220 GOSUB 1000:       REM brief delay
230 GOTO 100:         REM do it all again
1000 FOR I = TO 200 : NEXT: RETURN
```

Die auf dem Bildschirm erscheinenden Zahlen hängen vom genauen Wert des Kondensators ab. Wenn Sie die Verbindung zum Kondensator trennen, müssen die Zahlen auf 0 fallen. Schließen Sie jetzt ein 2-kOhm-Potentiometer als veränderlichen Widerstand in Reihe mit dem Kondensator an P0 an – d. h. das positive Ende des Kondensators an den Schleifer des Potentiometers, ein Ende des Potentiometers an P0, das negative Ende des Kondensators an 0 V. Wenn das Programm läuft, sollten sich die Zahlen beim Drehen am Potentiometer ändern.

EINE SCHNELLE VERSION IN MASCHINENSPRACHE

Einige Leser sind vielleicht mit Assembler- und Maschinensprache vertraut, anderen ist das Thema wohl ein völliges Rätsel. Verzweifeln Sie nicht, wenn es Ihnen nach der Lektüre der nächsten Abschnitte ein noch größeres Rätsel ist. Da die Software in Form von BASIC-Datenanweisungen angegeben ist, können Sie sie wie üblich eingeben und dann blind darauf vertrauen, daß das Programm läuft. Das Vertrauen in Ihre eigenen Tippfähigkeiten sollte nicht ganz so blind sein! Wenn Sie den Buchstaben O statt der Zahl Null tippen, stürzt das Programm ziemlich sicher ab und ist wahrscheinlich völlig verloren. Machen Sie deshalb eine Sicherungskopie, bevor Sie es laufen lassen.
Um mit annehmbarer Geschwindigkeit zu laufen, muß die Routine in Maschinensprache (d. h. Assemblersprache) umgeschrieben werden und sollte im Idealfall die Gestalt einer USR-Funktion haben, so daß Sie die Zeile

```
X=USR(CH)
```



Input geht auf 1,
wenn Schwelle überschritten wird.

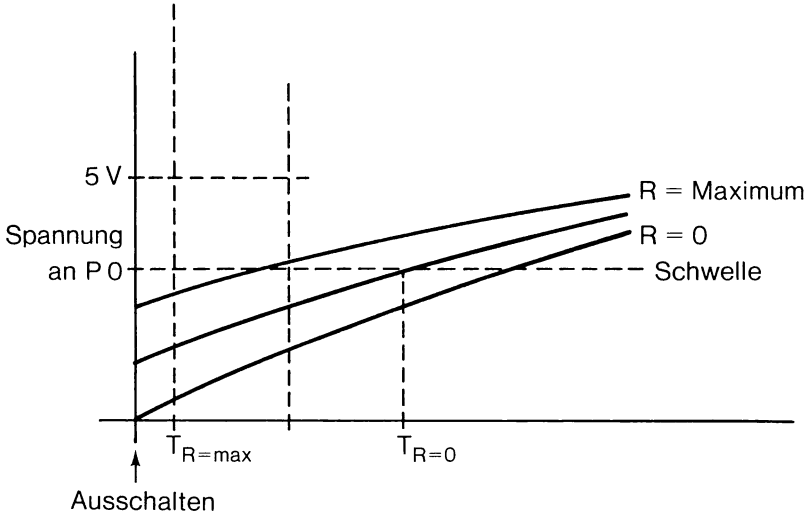


Abbildung 5.1 Eine einfache analoge Eingabe

in ein BASIC-Programm aufnehmen können, um den Wert von Kanal *CH* zu übergeben. Dazu müßte der Maschinencode an die USR-Sprungadresse gebunden und die Gleitkomma-Integer-Umwandlungsroutine (und umgekehrt) aufgerufen werden. Da unterschiedliche PETs zu berücksichtigen sind, ganz zu schweigen vom C 64, die alle ihre Routinen an anderen Stellen haben, ziehe ich es vor, das Problem anders anzugehen.
Variablen haben in allen Commodore-Geräten dieselbe Form. Insbesondere enthält das vierte Byte (d. h. Byte 3 – von 0 an gezählt) einer ganzzahligen Variablen das wertniedrigste Byte des Wertes. Wenn diese Variable nun als allererste vereinbart wurde, zeigen Bytes 42, 43 auf ihren Speicherplatz (PET 30xx, 40xx, 80xx), oder Bytes 45, 46 (CBM 64) oder Bytes 124, 125, falls Sie antike 2001er sammeln. Zwei Assembler-Anweisungen

```
LDY #$03
LDA (VARS),Y
```

laden jetzt ihren Wert in den Akkumulator. Falls die zweite zu vereinbarende Variable die ganze Zahl V% ist, kann man mit

```
LDY #$0A
STA (VARS),Y
```

das Ergebnis in V% ablegen.
Das nächste Problem ist, den Maschinencode an sicherer Stelle zu verstauen. Ein alter Favorit ist der zweite Kassettenpuffer, der in Adresse $033A=826 beginnt. In der Vergangenheit war das gewöhnlich risikolos, aber in den 40xx-PETs beginnt es dort buchstäblich von Diskettensignalen zu wimmeln. Andere bevorzugen das obere Ende des Speichers und ziehen die Decke herunter, damit keine Strings den Code zertreten. Ich selbst ziehe es vor, ein Sandwich im BASIC-Bereich zu machen, wobei die untere Scheibe bei der normalen BASIC-Startadresse liegt und eine Zeile wie die folgende (für PET 40xx) enthält:

```
10 POKE 41,12: RUN
```

Dann folgt eine dicke Scheibe Maschinencode, auf dem bei $0C01 diejenige Scheibe BASIC liegt, die den Maschinencode steuert und das Anzeigen oder Ablegen der Ergebnisse erleichtert. Zu Demonstrationszwecken jedoch, bei denen das Programm mit Datenanweisungen geladen werden muß, können wir genauso gut beim Kassettenpuffer bleiben und einen Bereich bei $0390 nutzen, der relativ verkehrsarm ist.
Zuerst muß der Maschinencode geladen werden. Wir wollen die Laderoutine oben bei 10000 unterbringen, wo sie aus dem Weg ist.

```
10 CH%=0:V%=0:GOTO 1000

10000 MC=3*16↑3+9*16:I=MC : REM $0390
10010 READ A$: IF LEN(A$)<> 2 THEN 100 :REM DONE IF XXXX
10020 GOSUB 10100
10030 POKE I,A: PRINT I,A$,A
10040 I=I+1:GOTO 10010

10100 A=ASC(A$)-48+7*(A$>":"): REM CONVERT FROM HEX
10110 B$=MID$(A$,2)
10120 A=16*A+ASC(B$)-48+7*(B$>":")
10130 RETURN
```

Nun können wir den Assembler in lesbarer, aus Datenanweisungen bestehender Form schreiben (natürlich brauchen Sie die REM-Anweisungen nicht einzutippen). Achten Sie darauf, die Nullen nicht als große O's zu schreiben!

```
10200 DATA A0, 03      :REM            LDY #3              PET
10210 DATA B1, 2A      :REM            LDA (VARS),Y        ****
10220 DATA A8          :REM            TAY
10230 DATA 4D, 43, E8  :REM            EOR DDR             *
10240 DATA 8D, 43, E8  :REM            STA DDR             *
10250 DATA 98          :REM            TYA
10260 DATA A2, 00      :REM            LDX #0
10270 DATA 78          :REM            SEI
10280 DATA 2C, 4F, E8  :REM LOOP       BIT PORT            *
10290 DATA D0, 05      :REM            BNE DONE
10300 DATA E8          :REM            INX
10310 DATA D0, F8      :REM            BNE LOOP
10320 DATA A2, FF      :REM            LDX #$FF            *
10330 DATA 0D, 43, E8  :REM DONE       ORA DDR             *
10340 DATA 8D, 43, E8  :REM            STA DDR
10350 DATA 58          :REM            CLI
10360 DATA 8A          :REM            TXA
10370 DATA A0, 0A      :REM            LDY #10             ****
10380 DATA 91, 2A      :REM            STA (VARS),Y
10390 DATA A9, 00      :REM            LDA #0              *
10400 DATA 8D, 4F, E8  :REM            STA PORT
10410 DATA 60          :REM            RTS
10420 DATA XXXXX       :REM            END
```

Diese Version läuft auf den PETs der 30xx-, 40xx- und 80xx-Serie. *Der alte PET* 2001 hat die Zeiger auf den Variablenanfang in den Adressen 124 und 125, so daß der in den Zeilen 10210 und 10380 enthaltene Code wie folgt geändert werden muß:

```
10210 DATA B1, FC
10380 DATA 91, FC
```

Für den C 64 müssen auch die Port- und Datenrichtungsadressen geändert werden. Die Änderungen sind so zahlreich, daß es sich lohnt, diesen Teil des Programms noch einmal aufzulisten:

```
10200 DATA A0, 03      :REM          LDY #3          CBM 64
10210 DATA B1, 2D      :REM          LDA (VARS),Y
10220 DATA A8          :REM          TAY
10230 DATA 4D, 03, DD:REM            EOR DDR
10240 DATA 8D, 03, DD:REM            STA DDR
10250 DATA 98          :REM          TYA
10260 DATA A2, 00      :REM          LDX #0
10270 DATA 78          :REM          SEI
10280 DATA 2C, 01, DD:REM LOOP       BIT PORT
10290 DATA D0, 05      :REM          BNE DONE
10300 DATA E8          :REM          INX
10310 DATA D0, F8      :REM          BNE LOOP
10320 DATA A2, FF      :REM          LDX #$FF
10330 DATA 0D, 03, DD:REM DONE       ORA DDR
10340 DATA 8D, 03, DD:REM            STA DDR
10350 DATA 58          :REM          CLI
10360 DATA 8A          :REM          TXA
10370 DATA A0, 0A      :REM          LDY #0
10380 DATA 91, 2D      :REM          STA (VARS),Y
10390 DATA A9, 00      :REM          LDA #0
10400 DATA 8D, 01, DD:REM            STA PORT
10410 DATA 60          :REM          RTS
10420 DATA XXXXX       :REM          END
```

Immer wieder wird der Prozessor unterbrochen, damit er Verwaltungsarbeiten wie das Lesen der Tastatur erledigt. In einem BASIC-Programm ist das nicht wahrnehmbar, aber wenn es mitten in der obigen Zeitgeberschleife passiert, wird das Ergebnis verpfuscht. Die Anweisung SEI blockiert den Interrupt – aber vergessen Sie nie, ihn danach wieder mit CLI zu aktivieren, sonst haben Sie die Konsequenzen zu tragen!

Nun können Sie eine Routine zum Austesten der Umwandlung hinzufügen. Erinnern Sie sich, daß die Kanalnummer in CH% gesetzt ist und ein Ergebnis in V% übergeben wird. Die tatsächliche Umwandlung geschieht durch einen Aufruf SYS MC, wobei MC die Adresse des Maschinencode enthält.

```
100 CH%=1:SYS MC:V1=V%: REM READ VALUE FROM P0
110 CH%=2:SYS MC:V2=V%: REM READ VALUE FROM P1
120 PRINT V1, V2
130 GOTO 100 : REM MEASURE THEM AGAIN
```

Geben Sie das Programm ein und lassen es laufen. Wenn nichts an den User-Port angeschlossen ist, sollten zwei Spalten mit Nullen auf dem Bildschirm erscheinen. Schalten Sie jetzt zwischen P0 und 0 V einen 2-Mikrofarad-Kondensator mit einem 2-kOhm-Potentiometer in Serie. Wenn Sie am Potentiometer drehen, müßten sich in Spalte 1 Zahlen zwischen 0 und 255 ergeben. Wenn die größte Zahl kleiner als 255 ist, nehmen Sie einen entsprechend größeren Kondensator. Wenn die kleinste Zahl nicht 0 ist, verwenden Sie ein Potentiometer mit größerem Widerstand.
Behandeln Sie jetzt P1 auf die gleiche Weise, und Sie haben das Rohmaterial zu einem Joystick. Natürlich läßt sich die Zahl der Eingabekanäle leicht bis auf acht erhöhen, einer für jedes Bit des User-Ports. Dazu braucht nicht einmal das Programm geändert zu werden – wie es ist, wird es mit 8 Kanälen fertig, wenn sie als 1, 2, 4, 8, 16 usw. aufgerufen werden.
Vorsicht müssen Sie in dem Moment walten lassen, wo Sie andere Ein- und Ausgaben haben, die die übrigen Bits des User-Ports benutzen. Wenn Sie beim Aufruf von SYS MC den Wert 240 für CH% zulassen, bleiben die oberen vier Bits des Ports als Ausgänge geschaltet. In der jetzigen Form legt auch das Programm Nullen auf alle Ausgangsbits; solange Ihr eigenes Programm sicherstellt, daß die als Analog-Eingänge zu lesenden Bits auf Null gesetzt werden, können Sie einfach Zeile 10400 weglassen.

EIN GENAUERER WANDLER

Die Analogeingabe-Einrichtung des C 64 und der bisher in diesem Kapitel beschriebene Wandler sind ganz gut für Joysticks zum Steuern von Spielen geeignet, aber sie sind nicht brauchbar für ernsthafte Anwendungen, bei denen Linearität erforderlich ist. Ihr Eingabesignal beruht auf der Änderung eines Widerstands, aber oft ist ein Signal in Form einer analogen Spannung verfügbar, sagen wir zwischen 0 und 5 Volt. Eine Ausgabe dieser Form hat ein Instrumentenverstärker, der etwa für ein Dehnungsmeßgerät oder Ähnliches gebraucht wird.

Um eine lineare Spannungseingabe zu erhalten, ist nur eine verhältnismäßig geringfügige Modifikation erforderlich; es ergibt sich aber auch nur ein Achtbit-Resultat. Das reicht für viele Anwendungen aus, aber mit einer weiteren Veränderung der Software ließe sich eine Genauigkeit von beliebig vielen Bits erreichen – wenn auch nach ungefähr zwölf Bits jede Verbesserung durch Rauschen verdeckt wird. Wachsende Auflösung wird mit wachsender Umwandlungszeit bestraft. Für acht Bits dauert die Umwandlung etwa vier Millisekunden, und im günstigsten Fall verdoppelt sich die Umwandlungszeit für jedes zusätzliche Bit. Bald ist dann der Punkt erreicht, an dem die Umwandlung besser in einem speziellen Hardware-Chip vor sich geht – obwohl man dabei für zusätzliche Bits mit zusätzlichen Kosten rechnen muß.
Um die Hardware so zu verfeinern, daß die Eingabe linearisiert wird, sind unter anderem ein Integrator und ein Komparator erforderlich – zwei Chips für zusammen weniger als vier Mark. Damit erhalten Sie vier analoge Eingabekanäle und belegen fünf Bits des User-Ports.
Ein weiteres Bit ist nötig, um den Integrator zu steuern – wir wollen Bit P4 verwenden. Wenn dieses auf High liegt, sinkt die Ausgangsspannung des Operationsverstärkers 747 (der den Integrator bildet) rasch ab, bis sie bei 0 V von der Diode aufgefangen wird. Wenn P4 auf Low gezogen wird, steigt die Ausgangsspannung an, bis sie +5 V in der Zeit 100k*.047 Mikrofarad = 10 ↑ 5*.04710 ↑ (−6) = 4.7 Millisekunden erreicht. Die Rampe des Integratorausgangs wird an den nichtinvertierenden Eingang jedes der vier Komparatoren gelegt, aus denen ein LM399 besteht. Sobald die Rampe die Eingabesignale überschreitet, die an die invertierenden Eingänge angeschlossen sind, wechselt das entsprechende Ausgangsbit von 0 auf 1. Das bedeutet, daß die oben angegebene Software nur geringfügig geändert werden muß.
Zu Beginn der Umwandlung muß der Ausgang von Bit P4 zunächst auf Low gezogen werden. Dazu ist die Software um folgende Zeile zu ergänzen:

```
10235 DATA 09, 10 :REM ORA #$10
```

Am Ende der Umwandlung muß das Bit P4 wieder auf High gehen können, um den Integrator auf 0 V rückzusetzen. Das erledigt die neue Zeile 10335:

```
10335 DATA 29, EF :REM AND #$EF
```

Jetzt ist die Software sowohl mit einfachen, wie bisher angeschlossenen Joysticks als auch mit linearer Spannungsumwandlung im Bereich 0 V bis 5 V kompatibel, wenn die Schnittstelle aus **Abbildung 5.2** verwendet wird – probieren Sie es aus.

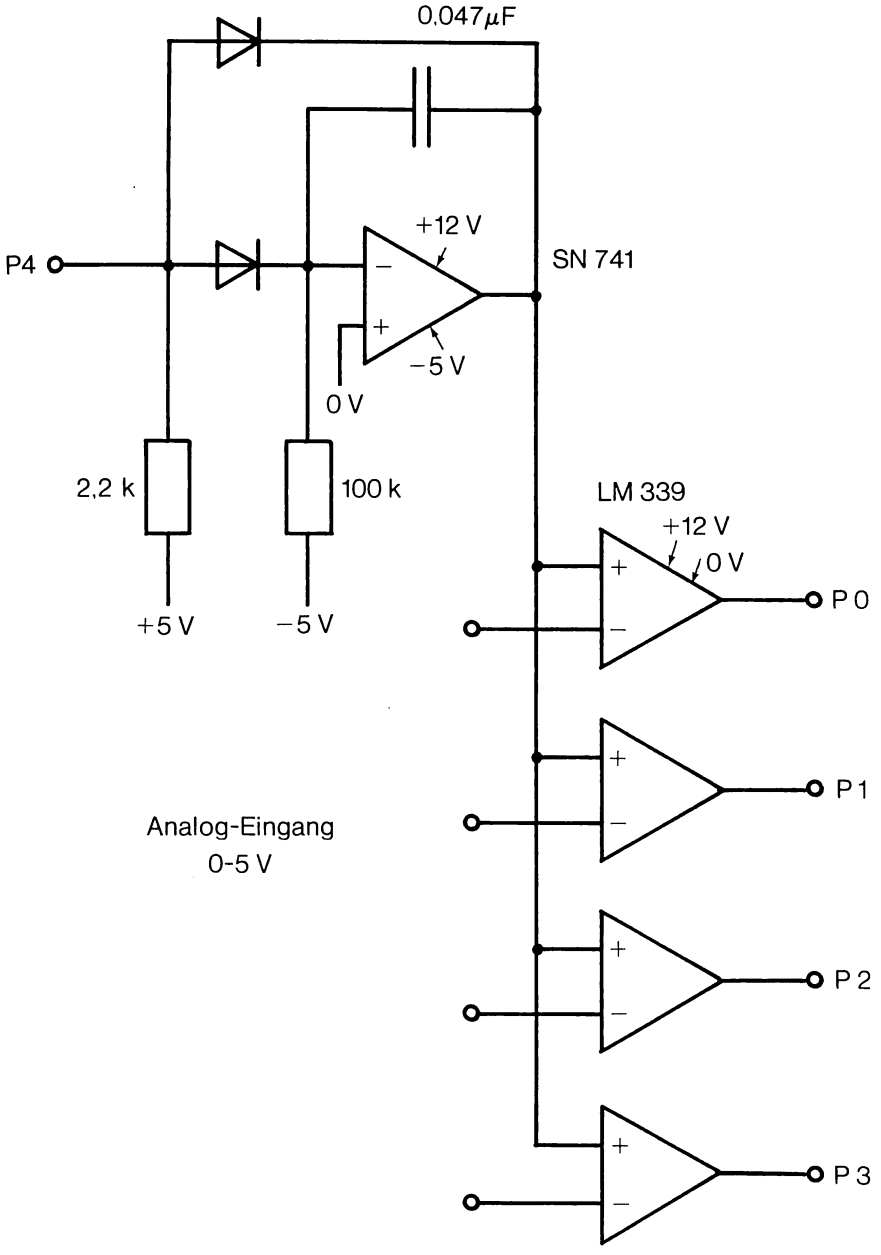


Abbildung 5.2 Linearer Analogeingang

LABORINSTRUMENTIERUNG

Nachdem jetzt die PET-Besitzer ein analoges Signal oder einen Joystick lesen können, werden sie sich davon Gebrauch machen wollen. Das Grafiksystem aus Kapitel 3 ist unereichbar für sie, wenn sie sich kein hochauflösendes Grafiksystem gekauft haben, doch es genügt schon ein viel einfacherer Einsatz der Grafik, um ein hochauflösendes Diagramm auf den Bildschirm zu zeichnen. Damit lassen sich physikalische Experimente beleben, etwa indem Übergangszustände mit bis zu fünfzig Ablesungen in der Sekunde erfaßt werden oder an den Computer die Aufgabe delegiert wird, geduldig jede halbe Stunde eine Ablesung vorzunehmen. Um die Daten zu speichern, muß durch Ersetzen von Zeilen ab 100 ein Feld eröffnet werden:

```
100 POKE 59468,12:PRINT CHR$(128+14):REM SET GRAPHICS
    MODE
110 PRINT CHR$(147) :REM CLEAR SCREEN
120 INPUT "HOW MANY DATA POINTS";NP
130 INPUT "HOW MANY CHANNELS (1 TO 4)";NC:NC=NC-1
140 DIM R(NP,NC),CC(NC)
150 FOR I=0 TO NC:CC(I)=2↑I:NEXT:REM SET UP CHANNEL CODES
160 INPUT "DELAY BETWEEN SAMPLES (WHOLE SECONDS)" D
170 D$=RIGHT$(STR$(1000000 + INT(D)),6):T0$="000000"
```

Jetzt können wir mit dem Protokollieren der Daten beginnen:

```
1000 FOR P=1 TO NP:PRINT
1010 FOR C=0 TO NC:CH%=CC(C):SYS MC
1020 R(R,C)=V%: GOSUB 2000 :REM STORE DATA, PLOT IT
1030 NEXT C:GOSUB 1500:NEXT P:REM DELAY BETWEEN POINTS
1040 END
```

Das Unterprogramm 1500 muß die Zeitgeberfunktion bereitstellen, die auf die richtige Sekunde wartet:

```
1500 IF TI$< D$ THEN 1500
1510 TI$=T0$:RETURN
```

Es bleibt eine Displayroutine bei 2000 zu schreiben, die die Resultate im besten Licht zeigt. Das allereinfachste Verfahren besteht darin, mit TAB(V%/7) über den Bildschirm zu 'tabulieren' und dann entsprechend der Kanalnummer ein Symbol anzuzeigen:

```
2000 PRINT CHR$(145) :REM CURSOR UP RETURN FOR LEFT
     MARGIN
2010 PRINT TAB(V%/7); CHR$(C+49);
2020 RETURN
```

Testen Sie das bisher entwickelte Programm.
Bei einer Auflösung von nur vierzig Punkten über den Bildschirm hinweg ist das Ergebnis nicht sonderlich beglückend. Eine höhere Auflösung bekommen wir mit den Grafikzeichen, die jeweils eine senkrechte Linie in einer von acht verschiedenen Positionen darstellen – das ergibt 320 Displaywerte, und auf diese Weise braucht von der analogen Auflösung nichts geopfert zu werden. Die Grafikzeichen werden durch Hinzufügen von Zeilen ab 180 definiert:

```
  180 DIM G$(7):FOR I=0 TO 7
  190 READ C:G$(I)=CHR$(C):NEXT
10300 DATA 165,212,199,194,221,200,217,167
```

Zeile 2010 wird jetzt ersetzt durch

```
2010 PRINT TAB(V%/8);G$(V% AND 7)
```

Leider lassen sich die Kanäle nicht unterscheiden; deshalb sollten die Abläufe alle 60 Zeilen protokolliert werden. Dazu verwenden wir die folgende Variante der alten Zeile 2010:

```
2005 IF (P AND 15)=0 THEN PRINT TAB(V%/8);CHR$(C+49)
:RETURN
```

Nachdem Sie Ihre Daten erhoben haben, möchten Sie sie vielleicht mit einer direkten Anweisung GOTO 3000 noch einmal auf den Bildschirm geben. Die Displayroutine bei 3000 ist einfach:

```
3000 FOR P=1 TO NP:PRINT:FOR C=0 TO NC
3010 V%=R(P,C):GOSUB 2000:NEXT:NEXT
3020 END
```

Probieren Sie jetzt diese endgültige Version aus. Falls Sie bei ihrer Entwicklung den Faden verloren haben, ist sie am Ende dieses Kapitels vollständig aufgelistet. Ihre Struktur ist so einfach, daß Sie sie sicher ohne weiteres so modifizieren können, daß sie die Daten automatisch analysiert, vielleicht indem sie den Logarithmus oder das Quadrat eines Kanals zeichnet, und eventuell das Verhältnis weiterer Kanäle. Sie ist der Ausgangspunkt für vollautomatische Experimente, bei denen der Com-

puter Signale aussendett, um den beobachteten Prozeß anzuregen, *und gleichzeitig* die Ergebnisse registriert und analysiert.

Analoge Eingabe für PET

```
10 CH%=0:V%=0:GOTO 10000: REM *** PET
100 CH%=1:SYSMC:V1=V%
110 CH%=2:SYSMC:V2=V%
120 PRINT V1,V2:GOTO 100
130 GOTO 120
10000 MC=3*16^2+9*16:I=MC
10010 READ A$:IF LEN(A$)<>2THEN100
10020 GOSUB 10100
10030 POKE I,A:PRINT I,A$,A
10040 I=I+1:GOTO 10010
10100 A=ASC(A$)-48+7*(A$>":")
10110 B$=MID$(A$,2)
10120 A=16*A+ASC(B$)-48+7*(B$>":")
10130 RETURN
10200 DATA A0,03, B1,2A, A8, 4D,43,E8
10210 DATA 8D,43,E8, 98, A2,00, 78
10220 DATA 2C,4F,E8, D0,05, E8, D0,F8
10230 DATA A2,FF, 0D,43,E8, 8D,43,E8
10240 DATA 58, 8A, A0,0A, 91,2A, A9,00
10250 DATA 8D,4F,E8, 60
10260 DATA XXXX
```

Analoge Eingabe für CBM 64

