COMPUTING EXPERIMENTAL

Handbook for Apple II computers

fischertechnik®
COMPUTING

Please note:

This manual describes the operation of the fischertechnik COMPUTING EXPERIMENTAL kit using the Apple computers Apple][, Apple][+, Apple //e and Apple //gs. Make sure that you use one of the above computers. The kit can **not** be used on the Apple //c computer. The fischertechnik COMPUTING EXPERIMENTAL kit can also be supplied for Commodore C64 and C128 computers and for IBM-PC and compatible computers.
All measures and efforts have been taken to guarantee the correctness of this product documentation. However since fischerwerke Artur Fischer GmbH&Co KG constantly work on the improvement of their products, we cannot guarantee the completeness and correctness of this documentation from the time of printing onwards.

In case of suggestions or proposals or requiring additional information customers may address themselves to one of the distributors listed below:

U.S.A.:
fischer america Inc.
175 Route 46 West
Fairfield, New Jersey 07006

United Kingdom:
Economatics Education
Epic House, Orgreave Road
GB-Handsworth, Sheffield S13 9LQ

Australia:
Modern Teaching Aids PTY
26 Chard Road
Brookvale, NSW 2100

Sverege:
Playmaster AB
Box 8294
S-16308 Spånga

Danmark:
Playmaster AB
Holsted Erhversnomrade 15b
DK-4700 Naestved

Canada:
Spectrum Educational Supplies Ltd.
125 Mary Street
Aurora, Ontario L4G 1G3

Nederland:
Technisch Handelsbureau
Smitfoort BV
L.F.D. Montignylaan 197
NL-3055 NC Rotterdam

Belgium:
La Compagnie Européenne S.C.
Toys & Hobby Division
Rue du Cerf, 85
B-1320 Genval

España:
fischertechnik España S.A.
Rambla Catalunya, 127 Atico 1
E-08008 Barcelona

Dear fischertechnik enthusiast,

You've probably seen the reports in the newspapers and the programs on television: The subject is computers, robots, automation and the factory of the future. Each of the work operations moves smoothly into the next: welding, bolting, assembling, painting.
And this is in fact the end of a process that started in the office. The designer feeds the computer with dimensions which the computer converts into pictures on the screen and detail drawings or data streams. It is linked on-line and directly to a fellow computer on the factory floor which calculates the movements and activities for a robot from this data. Then it runs its programs - hour after hour, day by day.

When Karel Capek invented the word "robot" in 1921, he imagined an artificial man, a puppet which is seemingly capable of moving automatically and partly performing functions which man tells it to. For many years the human appearance of robots was a special feature. This can be seen by the hundreds of science-fiction stories and films. The actual robots today have very little in common with these pictures and neither are they as intelligent.
Modern robots are very hard workers. They build cars and transport loads. But (luckily), they can't think like a man. Expressing feelings, solving problems using the imagination, that is impossible to a computer. A computer can only perform what it is told by a program. It's our task to develop the program using our imagination and creativity.
This experimenting kit demonstrates in practice all the possibilities of robots in miniature. And if you always follow the instructions, you will very quickly understand the way of programming. And immediatly the first experiments start. On top of it we will give you a lot of exciting proposals. Try them, you will have a lot of fun
Yours sincerely,

Artur and Klaus Fischer

Table of Contents

2 What the kit contains

You should read this first before starting with the most interesting model. We want to give you a few tips before you start.

First we want to give you an overview of the contents of your fischertechnik COMPUTING EXPERIMENTAL kit.
You will find three manuals in the box, one of which is the present one. The present one is your guide through all the experiments, programming, explanation, suggestions etc. For reasons of printing technology the assembly of the fischertechnik models is detailed in a different manual, the fischertechnik COMPUTING EXPERIMENTAL assembly manual. The third manual is the interface manual. Normally you won't need this manual, as the operation of the interface in the context with the present kit is detailled in the actual manual, the fischertechnik COMPUTING EXPERIMENTAL handbook. The instructions of the interface manual describe how to operate the interface on your computer using a different software approach. You'll only need the interface instructions when you want to find out more details on how the interface works or if you want to add other fischertechnik COMPUTING kits. So put this manual apart, but not to far; you may need it on later occasions.
You may also find a registration card. Customers in U.S.A. are asked to fill out the registration card and mail it to the address given on page 2. They will obtain a free copy of the fischertechnik NEWSLETTER.
Then there is the software package, consisting of a disk in the format of Disk II drive (5¼" diskette). We will come to the software in the next chapter.
What´s more about hardware? Of course all the hundreds of fischertechnik parts for the models. If you like to check the completeness of the parts you should refer to the parts list in the assembly manual. The parts list provides you with a photo of the parts, its part number, the quantity in the box and the part's name.
The big box with the printed circuit in it is the fischertechnik interface. The interface is used to link the model to the computer. It is also connected to the fischertechnik COMPUTING power supply. Thus, under the management of the software and the computer the interface directs the electric power to the fischertechnik model. More about that in chapter 4.
The package also contains a small but very important bit of hardware. This is an adapter which matches the interface to your

computer. It consists of a small printed circuit board with a connector, an integrated circuit chip and a cable. The connector is composed of twenty pins mounted in a housing. Take the fischertechnik interface out of the kit. It has a cable attached to it and it is mounted with another connector. This connector fits exactly onto the twenty pins. There is a knockout in the housing and a guide in the connector so that the two connectors can be fitted together correctly. The cable end of the adapter connector fits to the game controller port of your computer.

CAUTION: Before fitting the connector, switch off the computer first!