```
10 CH%=0:V%=0:GOTO 10000: REM *** 64
100 CH%=1:SYSMC:V1=V%
110 CH%=2:SYSMC:V2=V%
120 PRINT V1,V2:GOTO 100
130 GOTO 120
10000 MC=3*16^2+9*16:I=MC
10010 READ A$:IF LEN(A$)<>2THEN100
10020 GOSUB 10100
10030 POKE I,A:PRINT I,A$,A
10040 I=I+1:GOTO 10010
10100 A=ASC(A$)-48+7*(A$>":")
10110 B$=MID$(A$,2)
10120 A=16*A+ASC(B$)-48+7*(B$>":")
10130 RETURN
10200 DATA A0,03, B1,2D, A8, 4D,03,DD
10210 DATA 8D,03,DD, 98, A2,00, 78
10220 DATA 2C,01,DD, D0,05, E8, D0,F8
10230 DATA A2,FF, 0D,03,DD, 8D,03,DD
```

```
10240 DATA 58, 8A, A0,0A, 91,2D, A9,00
10250 DATA 8D,01,DD, 60
10260 DATA XXXX
```

Datenerhebung und Plotten – PET

```
10 CH%=0:V%=0:GOTO 10000: REM *** PET
100 POKE 59468,12:PRINT CHR$(128+14)
110 PRINT CHR$(147)
120 INPUT"HOW MANY DATA POINTS";NP
130 INPUT"HOW MANY CHANNELS(1 TO 4)";NC:NC=NC-1
140 DIM R(NP,NC),CC(NC)
150 FOR I=0 TO NC:CC(I)=2^I:NEXT
160 INPUT"SAMPLING INTERVAL (WHOLE SECONDS)";D
170 D$=RIGHT$(STR$(1000000+INT(D)),6)
175 T0$="000000"
180 DIM G$(7):FOR I=0 TO 7
190 READ C:G$(I)=CHR$(C):NEXT
1000 FOR P=1 TO NP:PRINT
1010 FOR C=0 TO NC:CH%=CC(C):SYS MC
1020 R(P,C)=V%:GOSUB 2000
1030 NEXT C:GOSUB 1500:NEXT P
1040 END
1500 IF TI$<D$ THEN 1500
1510 TI$=T0$:RETURN
2000 PRINT CHR$(145);:IF (P AND 15)>0 THEN 2010
2005 PRINT TAB(V%/8);CHR$(C+49):RETURN
2010 PRINT TAB(V%/8);G$(V% AND 7):RETURN
3000 FOR P=1 TO NP:PRINT:FOR C=0 TO NC
3010 V%=R(P,C):GOSUB 2000:NEXT:NEXT
3020 END
10000 MC=3*16^2+9*16:I=MC
10010 READ A$:IF LEN(A$)<>2THEN100
10020 GOSUB 10100
10030 POKE I,A:PRINT I,A$,A
10040 I=I+1:GOTO 10010
10100 A=ASC(A$)-48+7*(A$>":")
10110 B$=MID$(A$,2)
10120 A=16*A+ASC(B$)-48+7*(B$>":")
10130 RETURN
10200 DATA A0,03, B1,2A, A8, 4D,43,E8
10210 DATA 8D,43,E8, 98, A2,00, 78
10220 DATA 2C,4F,E8, D0,05, E8, D0,F8
10230 DATA A2,FF, 0D,43,E8, 8D,43,E8
```

```
10240 DATA 58, 8A, A0,0A, 91,2A, A9,00
10250 DATA 8D,4F,E8, 60
10260 DATA XXXX
10300 DATA 165,212,199,194,221,200,217,167
```

KAPITEL 6

SCHRITTMOTOREN UND IHRE VERWENDUNG

Schrittmotoren sind als Antrieb sehr beliebt. Sie benötigen nur Logiksignale zu ihrer Steuerung, also ist keine Digital-Analog-Umwandlung erforderlich. Bis vor kurzem waren nur Präzisionsmotoren der 'Oberklasse' zu exorbitanten Preisen erhältlich, aber mit dem Mikrocomputer und dem Bedarf an billigen Peripheriegeräten ist eine Nachfrage nach preiswerten Schrittmotoren einhergegangen, die eilig von der Industrie befriedigt worden ist. Ein geeigneter Motor für Turtles und Mikromäuse ist der Philips ID35, der zum Preis von etwa 50 Mark vertrieben wird.

PROBLEME UND GRUNDLAGEN

Trotz ihrer offensichtlichen Vorteile sind Schrittmotoren nicht problemlos. Ihre Höchstgeschwindigkeit ist strikt beschränkt, und wenn man sich ihr annähert, fällt das nutzbare Drehmoment dramatisch ab. Plötzliche Geschwindigkeitsänderungen, selbst bei ziemlich geringen Geschwindigkeiten, können den Motor zum Stillstand bringen. Leider erkennt der Computer nicht, daß der Motor 'aus dem Takt' geraten ist, wenn nicht besondere Sensoren vorhanden sind. Alle folgenden Bewegungen finden deshalb mit einem Positionsfehler statt, bis ein Rückstellmanöver vollzogen wird. Ein weiterer Nachteil bei einem batteriebetriebenen System ist der Energieverbrauch; selbst stationär entnimmt es soviel Enerige wie unter Vollast.
Wie arbeitet ein Schrittmotor nun? Der Rotor ist ein Dauermagnet, während der Stator (das feste Gehäuse) eine Anzahl elektrischer Windungen besitzt, die ein Magnetfeld erzeugen, wenn ihnen Energie zugeführt wird. Das Feld richtet den Rotor aus, und wenn die Auswahl der gespeisten Windungen in geeigneter Reihenfolge geändert wird, dreht sich der Rotor Schritt für Schritt. Wenn die Schrittfolge stoppt, wird der Rotor durch das Magnetfeld festgehalten.

EINE EINFACHE DEMONSTRATION

Die Bewegung des Dauermagneten läßt sich mit der Drehung eines magnetischen Kompasses vergleichen – und tatsächlich kann man einen Kompaß für ein Experiment verwenden, das die Arbeitsweise eines Schrittmotors verdeutlicht. Besorgen Sie sich einen billigen Kompaß – am besten die einfache Ausführung mit einem Zeiger statt einem verzierten Blatt. Wickeln Sie 50 Windungen feinen Kupferdrahtes

– 36 SWG oder feiner – über den Kompaß. Natürlich darf der Draht nicht die Sicht auf die Nadel verdecken.
Schalten Sie einen 47-Ohm-Widerstand in Reihe mit der Wicklung und legen Sie 5 V an die Enden. Sie können 100 Milliampere aus dem 5 V-Pin des User-Ports entnehmen – das ist allerdings die Obergrenze. Es experimentiert sich natürlich leichter, wenn Sie das in Kapitel 4 beschriebene Kabel mit einer Anschlußleiste verwenden.
Wenn Spannung angelegt wird, müßte sich die Nadel drehen und fast senkrecht zur Wicklung, d. h. entlang der Wicklungsachse, ausrichten. Kehren Sie die angelegte Spannung um, und die Nadel kehrt sich um. Könnte man die Wicklung und also die Nadel direkt aus zwei Bits des User-Ports betreiben? Leider ist der aus PB0-7 verfügbare Strom auf ungefähr 3 mA begrenzt: Falls Sie nicht bereit sind, Wicklungen mit mehreren Hundert Windungen herzustellen, überwiegt das nicht die Wirkung des irdischen Magnetfeldes auf die Nadel. Der PET benutzt PA0-7, die sogar noch weniger Drive haben. Wir brauchen deshalb einen Verstärker – keine schlechte Sache, um sich auf das Betreiben richtiger Schrittmotoren vorzubereiten. Der einfachste Verstärker besteht nur aus einem Widerstand und einem Transistor pro Ausgangsbit – jeweils vier pro Motor. (Später können wir uns überlegen, statt dessen einen Darlington-Verstärkerchip zu verwenden.) Ein guter Allzweck-pnp-Transistor ist der 2N 3703 (RS 294-334), der deutlich weniger als vier Mark pro Fünferpackung kostet. Verbinden Sie zunächst nur einen Transistor mit der Wicklung, und betreiben Sie ihn aus PB0 über einen 1-kOhm-Widerstand wie in **Abbildung 6.2.**
Schließen Sie die Schaltung an und schalten sie ein. Zunächst sollte nichts mit dem Kompaß passieren. Jetzt teilen sie dem Computer die für das Datenrichtungsregister zu verwendende Adresse als Variable DD und die für den Ausgabeport als Variable P0 mit, indem Sie tippen:

```
DD=56579 :    PO=56577 :    REM***** CBM 64
```

oder

```
DD=59459 :    PO=59471 :    REM*****PET
```

Nun können Sie alle Bits des Ausgabedatenregisters auf High setzen, indem Sie tippen:

```
POKE PO,255
```

Als nächstes schalten Sie Bits 0–3 als Ausgänge durch Tippen von:

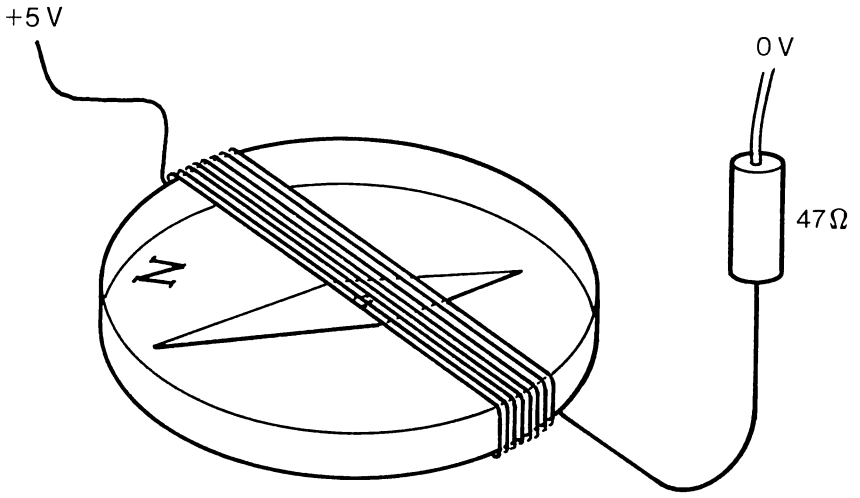


Abbildung 6.1 Kompaß und Wicklung

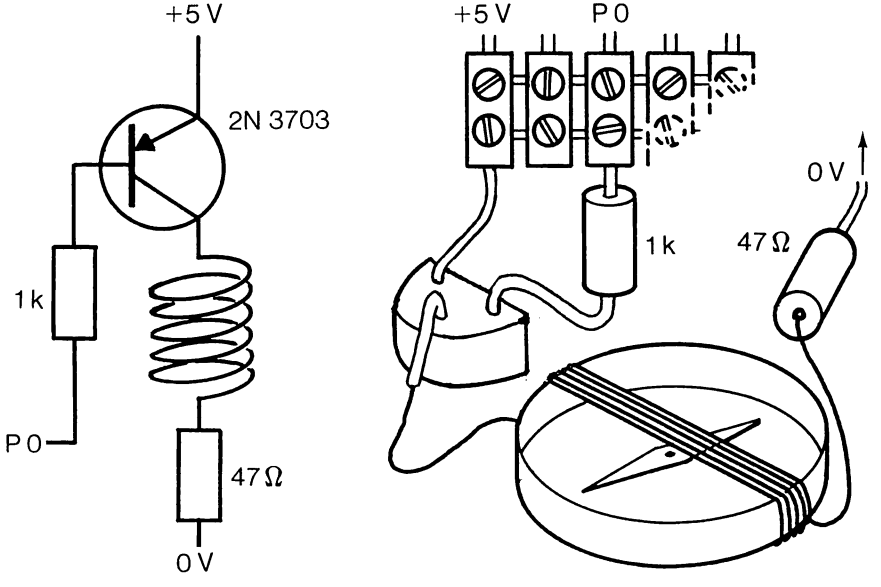


Abbildung 6.2 Transistorverstärker

```
POKE DD,15
```

Noch immer sollte nichts passieren, da der Ausgang von Bit 0 High ist und noch keinen Strom über die Transistorbasis aufnimmt. Jetzt tippen Sie:

```
POKE PO,255-1
```

Das zieht Bit 0 auf Null, und es fließt Strom von +5 V durch die Transistorbasis und R1 nach PB0 (oder PA0 beim PET). Der Transistor wird eingeschaltet und legt 5 V vom Kollektor des Transistors an die Wicklung und den Widerstand. Die Nadel müßte anspringen. Schalten Sie den Strom mit:

```
POKE PO,255
```

wieder ab, bevor der Widerstand R2 zu kochen beginnt.
Um die Nadel umzukehren, müssen wir den Strom in umgekehrter Richtung führen. Mit einer so einfachen Schaltung wie dieser können wir den Strom im Draht nicht umkehren, und deshalb brauchen wir eine zweite Wicklung direkt über der ersten. Wickeln Sie weitere fünfzig Drahtwindungen und verbinden ein Ende mit dem Widerstand, und zwar so, daß die Verbindungsstelle zum Mittelpunkt der jetzt aus hundert Windungen bestehenden Wicklung wird. Verdrahten Sie eine Zwillingsversion der Transistorschaltung und betreiben sie aus Bit 1 des User-Ports.
Die Anweisungen:

```
POKE PO,255-2
```

gefolgt von:

```
POKE PO,255-1
```

müßten die Kompaßnadel erst in eine Richtung (sagen wir Nord) und dann in die andere (Süd) drehen. Eine weitere Anweisung:

```
POKE PO,255
```

schaltet beide Zweige der Wicklung ab, und der Kompaß ist wieder dem irdischen Magnetfeld ausgeliefert.
Bis jetzt scheint die Spielerei mit dem Kompaß nicht viel mit Motoren zu tun zu haben. Aber die Sache wird gleich interessanter. Bringen Sie eine zweite Wicklung, ebenfalls mit 50+50 Windungen, quer über der ersten an und verbinden den Mittelpunkt (oder die 'Mittelabzapfung') der neuen Wicklung mit demselben

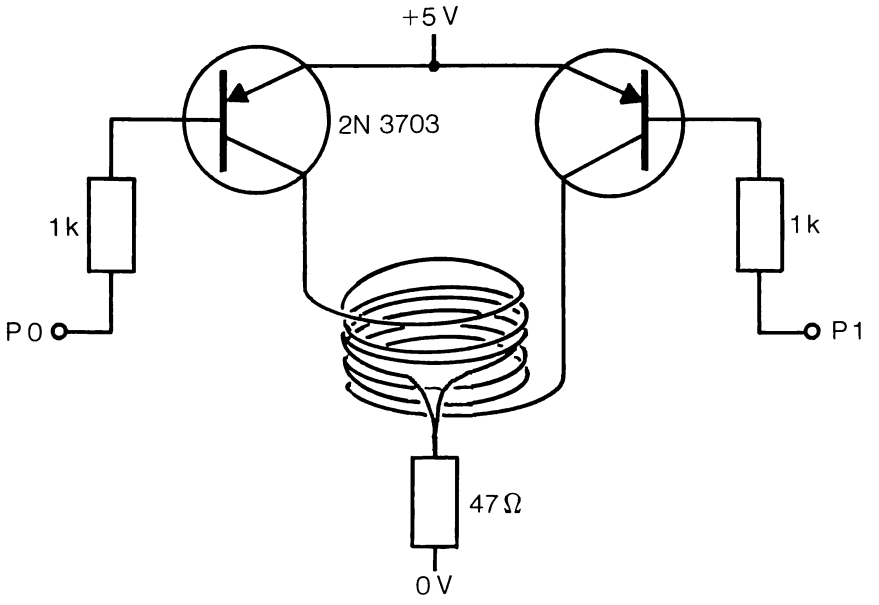


Abbildung 6.3 Zweirichtungsantrieb

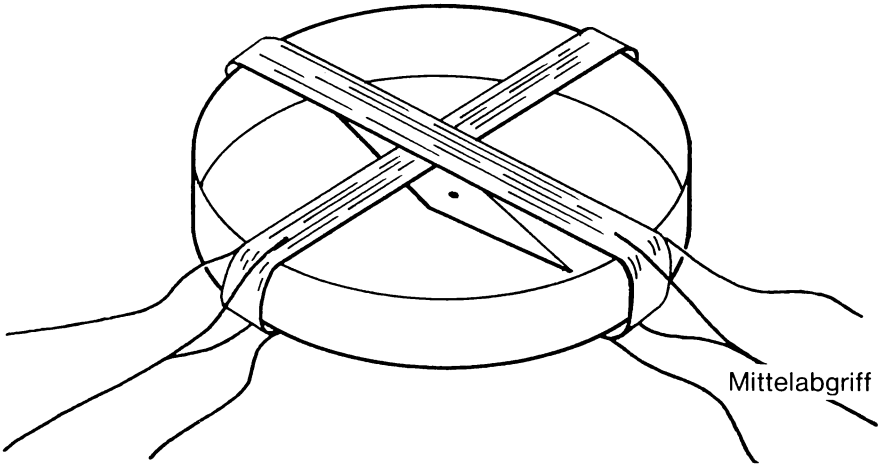


Abbildung 6.4 Zwei Wicklungen

47-Ohm-Widerstand und der Mittelabzapfung der ersten Wicklung. Wenn nun die Enden der neuen Wicklung an die Kollektoren von zwei weiteren, von Bit 2 und 3 betriebenen Transistoren angeschlossen werden, läßt die Anweisung:

```
POKE PO,255-4
```

die Nadel in die neue Richtung zeigen. Wenn die erste Wicklung die Nadel nach Nord oder Süd zeigen ließ, läßt die zweite Wicklung die Nadel nach Ost oder West zeigen. Durch gleichzeitiges Einschalten einer der N-S-Wicklungen und einer der O-W-Wicklungen erhalten wir zusätzlich NO, SO, SW und NW.

STEUERN DER GESCHWINDIGKEIT UND BESCHLEUNIGUNG

Geben Sie das folgende Programm ein und lassen es laufen:

```
10 DD=56579:PO=56577      :REM **** IF CBM 64
```

oder

```
10 DD=59459:PO=59471      :REM **** IF PET

20 POKE DD,15             :REM BITS 0-3 ARE OUTPUTS
30 POKE PO,255-5          :REM BOTH COILS ON TO GIVE NE
40 GOSUB 200              :REM 1 SECOND DELAY
50 POKE PO,255-9          :REM NOW COILS GIVE NW
60 GOSUB 200
70 POKE PO,255-10         :REM NOW COILS GIVE SW
80 GOSUB 200
90 POKE PO,255-6          :REM NW
100 GOSUB 200
110 GOTO 30               :ROUND AGAIN FOR ANOTHER
                           REVOLUTION
200 FOR I=1 TO 1000: NEXT:RETURN : REM DELAY 1 SECOND OR
    SO
```

Die Kompaßnadel müßte jetzt rotieren, wenn auch etwas ruckartig, sich also wie ein Schrittmotor verhalten.
Sie können nun in Zeile 200 unterschiedliche Zahlen ausprobieren, die die Motorgeschwindigkeit bestimmen. Sie werden bemerken, daß der Motor nicht einmal startet, wenn Sie zu hoch zielen. Versuchen Sie mit Hilfe der folgenden Änderungen stetig zu beschleunigen:

```
Einfügen     5          V=2000
Ändern       200        FOR I=1 TO V:NEXT
Einfügen     210        V=V-1
Einfügen     220        IF V<50 THEN V=50
Einfügen     230        RETURN
```

Jetzt nimmt die Verzögerung immer weiter ab, bis die Höchstgeschwindigkeit erreicht ist. Probieren Sie in Zeile 220 verschiedene Werte aus.
Die Geschwindigkeit wird zunächst nur langsam wachsen und erst gegen Ende schnell zunehmen. Eine gleichmäßigere Beschleunigung wird mit:

```
210 V=V*.995
```

erzielt. Sie experimentieren jetzt mit Techniken, die Sie brauchen werden, wenn Sie zu echten Schrittmotoren kommen. Natürlich ist das Programm immer noch höchst unelegant und nicht gerade vielseitig. Dennoch haben Sie sicher schon am Kompaßmotor einige der Fallen bemerkt, auf die Sie achten müssen:

1. Ohne Antrieb hält der Motor seine Position nicht.

2. Eine neue Position wird in Form einer schwach gedämpften Schwingung eingenommen. Bei bestimmten Geschwindigkeiten entsteht eine Resonanz, bei der die Schwingungen sich aufschaukeln; der Motor setzt dann aus.

3. Bei niedrigen Geschwindigkeiten ist die Bewegung unruhig. Das läßt sich durch Verwendung einer doppelten Schrittzahl etwas verbessern; die Schrittfolge ist dann N, NO, O, SO, S, SW, W, NW und wieder N.

4. Plötzliche Geschwindigkeitsänderungen würgen den Motor ab.

5. Es fehlt ein absoluter Positionsbezug. Alles hängt davon ab, daß der Motor im Schritt bleibt.

STRUKTURIEREN DER SCHRITTMOTOR-SOFTWARE

Nun wollen wir dem neuen Programm etwas 'Stil' geben, damit es allgemeiner verwendbar ist. Die Codes zur Bestimmung der Wicklungspolaritäten werden am besten in einem Feld abgelegt. Ich habe eine persönliche Vorliebe dafür, alle Initialisierungsdaten ans Ende des Programms zu verlegen, damit der funktionelle Teil nicht unübersichtlich wird. Deshalb beginnt das Programm mit GOTO 10000, und alle Definitionen fangen bei Zeile 10000 an.

```
10000 DD=56579:PO=56577: REM **** CBM 64
```

oder

```
10000 DD=59459:PO=59471: REM **** PET

10010 DIM DR(7):FOR I=0 TO 7: REM DRIVE CODES
10020 READ J:DR(I)=255-J: NEXT I
10030 DATA 1,5,4,6,2,10,8,9:REM COILS IN ORDER N-S-E-W
10040 POKE DD,15: REM MAKE BITS 0-3 OUTPUTS
10050 GOTO 100 : REM HOUSEKEEPING DONE
```

Die auszuführende Schrittzahl wird in der Variablen DI (DIstance) gehalten, und die Richtung in RO (ROtation) als Wert +1 oder −1. Die jeweilige Position liegt in HE (HEre), und die Geschwindigkeit (SPeed) wird durch die Variable SP bestimmt. Ein geeigneter Programmabschnitt zur Steuerung der Bewegung wäre etwa:

```
200 GOSUB 5000 : REM MOVE DISTANCE, ROTATION, SPEED
```

wobei das Unterprogramm wie folgt definiert ist:

```
5000 IF (DI< 1) OR (SP< 1) THEN RETURN
5010 FOR I=1 TO DI
5020 HE=HE + RO :REM ADD ROTATION TO HERE
5030 POKE PO,DR(HE AND 7) :REM OUTPUT CODE TO PORT
5040 FOR J= 1 TO 1000 STEP SP:NEXT J
5050 NEXT I
5060 RETURN
```

Es sieht vielleicht unbeholfen aus, daß die Geschwindigkeit durch eine variable Verzögerung wie in Zeile 5040 gewählt wird, aber es ist wirkungsvoll, solange der Wert von SP nicht übertrieben groß ist. Ein eleganteres Verfahren, der 'Bitratenverstärker', wird im nächsten Kapitel beschrieben. Es ist zu Koordinierung der Bewegungen mehrerer Schrittmotoren nützlich, aber aufgrund einer ungleichmäßigen Schrittrate ist die Höchstgeschwindigkeit herabgesetzt.
Zur Vervollständigung des Programms fügen Sie hinzu:

```
 10 GOTO 10000
100 PRINT "DISTANCE,ROTATION DIRECTION,SPEED"
110 INPUT DI,RO,SP
200 GOSUB 5000
210 GOTO 100
```

und haben damit ein Demonstrationsprogramm, mit dem Sie Bewegungen von der Tastatur aus steuern können. Sie sollten jetzt imstande sein, ein ausführlicheres Programm zu schreiben, das ein Feld programmierter Bewegungen aufbaut und dann ausführt.
Mit den Bits 4 bis 7 können Sie einen zweiten Schrittmotor betreiben. Das reicht für einen Plotter oder eine Turtle, ist aber etwas einschränkend für einen Roboter. Um bis zu acht Motoren aus einem einzigen User-Port zu steuern, brauchen Sie raffinierte Adressierverfahren.

STROMVERSORGUNG

Bevor wir uns weiter in die Einzelheiten der Software vertiefen, wollen wir uns den elektronischen Problemen zuwenden, die der Anschluß eines oder mehrerer echter Schrittmotoren an den Computer mit sich bringt. Die Grundzüge bleiben dieselben, aber wir müssen jetzt viel größere Ströme bereitstellen, die jenseits des erlaubten Abflusses aus dem Computer liegen. Für unter 120 Mark müßten Sie sich ein 1-A-Netzgerät, einstellbar zwischen 4 V und 10 V, kaufen können. Selbst das ist kaum genug Strom – obwohl eine Überbelastung lediglich die Ausgangsleistung 'umstülpt'. Vielleicht ist es am besten, das in Kapitel 1 beschriebene unstabilisierte Netzgerät zu bauen, das ungefähr drei Ampere bei +7 V und −7 V abgibt. Viele Schrittmotoren brauchen 12 V, um ihr Bestes zu geben, und das Netzgerät kann so geschaltet werden, daß sich 14 V entnehmen lassen – einfach indem die −7-V-Klemme als negativer Anschluß genommen und die mittlere Klemme ignoriert wird. Eine bequeme, aber riskante Alternative besteht darin, das Leben aufs Spiel zu setzen und ein Autobatterie-Ladegerät zu benutzen. Das gibt Ihnen vielleicht bis zu vier Ampere, aber Sie brauchen einen großen externen Kondensator – ungefähr 10000 Mikrofarad. Die Einstellung ist ebenfalls schlecht, und Wohlergehen oder Vernichtung Ihres Schaltersystems hängen von nichts Besserem als einer 4-A-Sicherung ab. Das ist trotzdem noch besser, als einen unerschöpflichen Vorrat an Batterien anzuschaffen, es sei denn, Sie können sich wiederaufladbare leisten.

SCHNITTSTELLE FÜR DIE HARDWARE

Ein gewöhnlicher Transistor wird kaum genug 'Beta' haben, um einen Schrittmotor mit der obigen Schaltung zu betreiben. Man kann jedoch Darlington-Transistoren mit viel höherer Leistungsverstärkung kaufen. Sie sind nichts weiter als ein Paar in Kaskade geschalteter Transistoren, haben aber den Nachteil einer viel höheren 'unteren' Spannung – sind also in Niederspannungsschaltungen weniger leistungsfähig. Heutzutage ist der Kauf von Mehrfunktionschips ökonomischer als der

von einzelnen Transistoren, und der Chip RS 307-109 enthält sieben *Darlingtons*, komplett mit Eingangswiderständen und Schutzdioden, für deutlich weniger als acht Mark. Dabei ereilt Sie natürlich Murphys Gesetz, denn zum Betreiben von zwei Schrittmotoren brauchen Sie acht Ausgänge, nicht sieben.
Eine weitere Komplikation besteht darin, daß diese Schaltkreise Senken und nicht Quellen sind. Die Mittelanzapfungen der Motorwicklungen müssen deshalb an die positive Zuleitung angeschlossen werden, und die Wicklung wird mit Spannung versorgt, wenn das Ausgangsbit des User-Ports auf High liegt, nicht auf Low. Die Zeile des Programms, die die Ausgänge schaltet, muß so geändert werden, daß die Umkehrung '255-' wegfällt, und wird zu

```
10010 READ J:DRIVE(I)=J:NEXT I
```

Überdies muß das Programm PO und DD so früh wie möglich mit 255 poken, damit Nullen ausgegeben werden und der Motor nicht unter dem Doppelten seines gerechten Anteils an Wirkströmen in Flammen aufgeht.
Mit Hilfe der oben beschriebenen Programmänderung und dem unten abgebildeten Schaltkreis sollten Sie sich nun zutrauen, Schrittmotoren zu steuern und eine einfache Turtle zu bauen.

MEHRPOLIGE SCHRITTMOTOREN

Schrittmotoren können bis zu 200 Schritte pro Umdrehung haben. Wenn die Eingänge durch eine 'elektrische Umdrehung' geschaltet werden, dreht sich der Motor nur um einige Grad. Wie kommt das zustande? **Abbildung 6.6** zeigt einen Rotor, der anders als eine einfache Kompaßnadel zwei Nord- und zwei Südpole hat. Die Motorwicklungen verlaufen nicht mehr direkt über dem Rotor, sondern sind auf Paare von Einzelpolen gewickelt. Wenn Wicklung 1 in positiver Richtung erregt wird, werden z. B. die Pole 'A' Süd und die Pole 'a' Nord. Wenn statt dessen Wicklung 2 in positiver Richtung erregt wird, werden die Pole 'B' Süd, und der Rotor wird um 45 Grad gedreht. Nachdem die Wicklungen die Schrittfolge einer elektrischen Umdrehung durchlaufen haben, wird Wicklung 1 wieder positiv erregt, und der Rotor hat genau eine halbe Drehung vollzogen. Wird der Rotor mit mehr Polen versehen, wächst das Verhältnis zwischen elektrischen Schritten und dem Rotationswinkel an.

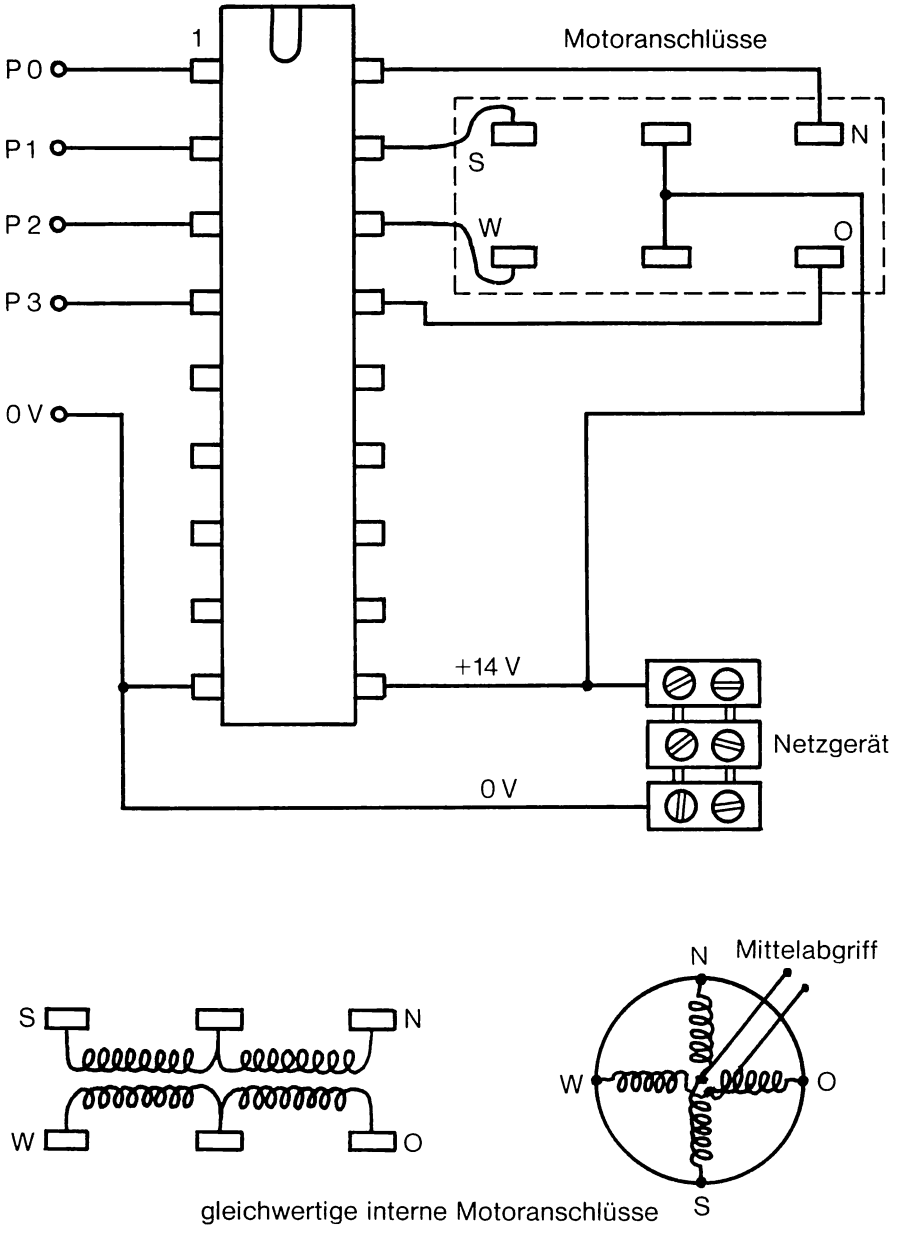


Abbildung 6.5 Darlington-Verstärkerchip

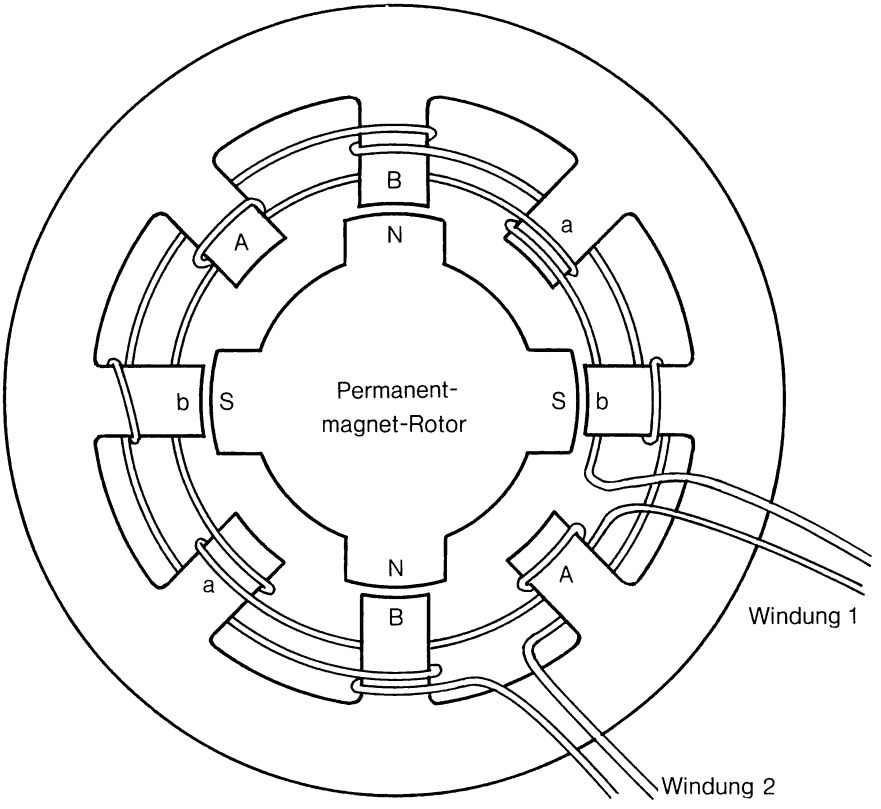


Abbildung 6.6 Ein vierpoliger Schrittmotor

KAPITEL 7

EINE EINFACHE TURTLE

Wenn Ihnen einmal eine umgedrehte Suppenschüssel begegnet, die umherwandert und vielleicht noch Kurven auf ein großes Blatt Papier zeichnet, so war das eine Turtle (Schildkröte). Wir machen hier keinen Versuch, auf die Probleme der Turtle-Grafik einzugehen; vielmehr dient das Prinzip der Turtle als guter Vorwand, ein Paar Schrittmotoren einzusetzen.

GRUNDSÄTZLICHES ZUR TURTLE

Die Turtle ist ein einfaches 'Rollstuhl'-System, das von zwei unabhängigen Rädern auf einer Achse angetrieben wird. Kugellager oder Kufen schränken das dabei mögliche Wackeln längsschiffs ein. Zur Geradeausbewegung drehen sich beide Räder synchron. Um auf der Stelle zu drehen, rotiert ein Rad vorwärts und das andere mit genau der gleichen Geschwindigkeit rückwärts. Dreht sich ein Rad genau doppelt so schnell wie das andere, so beschreibt die Turtle einen Kreis, dessen Mittelpunkt einen Radabstand vom langsameren Rad entfernt liegt. Präzise Bewegungen erfordern Motoren, die exakt im Gleichschritt arbeiten – genau das Richtige für Schrittmotoren!

Im Mittelpunkt einer 'echten' Turtle befindet sich ein zurückziehbarer Schreibstift, so daß sie beim Umherwandern Figuren zeichnen kann, oder sogar Diagramme und Illustrationen. Diesem Problem wollen wir uns später zuwenden.

Zwei Schrittmotoren können problemlos durch acht Bits des User-Ports gesteuert werden. Mit Hilfe zweier Darlington-Chips und den Erfahrungen aus dem letzten Kapitel sollte es wenig Mühe machen, die Motoren zum Laufen zu bringen. Schwieriger ist es, die Software 'sinnvoll' zu machen, damit die Befehlsstruktur sich auf die gewünschten Bewegungen der Turtle stützen kann, ohne sich in den grauenvollen Einzelheiten der für jede Kreisbewegung erforderlichen Motorschritte zu verlieren. Um das Problem von hinten aufzuzäumen: Wir möchten 'vorwärts 100' eintippen können, um uns 100 mm vorwärts zu bewegen, oder vielleicht 'drehen, Uhrzeigersinn, 90'. Es wäre schön, wenn auch Kreise möglich wären, zum Beispiel 90 Grad eines Kreises mit dem Radius 200 mm durch 'Kreis, Uhrzeigersinn, 200, 90'. Vielleicht sind sogar Cornusche Spiralen realisierbar, um einen Radius mit einem anderen zu verschmelzen – aber nicht jetzt gleich. Mit den weiteren Befehlen 'Stift auf' und 'Stift ab' für grafische Anwendungen ist das Sortiment vollständig. Wenn man will, kann man es einem zusätzlichen Unterprogramm überlassen, aufgrund der Befehlsfolge auszurechnen, wo die Turtle nach einem bestimmten Manöver landen müßte, obwohl wegen der mechanischen Toleranzen das Ergebnis nach einem längeren Spaziergang nicht sonderlich genau sein wird.

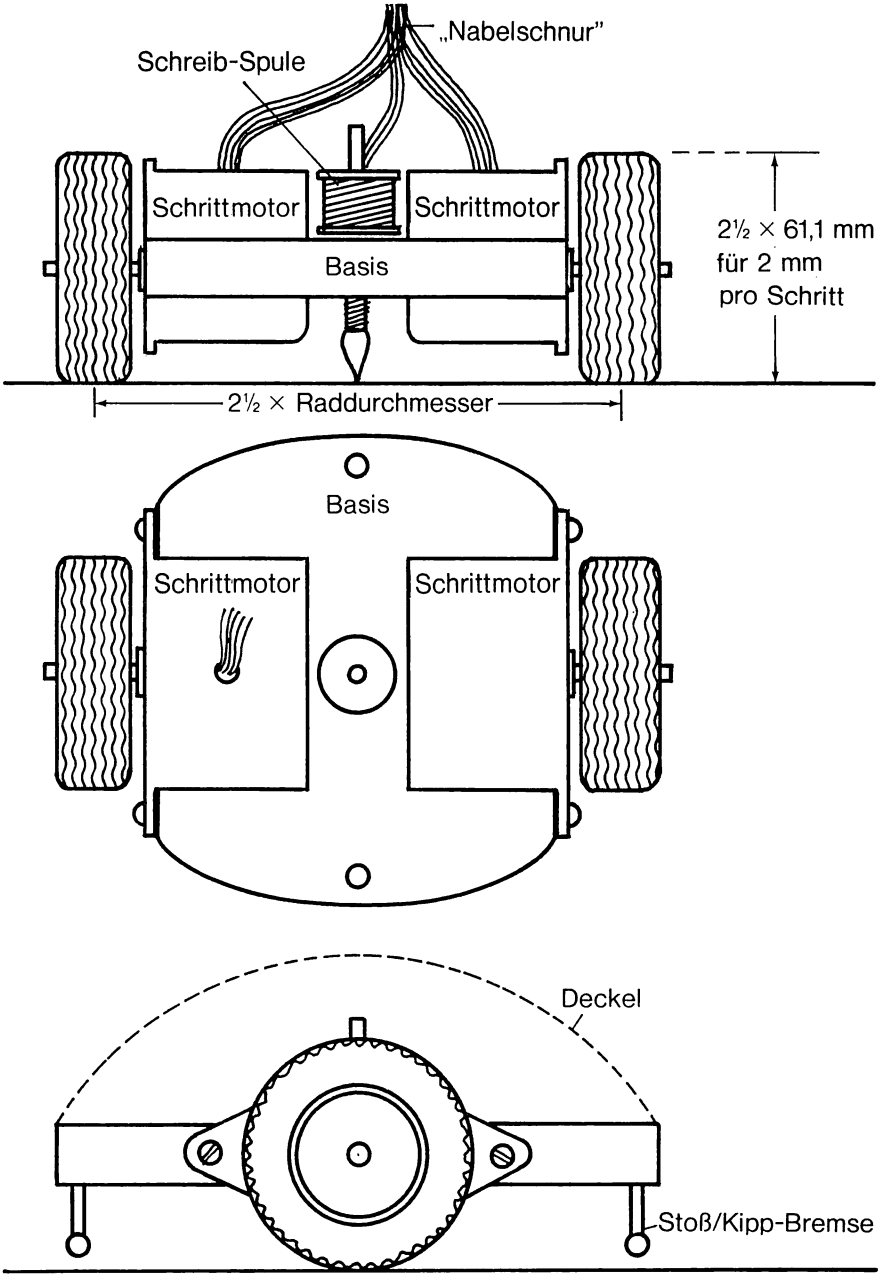


Abbildung 7.1 Ansichten einer Turtle

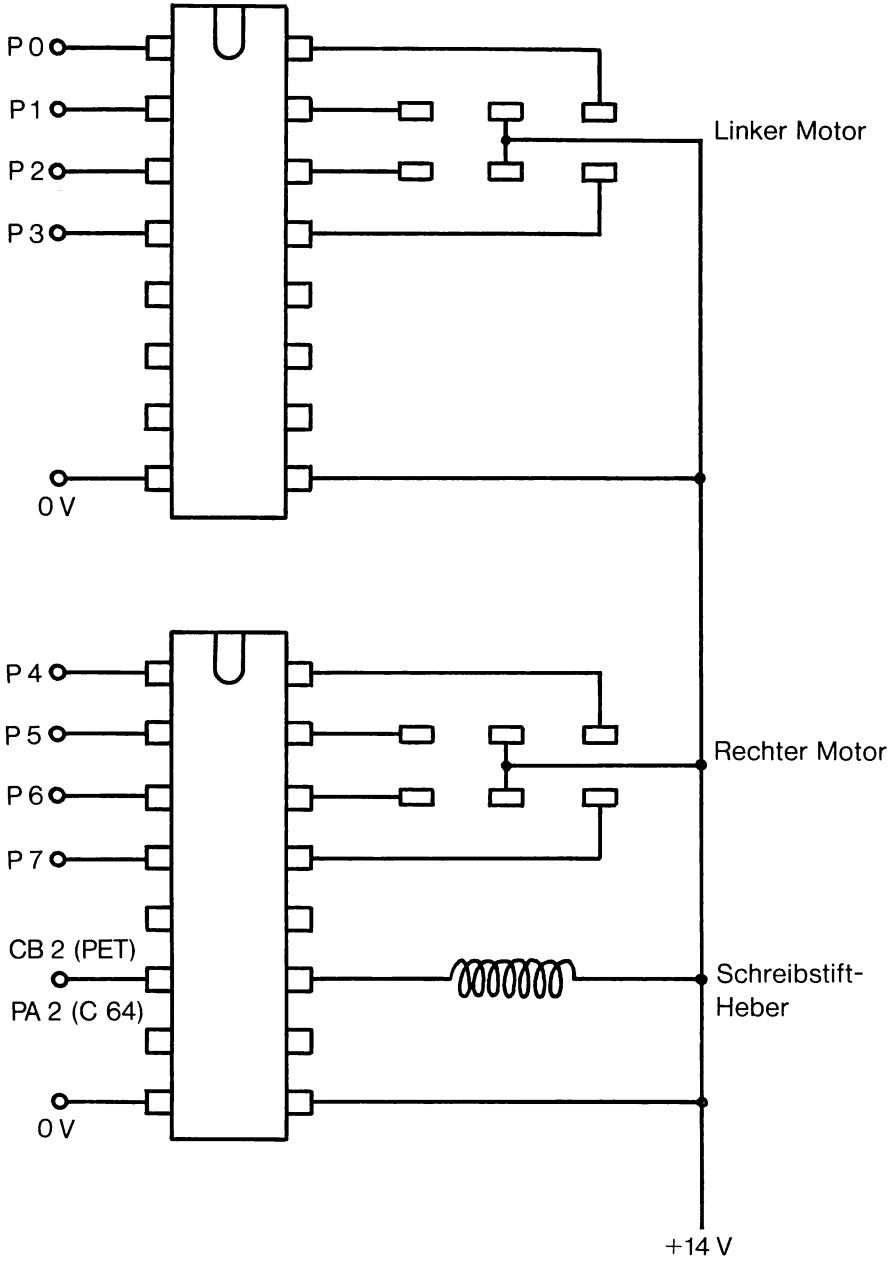


Abbildung 7.2 Schaltung von Turtle-Darlingtons

MECHANISCHE KONSTRUKTION

Die empfohlenen Schrittmotoren vom Typ ID35 werden von Philips hergestellt und zu einem Preis von ungefähr 50 Mark vertrieben. Sie haben 48 Schritte pro Umdrehung, das heißt 12 elektrische Umdrehungen pro mechanische Umdrehung. Wenn Sie die Motoren in Halbschritten betreiben, also N, NO, O, SO, S, SW, W, NW, haben Sie 96 Halbschritte pro Radumdrehung. Nehmen wir an, die Räder haben einen Durchmesser von 80 mm. Dann haben sie einen Umfang von etwa 250 mm, und jeder Halbschritt ergibt eine Bewegung von ungefähr 2,6 mm. Wenn es Ihnen nichts ausmacht, die Räder selbst herzustellen oder zu trimmen, ergibt ein Durchmesser von $2*96/\pi = 61{,}1$ mm genau 2 mm pro Halbschritt – aber am besten wählen Sie den nächstgrößeren Umfang aus dem Modellbaugeschäft und nehmen einen etwas seltsamen Skalenfaktor in Kauf.

Wenn die Motoren gleich schnell in entgegengesetzter Richtung betrieben werden, dreht sich die Turtle um ihren Mittelpunkt. Macht jedes Rad eine Umdrehung, so dreht sich die Turtle um genau ein Grad. Werden die Motoren mit unterschiedlichen Geschwindigkeiten betrieben, so ist die zurückgelegte Strecke gleich dem Durchschnitt der (mit Vorzeichen versehenen) Schrittzahl, wobei sich die Turtle durch einen Winkel dreht, der gleich der halben Differenz ist.

Das Fahrgestell kann aus Sperrholz oder sogar Balsaholz bestehen, da es wenig Arbeit leisten muß. Die Kufen kann man aus leichten Kugelrastungen für Schränke machen, obwohl ein paar zurechtgebogene Büroklammern es auch tun. Sie sollten nicht ganz bis zum Boden reichen, damit jeweils nur eine den Boden berührt. Die größte mechanische Beanspruchung wird durch das 'Nabelschnur'-Kabel verursacht, das hoch und zentral an der Turtle angebracht werden muß. Wenn Sie irgendeine Plastikschüssel als Schutzhaube opfern, darf das Kabel gefahrlos aus einem Loch in der Mitte herauskommen. Wenn aber Ihre Turtle nackt ist, sollten Sie einen Mast in der Mitte montieren – nicht zu hoch, damit die Turtle nicht umkippt. Damit sich das Kabel der Turtle von oben nähert, sollte es von einer an der Decke befestigten Tragschnur baumeln.

Auf den ersten Blick brauchen Sie ein Kabel mit mindestens einem Dutzend Leitungen, fünf für jeden Motor, zwei für den Stiftheber und weitere für beliebige Sensoren, die Sie vielleicht später noch anbringen möchten. Im Notfall kommen Sie mit zwei weniger davon, wenn Sie die positiven Stromleitungen zusammenfassen. Das ist aber möglicherweise Sparsamkeit am falschen Platz, denn der Widerstand des Kabels kann Kopplung zwischen den Motorantrieben verursachen. Flachkabel ist die sauberste, aber längst nicht die billigste Lösung. Vielleicht sollte ich an dieser Stelle ein gutes Buch über Flechten empfehlen.

STEUERSTRATEGIEN

Ein Algorithmus zur Umsetzung von Befehlen in erforderliche Motorhalbschritte (wir wollen sie von nun an einfach 'Schritte' nennen) läßt sich wie folgt bilden: Nehmen wir einen Raddurchmesser von 61 mm und einen Radabstand von 2,5*61 = 152,5 mm an. Siehe **Tabelle 7.1.**

Tabelle 7.1

BEFEHL	LINKE MOTORSCHRITTE	RECHTE MOTORSCHRITTE
Advance	Abstand/2	Abstand/2
Turn, cw	+ Winkel	− Winkel
Turn, acw	− Winkel	+ Winkel
Circle, cw	Winkel*(Radius/115+1)	Winkel*(Radius/115−1)
Circle, acw	Winkel*(Radius/115−1)	Winkel*(Radius/115+1)

Der Befehlsinterpret muß jetzt einen Motorsteuermodul 'ansprechen' können, der Befehle in Gestalt der von jedem Motor auszuführenden Schrittzahl annimmt. Ein weiterer Befehl, 'Geschwindigkeit, 20' kann eine allgemeine Variable bestimmen, die in der Syntax nicht aufzutauchen braucht. Wir wollen ein Unterprogramm zur Motorsteuerung verwenden und ab Zeile 8000 unterbringen.
Eine Geschwindigkeitsänderung eines einzelnen Motors läßt sich mit einer einfachen variablen Verzögerung realisieren, aber um zwei Motoren mit verschiedenen Geschwindigkeiten zu betreiben, ist ein anderes Konzept nötig, der Bitratenverstärker. Angenommen, der linke Motor soll 100 Schritte durchlaufen, während der rechte nur 67 durchläuft. Dann bilden wir zunächst das Verhältnis der beiden, in diesem Fall 0,67. Jedesmal, wenn die Schleife durchlaufen wird, macht der linke Motor einen Schritt, aber der rechte macht dabei nicht immer einen. Um das zu entscheiden, addieren wir das Verhältnis zu einer anderen Variablen, sagen wir T. Sobald T größer als 1 ist, macht der Motor einen Schritt, und T wird um 1 vermindert. Hört sich verwirrend an? Dann wollen wir ein Beispiel versuchen; siehe **Tabelle 7.2.**

Tabelle 7.2

LINKE MOTOR-POSITION		T	RECHTE MOTOR-POSITION		
	0	0		0	
Schritt	1	.67		0	
Schritt	2	1.34	Schritt	1	T wird 0.34
Schritt	3	1.01	Schritt	2	T wird 0.01

Schritt	4	.68	Schritt	2	
Schritt	5	1.35	Schritt	3	T wird 0.35
.....					
.....					
Schritt	97	.99	Schritt	64	
Schritt	98	1.66	Schritt	65	T wird 0.66
Schritt	99	1.33	Schritt	66	T wird 0.33
Schritt	100	1.00	Schritt	67	T wird 0.00

Damit ist die Bewegung vollständig ausgeführt, und jeder Motor hat die richtige Schrittzahl gemacht. Dieses Prinzip liegt den meisten Routinen zum Zeichnen geneigter Geraden zugrunde. Das Ergebnis wird leicht verbessert, wenn T mit dem Wert 0.5 beginnt, denn dadurch werden Unebenheiten symmetrisch über die ganze Linie verteilt. Im obigen Beispiel macht der rechte Motor nur ganz am Ende der Bewegung drei Schritte hintereinander; hätte T mit dem Wert 0.5 begonnen, wären nie in der Mitte aufgetreten.

SOFTWARE FÜR DIE MOTORSTEUERUNG

Wir können jetzt das Unterprogramm zur Motorsteuerung definieren. Dabei ist die Schrittzahl für den linken Motor in LM und für den rechten in RM abgelegt.

```
8000 AL=ABS(LM):AR=ABS(RM):REM ABSOLUTE VALUES OF
     STEPS
8010 SL=SGN(LM):SR=SGN(RM):REM AND SIGNS OF DIRECTIONS
8030 IF AR + AL=0 THEN RETURN:REM NO MOVE, GO HOME
8040 IF AR>AL THEN 8200:REM DEAL WITH THIS SEPARATELY

8100 RA=AR/AL:T=0.5:REM RATIO OF MOVES
8110 FOR M=1 TO AL:REM HERE WE GO
8120 GOSUB 9000:REM STEP LEFT MOTOR DIRECTION SL
8130 T=T+RA
8140 IF T>1 THEN GOSUB 9100:T=T-1: REM RIGHT MOTOR
8150 GOSUB 9500:NEXT M: RETURN:REM THAT'S ALL, GO HOME

8200 RA=AL/AR:T=0.5:REM RIGHT MOVE > LEFT MOVE
8210 FOR M=1 TO AR
8220 GOSUB 9100:REM RIGHT MOTOR EVERY TIME
8230 T=T+RA
8240 IF T>1 THEN GOSUB 9000:T=T-1:REM LEFT MOTOR
8250 GOSUB 9500:NEXT M:RETURN
```

Es bleibt das Schreiben des Motorantriebs nachzuholen. Wir fangen in Zeile 10000 mit Verwaltung an:

```
10000 LP=0:RP=0:SP=100        :REM MOTOR POSITIONS,SPEED

10010 PO=56577:DD=56579       :REM PORT,DATA DIR *** CBM 64
```

oder

```
10010 PO=59471:DD=59459       :REM                *** PET

10020 DIM LD(7),RD(7):        REM TWO ARRAYS FOR MOTOR
                              DRIVES

10030 FOR M=0 TO 7
10040 READ J:LD(M)=J:RD(M)=16*J:NEXT
10050 DATA 1,5,4,6,2,10,8,9:REM COILS IN ORDER N-S-E-W
10060 POKE DD,255:POKE PO,0:REM MAKE OUTPUTS, SET TO ZERO
10070 GOTO 100
```

Dann kommen wir zum Motorantrieb und einer von der Geschwindigkeit SP abhängigen Verzögerung:

```
9000 LP=(LP+SL) AND 7: REM LEFT MOTOR NEW POSITION
9010 POKE PO, (PEEK(PO)AND 240)+LD(LP)
9020 REM MIX NEW LEFT MOTOR DRIVE WITH OLD RIGHT, OUTPUT
9030 RETURN
9100 RP=(RP+SR) AND 7: REM RIGHT MOTOR NEW POSITION
9110 POKE PO, (PEEK(PO)AND 15) +RD(RP)
9120 RETURN
9500 FOR D=1 TO 1000 STEP SP:NEXT
9510 RETURN
```

(Mit etwas Geschick können Sie die Motorprozeduren in eine einzige umschreiben, in der SL, SR oder Null von zwei Argumenten QL und QR begleitet sind. Außerdem läßt sich die Steuerprozedur so anordnen, daß sie weniger unruhig ist. Die hier angegebenen, uneleganten Verfahren sind auf leichtere Verständlichkeit zugeschnitten.)
Bevor Sie die Feinheiten einarbeiten, suchen Sie mit einem 'Augenblicksprogramm' nach Fehlern in diesen Moduln:

```
 10 GOTO 10000
100 PRINT "LEFT MOTOR, RIGHT MOTOR"
110 INPUT LM,RM
120 GOSUB 8000
130 GOTO 100
```

und wenn das Resultat zu wünschen übrig läßt, versuchen Sie es mit etwas noch Einfacherem:

```
100 SPEED= 10:SL=1
110 GOSUB 9000:GOSUB 9500: GOTO 110
```

um der Sache auf den Grund zu gehen. Wenn alles andere versagt, greifen Sie zur Handsteuerung und poken PO mit verschiedenen Werten; holen Sie dann Ihr treues Prüfgerät hervor.

DER STIFTHEBER

Da wir gerade bei 'Nägeln und Schrauben' sind, wollen wir uns den Stiftheber ansehen. Nachdem wir Bits P0 bis P7 gründlich verbraucht haben, ist als einziges passendes User-Port-Bit Pin M übriggeblieben, Position 11 auf unserer Anschlußleiste, also PA2 (C 64) oder CB2 (PET). Ohne daß wir uns zu sehr in die Problematik des peripheren Kontrollregisters des VIA versenken möchten, können wir doch gefahrlos verraten, daß POKE DD+9,14*16(POKE $E84C,$E0) den CB2 des PET auf High setzt und POKE DD+9,12*16 ihn auf Low setzt. Wird Pin M jetzt mit einem weiteren Kanal eines Darlington-Chips verdrahtet (was einen der sechs übriggebliebenen Kanäle verbraucht), ist das resultierende Signal satt genug, um eine Zylinderspule zu betreiben. Für den C 64 kommt hinzu:

```
7000 D=PEEK(PO-1) AND 251:  REM PORT A,BIT 2 LOW: *** CBM 64
7010 IF P=0 THEN D=D+4 : REM ENERGISE TO LIFT PEN
7020 POKE PO-1,D=:RETURN:REM OUTPUT TO PORT A
```

Für den PET nehmen wir statt dessen:

```
7000 IF P=0 THEN POKE DD+9,224:RETURN:  REM ENERGISE TO
                                        LIFT PEN
7010 POKE DD+9,192:RETURN:  REM DEENERGISE, DROP PEN
```

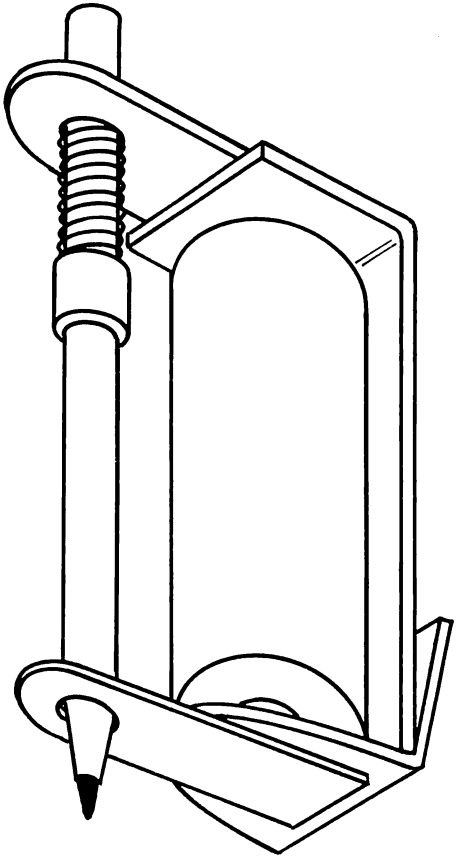


Abbildung 7.3 Stiftheber mit einem Postrelais

Damit ist das Problem nicht gelöst, wie wir eine Zylinderspule zum Betreiben des Stifthebers finden. Man kann einfach eine kommerzielle Zylinderspule kaufen, aber sie ist wahrscheinlich schwer und zu leistungsstark. Es braucht nicht viel, einen Kugelschreiber zu heben, und noch weniger, einen Filzstift zu heben, und mit etwas Geschicklichkeit läßt sich ein zu hoher Stromverbrauch vermeiden. Ein altes Postrelais liefert mehr als genug Hebekraft, wenn der Kontaktsatz entfernt ist. Sie müssen vielleicht ein wenig mit dem Stiftgewicht spielen, aber das Ergebnis müßte annehmbar sein, vorausgesetzt, die Räder sind nicht exzentrisch.

EIN EINFACHER BEFEHLSINTERPRET

Wir können uns jetzt dem Befehlsinterpreten zuwenden. Dieser ließe sich sehr elegant und kaum verständlich mit Suchanweisungen in Befehlslisten schreiben. Wir wollen lieber ein schlichteres Verfahren anwenden, das einfach jeden Befehl sofort ausführt. Es kann später modifiziert werden, so daß es eine Befehlsfolge speichert und editiert. Die Befehlseingabe wird nicht gerade durch das bunte Sortiment der erforderlichen Argumente erleichtert. Wir haben einerseits 'ADVANCE, 200' und andererseits 'CIRCLE, CW, 45, 150', und deshalb wird sich der Benutzer über eine benutzerfreundliche Anleitung freuen. Bringen wir die Zergliederungsroutine bei 1000 unter:

```
100 GOTO 1000

1000 PRINT"COMMAND: ";:INPUT A$
1010 IF A$<>"ADVANCE" THEN 1100
1020 PRINT"DISTANCE: ";:INPUT DS
1030 LM=DS/2:RM=DS/2
1040 GOSUB 8000:GOTO 1000

1100 IF A$<>"TURN" AND A$<>"CIRCLE" THEN 1300
1110 PRINT"CW/ACW: ";:INPUT B$
1120 PRINT"ANGLE(DEG): ";:INPUT AG
1130 IF B$="ACW"THEN AG=-AG
1140 IF A$="TURN" THEN LM=AG:RM=-AG:GOSUB 8000:GOTO
     1000
```

Turtle-Programm

```
10 GOTO 10000
1000 PRINT"COMMAND: ";:INPUT A$
1010 IF A$<>"ADVANCE" THEN 1100
1020 PRINT"DISTANCE: ";INPUT DS
1030 LM=DS/2:RM=DS/2
1040 GOSUB 8000:GOTO1000
1100 IF A$<>"TURN" AND A$<>"CIRCLE" THEN 1300
1110 PRINT "CW/ACW: ";:INPUT B$
1120 PRINT "ANGLE: ";:INPUT AG
1130 IF B$="ACW" THEN AG=-AG
1140 IF A$<>"TURN" THEN 1200
1150 LM=AG:RM=-AG:GOSUB 8000:GOTO 1000
1200 PRINT "RADIUS: ";:INPUT R
1210 IF B$="ACW" THEN R=-R
1220 LM=AG*(R/115+1):RM=AG*(R/115-1)
1230 GOSUB 8000:GOTO1000
1300 IF A$<>"SPEED" THEN 1400
1310 PRINT "SPEED WAS ";SP
1320 PRINT "NEW SPEED: ";:INPUT SP:GOTO 1000
1400 IF A$<>"PEN" THEN 1500
1410 PRINT "UP/DOWN: ";:INPUT B$
1420 P=1:IF B$="UP" THEN P=0
1430 GOSUB 7000:GOTO 1000
1500 PRINT "SORRY - CAN'T RECOGNISE COMMAND"
1510 PRINT "ADVANCE, TURN, CIRCLE, SPEED, PEN"
1520 GOTO 1000:REM ADD NEW COMMANDS AT 1500
7000 D=PEEK(PO-1) AND 251: REM *** CBM 64
7010 IF P=0 THEN D=D+4:REM LIFT PEN
7020 POKE PO-1,D: RETURN
8000 AL=ABS(LM): AR=ABS(RM)
8010 SL=SGN(LM): SR=SGN(RM)
8030 IF AR+AL=0 THEN RETURN
8040 IF AR>AL THEN 8200
8100 RA=AR/AL: T=.5
8110 FOR M=1 TO AL:GOSUB 9000:T=T+RA
8140 IF T>1 THEN GOSUB 9100: T=T-1
8150 GOSUB 9500:NEXT M:RETURN
8200 RA=AL/AR: T=.5
8210 FOR M=1 TO AR:GOSUB 9100:T=T+RA
8240 IF T>1 THEN GOSUB 9000: T=T-1
8250 GOSUB 9500:NEXT M:RETURN
9000 LP=(LP+SL) AND 7
9010 POKE PO,(PEEK(PO) AND MR)+LD(LP)
9020 RETURN
```

```
9100 RP=(RP+SR) AND 7
9110 POKE PO,(PEEK(PO) AND ML)+RD(RP)
9120 RETURN
9500 FOR D=1 TO 1000 STEP SP:NEXT:RETURN
10000 LP=0:RP=0:MR=240:ML=15:      REM MASKS
10010 PO=56577: DD=56579:     REM *** CBM 64
10020 SP=100:DIM LD(7),RD(8)
10030 FOR M=0 TO 7
10040 READ J: LD(M)=J: RD(M)=16*J: NEXT
10050 DATA 1,5,4,6,2,10,8,9:     REM N-S-E-W
10060 POKE DD,255: POKE PO,0
10070 GOTO 1000
```

KAPITEL 8

EINE SCHNITTSTELLE FÜR ROBOTER

Ein ausgewachsener Roboter hat sieben Freiheitsgrade, muß also mit sieben unabhängigen Motoren betrieben werden Der 'end effector' (ein Phantasiename für 'Hand') muß sich in drei Dimensionen bewegen können, und bei jeder gegebenen Position sollte er um drei weitere Achsen schwenkbar sein. Ein weiterer Kanal ist erforderlich für das 'Öffnen' und 'Schließen', wobei allerdings häufig einfach ein Ventil geöffnet oder geschlossen wird, das einen pneumatischen Greifer steuert. Lehrroboter wie der Armdroid opfern eine der 'Handgelenk'-Achsen, erlauben aber stetige Greifbewegungen. Das reduziert die Zahl der Kanäle auf sechs. Wenn diese mit Schrittmotoren betrieben werden, wie muß die Schnittstelle zum Computer aussehen? Im letzten Kapitel sind zwei Schrittmotorkanäle mit Hilfe aller acht Bits an den User-Port angeschlossen worden; zum Steuern von sechs Kanälen müssen wir also eine neue Technik finden. Dazu ist es erforderlich, in die User-Port-Daten eine Adresse einzubeziehen, die im Roboter selbst entschlüsselt werden kann.

STEUERN MEHRERER SCHRITTMOTOREN

Der User-Port hat acht Bits: ein Schrittmotor braucht vier Bits, um eine Halbschrittposition festzulegen (es sei denn, es macht Ihnen Spaß, eine Dreierskala für die Schnittstelle zu verwenden: N, aus, S). Es bleiben vier Bits für Verwaltung. Aus dreien dieser Bits läßt sich eine Adresse zum Ansprechen von drei Kanälen bilden. Der Adressenkanal fängt dann die Motorsignale in einem Vier-Bit-Latch und steuert danach weiter die Motorleitungen, bis er andere Anweisungen bekommt. Jetzt brauchen wir noch ein Strobe-Signal, um der Schaltungsanordnung mitteilen zu können, 'die Motorleitungen sind umgeschaltet, die Adreßleitungen sind gesetzt, übernimm jetzt diese Daten und verwende sie'.
Die Anschlüsse des schon etwas in die Jahre gekommenen Armdroiden, an dem die Programme dieses Kapitels ausprobiert worden sind, sind wie folgt belegt:

PB0	PB1	PB2	PB3	PB4	PB5	PB6	PB7
Strobe	------Kanal------			N---------O---------W---------S			

Beachten Sie, daß die Kompaß-Bits gegenüber dem letzten Kapitel vertauscht sind, und daß die Datenanweisungen für die Codes verändert sind. Das Vertauschen der 'Kanal'-Bits erschwert die Berechnung der Codes ein wenig, aber zum Schluß löst sich alles in Wohlgefallen auf.

Da das Strobe-Signal auf Low aktiv ist, verfährt man bei der *Ausgabe eines neuen* Befehls wie folgt:

1. Code für gewünschte Motorposition bestimmen.
2. 2*(Kanalnummer) addieren, Resultat in (sagen wir) CO ablegen.
3. Strobe-Bit (Bit 0) von CO auf High setzen.
4. CO auf den User-Port geben.
5. Bit 0 von CO auf Low setzen; CO auf den User-Port geben.
6. Bit 0 von CO wieder auf High setzen; CO auf den User-Port geben.

ÜBERSETZEN DES ALGORITHMUS IN SOFTWARE

Wie bisher definieren wir Daten und Felder 'ganz weit hinten' und legen die Verwaltung nach 10000:

```
10000 DD=59459:PO=59471:KB=547:KO=255:REM ANTIQUE
      PET ******
10000 DD=59459:PO=59471:KB=166:KO=255:REM 8000, 4000
      PET ****
10000 DD=56579:PO=56577:KB=197:KO=64:REM
      COMMODORE 64 *****
10010 DIM DR(7): REM DRIVE VALUES WITH STROBE ALREADY
      HIGH
10020 FOR I=0 TO 7: READ J:DR(I)=16*J+1: NEXT
10030 DATA 1,3,2,6,4,12,8,9:REM N-E-S-W
10040 MA=254:POKE DD,255 :REM SET TO OUTPUTS
```

Nun können wir uns ein Unterprogramm zum Betreiben eines der Motoren ansehen. Wenn die Kanalnummer in Variable CH gesetzt und der Schrittwinkel als Wert von V abgelegt ist (gezählt als Schrittzahl von der Einschaltposition an), dann sendet die folgende Routine den nötigen Code an den Motor:

```
9000 CO=DR(V AND 7)+CH*2
9010 POKE PO,CO: REM OUTPUT THE CODE,STROBE BIT HIGH
9020 POKE PO,CO AND MA :REM 'AND' WITH MASK-STROBE LOW
9030 POKE PO,CO: REM STROBE HIGH AGAIN
9040 RETURN
```

Diese Routine macht ein oder zwei Abkürzungen im eben beschriebenen Algorithmus und wählt einfach einen Motorantrieb zur Ansteuerung einer gegebenen Position.

AUSTESTEN DER ANSCHLÜSSE

Für die Anschlüsse auf der Seite des Armdroiden müssen Sie das Handbuch konsultieren – der Steckverbinder ist bei den neueren Versionen verändert worden. Falls Sie sich einen eigenen Schaltkreis gebaut haben (vielleicht nach dem später in diesem Kapitel angegebenen), sind Sie schon damit vertraut.
Der nächste Test überzeugt Sie nicht nur davon, daß tatsächlich etwas läuft; er stellt auch fest, welche Kanalnummer welche Achse des Roboters steuert, und in welcher Richtung. Geben Sie das folgende einfache Testprogramm ein. Da es im Verlaufe dieses Kapitels immer wieder umgeschrieben wird, sollten Sie es in jeder Phase zu späterem Gebrauch sicherstellen.

```
  10 GOTO 10000
 100 INPUT "CHANNEL NUMBER,DISTANCE":CH,DI
 110 FOR V=0 TO DI STEP SGN (DI)
 120 GOSUB 9000:NEXT
 130 GOTO 100
10900 GOTO 100:REM AT END OF HOUSEKEEPING
```

Prüfen Sie jetzt nacheinander jeden Kanal, indem Sie Werte von 0 bis 7 für die Kanalnummer und ungefähr 50 oder −50 für die Distanz eingeben. Zwei der acht Möglichkeiten sind natürlich ohne Wirkung, da nur sechs Kanäle in Gebrauch sind. Notieren Sie sich sorgfältig Achse und Richtung der Bewegungen auf, ab, Schwenk nach links, rechts, vorwärts, zurück, Greifer öffnen und schließen. Der Armdroid braucht jeweils zwei Motoren, um das Armgelenk zu drehen und es auf und ab zu schwenken. Halten Sie fest, welcher was tut. Ein Programm mit einer Ausgaberoutine in Maschinensprache kann sich die Zeit zum Entwirren getrennter Befehle zum Drehen und Kippen des Handgelenks leisten. In BASIC wird das System dadurch ziemlich langsam, und deshalb wollen wir zunächst nur jeweils einen einzigen Motor betreiben.

TASTENBEFEHLE

Wir wollen jetzt eine Routine hinzufügen, die es ermöglicht, einen Motor durch Festhalten einer Taste zu betreiben. Es ist wichtig, ein Menü von Tasten auf den Bildschirm zu geben, das anzeigt, welche Taste was tut. Um dem Text durch

Verwendung von Groß- und Kleinschreibung ein eleganteres Aussehen zu verleihen, schalten wir den Kleinschreibmodus ein, bevor wir das Programm eintippen. Natürlich erscheinen beim Auflisten in diesem Modus alle Befehle klein geschrieben, doch sind sie der Deutlichkeit halber hier in den üblichen Großbuchstaben anzugeben.
Nun müssen wir jeder Bewegung eine Taste zuordnen: auf, ab, links, rechts, vorwärts, zurück, öffnen und schließen, außerdem Handgelenk auf, ab und drehen. Wir nehmen dazu die Tasten U, D, L, R, F, B, O und C (up, down, left, right, forwards, backwards, open, close) und benötigen weitere vier für das Handgelenk – warum nicht W, Q, T und Y? Jeder Taste entspricht eine Kanalnummer und eine Richtung. Um sie zu ordnen, müssen wir die Verwaltung um einen Satz von Datenanweisungen ergänzen – korrigieren Sie die unten angegebenen Werte anhand der oben erhaltenen Ergebnisse, um die richtigen Bewegungen zu bekommen. Die hier gegebenen Werte sind die, die Richard für seinen speziellen Armdroiden erhalten hat:

```
10100 DIM CH(5), HE(5):FOR I=0 TO 5:READ J
10110 CH(I)=2*J:NEXT:REM CHANNEL CODES
10120 DATA 1,3,5,4,2,6:REM U/D, L/R, F/B, O/C, WRIST
10130 C$="UDLRFBOCWQTY":REM COMMAND KEYS
```

Vertauschen Sie nötigenfalls die Buchstabenpaare in C$, damit der erste der positiven Richtung und der zweite der negativen entspricht. Das Feld HE(I) (für HEre) merkt sich die aktuelle Position jedes Motors und erweist sich später als sehr nützlich. Bei 1000 fügen wir eine Routine ein, um das Menü anzuzeigen, die gedrückte Taste zu lesen, den Befehl zu interpretieren und ihn auszuführen:

```
1000 PRINT CHR$(147);    "Up      Down"
1010 PRINT"Left           Right"
1020 PRINT"Forward        Back"
1030 PRINT"Open           Close"
1040 PRINT"Wrist-         Q"
1050 PRINT"Twist-         Y"
1100 GET A$:K=PEEK(KB):IF K= KO OR A$="" THEN 1100
1150 FOR I=1 TO LEN(C$):IF A$<> MID$(C$,I,1) THEN NEXT:
     GOTO1000
1160 J=I:I = LEN(C$):NEXT: REM CLOSE LOOP NEATLY
1170 CH=INT((J-1)/2):DI=2*(J AND 1)-1:V=HE(CH)
1180 V=V+DI:GOSUB9000:REM TAKE A STEP
1190 IF PEEK(KB)= K THEN 1180:REM KEEP STEPPING IF PRESSED
1200 HE(CH)= V:GOTO1000
```

Damit das funktioniert, muß Zeile 9000 zum Entwirren der Kanäle geändert werden:

```
9000 CO=DR(V AND 7)+CH(CH)
```

Außerdem müssen wir die alten Zeilen 100–130 ersetzen durch:

```
100 PRINT CHR$(14):GOTO 1000
```

In der jetzigen Form müßte das Programm einigermaßen schnell arbeiten. Mit einem kleinen Trick läßt es sich sogar noch ein wenig beschleunigen. Um auf eine Variable zuzugreifen, muß BASIC die Liste der Variablen in der Reihenfolge durchgehen, in der sie zuerst angetroffen wurden, bis die richtige erreicht ist. Also werden die zuerst definierten Variablen am schnellsten verarbeitet. Beim Durchgehen des Programms erkennen wir, daß CO, PO, V, DI, CH, KB und K bei jedem Schritt gebraucht werden, manche sogar mehrfach. Wenn wir Zeile 10 in die seltsam aussehende Form:

```
10 DIM CO,PO,V,DI,CH,I,R,KB,K:GOTO 10000
```

bringen, sollte sich ein hinreichender Geschwindigkeitszuwachs ergeben. (I und R sind für später mit aufgeführt.)

PROGRAMMIERTE BEWEGUNGEN

Wenn jede Bewegung über die Tastatur gesteuert werden muß, ist der Roboter nur ein Spielzeug. Wenn wir ihm jedoch eine Folge von Bewegungen einprogrammieren können, die er automatisch ausführt, können wir allmählich seinen ernsthaften Einsatz in Erwägung ziehen. Wir müssen dazu jeden Zielpunkt aufzeichnen können und benötigen eine zweite Befehlsebene, auf der wir zwischen dem Abschnitt im 'Teach'-Modus (dem schon erprobten Programm) und den Routinen umschalten können, die den Roboter automatisch betreiben. Dafür brauchen wir ein zweites Menü, das bei Zeile 100 beginnt:

```
100 PRINT CHR$(147);      "Teach,      Perform,"
110 PRINT "Repeat,        Clear,"
120 PRINT "Save,          Input"
130 GET A$: IF A$=""THEN 130: REM WAIT FOR A KEY-PRESS
140 FOR I=1 TO 6:IF A$<> MID$("TPRCSI",I,1) THEN NEXT:
    GOTO100
150 J=I:I = 6:NEXT: REM CLOSE THE FOR...NEXT LOOP NEATLY
160 ON J GOTO 1000,5000,5100,2000,6000,7000
```

Bisher haben wir nur die 'Teach'-Routine bis 1000 geschrieben, und selbst sie muß leicht modifiziert werden. Wir müssen die Wahl einer beliebigen Stelle im Manöver möglich machen, und wir müssen eine Rückkehr zum Befehlsmodus erlauben. Deshalb füllen wir die Lücken im Programm mit:

```
1060 PRINT"Point        End teach"
1110 IF A$="E" THEN 100
1120 IF A$<>"P"THEN 1150
1130 IF NP=MP THEN 1000:REM TOO MANY POINTS
1140 NP=NP+1:FOR I=0 TO 5:PT(I,NP)=HE(I):NEXT:GOTO 1000
```

Jetzt noch etwas zur Verwaltung: Um die Punkte abzulegen, muß das Feld PT(5,MP) dimensioniert werden. Ein brauchbarer Wert für MP ist 20, aber wenn Sie wollen, können Sie ihn viel größer wählen.

```
10300 NP=0:MP=20:DIM PT(5,MP)
```

Um über den Fortgang auf dem laufenden zu bleiben, können wir die aktuellen Koordinaten und die Anzahl der gesetzten Punkte mit:

```
1080 PRINT:PRINT NP;"POINTS"
1090 FOR I=0 TO 5:PRINT HE(I):NEXT
```

anzeigen. Damit ist der Befehlsmodus-Abschnitt des Programms abgeschlossen. Es bleibt die Routine zu schreiben, die einen Satz von Bewegungen steuert:

ROUTINEN FÜR DIE AUSFÜHRUNG EINES MANÖVERS

Nachdem wir die Bewegungen gespeichert haben, können wir sie ausführen, indem wir nacheinander Zielpunkte aufsuchen und dann eine Routine bei 8000 aufrufen, um uns von HE(I) nach TA(I) (von HEre nach TArget) zu bewegen. Wenn wir sie nur einmal ausführen möchten, gehen wir nach 5000:

```
5000 IF NP=0 THEN GOTO 100:REM NO POINTS
5010 FOR P=1 TO NP
5020 FOR I=0 TO 5:TA(I)=PT(P,I):NEXT
5030 GOSUB 8000
5040 NEXT P
5050 GOTO 100
```

Wenn wir den Zyklus wiederholen möchten, bis eine Taste angeschlagen wird, können wir einen Programmabschnitt bei 5100 benutzen:

```
5100 IF NP=0 THEN GOTO 100
5110 FOR P=1 TO NP
5120 FOR I=0 TO 5
5130 TA(I)=PT(P,I):NEXT
5140 GOSUB 8000
5150 NEXT P
5160 GET A$:IFA$=""THEN 5110: REM NO KEY, ROUND AGAIN
5170 GOTO 100
```

Damit ist noch immer nicht das Problem geklärt, was wir bei 8000 unterbringen müssen. Um die Reaktionszeit zu optimieren, bewegen wir zunächst nur jeweils eine Achse; allerdings ziehen wir später auch diagonale Bewegungen in Betracht. Nachdem das Feld TA(5) mit:

```
10200 DIM TA(5)
```

dimensioniert ist, können wir jeden Kanal von TA mit HE vergleichen und nötigenfalls eine entsprechende Bewegung ausführen.

```
8000 FOR CH=0 TO 5
8010 IF TA(CH)=HE(CH)THEN NEXT:RETURN
8020 FOR V=HE(CH) TO TA(CH) STEP SGN(TA(CH)-HE(CH))
8030 GOSUB 9000:NEXT:HE(CH)=TA(CH)
8040 NEXT:RETURN
```

Nun können wir die Befehlsroutinen mit einem Einzeiler 'Clear' abschließen:

```
200 NP=0:GOTO 100
```

Wir sollten auch Routinen bei 6000 und 7000 einschieben, um einen Satz von Bewegungen abzulegen und zurückzuholen. Fürs erste füllen Sie die Zeilen mit:

```
6000 GOTO 100
7000 GOTO 100
```

und schreiben eigene Routinen, wenn Sie das übrige Programm ausgetestet haben. In der jetzigen Form läßt sich mit dem Programm ein Roboter ganz annehmbar steuern, aber daß die Manöver eine Achse nach der anderen ausgeführt werden,

sieht unelegant aus. Es wäre viel besser, die Bewegungen durch gleichzeitiges Ansprechen aller Achsen zu interpolieren. Leider ist das Programm im nächsten Abschnitt unerträglich langsam in der Ausführung, und nur mit Hilfe von Maschinensprache läßt sich ein wirklich befriedigendes Ergebnis erzielen.

GLEICHZEITIGE BEWEGUNGEN

Wir hätten gern eine Routine, die alle sechs Kanäle des Roboters anspricht und ein Manöver so interpoliert, daß alle Kanäle gleichzeitig arbeiten können. Das ist ebenfalls eine Aufgabe für den Bitratenverstärker, der diesmal aus sechs Rohren feuert. Die Startposition ist HEre, ein Feld mit sechs Elementen, eines für jede Motorachse. Das Ziel wird in TArget gehalten, und wir können alle sechs Achsen durchgehen, um diejenige zu finden, die die größte Änderung verlangt. Es ist dann eine einfache Aufgabe, die Verhältnisse zu berechnen und einen Schritt aufgrund des Überlaufs eines Registers zu machen, genau wie im letzten Kapitel. Unterwegs brauchen wir einige weitere Felder, und die Verwaltungsroutine bei 10200 erhält die folgende Gestalt:

```
10200 DIM TA(5),RA(5),WA(5),RE(5)
10210 FOR CH=0 TO 5:HE(CH)=0:RE(CH)=.5
10220 V=0:GOSUB9000:NEXT:REM ZERO MOTORS
```

Jetzt können wir ein Unterprogramm schreiben, das alle sechs Kanäle von HEre bis zur TArget-Position steuert:

```
8000 REM MOVE FROM HERE TO TARGET : FIRST FIND LONGEST
     MOVE
8010 RM=0:FOR CH=0 TO 5
8020 RA(CH)=ABS(TA(CH)-HE(CH)): REM WORK OUT DISTANCE
     AND
8030 WA(CH)=SGN(TA(CH)-HE(CH)): REM DIRECTION FOR EACH
     AXIS
8040 IF RA(CH)>RM THEN RM=RA(CH): REM FIND MAX DISTANCE
8050 NEXT:IF RM=0 THEN RETURN: REM NO MOVE
8060 FOR CH=0 TO 5
8070 RA(CH)=RA(CH)/RM:NEXT:REM RATES NOW IN RANGE 0 TO 1

8100 FOR R=1 TO RM: REM NOW WE ARE READY TO MOVE
8110 FOR CH=0 TO 5
8120 RE(CH)=RE(CH)+RA(CH)
8130 IF RE(CH)< 1 THEN 8160
```

```
8140 RE(CH)=RE(CH)-1:HE(CH)=HE(CH)+WA(CH)
8150 V=HE(CH):GOSUB 9000 :REM MOVE MOTOR
8160 NEXT CH
8170 NEXT R
8180 RETURN : REM NOW HERE= TARGET
```

Die Zeilen 8100 bis 8180 sind so geschrieben, daß sie leicht verständlich sind. Die Geschwindigkeit kann auf Kosten der Klarheit etwas verbessert werden. Versuchen Sie es nach dem Austesten der ersten Version statt dessen mit folgendem:

```
8100 FOR R=1 TO RM:FOR CH=0 TO 5:I=RA(CH):IF I=0 THEN 8130
8110 I=I+RE(CH):IF I<1 THEN RE(CH)=I:NEXT:NEXT:RETURN
8120 RE(CH)=I-1:V=HE(CH)+WA(CH):HE(CH)=V:GOSUB 9000
8130 NEXT:NEXT:RETURN
```

8140–8180 sind jetzt überflüssig und sollten gelöscht werden.
Selbst wenn Sie nur das hier aufgelistete Programm verwenden, können Sie dem Roboter eine Routine beibringen, die er dann mit interpolierten Bewegungen ausführt. Ich will noch immer nicht behaupten, daß sie schnell arbeitet – Sie möchten sicher bald die Funktionen des Bitratenverstärkers in Maschinensprache realisieren und eine lineare Beschleunigungsroutine einfügen, mit der die Höchstgeschwindigkeit ohne Abbrechen erreicht wird. Einige elegante Programme (z. B. MEMROB, das einen Armdroiden mit einem PET steuert, in Zusammenarbeit mit Tim Dadd geschrieben wurde und von Colne vertrieben wird) verbinden die Interpolations- und Ausgabefunktionen mit dem Interrupt des Computers und kommunizieren zwischen BASIC und Maschinensprache, indem sie Werte in ein Variablenfeld setzen. Dadurch hat der BASIC-Teil Zeit zum 'Nachdenken', nämlich zum Bestimmen der Geschwindigkeitsverhältnisse für die nächste Bewegung, während gleichzeitig eine Bewegung ausgeführt wird. Das MEMROB-Listing ist vielen ein Rätsel und hat wenig pädagogischen Wert.

ROBOTER IM EIGENBAU

Zunächst brauchen Sie ein halbes Dutzend Schrittmotoren. Der im letzten Kapitel erwähnte ID35 ist gut geeignet – es ist der, der beim Armdroiden verwendet wird. Die Darlington-Verstärker sind ziemlich gründlich in Kapitel 6 besprochen worden. Es bleiben die Kanaldecodierer und die Vierbit-Latches – und die Stromversorgung. Das im Kapitel 1 beschriebene Netzgerät müßte ausreichen und kostet weniger als ein einziger Schrittmotor. Ein Schaltdiagramm für die 'Innereien' des Roboters ist in **Abbildung 8.1** gezeichnet und sollte alles Erforderliche leisten.

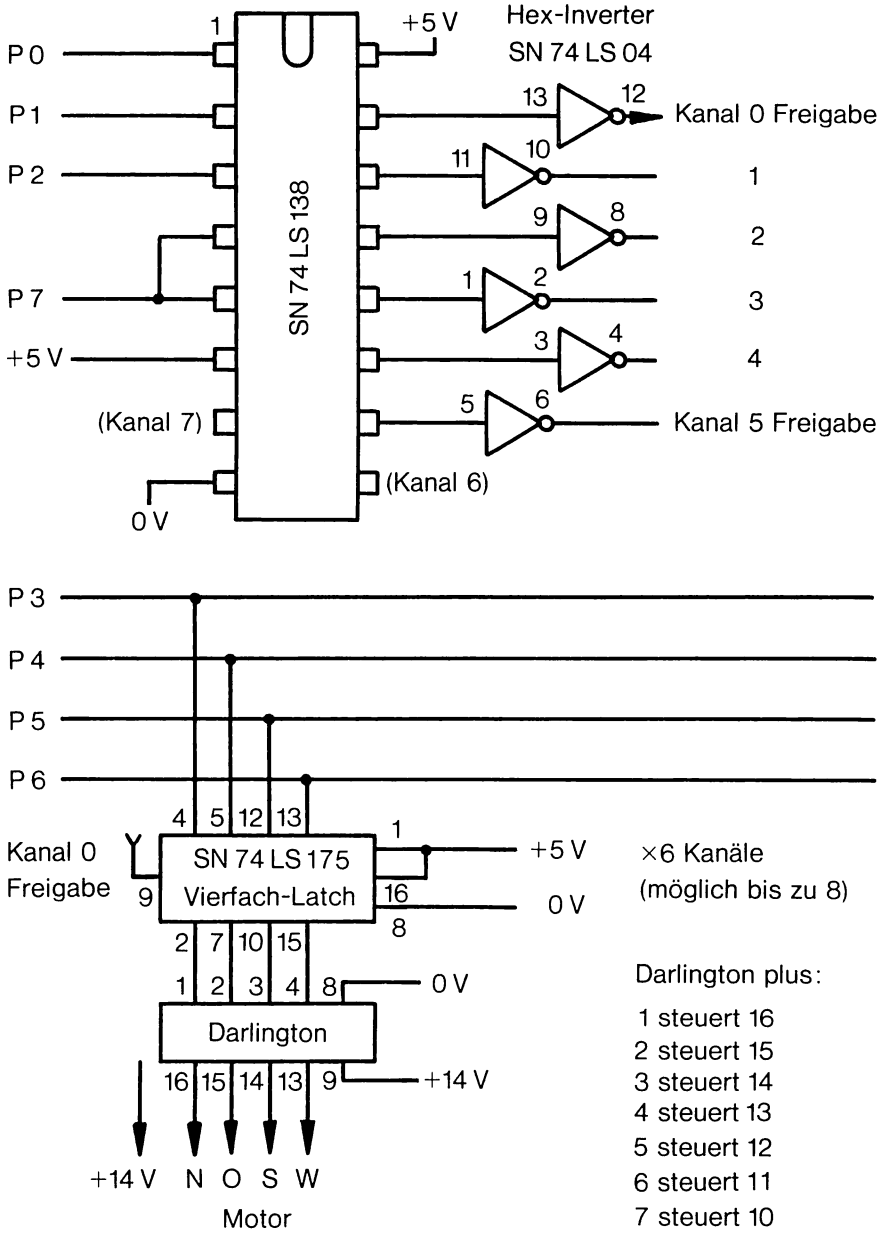


Abbildung 8.1 Schaltung für einen 6achsigen Roboter: Decodierer, Latches, Verstärker

Um die Mechanik für den Eigenbau selbst zu entwerfen, können Sie entweder auf den Spuren kommerzieller Roboter wandeln oder auch etwas mutiger sein. Wenn Sie sich einmal überlegen, wie viele verschiedene Möglichkeiten es gibt, sechs Motoren miteinander zu verbinden, werden Sie auf eine erstaunliche Vielfalt kommen. Ihre Geometrie kann kartesisch, polar, zylindrisch-polar oder irgendein seltsamer Zwitter sein. Zuerst wollen wir uns 'konventionelle' Roboter ansehen.

ROBOTERANATOMIE

Die erste Bewegungsachse ist eine Drehung der ganzen Vorrichtung um die Senkrechte. Das läßt sich als Hüftdrehung 'vermenschlichen'. Als nächstes läßt das Schultergelenk den Arm auf und nieder schwenken, so daß allein mit diesen zwei Motoren die Hand jeden Punkt auf der Oberfläche einer Kugel erreichen könnte. Dann kommt das Ellbogengelenk. Wenn es gebeugt ist, wird der Arm im Prinzip verkürzt, obwohl mehr und mehr Trigonometrie nötig ist, um die Bewegung der Hand zu beschreiben. Grundsätzlich sollte der Roboter jetzt jeden Punkt innerhalb einer Kugel erreichen können, aber wenn zum Beispiel der Oberarm nicht dieselbe Länge wie der Unterarm hat, sind gewisse Bereiche nicht zugänglich. Das Handgelenk müßte jetzt sowohl auf und ab als auch von links nach rechts schwenken können. Der Unimation Puma verwendet statt dessen eine Bewegung wie das menschliche Handgelenk, bei dem das Auf-und-ab-Scharnier eine Achse ist, die wiederum um die Richtung des Unterarms geschwenkt werden kann. Beim Armdroiden fehlt eine dieser Bewegungen. Der Puma kann im Prinzip einen Schraubenzieher auf eine Schraube ausrichten; die letzte Achse dreht den Schraubenzieher und damit die Schraube.

Einige hochentwickelte Roboter wie der Puma stellen umfangreiche Berechnungen an, damit der Benutzer die Hand in gerader Linie bewegen kann und das Werkzeug sich nicht im Raum dreht. Bis zu vierzig Mal in der Sekunde werden neue Positionen für die Motorachsen bestimmt, und die Bewegung wird dann durch 'Steuerung der Verhältnisse' ähnlich den früher beschriebenen Techniken geglättet. Eine verlokkende Aufgabe!

Selbst wenn die Geometrie festgelegt ist, gibt es noch viele Varianten, die Motoren mit den Achsen zu verbinden. Der Puma braucht rohe Gewalt, und die gesamte Hebelwirkung des Armes einschließlich Last greift am Schultergelenk an. Beim Armdroiden hingegen bleibt durch geschickte Seilführung der Unterarm parallel zur Ausgangsstellung, wenn die Schulter der Oberarm dreht. Dadurch wird im Prinzip die Hebelwirkung der Last auf den Schultermotor halbiert – obwohl das dem Seil nicht gut tut, das ärgerlicherweise zum Reißen neigt.

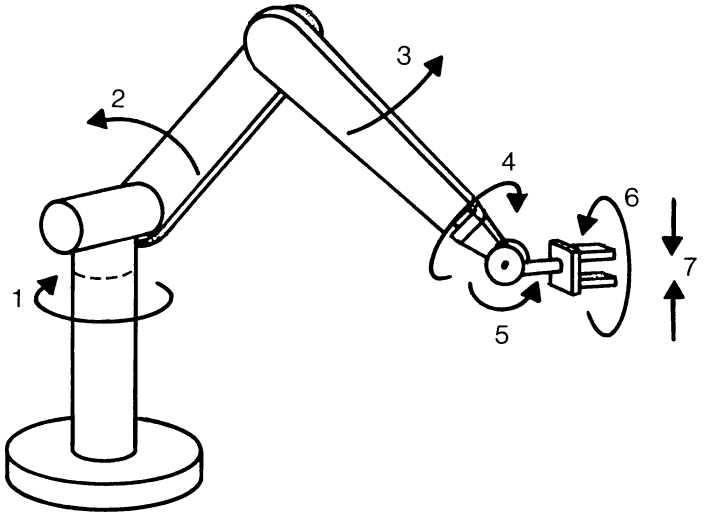


Abbildung 8.2 Die Achsen eines 'konventionellen' Roboters

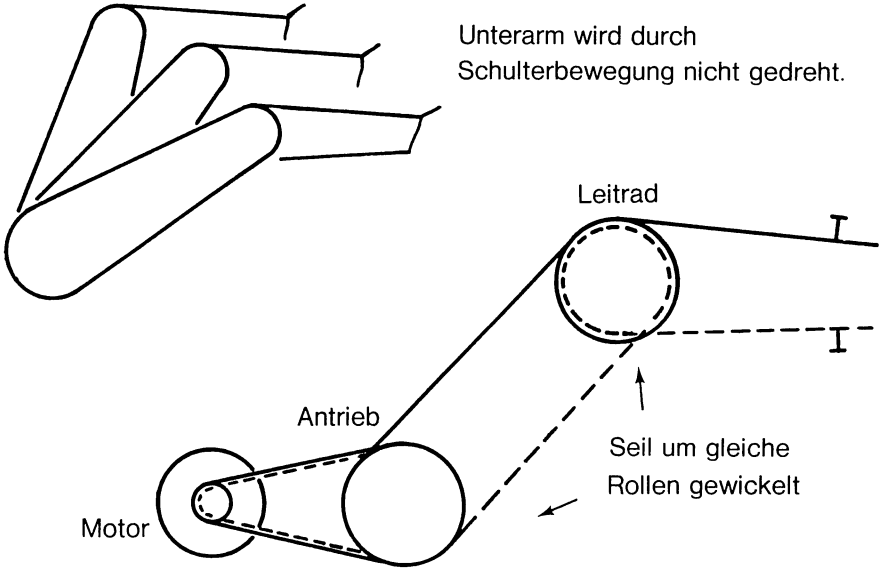


Abbildung 8.3 Seilführung für parallele Unterarmbewegungen

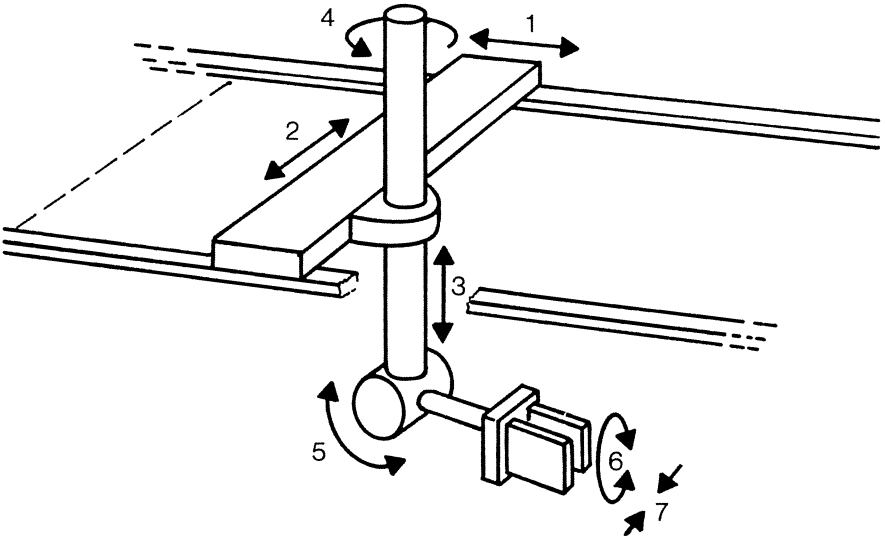


Abbildung 8.4 Anordnung beim kartesischen Roboter

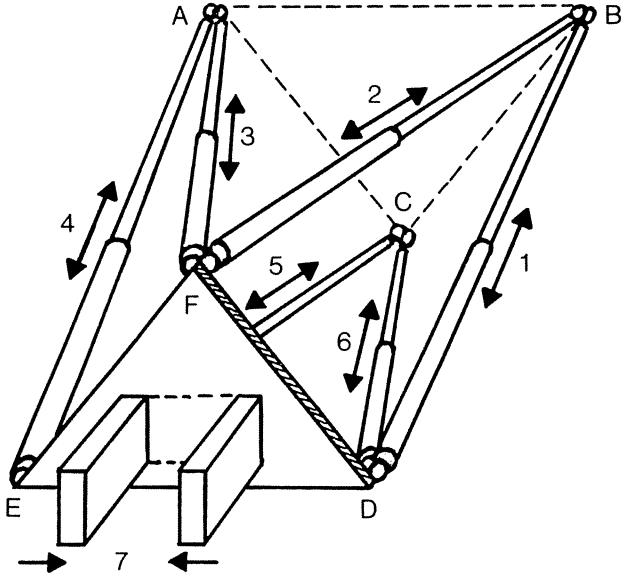


Abbildung 8.5 Eine Variante der 'Gadfly'

Der IBM-Roboter ist kartesisch, und die Steuerung der X- und Y-Achse ähnelt sehr einem überlebensgroßen Grafikplotter. Die Z-Achse ist ein noch riesenhafterer Stiftheber, der eine Stange hebt und senkt, die sich dreht und so die erste Handgelenkachse bildet. Alle Probleme bei geradlinigen Bewegungen sind mit einem Schlag gelöst, aber es sind jetzt Schienen und Flaschenzüge statt Zapfen und Hebel erforderlich.
Wenn Rechenkapazität entfesselt wird, ist alles möglich. Die Industrieforschung ist intensiv mit der Entwicklung eines Geräts mit sechs ausfahrbaren motorgetriebenen Stäben und Leitspindeln beschäftigt. Stellen Sie sich vor, ein Dreieck ABC ist am Boden befestigt, und das Werkzeug ist auf einer beweglichen dreieckigen Platte DEF angebracht. Die Platte wird durch sechs Stäbe AE, AF, BF, BD, CD und CE gehalten. Wenn diese ihre Länge verändern, kann die Platte in drei Dimensionen bewegt und um drei Achsen gedreht werden. Versuchen Sie in einer ruhigen Minute, die Beziehung zwischen diesen Längen und der Position der Platte zu berechnen, oder spezieller die Längen, die erforderlich sind, um die Platte in eine bestimmte Position zu bringen – es ist ein Preis auf die einfachste Lösung ausgesetzt! Nicht umsonst bedeutet 'Gadfly' im Englischen 'Viebremse' oder 'Störenfried'.
Diese Varianten erschöpfen nicht annähernd die möglichen Kombinationen. Wenn Sie sechs Schrittmotorkanäle anschließen, können Sie beliebig viele Versionen aus Pappe, Fäden und Balsaholz ausprobieren, bevor Sie Ihren Entwurf in Aluminium oder Stahl verewigen. Viel Glück!

Programm zur Robotersteuerung

Anmerkung: Für den PET muß Zeile 10000 geändert werden.

```
10 DIM CO,PO,V,DI,CH,I,R,KB,K: GOTO 10000
100 PRINT CHR$(147);"TEACH,     PERFORM,"
110 PRINT"REPEAT,    CLEAR"
120 PRINT"SAVE,      INPUT"
130 GET A$: IF A$="" THEN 130:REM AWAIT KEYPRESS
140 FOR I=1 TO 6
145 IF A$<>MID$("TPRCSI",I,1) THEN NEXT:GOTO 100
150 J=I:I=6:NEXT: REM CLOSE 'FOR' LOOP NEATLY
160 ON J GOTO 1000,5000,5100,2000,6000,7000
1000 PRINT CHR$(147);"UP          DOWN"
1010 PRINT"LEFT       RIGHT"
1020 PRINT"FORWARD    BACKWARD"
1030 PRINT"OPEN       CLOSE"
1040 PRINT"WRIST  -   Q"
1050 PRINT"TWIST  -   Y"
1060 PRINT"POINT  END TEACH"
```

```
1080 PRINT:PRINT NP;"POINTS"
1090 FOR I=0 TO 5:PRINT HE(I): NEXT
1100 GET A$:K=PEEK(KB):IF K=KO OR A$=""THEN 1100
1110 IF A$="E" THEN 100
1120 IF A$<>"P"THEN 1150
1130 IF NP=MP THEN GOTO 1000:REM TOO MANY POINTS
1140 NP=NP+1
1145 FOR I=0 TO 5:PT(NP,I)=HE(I):NEXT:GOTO 1000
1150 FOR I=1 TO LEN(C$)
1155 IF A$<>MID$(C$,I,1) THEN NEXT: GOTO 1000
1160 J=I: I=LEN(C$): NEXT:REM CLOSE LOOP NEATLY
1170 CH=INT((J-1)/2): DI=2*(J AND 1)-1: V=HE(CH)
1180 V=V+DI:GOSUB 9000:          REM TAKE A STEP
1190 IF PEEK(KB)=K THEN 1180:  REM KEEP STEPPING
1200 HE(CH)=V:GOTO 1000
2000 NP=0:GOTO 100
5000 IF NP=0 THEN GOTO 100:         REM NO POINTS
5010 FOR P=1 TO NP
5020 FOR I=0 TO 5: TA(I)=PT(P,I): NEXT
5030 GOSUB 8000
5040 NEXT P
5050 GOTO 100
5100 IF NP=0 THEN GOTO 100
5110 FOR P=1 TO NP:PRINT P;
5120 FOR I=0 TO 5
5130 TA(I)=PT(P,I): NEXT
5140 GOSUB 8000
5150 NEXT P
5160 GET A$:IF A$="" THEN 5110
5170 GOTO 100
6000 GOTO 100:REM PUT 'SAVE' HERE
7000 GOTO 100:REM PUT 'INPUT' HERE
8000 REM MOVE FROM HERE TO TARGET
8010 M=0: FOR CH=0 TO 5
8020 RA(CH)=ABS(TA(CH)-HE(CH))
8030 WA(CH)=SGN(TA(CH)-HE(CH))
8040 IF RA(CH)>RM THEN RM=RA(CH)
8050 NEXT: IF RM=0 THEN RETURN:       REM NO MOVE
8060 FOR CH=0 TO 5
8070 RA(CH)=RA(CH)/RM: NEXT
8100 FOR R=1 TO RM
8105 FOR CH=0 TO 5:I=RA(CH):IF I=0 THEN 8130
8110 I=I+RE(CH):IF I<1 THEN RE(CH)=I:NEXT:RETURN
8120 RE(CH)=I-1: V=HE(CH)+WA(CH): HE(CH)=V
8125 GOSUB 9000
```

```
8130 NEXT: NEXT: RETURN
9000 CO=DR(V AND 7)+CH(CH)
9010 POKE PO,CO:POKE PO,CO AND MA:POKE PO,CO
9020 RETURN
10000 DD=56579:PO=56577:KB=197:KO=64: REM CBM 64
10010 DIM DR(7): REM DRIVE VALUES WITH STROBE HI
10020 FOR I=0 TO 7: READ J: DR(I)=16*J+1: NEXT
10030 DATA 1,3,2,6,4,12,8,9: REM N-E-S-W
10040 MA=254: POKE DD,255:     REM SET TO OUTPUTS
10100 DIM CH(5),HE(5): FOR I=0 TO 5: READ J
10110 CH(I)=2*J: NEXT:          REM CHANNEL CODE
10120 DATA 1,3,5,4,2,6
10130 C$="UDLRFBOCQWTY":        REM COMMAND KEYS
10200 DIM TA(5),RA(5),WA(5),RE(5)
10210 FOR CH=0 TO 5: HE(CH)=0: RE(CH)=.5
10220 V=0: GOSUB9000: NEXT:  REM ZERO THE MOTORS
10300 NP=0:MP=20:DIM PT(5,MP)
10900 GOTO 100
```

Einfache Steuerung und Kanalidentifizierung

Anmerkung: Für den PET muß Zeile 10000 geändert werden.

```
10 GOTO 10000
100 INPUT"CHANNEL NUMBER,DISTANCE";CH,DI
110 FOR V=0 TO DI STEP SGN(DI)
120 GOSUB 9000:NEXT
130 GOTO 100
9000 CO=DR(V AND 7)+CH*2
9010 POKE PO,CO:POKE PO,CO AND MA:POKE PO,CO
9020 RETURN
10000 DD=56579:PO=56577:KB=197:KO=64: REM CBM 64
10010 DIM DR(7): REM DRIVE VALUES WITH STROBE HI
10020 FOR I=0 TO 7: READ J: DR(I)=16*J+1: NEXT
10030 DATA 1,3,2,6,4,12,8,9: REM N-E-S-W
10040 MA=254: POKE DD,255:     REM SET TO OUTPUTS
10900 GOTO 100
```

KAPITEL 9

ANALOGE AUSGABE UND POSITIONSREGLER

Ein Schrittmotor hat zwar eine einfache Schnittstelle, aber keinen absoluten Positionsbezug, und weitere Probleme tauchen auf, wenn schnelle Reaktionen nötig sind. Ein analoger Servomotor erhält seinen Bezug aus einem einfachen Potentiometer, allerdings kann man auch viel subtilere Einrichtungen wie Drehmelder und Drehregler einsetzen. Das Rückkopplungssignal wird vom Befehlssignal subtrahiert, und die Differenz entspricht dem Positionsfehler des Servomotors. Der Motor wird dann im Verhältnis zum Fehler betrieben, so daß sich der Fehler dabei vermindert. Wenn die gewünschte Position erreicht ist, verlangt der Motor keine Energie mehr. Der Nachteil ist, daß bei einer stationären Belastung des Motors ein systematischer Fehler entsteht, dessen Wert dem zum Ausgleich erforderlichen Antrieb entspricht. Um diesen Fehler zu minimieren, muß der Regelkreis 'steif' sein, das heißt, ein kleiner Fehler muß ein großes Drehmoment des Motors ergeben. Dadurch entstehen weitere Probleme, denn ein kleiner Fehler kann ein so großes Drehmoment hervorrufen, daß der Motor Geschwindigkeit aufnimmt, am Ziel vorbeischießt und auf der anderen Seite mit einem größeren Fehler zur Ruhe kommt. Dann wirbelt er natürlich wieder zurück, und wieder . . . Der Motor hat angefangen, zu oszillieren.

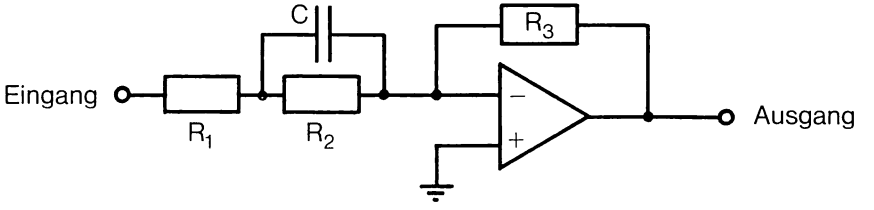
RÜCKKOPPLUNG UND STABILITÄT

Um ein Oszillieren zu vermeiden, muß entweder die Steifigkeit des Systems verringert oder ein Geschwindigkeitssignal hinzugefügt werden. Ein Geschwindigkeitsanteil bremst den Motorantrieb, wenn er Geschwindigkeit aufnimmt, so daß es zu jedem gegebenen Fehler eine Geschwindigkeit gibt, bei der der Motor frei läuft. Wenn die Freilaufgeschwindigkeit überschritten wird, arbeitet der Motor umgekehrt, um die Bewegung zu verlangsamen. Bei der richtigen Mischung aus Position und Geschwindigkeit kommt der Motor rasch in der gewünschten Position zum Stehen. Ein Geschwindigkeits- oder 'Tacho'-Signal ist gewöhnlich teuer, aber ein ähnlicher Effekt läßt sich auch billiger erzielen. Ein gewisser Betrag an Geschwindigkeitsrückkopplung kommt vom Motor selbst, nämlich die durch die Umdrehung eines Gleichstrommotors erzeugte 'Gegen-Spannung'. Sie begrenzt die Geschwindigkeit, die der Motor bei gegebener Verstärkerausgangsspannung erreicht, und fügt einem System geringer Steifigkeit die nötige Dosis Dämpfung hinzu. Wird die 'Entdämpfung' des Regelkreises (d. h. die Ausgangsspannung pro Einheit des Positionsfehlers) erhöht, so ist die Eigendämpfung weniger wirksam. Im Grenzwert ist das System 'ein-aus' und reagiert mit voller Aussteuerung auf den geringsten

Fehler, und wenn auf den Verstärkereingang kein Geschwindigkeitssignal gesetzt wird, ist ein Oszillieren unausweichlich.
Ein Hochleistungs-Servomotor hat gewöhnlich einen eigenen Tacho für das Geschwindigkeitssignal. Das braucht allerdings nichts Exotischeres als ein zusätzlicher, viel kleinerer Motor zu sein, der auf der Welle des Hauptmotors sitzt und als Spannungsquelle arbeitet. So ein System ist leicht zu berechnen und läßt sich außerordentlich steif machen, aber für Lehrroboter lohnt es sich, den Kosten für eine doppelte Anzahl von Motoren aus dem Wege zu gehen.

PHASENVOREILUNG

Welche anderen Möglichkeiten gibt es dann? Wesentlich gesteigert werden kann die Steifigkeit, unter Beibehaltung der Stabilität, mit Hilfe der 'Phasenvoreilung'. Der Rückkopplungswiderstand wird in zwei Serienwiderstände zerlegt, und ein Kondensator wird zu einem der beiden parallelgeschaltet. Da der Strom in einem Kondensator proportional zur Änderungsrate der Spannung ist, enthält der Rückkopplungssignalstrom in einen Verstärker mit 'virtueller Erde' einen Anteil aus der Änderungsrate des Positionsfehlers – einen Geschwindigkeitsanteil. Leider vergrößert Phasenvoreilung auch die Wirkung des Rauschens auf die Signale, und wenn die Motoren aus derselben Quelle betrieben werden, die das Positionspotentiometer erregt, leisten die Stromversorgungsleitungen selbst bald Oszillationen Vorschub.



$$\text{Hochfrequenz-Verstärkung} = \frac{R_3}{R_1}; \quad \text{Niederfrequenz-Verstärkung} = \frac{R_3}{R_1 + R_2}$$

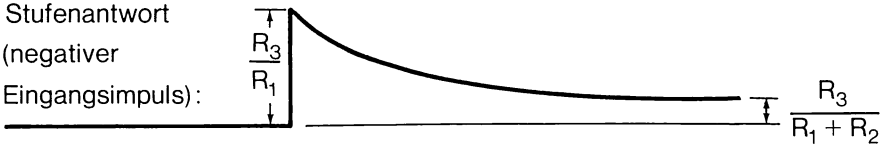


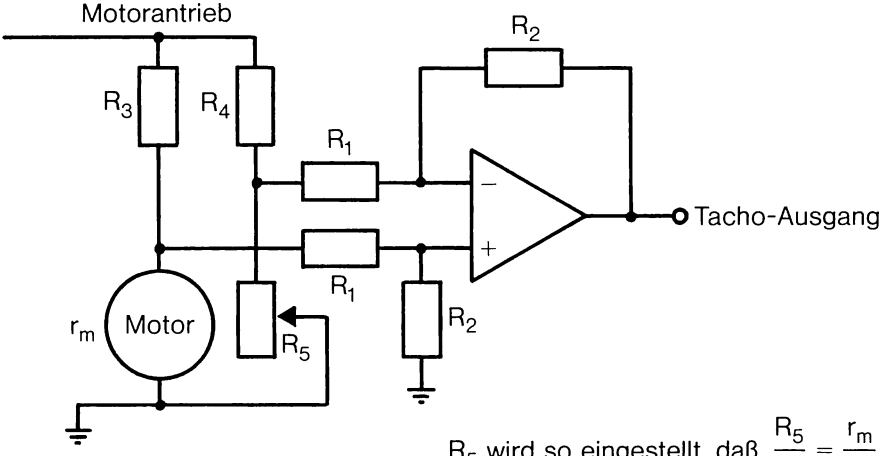
Abbildung 9.1 Phasenvoreilung

TACHOSIGNAL AUS DER GEGEN-SPANNUNG

Raffinierter ist die Technik, zwar wieder die Gegen-Spannung zu benutzen, aber sie von der durch den Treibstrom erzeugten Spannung zu trennen. Dazu ist eine Widerstandsbrücke und ein Differenzverstärker erforderlich – tatsächlich braucht man nicht mehr als drei Widerstände, um die Brücke zu vervollständigen, und einen weiteren Widerstand zusammen mit einem TL081-Chip für zwei Mark. Das kann sehr wirkungsvoll sein, wenn die Brücke genügend sorgfältig abgeglichen wird, aber das Einstellen erfordert Fingerspitzengefühl.

EIN EINFACHER SCHALTKREIS

Bevor Sie zur Tat schreiten und ein Servosystem zu bauen beginnen, verlangt es die Fairness, Sie zu warnen, daß das folgende Verfahren der analogen Ausgabe nur für den PET funktioniert. Gegen Ende des Kapitels werden Sie jedoch andere Techniken beschrieben finden.



R_5 wird so eingestellt, daß $\frac{R_5}{R_4} = \frac{r_m}{R_3}$

worin r_m den Motorwiderstand bedeutet. R_3 ist kleiner als r_m gewählt, um Leistungsverlust am Motor zu vermeiden.

$$\text{Ausgang} \mathrel{\widehat{=}} \frac{R_2}{R_1} \times \frac{R_3}{R_3 + r_m} \times \text{Gegenspannung}$$

(für 10 Ω - Motor probieren Sie $R_3 = 3{,}3\,\Omega$, $R_4 = 2{,}2\,k\Omega$
$R_1 = 10\,k\Omega$, $R_2 = 39\,k\Omega$, $R_3 = 1\,k\Omega$ -Potentiometer)

Abbildung 9.2 Tachosignal aus der Gegen-Spannung des Motors

Geben Sie sich für den Augenblick damit zufrieden, ein weniger genaues Servosystem zusammenzusetzen, das Ihnen vielleicht schon reicht. Ein Servobauelement mit Motor, Schaltgetriebe und Rückkopplungspotentiometer kann man im Modellbau-Geschäft für 40–80 Mark kaufen. Um das hier beschriebene System auszuprobieren, wurde der Motor Skyleader SRC 4BB verwendet. Ein Treiberverstärker läßt sich aus einem Einzelchip-Leistungs-Operationsverstärker 759 (RS-Nummer 303-258) herstellen, und das in Kapitel 1 beschriebene Netzgerät kann so geschaltet werden, daß es (ungefähr) +7 V und −7 V abgibt. Ihr einziges Problem ist, vom Computer ein analoges Treibersignal zu bekommen. Bevor Sie sich damit befassen, bringen Sie den Regelkreis mit einem Befehlssignal aus einem zweiten Potentiometer ans Laufen. Schalten Sie das neue Potentiometer (Wert 1 kOhm) zwischen die 0 V- und +5-V-Positionen der Anschlußleiste des User-Ports und nehmen das Befehlssignal vom mittleren (Schleifer-)Abgriff des Potentiometers. Mit der unten abgebildeten Schaltung müßte der Servomotor dem Befehlspotentiometer über den ganzen Bereich folgen. Dieser Schaltkreis erhöht die Steifigkeit mit einem anderen Trick: 'integrierendem Verhalten'. Kurzfristig ist die Ausgangsspannung ungefähr das Dreißigfache der Fehlerspannung – das scheint vielleicht reichlich, beachten Sie aber, daß der Positionsfehler für volle Aussteuerung 1/30 von 270 Grad beträgt, und neun Grad bedeuten ziemlich viel Bewegung für einen Servomotor. Mit 'integrierend' ist gemeint, daß etwa eine Sekunde lang die Treibspannung verdoppelt und dabei jeder Fehler aufgrund einer stationären Motorbelastung heruntergeschraubt wird; nach etwa zehn Sekunden kann ein Fehler von einem Grad das volle Drehmoment nach sich ziehen.

ANALOGE AUSGABE AUS CB2 – NUR PET

Nun zum analogen Ausgabesignal. Es ist schon in Kapitel 4 erwähnt worden, daß sich CB2 überreden läßt, als Schieberegister zu arbeiten. Bei passend initialisierten Kontrollregistern wird jedes in den Speicherplatz $E84A (59466) gesetzte Datenbyte wiederholt als serielles Bitmuster auf CB2 ausgegeben. Wenn jetzt $E84A den Wert 1 enthält, ist das Augabemuster bei sieben Impulsen auf Low und bei einem achten auf High – dasselbe gilt für 2, 4, 8, 16, 32, 64 und 128. Bei dem Wert 3 ist der Ausgang für sechs Impulse auf Low und für zwei auf High, allerdings ergibt 17 eine bessere Anordnung. 73 ist binär 01001001, also dreimal High und fünfmal Low, und so weiter bis 255, wofür der Ausgang immer auf High ist. Glättet man die Ausgabe mit einem Kondensator, erhält man neun Ausgangspegel: 0, 1/8, 1/4, 3/8, 1/2, 5/8, 3/4, 7/8 und 1mal die 5V-Versorgung. Neun Pegel sind kaum ausreichend; wie läßt sich eine bessere Auflösung erzielen? Eine Auflösung auf 256 Werte können wir mit unserem alten Bekannten, dem Bitratenverstärker, bekommen.

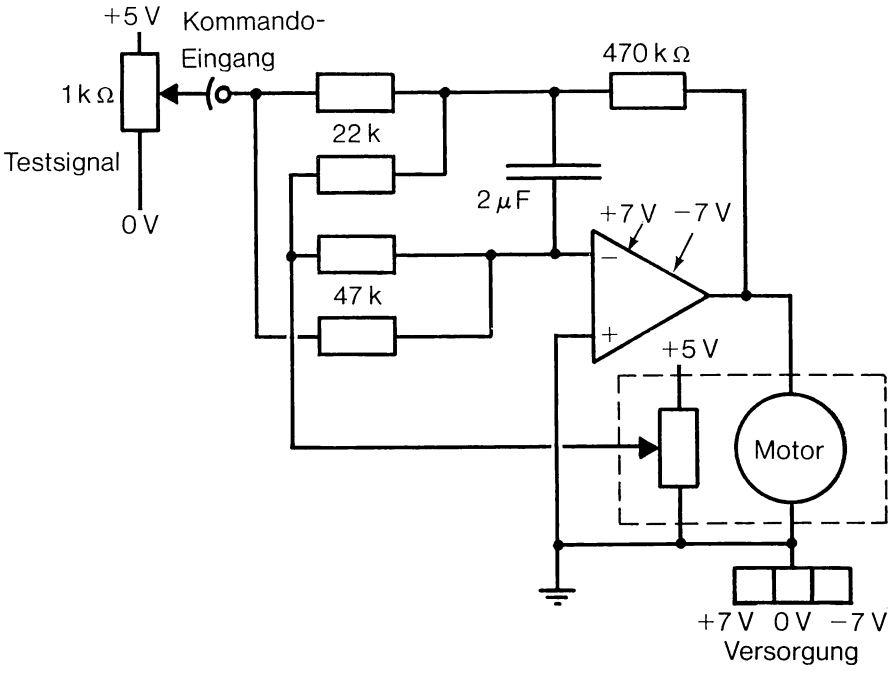


Abbildung 9.3 Servoverstärker

Der Ausgang ist jetzt durch eine ganze Zahl, sagen wir V%, festgelegt. Mit den oberen drei der acht Bits des niederwertigen Bytes von V% wählen wir eines der acht 'Muster' 0, 1, 17, 73, 85, 182, 238 und 254 – ein neuntes Muster ist 255. Die untersten fünf Bits von V% maskieren wir weg und addieren sie zu einer Variablen, sagen wir S. Immer wenn S den Wert 31 übersteigt, wird das nächsthöhere Muster im Schieberegister verwendet und 32 von S subtrahiert. Um einen glatten Mittelwert zu erhalten, muß diese Operation etwa fünfzig Mal in der Sekunde wiederholt werden. Wir wollen diese Technik zuerst mit einer einfachen BASIC-Schleife ausprobieren.

```
 10 V%=0: PO=59466
 20 DIM PA(8):FOR I=0 TO 8:READ PA(I):NEXT
 30 DATA 0,1,17,73,85,182,238,254,255
 40 POKE 56467,16: REM MAKE CB2 A SHIFT REGISTER
 50 POKE 59464,0: REM SET MAX SPEED
100 FOR V=0 TO 255:V%=V:GOSUB 1000:NEXT
110 GOTO 100: REM OUTPUT A REPEATED RAMP
```

```
1000 A=V%/32:B=V% AND 31: REM SPLIT UP V%
1010 FOR I=1 TO 20: REM TRY VARYING THE LOOP
1020 C=A:S=S+B
1030 IF S>31 THEN S=S-32:C=C+1: REM NEXT PATTERN UP
1040 POKE PO,PA(C)
1050 NEXT I:RETURN
```

Wenn Sie dieses Programm eingeben und laufen lassen und Ihr treues Meßgerät (6V-Skala) zwischen CB2 und 0 V schalten, müßten Sie die Nadel wiederholt langsam auf 5 V steigen und dann wieder fallen sehen. Beachten Sie, daß das Programm die meiste Zeit in einer Schleife verbringt, und immer wenn es eine längere Berechnung vornehmen muß, nimmt der Augang einen der neun 'stabilen' Ausgangswerte an – jeder Servomotor würde dabei um bis zu dreißig Grad zucken.

INTERRUPTGESTEUERTE AUSGABE

Um eine Schleife im Programm zu vermeiden, kann man die Interpolation (d. h. Durchschnittsbildung) mit einem Interrupt machen. Das BASIC-Programm verrichtet fröhlich sein Werk, und jedesmal, wenn ein bestimmter Zeitgeber einen Impuls sendet, wendet der Computer sich einer Interruptroutine zu.
Diese speichert zunächst den Status und alle nötigen Register weg, dann springt der Prozessor in den passenden maschinensprachlichen Abschnitt. Nach der Ausführung muß die Kontrolle an das Schwanzende der Interruptroutine zurückgegeben werden, die dann alle Register und den Status wiederherstellt und das ursprüngliche Programm fortsetzt, als ob nichts gewesen wäre. Die Interruptroutine kann den vom BASIC-Programm gespeicherten Wert von V% aufsuchen, teilen und mitteln, und bei jedem Interrupt ein neues Muster ausgeben. Die Ausgabe geschieht wie durch Zauberei, da sie bei jeder Änderung des Wertes von V% ohne sichtbare Ausgabeanweisung erfolgt.
Ein solcher Interrupt kommt im PET (und auch im C 64) sechzig Mal in der Sekunde vor. Seine normale Aufgabe ist das Abtasten der Tastatur und Aktualisieren der Zeit TI$. Nachdem alle Register gerettet sind, macht die Maschine einen indirekten Sprung in eine Adresse, die in den Plätzen $90 und $91 (144 und 145) gehalten wird. Ein neues Stück Software kann an die Interruptroutine angeschlossen werden, indem seine Adresse in diese Plätze gesetzt wird. Am Ende jeder neuen Routine muß in die ursprüngliche Adresse gesprungen werden, damit die Verwaltung der Tastatur erledigt wird.
Das Setzen der neuen Adresse erfordert ganz besondere Vorsicht: falls auch nur einer der Speicherplätze verändert ist, wenn ein Interrupt geschieht, ist ein Absturz fast unvermeidlich. Deshalb muß die Adresse durch ein weiteres Programm in Maschinensprache gesetzt werden, das Interrupts zeitweilig unterdrückt. (Wenn Sie

sehen wollen, was alles passieren kann, versichern Sie sich zunächst, daß jedes nützliche Programm gerettet ist, und poken dann 145 mit irgendeiner Zahl, die Ihnen gefällt.)
Das nächste Problem ist, Platz für den Maschinencode zu schaffen. Wenn der Code die Form eines Satzes von Datenanweisungen hat, die durch einen Teil des BASIC-Programms gesetzt werden, ist der zweite Kassettenpuffer sehr gut geeignet (allerdings kann er da von Diskettenbefehlen zertreten werden). Wenn der Code häufiger gebraucht wird, ist die Möglichkeit vorzuziehen, sowohl die Teile in BASIC als auch die in Maschinensprache in sofort verwendbarer Form wegzuspeichern. Ist die erste Zeile des Programms:

```
10 POKE 41,5: RUN
```

wird der BASIC-Anfang auf $0501 gehoben, und dort kann das Hauptprogramm ruhen. Die direkte Anweisung

```
POKE 41,5: POKE 5*256,0: NEW
```

setzt die Zeiger so, daß Sie den Rest des Programms eintippen können, und läßt eine Lücke von über 200 Bytes, in die Sie Ihren Maschinencode setzen können. Die erste Zeile des oberen BASIC-Programms muß lauten:

```
10 V%=0
```

damit V% die erste Variable des Programms ist. Der Maschinencode kann sie dann über die Zeiger in $2A, $2B (42,43) finden. Als nächstes kommt ein Aufruf SYS 1056 an die maschinensprachliche Routine, der die Interrupt-Links setzt. Dadurch wird die Adresse des Sprungs, der den Interrupt beendet, mit der schon in $90, $91 gehaltenen Adresse ausgetauscht. (Dieses Verfahren funktioniert bei den PETs der Serien 30xx, 40xx und 50xx.) Durch Aufruf derselben SYS-Adresse findet dann der Austausch statt, der den Normalzustand wiederherstellt.

```
 20 SYS 1056
 30 POKE 59467,16:POKE59464,0
100 INPUT "NEW VALUE (0 TO 255)";V%
110 GOTO 100
```

Bis auf den maschinensprachlichen Teil ist das schon alles! Statt Werte einzugeben, können Sie wie vorhin eine Rampe ausgeben – die POKE-Anweisung ist allerdings nicht nötig, und das Unterprogramm 1000 erhält einfach die Gestalt einer kurzen Verzögerungsschleife:

```
1000 FOR I=1 TO 100:NEXT:RETURN
```

Andernfalls führen Sie irgendeine Berechnung durch, zum Beispiel:

```
100 FOR I=1 TO 10000
110 V%=128+127*SIN(I/100)
120 NEXT: GOTO 100
```

und das Ergebnis wird ganz allein dadurch ausgegeben, daß es in V% gespeichert wird.
Seien Sie auf der Hut! Läuft das Programm ein zweites Mal, werden die Links wieder ausgetauscht, und die Routine bricht ab. Beim dritten Mal werden sie wieder gesetzt, und so weiter. Überzeugen Sie sich, daß sie wieder normal sind, bevor Sie das Programm addieren, insbesondere bevor Sie ein anderes Programm laden. Interrupts können geradewegs zum Absturz führen.
Natürlich fehlt noch immer etwas: wir haben den Maschinencode noch nicht geladen. Das läßt sich am einfachsten mit dem im PET eingebauten Monitor machen. Den Monitor bekommen Sie, wenn Sie tippen:

```
SYS 1024
```

Die Maschine antwortet mit B* und zwei Zeilen von Zeichen, die den Status der Maschine anzeigen. Geben Sie jetzt die Stelle des Speichers auf den Bildschirm, in die Sie den Code setzen wollen, indem Sie tippen:

```
.M 0420 0460
```

Das Display sieht dann ähnlich wie das unten abgedruckte aus, jedoch mit anderen Zahlen rechts von den Adressen. Fahren Sie jetzt mit dem Cursor in jede Zeile, tippen Sie die hier abgedruckten Werte statt der bei Ihnen angezeigten ein, und vergessen Sie nicht, nach jeder geänderten Zeile RETURN anzuschlagen.

```
.M  0420  0460
.:  0420  78  AD  5B  04  A6  90  85  90
.:  0428  8E  5B  04  AD  5C  04  A6  91
.:  0430  85  91  8E  5C  04  58  60  EA
.:  0438  A0  03  B1  2A  4A  4A  4A  4A
.:  0440  4A  AA  B1  2A  29  1F  18  6D
.:  0448  5D  04  C9  20  30  03  E8  29
.:  0450  1F  8D  5D  04  BD  5E  04  8D
.:  0458  4A  E8  4C  38  04  00  00  01
.:  0460  11  49  55  B6  EE  FE  FF  FF
```

Wenn Sie fertig sind, tippen Sie wieder

```
.M 0420 0460
```

und prüfen, ob die Ausgabe mit der obigen übereinstimmt. Zum Abschluß geben Sie

```
.X
```

ein, um zu BASIC zurückzukehren.
Um das Programm einschließlich Maschinencode sicherzustellen, tippen Sie

```
POKE 41,4
```

Das setzt den Zeiger auf den BASIC-Beginn auf normal, und Sie können das Programm wie üblich auf Kassette oder Diskette speichern.
Wenn Sie den Maschinencode auch nicht zu verstehen brauchen, um ihn einzugeben, möchten Sie doch sicher wissen, woraus er besteht. Die PLANT-Routine fängt bei $0420 an – deshalb die Anweisung SYS 1056.

```
0420    PLANT   SEI             ;INHIBIT INTERRUPTS
0421            LDA     J+1     ;SWAP JUMP POINTERS
0424            LDX     $90
0426            STA     $90
0428            STX     J+1
042B            LDA     J+2
042E            LDX     $91
0430            STA     $91
0432            STX     J+2
0435            CLI             ;PERMIT INTERRUPTS
0436            RTS             ;END OF PLANT,RETURN

0438    DTOA    LDY     #3      ;READY FOR V%
043A            LDA     ($2A),Y ;LOAD VALUE OF V%
043C            LSR     A
043D            LSR     A
043E            LSR     A
043F            LSR     A
0440            LSR     A       ;DIVIDE BY 32
0441            TAX             ;PARK IT
0442            LDA     ($2A),Y ;GET V% AGAIN
```

```
0444            AND     #$1F    ;MASK WITH 31
0446            CLC             ;CLEAR CARRY
0447            ADC     S       ;ADD S
044A            CMP     #$20    ;OVER 31?
044C            BMI     NOMORE  ;NO
044E            INX             ;INCREASE OUTPUT ONE
                                NOTCH
044F            AND     #$1F    ;CHOP DOWN S
0451    NOMORE  STA     S       ;SAVE NEW S
0454            LDA     PA,X    ;LOAD X'TH PATTERN
0457            STA     #$E84A  ;PLANT IN SHIFT
                                REGISTER
045A    J       JMP     DTOA    ;NOTE THIS IS NOBBLED
                                BY
                                ;'PLANT' TO POINT TO
                                THE REST
                                ;OF THE NORMAL
                                ROUTINE.
045D    S       BYTE    0
045E    PA      BYTE    0,1,17,73,85,182,238,254,255
```

FERNSTEUERUNG VON MOTOREN

Wenn Sie sich durch die letzten Abschnitte gekämpft haben, finden Sie es vielleicht unfair, daß ich erst jetzt mit einer anderen Möglichkeit herausrücke, einen Servomotor zu betreiben. Diese Methode wird von Alan Dibley bevorzugt, der sich enorm wirkungsvoll beim Bau von Mikromäusen einsetzt. Zudem freut es sicher die C 64-Besitzer, daß sie sie auch verwenden können – obwohl ihnen der Maschinensprache-Monitor fehlt, mit dem sich der PET so leicht laden läßt.

Zur Fernsteuerung können Sie sich einen Servomotor, komplett mit Treiberverstärker, aus dem Regal kaufen. Um Rückkopplung, Stabilität oder Tachosignale brauchen Sie sich keine Sorgen zu machen. Der Motor hat einen dreiadrigen Anschluß, zwei für Energieversorgung und die dritte zur Befehlseingabe.

Befehle an den Motor haben die Form von Impulsen im Abstand von etwa 20 Millisekunden. Die Länge jedes Impulses bestimmt die befohlene Position und variiert zwischen einer Millisekunde für volle Aussteuerung in einer Richtung und zwei Millisekunden für volle Aussteuerung in der anderen Richtung. Wieder können wir zur wiederholten Ausgabe an den Motor einen Interrupt einsetzen, und wir können ebenfalls wieder den Trick anwenden, mit der Interruptroutine durch Spei-

chern des Wertes in V% zu kommunizieren. Da bei CB2 die Ausgabe in Form einer Impulsfolge erfolgt, kann der Ausgang an den Steuereingang des Motors angeschlossen werden.

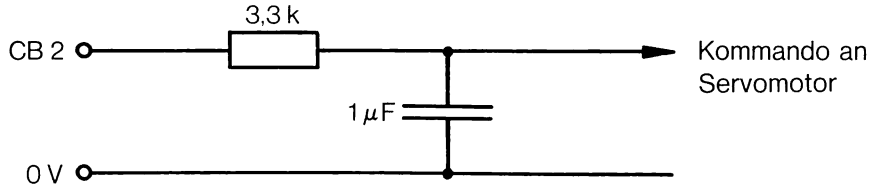


Abbildung 9.4 Glätten für analoge Ausgabe

PROGRAMM ZUR FERNSTEUERUNG VON MOTOREN

Um das Ladeproblem zu umgehen, kehren wir zur Technik aus Kapitel 3 zurück, bei der der Maschinencode durch Datenanweisungen innerhalb des BASIC-Programms definiert ist. Der Maschinencode wird durch den Loader und Hexadezimal-Dezimal-Wandler bei 10000 in den Kassettenpuffer geschrieben.

```
   10 V%=0:GOTO 1000

10000 MC=3*16 ↑2+9*16 : I=MC : REM $0390
10010 READ A$:IF LEN(A$)<>2 THEN 100 :REM DONE IF XXXX
10020 GOSUB 10100
10030 POKE I,A: PRINT I,A$,A
10040 I=I+1: GOTO 10010

10100 A=ASC(A$)-48+7*(A$>":"): REM CONVERT FROM HEX
10110 B$=MID$(A$,2)
10120 A=16*A+ASC(B$)-48+7*(B$>":")
10130 RETURN
```

Wir können die Sache weiter vereinfachen, wenn wir statt CB2 eines der User-Port-Bits nehmen. Wie bisher gibt es mehrere Unterschiede zwischen PET und C 64, deshalb geben wir beide Sätze von Datenanweisungen an:

```
10200 DATA 78          :REM PLANT  SEI          *** PET
10210 DATA AD, CO, 03  :REM        LDA J+1
10220 DATA A6, 90      :REM        LDX $90
10230 DATA 85, 90      :REM        STA $90
```

```
10240 DATA 8E, C0, 03 :REM        STX J+1
10250 DATA AD, C1, 03 :REM        LDA J+2
10260 DATA A6, 91     :REM        LDX $91
10270 DATA 85, 91     :REM        STA $91
10280 DATA 8E, C1, 03 :REM        STX J+2
10290 DATA 58         :REM        CLI
10300 DATA 60         :REM        RTS
10310 DATA A0, 03     :REM DTOA   LDY #3       V%
10320 DATA B1, 2A     :REM        LDA (VARS),Y
10330 DATA AA         :REM        TAX          -PUT IN X
10340 DATA E8         :REM        INX          -IN CASE 0
10350 DATA A9, 01     :REM        LDA#1        -SET BIT 0
10360 DATA 8D, 4F, E8 :REM        STA PORT       HIGH
10370 DATA CA         :REM LOOP   DEX          -COUNT X
10380 DATA D0, FD     :REM        BNE LOOP
10390 DATA A2, 7F     :REM        LDX #127     *VARY TO
                                                 GIVE 1 MS
10400 DATA CA         :REM LP2    DEX
10410 DATA D0, FD     :REM        BNE LP2
10420 DATA A9, 00     :REM        LDA #0
10430 DATA 8D, 4F, E8 :REM        STAPORT      -CLEAR
10440 DATA 4C, A7, 03 :REM        JMPDTOA      -GETS
                                                 NOBBLED
10450 DATA XXXXX      :REM        END            BY PLANT
```

Beachten Sie, daß der in Zeile 10390 geladene Wert eventuell geändert werden muß – der Motor erfordert möglicherweise einen anderen Bereich von Impulslängen.
Die Version des Codes für den C 64 ist:

```
10200 DATA 78         :REM PLANT  SEI          *** CMB 64
10210 DATA AD, 14, 03 :REM        LDA $0314    IRQ
                                                 VECTOR
10220 DATA AE, C4, 03 :REM        LDX J+1
10230 DATA 8D, C4, 03 :REM        STA J+1
10240 DATA 8E, 14, 03 :REM        STX $0314
10250 DATA AD, 15, 03 :REM        LDA $0315    IRQ HI
10260 DATA AE, C5, 03 :REM        LDX J+2
10270 DATA 8D, C5, 03 :REM        STA J+2
10280 DATA 8E, 15, 03 :REM        STX $0315
10290 DATA 58         :REM        CLI
10300 DATA 60         :REM        RTS
```

```
10310 DATA A0, 03       :REM DTOA   LDY #3          GET V%
10320 DATA B1, 2D       :REM        LDA (VARS),Y
10330 DATA AA           :REM        TAX             -PUT IN X
10340 DATA E8           :REM        INX             -IN CASE 0
10350 DATA A9, 01       :REM        LDA #1          -SET BIT 0
10360 DATA 8D, 01, DD   :REM        STA PORT        HIGH
10370 DATA CA           :REM LOOP   DEX             -COUNT X
10380 DATA D0, FD       :REM        BNE LOOP
10390 DATA A2, 7F       :REM        LDX #127        *VARY TO
                                                    GIVE 1 MS
10400 DATA CA           :REM LP2    DEX
10410 DATA D0, FD       :REM        BNE LP2
10420 DATA A9, 00       :REM        LDA #0
10430 DATA 8D, 01, DD   :REM        STA PORT        -CLEAR
10440 DATA 4C, AB, 03   :REM J      JMP DTOA        -GETS
                                                    NOBBLED
10450 DATA XXXXX        :REM        END             BY PLANT
```

Jetzt müssen wir noch Bit 0 des Ports als Ausgang schalten und den Interrupt wecken, und wir sind startbereit:

```
100 POKE 56471,1    :REM IF PET
```

oder

```
100 POKE 56579,1    :REM IF 64

110 SYS MC          :REM WAKE UP
```

Jetzt müßte der Servomotor auf jeden in die Variable V% abgelegten Wert reagieren, und Sie können das folgende Programm als einfachen Test verwenden:

```
1000 FOR I=0 TO 255
1010 V%=I:REM YES, THATS'S ALL IT TAKES
1020 FOR J=1 TO 100:NEXT: REM BRIEF DELAY
1030 NEXT I
1040 GOTO 1000:REM OUTPUT ANOTHER RAMP
```

Wenn Sie BREAK drücken, nachdem das Programm gelaufen ist, gibt der Computer weiterhin bei Interrupt V% aus. Um auf normal rückzusetzen, tippen Sie

```
SYS MC
```

als Direktbefehl.

Analoge Ausgabe für PET – BASIC

```
10 V%=0:PO=59466
20 DIM PA(8):FORI=0TO8:READ PA(I):NEXT
30 DATA 0,1,17,73,85,182,238,254,255
40 POKE59467,16
50 POKE59464,100
100 FOR V=0 TO 255:V%=V:GOSUB1000:NEXT
110 GOTO100
1000 A=V%/32:B=V%AND31
1010 FORI=1TO20
1020 C=A:S=S+B
1030 IFS>31THENS=S-32:C=C+1
1040 POKEPO,PA(C)
1050 NEXT:RETURN
```

Motor-Impulslängen-Ausgabe – PET

```
10 V%=0:GOTO 10000
100 POKE59471,1 :REM **** PET
110 SYS MC
120 INPUT"POSITION DEMAND (0 TO 255)";V%
130 GOTO 120
10000 MC=3*16^2+9*16:I=MC
10010 READ A$:IF LEN(A$)<>2THEN100
10020 GOSUB 10100
10030 POKE I,A:PRINT I,A$,A
10040 I=I+1:GOTO 10010
10100 A=ASC(A$)-48+7*(A$>":")
10110 B$=MID$(A$,2)
10120 A=16*A+ASC(B$)-48+7*(B$>":")
10130 RETURN
10200 DATA 78, AD,C0,03, A6,90, 85,90
10210 DATA 8E,C0,03, AD,C1,03, A6,91
10220 DATA 85,91, 8E,C1,03, 58, 60
10310 DATA A0,03, B1,2A, AA, E8, A9,01
10320 DATA 8D,4F,E8, CA, D0,FD, A2,7F
10330 DATA CA, D0,FD, A9,00, 8D,4F,E8
10340 DATA 4C,A7,03
10350 DATA XXXX
```

Motor-Impulslängen-Ausgabe – C 64

```
10 V%=0:GOTO 10000
100 POKE56579,1 :REM **** CBM 64
110 SYS MC
```

```
120 INPUT"POSITION DEMAND (0 TO 255)";V%
130 GOTO 120
10000 MC=3*16^2+9*16:I=MC
10010 READ A$:IF LEN(A$)<>2THEN100
10020 GOSUB 10100
10030 POKE I,A:PRINT I,A$,A
10040 I=I+1:GOTO 10010
10100 A=ASC(A$)-48+7*(A$>":")
10110 B$=MID$(A$,2)
10120 A=16*A+ASC(B$)-48+7*(B$>":")
10130 RETURN
10200 DATA 78, AD,14,03, AE,C4,03
10210 DATA 8D,C4,03, 8E,14,03, AD,15,03
10220 DATA AE,C5,03, 8D,C5,03, 8E,15,03
10230 DATA 58, 60
10310 DATA A0,03, B1,2D, AA, E8, A9,01
10320 DATA 8D,01,DD, CA, D0,FD, A2,7F
10330 DATA CA, D0,FD, A9,00, 8D,01,DD
10340 DATA 4C,AB,03
10350 DATA XXXX
```

KAPITEL 10

EIN EINFACHES AUGE FÜR ROBOTER

Ein Großteil der in diesem Kapitel beschriebenen Arbeit gehört zur Forschung von Ali Hosseinmardi. Die Veröffentlichung hier greift seinem Anspruch auf Originalität nicht vor. Das einfachste System zur optischen Wahrnehmung ist von Upperdata Ltd. aufgegriffen worden und wird zum unglaublichen Preis von 200 Mark vertrieben. Wenn Sie fleißig sind, können Sie sich ein eigenes 'Auge' mit den hier gegebenen Hinweisen ganz von vorne bauen, aber vielleicht wählen Sie lieber den bequemen Ausweg.

PROVOKATIVE INSTRUMENTIERUNG

Es gibt viele und unterschiedliche Techniken, analoge Signale mit dem Computer einzufangen, und der Computer beginnt rasch, die Meßaufzeichnungsgeräte in Prozeßanlagen zu ersetzen. Die traditionelle Instrumentierung hat sich mit der Entwicklung von Sensoren beschäftigt, die auf Temperaturen, Druck, Niveau, Säuregehalt usw. reagieren, und mit deren Einbau in Systeme, die eine stetige Spannung oder einen stetigen Strom proportional zur gemessenen Größe abgeben. Der Computer wird gegenwärtig häufig dazu verwandt, elektrische Signale, die sonst Registrierstreifen oder Zeiger betrieben hätten, unter seine Nase zu setzen, damit er davon kostet, wenn es ihm paßt. Doch der Computer ist zu Besserem fähig. Der Computer kann Experimente zur Gewinnung der notwendigen Daten durchführen, indem er das System zu einer Reaktion provoziert – daher der Ausdruck 'provokative Instrumentierung', den ich geprägt habe. Das optische Wahrnehmungssystem ist ein wichtiges Beispiel. Ein Sehender analysiert die Signale, die sein Auge gerade erreichen. Ein Blinder muß mit seinem Stock umhertappen und aus den Reaktionen ein Bild seiner Umgebung aufbauen. Dieses Bild ist vielleicht nicht so detailliert wie das eines Sehenden, aber es ist besser als gar kein Bild.
Das System Cyclops zur optischen Wahrnehmung stattet den Roboter mit einer einzigen fokussierten Fotozelle aus. Mit diesem einen Sehpunkt tastet der Roboter selbst seine Umgebung ab und kann daraus ein Bild aufbauen. Die langsame Methode besteht darin, den Roboter ein Raster abtasten zu lassen und ein normales Bild dadurch zu erzeugen, daß die Grautöne auf den Bildschirm geschrieben werden. Interessanter ist die Technik, den Roboter dem Rand irgendeines kontrastierenden Objekts folgen zu lassen, so daß das Bild auf seinen Umriß hin analysiert werden kann, noch bevor es vollständig eingegeben ist. Mit ein oder zwei subtilen Techniken kann man Rändern selbst dann folgen, wenn sie grau in grau sind, und selbst wenn durch ungleichmäßige Beleuchtung die Änderung der Lichtstärke über das Sehfeld hinweg stärker variiert als beim Objekt selbst.

KONSTRUKTION DES OPTISCHEN SYSTEMS

Wie Mrs. Beeton beinahe gesagt hätte: 'First catch your robot.' Sie müssen Linse und Fotozelle irgendwie bewegen, um das Sehfeld abzutasten. Sie können das Auge am Unterarm eines fertigen Roboters, etwa eines Armdroiden, befestigen, aber Sie können auch ziemlich leicht ein eigenes Ablenkungssystem mit zwei Kanälen herstellen; es könnte am Ende ganz wie der selbstgebaute Joystick aussehen, nur daß Schrittmotoren die Stelle der Potentiometer einnehmen und das Auge den Joystick selbst ersetzt. Leider macht ein Schrittmotor ziemlich große Schritte, und vierundzwanzig Halbschritte überstreichen volle neunzig Grad. Deshalb müssen Sie einen Weg finden, die Bewegung zu untersetzen, entweder mit einem konventionellen Getriebe oder mit einem einfachen System aus Riemenscheiben und Riemen. Wenn Sie die Übersetzung übertreiben, können Sie immer noch die Software so ändern, daß sie eine Anzahl Schrittbewegungen zwischen die vorgegebenen Punkte legt.

Das Auge ist nichts weiter als eine Linse, eine Röhre, eine Fotozelle (OP500 eignet sich gut) und ein Verbindungskabel. Für Maschinen wie den PET ist es in der kommerziellen Version mit einer zusätzlichen Schaltungsanordnung zur Analog-Digital-Umwandlung (ganz ähnlich dem in Kapitel 5 beschriebenen Verfahren) versehen, aber für Geräte mit eingebautem Wandler ist das eigentlich nicht nötig. Bei starkem Licht kann ein OP500 direkt an einen der Drehregler-Eingänge angeschlossen werden, und bei geringerer Lichtstärke kann man einen einzigen Transistor und zwei Widerstände dazuschalten.

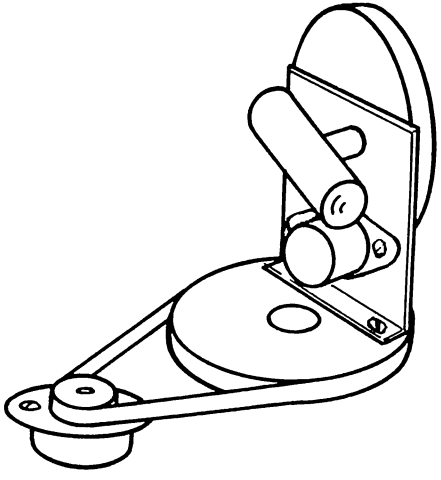


Abbildung 10.1 Aufhängung des Auges

Die Linse sollte 10 Dioptrien oder etwas mehr haben – also eine Brennweite von 10 cm oder weniger. Ein einfaches Vergrößerungsglas aus Plastik wäre ideal. Wenn Sie zu ernsthaft sind, um alles in eine Smarties-Dose zu montieren, können Sie mit Klebeband eine Papierrolle anfertigen. Sobald die Fotozelle im Brennpunkt befestigt und die Röhre am Roboter oder dem Abtastgerät angebracht ist, ist bis auf die Software fast alles getan. Um die Anschlüsse ans Auge auszutesten und sich zu überzeugen, daß der Widerstand der Lichtstärke angemessen ist, können Sie den folgenden Einzeiler eingeben und laufen lassen:

```
10 PRINT PEEK(54297):GOTO10:REM CBM 64
```

Beim PET müßten Sie Zeile 10 und die Zeilen ab 10000 aus Kapitel 5 eingeben und die Werte mit der nächsten Zeile testen:

```
100 CH%=1:SYS MC:PRINT V%:FOR I=1 TO 500:NEXT: GOTO 100
```

Wenn aber der User-Port zum Ablesen der Fotozelle benutzt wird, wo kann man dann den Roboter anschließen? Tatsächlich sind sehende Roboter für PET-Besitzer recht schwierig zu handhaben, nicht weil das Wahrnehmen des Bildes problematisch wäre, sondern weil die Grafik zum Ausgeben auf den Bildschirm fehlt. Wenn Sie sich mit einer schwarzweißen Szenerie abfinden wollen, können Sie einfach die Fotozelle OP500 an einen Komparator LM339 anschließen, den Schwellenwert mit einem Potentiometer setzen und die Lichtstärke mit einem einzigen PEEK ablesen. Um den User-Port freizuhalten, verbinden Sie das Auge mit dem Kassetten-Anschluß und stehlen die +5 V für den LM339 aus Pin B/2. Der Ausgang wird dann mit Pin F/6 verbunden, der gewöhnlich den Kassettenschalter auf Pin PA4 des PIA 6520 liest. Dieser Port erscheint in Adresse $E810, also kann die Eingabe mit PEEK(59408) AND 16 gelesen werden. Genau diese Technik wird in **Abbildung 10.2** beschrieben.

ZWEI STRATEGIEN ZUR OPTISCHEN WAHRNEHMUNG

Eine schnelle und wirksame Methode, ein Bild auf den Bildschirm zu bringen, besteht im Abtasten eines Rasters. Das Auge wird hin- und herbewegt und senkt sich dabei stetig nach unten, und dabei wird in jeden Punkt des Bildschirms ein Fleck mit einer dem Fotozellensignal entsprechenden Intensität geschrieben. Im Prinzip läßt sich eine Abtastzeile von links nach rechts und die nächste von rechts nach links führen. In der Realität führt der tote Gang dazu, daß abwechselnde Zeilen ein wenig gegeneinander verschoben sind; deshalb ist ein konventionelles Raster vorzuziehen. Die Motorbewegung wird jetzt durch zwei geschachtelte FOR- . . . NEXT-Schleifen gesteuert. Die Lichtstärke wird in jedem Punkt abgelesen und

durch einen Skalenfaktor dividiert, damit das Ergebnis im Bereich von, sagen wir, 0 bis 7 liegt. Mit dieser Zahl kann ein Zeichen aus einem String immer dichter werdender Symbole gewählt werden, das dann in die entsprechende Position auf dem Bildschirm geschrieben wird. Alternativ kann man mit dem Wert auch die Vordergrundfarbe wählen und ein Zeichen schreiben, das aus einem vollen Block Farbe besteht. Bei einer schwarzweißen Anzeige ergibt die richtige Farbwahl eine Skala von Grautönen – um die Farben zu bestimmen, brauchen Sie ein Feld als Umwandlungstabelle, aber das läßt sich leicht machen.

Ein randverfolgendes Programm ist wesentlich interessanter. Um Ränder zu erkennen, die hellgrau auf dunkelgrau sind, können Sie die volle Auflösung der Fotozelle ausnutzen. Nehmen wir zunächst an, das Bild ist ausschließlich schwarz und weiß. Wir möchten die Grenze eines weißen Gebietes entlanglaufen und dabei schwarz zur Linken und weiß zur Rechten lassen. Da wir nur jeweils einen Punkt ansehen können, müssen wir offensichtlich ständig im Zickzack über die Grenze laufen, damit wir sie nicht aus dem Auge verlieren. Ali und ich haben festgestellt, daß das folgende Verfahren effektiv ist.

Stellen Sie sich vor, Sie haben einen Kompaß in der Hand, der auf ein Stück Papier gezeichnet ist, und es sind Nord und Süd markiert. Sie stehen auf einem Feld

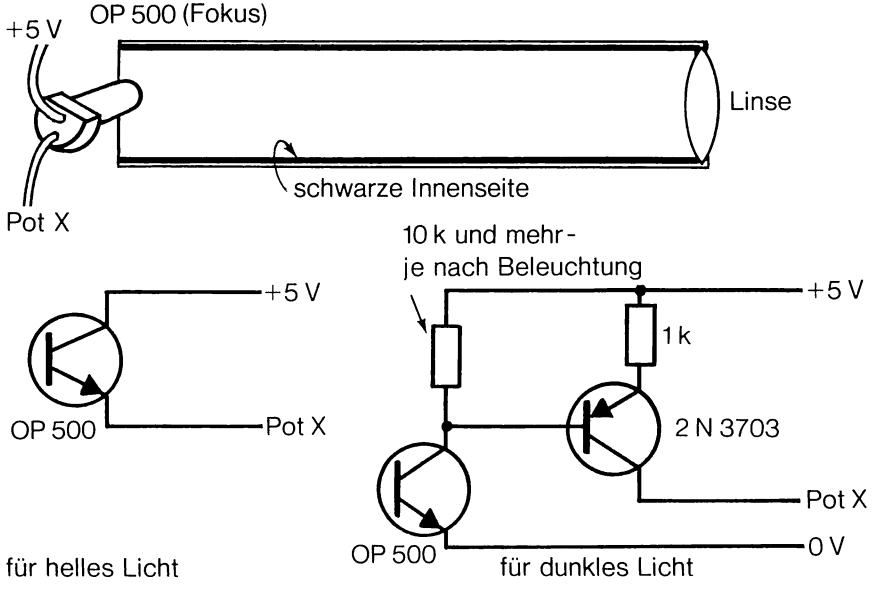


Abbildung 10.2 Ein billiges 'Auge'

quadratischer Fliesen. Die Fliese, auf der Sie stehen, ist weiß, und Sie schauen nach Norden (wie es der Kompaß anzeigt), eine Reihe weiterer weißer Fliesen entlang. Auf das Papier sind folgende Regeln geschrieben:

1. Stehen Sie auf einem weißen Quadrat, drehen Sie den Kompaß 45 Grad nach West. Machen Sie einen Schritt 'Kompaß Nord' in die Mitte des nächsten Quadrats.

2. Stehen Sie auf einem schwarzen Quadrat, drehen Sie den Kompaß 45 Grad nach Ost. Machen Sie einen Schritt 'Kompaß Ost' in die Mitte des nächsten Quadrats.

Wenn Sie diesen Regeln folgen, drehen Sie sich halb nach links und gehen vorwärts – diagonal. Ist dieses Quadrat schwarz, drehen Sie sich halb nach rechts (in Ihre ursprüngliche Richtung) und machen einen Schritt seitlich nach rechts. Sie sind dann genau ein Quadrat vor Ihrer ursprünglichen Position. Falls links von Ihrer Reihe weißer Fliesen eine Reihe schwarzer Fliesen liegt, gehen Sie im Zickzack vorwärts und kommen dabei mit zwei Schritten um ein Quadrat voran. Wenn kein weißes Quadrat mehr kommt, drehen Sie sich immer weiter halbrechts, bis Sie wieder auf ein weißes Feld kommen, und so bewegen Sie sich entlang jeder Biegung der Grenzlinie. Das ist schwer zu erklären, aber leicht zu programmieren. Wie kann die Strategie nun einen Rand in wechselnden Grautönen verfolgen? Das Verfahren besteht darin, einen lokalen Schwellenwert TH zu setzen, der in der Mitte von 'lokal schwarz' und 'lokal weiß' liegt. Ist der Punkt, den Sie gerade ansehen, schwärzer als lokal schwarz, wird lokal schwarz sofort in diesen Wert abgeändert. Entsprechend wird lokal weiß sofort geändert, wenn ein Punkt gefunden worden ist, der weißer als weiß ist. Wenn die Lichtstärke irgendwo zwischen diesen Werten liegt, dürfen beide Werte etwas nach innen rücken, etwa durch folgende Zuordnung:

```
LB = LB + (TH-LB)/20:REM THRESHOLD-LOCALBLACK
```

Auf diese Weise wird die Entscheidung über schwarz und weiß um einen Schwellenwert herum gefällt, der sich den Beleuchtungsveränderungen über das Bild hinweg anpassen kann. Alis experimenteller Aufbau kann den Konturen eines schwarzen Buchstaben folgen, selbst wenn ein weiteres Blatt Papier darüberliegt – aber der Roboter neigt dazu, nach einer Falte im Papier wegzulaufen.
Aufgrund der bisher gegebenen Hinweise können Sie sicher ein eigenes randverfolgendes Programm schreiben, nachdem Sie das unten angegebene Rasterprogramm ausprobiert haben. Es läßt sich schwer sagen, wann so ein Programm vollständig ist, denn wenn ein Satz Datenpunkte eingefangen ist, der den Rand

eines Objektes repräsentiert, müssen die Punkte geglättet werden, um das 'Saumstich'-Muster zu beseitigen. Sie können dann verarbeitet werden, um unerhebliche Punkte zu entfernen – Geraden können hinreichend gut durch einen Punkt an jedem Ende dargestellt werden, und mit etwas Scharfsinn läßt sich der Umriß eines K von mehreren hundert Punkten auf vierzehn reduzieren. (Irgendwie scheint das Programm mit elf nicht zurechtzukommen.)

DAS RASTERPROGRAMM

Bei jeder Strategie ist ein Verfahren zum Bewegen des Roboters erforderlich. Wenn Sie einen vollständigen Armdroiden oder etwas Ähnliches verwenden, kann das Unterprogramm 8000 aus Kapitel 8 über eine Routine bei 6000 aufgerufen werden, die X und Y zu TA(0) und TA(1) addiert, wobei X und Y die gewünschten Verschiebungen sind. Zusätzlich zu den Zeilen 10000 bis 10900 und 8000 bis 8190 aus Kapitel 8 brauchen Sie:

```
6000 TA(0)=TA(0)+X
6010 TA(1)=TA(1)+Y
6020 REM: THE CHANNEL NUMBERS 0,1 MAY NEED CHANGING-
     CHECK
6030 GOSUB 8000
6040 RETURN
```

Wenn Sie statt eines fertigen Roboters zwei eigene Schrittmotoren verwenden, können Sie das Unterprogramm 8000 aus Kapitel 7 fast unverändert übernehmen (Sie müssen die Zeilen ab 8000 eingeben). Um den Rest dieses Kapitels so unkompliziert wie möglich zu halten, fügen Sie den folgenden Trick bei 8000 ein (später werden Sie vielleicht das Programm leicht abändern wollen, um ihn zu vermeiden):

```
6000 LM=X:RM=Y:GOSUB 8000:RETURN
```

Da das Programm sonst zwei verschiedene Dachgeschosse hätte, wollen wir den Rest der Verwaltung im Erdgeschoß halten, so daß das Programm (für den C 64) so beginnt:

```
  10 GOTO 10000
 100 LS=16/256:EY=54297:REM LIGHT SCALE, EYE ADDRESS
 110 CC=55296:DIM CC(7):REM COLOUR MAP, CODES
 120 FOR I=0 TO 7:READ CC(I):NEXT
```

```
130 DATA 0,2,6,14,5,13,7,1
140 REM BLACK, RED, BLUE, LT BLUE, GREEN, LT GREEN, YEL-
    LOW, WHITE
```

Für den PET, ohne Schattierung, vereinfacht sich das zu:

```
 10 GOTO 10000: REM PET ****
100 EY=59408:CC=32768:REM EYE, SCREEN ADDRESS
```

und wird für beide Geräte wie folgt fortgesetzt:

```
1000 PRINT CHR$(147);CHR$(18);:REM CLEAR SCREEN,
     REVERSE
1010 FOR RO=0 TO 24:REM ROW
1020 FOR CM=0 TO 39:REM COLUMN
1030 POKE CC+40*RO+CM,CC(PEEK(EY)*LS):REM SET COLOUR
     *** 64 ONLY
1040 PRINT"  ";:REM PRINT (REVERSED)SPACE *** 64 ONLY
1050 X=1:Y=0:GOSUB 6000:REM CAN CHANGE X VALUE TO
     SCALE
1060 NEXT CM:REM NEXT COLUMN
1070 X=-20*X:Y= -1:GOSUB 6000:REM SCALE Y VALUE TOO
1080 NEXT RO
1090 X=0:Y= - 25*Y:GOSUB 6000:REM MOVE BACK TO START
1100 GET A$:IF A$=""THEN 1100
1110 GOTO 1000
```

Für den PET wird Zeile 1040 weggelassen und Zeile 1030 abgeändert:

```
1030 POKE CC+40*RO+CM,32+8*(PEEK(EY) AND 16)
```

Dadurch wird ein Leerzeichen gesetzt, wenn die Eingabe Low ist, und ein inverses Leerzeichen, wenn sie High ist.
Das müßte reichen, um ein Bild auf den Bildschirm zu bekommen. Wenn Sie ein Fernsehgerät als Monitor benutzen, brauchen Sie nur die Farbe herunterzudrehen, um eine Skala von Grautönen zu erhalten. Falls Sie ein Schwarzweißgerät haben, brauchen Sie keinen Handschlag zu tun!
Natürlich wird aufgrund jenes berühmten Gesetzes die Fotozelle zu Beginn in die falsche Richtung zeigen. Deshalb möchten Sie vielleicht ergänzen:

```
200 PRINT CHR$(147);"SET EYE TO TOP LEFT OF SCENE"
210 INPUT "MOVE RIGHT HOW MANY?";X
215 INPUT "MOVE DOWN HOW MANY?";Y
220 GOSUB 6000
230 INPUT "OK?";A$:IF LEFT$(A$,1)<>"Y"THEN 200
```

Sobald Sie ein Bild mit dem Computer einfangen können, eröffnet sich eine Unzahl von Möglichkeiten: Sie können Ränder weiterverarbeiten, Bilder vergleichen, zweidimensional filtern und eine Reihe fortgeschrittener Techniken ausprobieren, die Gegenstand der aktuellen Forschung sind. Der Computer wird vielleicht einer Verarbeitung in 'Realzeit' um einen Geschwindigkeitsfaktor von mehreren Hundert hinterherhinken, und die 'Pixel'-Auflösung ist möglicherweise auch nicht gerade fabelhaft, aber prinzipiell sollte jede Strategie innerhalb der Fähigkeiten des Geräts liegen.

KAPITEL 11

WAS BRINGT DIE ZUKUNFT?

Die großen Computerhersteller haben viel Zeit gebraucht, um die Existenz des Mikrocomputers anzuerkennen. Sie haben ihn angestrengt ignoriert, aber er wollte einfach nicht verschwinden. Zuerst waren billige Mikrocomputersysteme nicht viel mehr als Spielzeuge, und sie waren so offensichtlich durch ihre Achtbit-Unterlegenheit behindert, daß sie den Großrechnern nie gefährlich werden konnten – oder etwa doch? Daß sie so leicht zugänglich waren, hat viele Leute gereizt, Software zu schreiben – unter ihnen manche, die die Bezeichnung 'Computerfachmann' weit von sich gewiesen hätten –, und bald war eine Vielfalt cleverer Programme auf dem Markt, von Textverarbeitungsprogrammen zur Umsatzplanung, von Lohnlisten zur Lagerhaltung. Als Peripheriegeräte in der Leistung herauf- und im Preis heruntergingen, wurde deutlich, daß der Mikrocomputer kein bloßes Leichtgewicht war, sondern der Eckstein der Industrie, der allabendlich zahllose Werbesendungen im Fernsehen wert war. Als Sechzehnbit-Chips erschienen, begannen sich die Riesen zu rühren – obwohl die Software hauptsächlich aus umgeschriebenen Achtbit-Routinen bestand und selten einen Geschwindigkeitsgewinn brachte. Jetzt strengen sich die Großgerätehersteller an, Mikrocomputer in ihre Marktstrategien mit aufzunehmen.
Die gleiche Entwicklung beginnt sich für Roboter abzuzeichnen.

WEITERENTWICKLUNG VON ROBOTERN

Betriebsingenieure sind seit langem mit numerisch gesteuerten Werkzeugmaschinen vertraut. Diese werden mit so altertümlichen Techniken wie Lochstreifen gesteuert und ähneln im Konzept den übrigen Industrieroboter. Einmal programmierte Anweisungen werden wiederholt, um eine Serie identischer Produkte zu erzeugen. Werden die Anweisungen geändert, kann dieselbe teure Werkzeugmaschine ein anderes Produkt herstellen – der Beginn eines flexiblen Produktionssystems. Erst als der anthropomorphe Roboterarm aufkam, begann sich die Bezeichnung 'Roboter' für diesen Automatentyp allgemein durchzusetzen. Diesen Namen trug als erster ein IBM-Roboter, der mehr der Anordnung einer Fräsmaschine als einem menschlichen Arm glich. Industrieroboter dieser Kategorie tragen Preisschilder über Zehntausende von Mark, und ein sehfähiges System kann ohne weiteres ein Vielfaches davon kosten.
Dann erschienen Lehrroboter auf der Szene. Für etwa tausend Mark konnte man ein ziemlich blechernes Gerät kaufen, daß zugegebenermaßen einer Spielzeugversion der Mechanik eines JCB glich. Es konnte an fast jeden guten Mikrocomputer

angeschlossen und ähnlich wie seine größeren Verwandten programmiert werden. Seine Hebekraft war praktisch Null und seine Geschwindigkeit nicht bemerkenswert, aber Roboter waren nun nicht länger ausschließliches Eigentum der Großindustrie. Selbst diese bescheidenen Geräte konnten mit Sensoren versehen und durch Programm gesteuert werden, die tendenziell Intelligenz besaßen. Als kleine Firmen (und einige große Gesellschaften) mit den Möglichkeiten billiger Automaten experimentierten, wurde deutlich, daß die Nachfrage nach Robotern wuchs, die einen relativ niedrigen Preis mit brauchbarer Leistung vereinigten. Wie der Mikrocomputer wuchs, um seine Marktposition auszufüllen, so streckt der Mikroroboter jetzt seine Muskeln, um Anwendung in der Industrie zu finden. Anders als seine marktbeherrschenden Vorfahren ist der Mikroroboter nicht an veraltete und teure Minicomputer gefesselt; er kann die neuesten und billigsten Mikrocomputer-Technologien nutzen. Komplett mit Computer, Sprache und einer den heutigen Marktführern vergleichbaren Leistung wird die neue Robotergeneration vermutlich im Preis unter 20000 Mark liegen.

Die Verfügbarkeit mechanischer Roboterleistung ist nur ein Aspekt, was die industrielle Ausbeutung betrifft. Bis jetzt nutzen wenige Anwendungen auch nur einen Bruchteil der Raffinesse, deren ein Roboter fähig ist, und der Mangel an fähigen Roboterprogrammierern bedeutet, daß das Einprogrammieren mechanisch wiederholter Vorgänge üblich ist. Das Ende dieses Problems ist jedoch in Sicht. Mit dem Mikrocomputer ist eine Generation herangewachsen, die in der Schule und zu Hause mit dem Programmieren vertraut geworden ist. Einige haben ein Vermögen als Unternehmer gemacht, andere haben betrübt festgestellt, daß Programmieren gelegentlich so niedrig wie Stenotypie bezahlt wird. Wenn Mirkocomputer in Schulen und privat mit Robotern ausgerüstet werden, wird die nächste Generation die industrielle Automation mühelos bewältigen. Bald wird jede kleine Werkstatt sich einen oder zwei Roboter leisten können, und es wird Sachkenntnis in Fülle vorhanden sein, um sie zu programmieren. Eine Zeitlang jedoch wird Erfahrung mit Robotern eine hochbezahlte Ware sein, und ich hoffe, daß dieses Buch Ihnen einen Vorsprung verschafft.

ROBOTER-INTELLIGENZ

Die Definition eines Roboters läßt sich so erweitern, daß sie jede Maschine umfaßt, die ein 'Robotnik' ist – ein Arbeiter. Es ist nicht schwer, automatische Waschmaschinen und Spülmaschinen einzubeziehen, die schließlich Variablen wie Wasserstand und Temperatur messen und durch ein Programm den Ablauf entsprechend steuern. Und wenn auch eine Mikromaus wenig Arbeit verrichtet, so ist sie doch sicher ein Roboter. Die Mäuse, die sich ins Zentrum des 'Euromaus'-Labyrinths der Euromicro vorgekämpft haben, haben dazu Sensoren und Motoren mit einem hohen Maß an Intelligenz verwendet – wenn auch nur in Vertretung ihrer Schöpfer.

Auch Industrieroboter beginnen sich von ihrem 'Tu genau was ich dir sage'-Image zu lösen und unterwerfen ihre Arbeitsweise Korrekturen und Modifikationen, um ein allgemeiner spezifiziertes Ziel zu erreichen. (Mit einiger Mühe unterdrücke ich mein Bedürfnis, die Einzelheiten des Projekts 'Craftsman Robot' auszubreiten, für das meine Gruppe am Portsmouth Polytechnic von der SERC unterstützt wird.)
Der wichtigste Algorithmus zur Auflösung eines Labyrinths stammt von Nick Smith, dem ersten Euromaus-Champion von 1980. Er ordnet den Quadraten zunächst Zahlen zu, die im Zentrum mit Null beginnen. Jedes vom Zentrum aus erreichbare Quadrat wird mit Eins numeriert, und jedes von einem Einser-Quadrat erreichbare Quadrat mit zwei, und so weiter. Wenn neue Wände gefunden werden, werden Verbindungen zu niedriger numerierten Quadraten unterbrochen, und die Werte wachsen an. Der kürzeste Weg zum Zentrum folgt den Zahlen abwärts – bis eine neue Wand angetroffen wird. Ein Pedant würde darauf bestehen, die Technik 'rekursives dynamisches Programmieren' zu nennen. Mikromäuse gewannen daraufhin durch Wendigkeit und insbesondere durch Verläßlichkeit. David Woodfields 'Thumper' arbeitet noch zwei Jahre nach seinem Sieg 1981 einwandfrei. Alan Dibley hat das Konzept der 'Sparmaus' eingeführt. Er entfernt die Tastatur des billigsten erhältlichen Mikrocomputers, montiert ihn auf einen Sperrholz- und Balsaholzrahmen und verwendet kommerzielle Modellflugzeugmotoren. Mit der in Kapitel 9 beschriebenen Technik erzielt er beträchtliche Erfolge – wenn auch in letzter Zeit nicht genug, um die finnischen Champions zu besiegen. Es ist sehr bezeichnend, daß Schulteams am Wettkampf teilzunehmen beginnen und sogar bis nach Madrid pilgern. Ihre Mäuse sind vielleicht noch verbesserungsfähig, aber ein Anfang ist gemacht. Möglicherweise wird sogar ein Schulteam Champion in Kopenhagen und unternimmt dann einen kostenlosen Ausflug, um am Wettbewerb in Japan teilzunehmen.

ROBOTER-PINGPONG

Jetzt ist ein Turnier nötig, das die Professionellen auf die Probe stellt. Ich habe ein Pingpongturnier für Roboter vorgeschlagen und schon mehrere Dutzend ernsthafte Anfragen potentieller Bewerber (oder sollte ich sagen: potentieller Designer) bekommen. Das Datum des ersten Spiels war zunächst auf 1986 festgesetzt worden, wird aber jetzt wohl fast sicher auf 1985 vorgezogen. Das Turnier ist nicht so weit hergeholt, wie es auf den ersten Blick erscheinen mag. Die Platte ist nur einen halben Meter breit und zwei Meter lang. Halbmeterbreite Rahmen an jedem Ende und über dem Netz begrenzen die erlaubten Ballbewegungen und verkleinern das Gebiet, das die Roboter erreichen können müssen. Das Netz ist einen Viertelmeter hoch, und deshalb führt ein Schmetterball unweigerlich zu einem verlorenen Punkt. Simulationen zeigen, daß ein Roboter den Ball sehr präzise schlagen muß, um das Zurückschlagen zu erschweren.

Den Aufschlag macht die Platte selbst. Der Ball hängt zu Beginn unbewegt vom mittleren Rahmen über dem Netz. Wenn die optischen Erkenntnissysteme beider Roboter sich darauf eingestellt haben, schlägt eine fast durchsichtige 'Fliegenklappe' den Ball zu dem Roboter, der 'Aufschlag' hat. Der Ball springt einmal, bevor er das 'Spielfeld' verläßt, und der Roboter muß ihn so zurückschlagen, daß er einmal aufschlägt, bevor er das gegnerische Spielfeld verläßt. Und so geht das Spiel weiter.
Ein paar Berechnungen zeigen, daß keine übermäßige Geschicklichkeit erforderlich ist. Ein guter X-Y-Plotter, nahe dem Spielrahmen montiert, könnte schon fast alles an Hardware sein. Der Schläger kann in der Mitte von einem besseren Stiftheber gehalten werden, der mit einem kleinen Motor ausgestattet ist und feuert, wenn der Ball sich dem Schläger nähert. Zusätzlich ließe sich das Anschneiden realisieren; denselben Effekt könnte man auch mit einer gewölbten Schlägeroberfläche und präziser Steuerung erzielen. Die Anzahl unterschiedlicher Entwürfe ist mindestens so groß wie bei den Mikromäusen, und es gibt keinen Grund anzunehmen, daß die Meldungen sich auf 'Profis' beschränken. Klar ist, daß der Erfolg von einer Kombination aus flexibler optischer Nachführung und wohldurchdachtem strategischem Spiel abhängt.
Das Interesse an diesem Wettbewerb wird allmählich international. Geht man nach dem Interesse, das die japanischen Delegierten auf der Madrider Euromicro gezeigt haben, werden sie Roboter-Pingpong so schnell aufgreifen, wie sie Euromouse übernommen haben.

ZUM SCHLUSS

Mikroprozessoren und Roboter werden wohl den vorausgesagten Beitrag zu einer neuen industriellen Revolution leisten. Maschinell erzeugte Güter werden wahrscheinlich weiter im Preis sinken, und nur eine Nation von Maschinenstürmern würde sich in Zukunft noch auf monotone Fließbandarbeit als Grundlage der Nationalökonomie verlassen. Die Revolution läßt sich vielleicht ein wenig beschleunigen; sie zu verzögern wäre schwer. Aber welche wirtschaftliche Bedeutung Roboter auch immer haben, sie machen enorm viel Spaß.

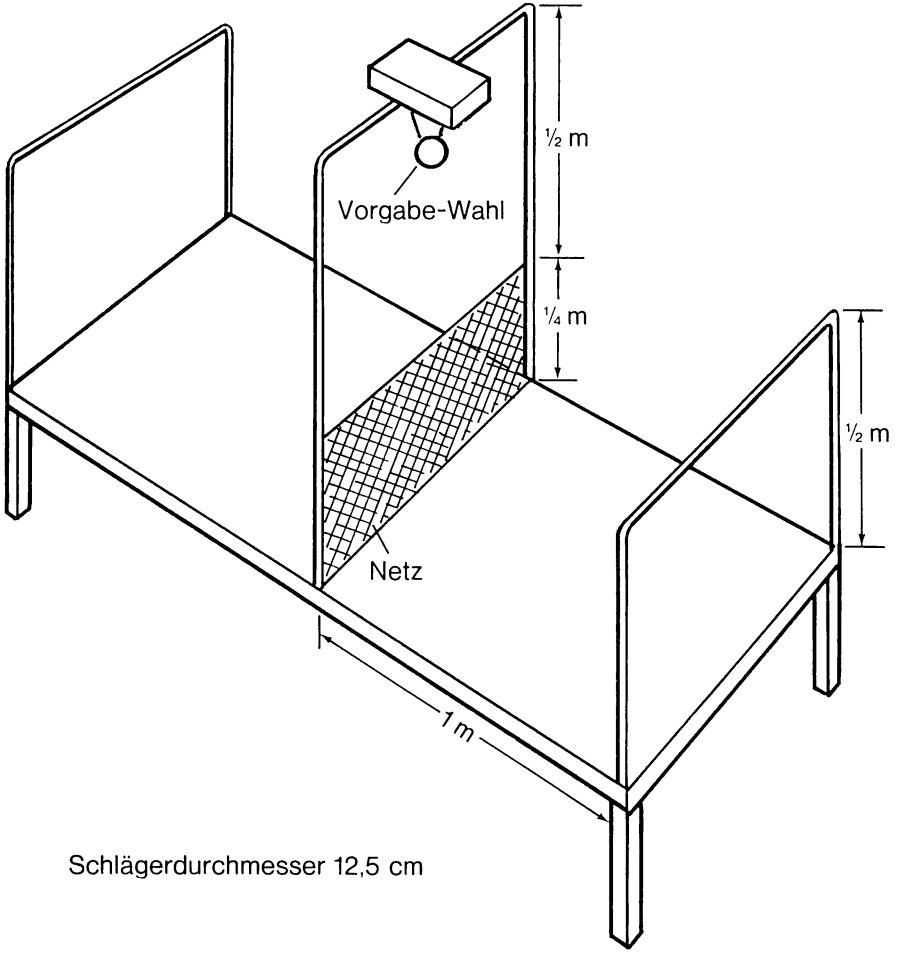


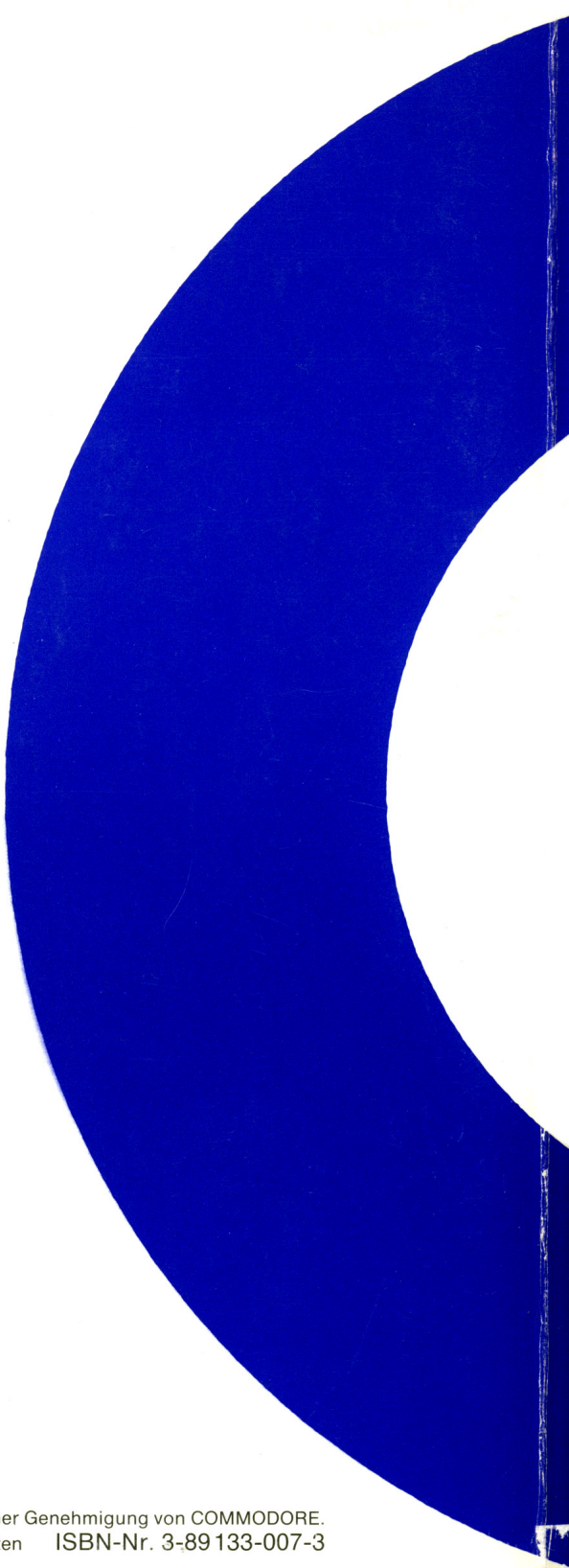
Abbildung 11.1 Platte für Roboter-Pingpong

Wie schließt man einen selbstgebauten Joystick, Schrittmotor oder ausgewachsenen Roboter an den Commodore-Computer an? Wie schreibt man Programme zur Steuerung von Schrittmotoren, und wie kann man mit diesen Programmen und Bausteinen für ein paar Groschen analoge Signale senden?

Schritt für Schritt lernen Sie, eine Fülle von Geräten zu bauen. Gleichzeitig wächst Ihr Verständnis der Grundlagen der digitalen und analogen Ein- und Ausgabe.

Obwohl John Billingsley acht Jahre als Don in Cambridge verbracht hat, hat er eine praktische Einstellung zur Technik. Seine kommerziellen Entwürfe reichen von Autopiloten und Computersystemen für Krankenhäuser bis zu Einchip-Zeitschaltern für Kocher und einem Meßgerät zur Bestimmung des Feuchtigkeitsanstiegs.

Er ist Mitglied mehrerer IEE-Ausschüsse, leitet eine Gruppe zur Erforschung von Robotern, und ist bekannt als Organisator des „Euromouse Maze"-Wettbewerbs.



Commodore

Commodore GmbH
Lyoner Straße 38
D-6000 Frankfurt/M. 71

Commodore AG
Aeschenvorstadt 57
CH-4010 Basel

Commodore GmbH
Kinskygasse 40–44
A-1232 Wien

Artikel-Nr. 580005/11.84 Änderungen vorbehalten ISBN-Nr. 3-89133-007-3