Open the lid of your computer and have a close inspection of the printed circuit board. On the Apple][and the Apple //e you will find an empty socket at the right back end of the board, close to the long card slots, where your disk drive interface is inserted. On the Apple //gs the socket is more close to the middle of the board. The socket is just of the same type as is used for inserting the integrated circuit chips and has 16 holes.
Now remove the piece of protective foam from the adapter cable. Bring the cable through one of the openings or slots on the rear side of the housing to the interior of the computer. The openings on the rear plane are rather small, so take care not to damage the connector.
The connector has 16 pins and will fit to the socket identified beforehand. There is no indication of the orientation, however. Have a look to Fig. 2.1. On the connector a dot is indicated, which identifies pin number one of the connector. In reality there is no dot, rather the pin numbers are clearly visible. Now you know about pin number one of the connector, you have to locate pin number one of the socket as well. To our best knowledge all integrated circuit chips on all Apple circuit boards are oriented in uniform direction, which holds for the game controller socket too. So have a look to the labelling of the chips (you don't need to understand the meaning of the labels). Turn the computer, so that the labels appear to be in normal reading direction. Now pin number one of each socket is in the left lower corner. By the way: for most Apple II computers the cable will be oriented as is given in Fig. 2.1.
Now insert the connector to the socket. This is much easier said than done. The pins of the connectors are very delicate and break

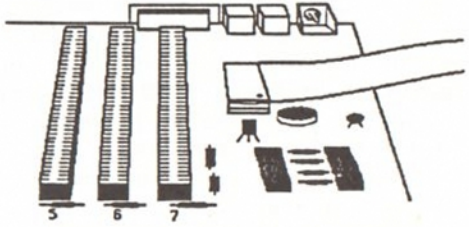


Fig 2.1: Fitting the interface adapter to the Apple II computer (housing omitted for clarity).

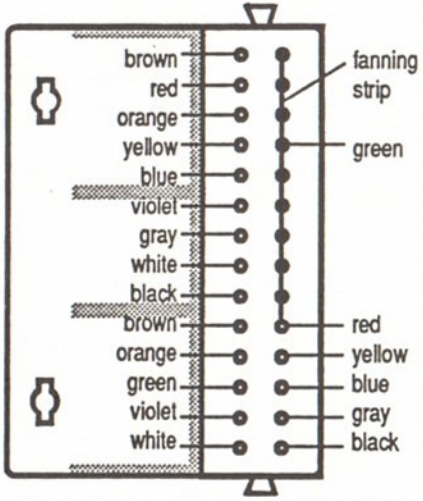


Fig. 2.2: Connection of the 20-wire cable to the 28-pole socket. The two cables green 1 and red 2 are mounted together with the fanning strip (View from bottom side).

when they are bended. Also there is no guiding and you have to rely on careful visual inspection. We advise you to remove any interface card according the manufacturer's instructions from the slots in order that you have more room to maneuvre. Adjust the connector and then push it down. **Do not force it in.**

When you have installed the interface adapter, assemble any interface card and close the lid of the computer.

We give you the advice that you let the interface adapter cable installed even if you do not use the fischertechnik interface. The interface may be detached by removing the interface connector from the adapter printed circuit board.

Now take the power supply out of the box. The power cable is terminated with a red and a green plug. Insert the red plug into one of the two interface sockets marked with a +. Insert the green plug into a socket marked with a -. It doesn't matter which of the sockets you use. The number of sokkets is doubled for the larger fischertechnik COMPUTING models which require more power.

All you have to do now is connect the interface to the model. In the kit you'll find a 20-wire cable 2 meters long with conductors of different colors. One end is connectorized for plugging into the interface.

Stop! Don't plug in yet! You must first fit a connector to the other end of the cable. In the kit there is a 28-fold plug connector which fits to the fischertechnik connectors. There is also a metal fanning strip to connect several plugs together. Screw the twenty conductors of the ribbon cable and the fanning strip into the plugs. Have a careful look at Fig. 2.2 which shows you exactly which conductor belongs to which plug. You can use the cable colors as a guide. The conductors are colored brown, red, orange, yellow, green, blue, purple, gray, white, black and then the colors are repeated. When you are screwing the conductor into the plug, make sure that you don't screw down too tightly, otherwise you will squash and break the cable. Then connect the green and red wire of the ribbon cable (both labelled with +5V on the interface lid) to the plugs which receive the fanning strip. It doesn't matter to the fischertechnik interface whether the two lines are connected in this way. With similar interfaces of other manufacturers supplied for other fischertechnik COMPUTING kits (e.g. the interface manufactured by Parsec Research), care must be taken at this point. In that

case, follow the instructions of the manufacturer.
When you have finished, make a careful visual inspection. There is a label to be fastened to the top of the connector indicating the color code of the connected cable to each socket. Make the effort to really check carefully and precisely whether the colors match. You will save a lot of trouble later and prevent damage to the interface. Only when you are absolutely sure, plug the connector into the interface. But first, press the ribbon cable flat against the red plate, place the 45 mm long strut across it and secure with two locking clips. This relieves strain on the cable-socket connection, which is otherwise easily damaged.
The step-by-step mounting instructions for the models are detailed in the assembly manual. In each step, the components to be used in this construction step are shown in a box. Here is a tip for you: first find the components for each construction step and then assemble them together. Don't proceed to the next construction step until all the components of the previous step have been mounted. If there are any left over, have another careful look at the photos; you may see what you've missed out. Also make sure that you assemble the building blocks in the right orientation, otherwise you may find that you have blocked a space in mounting one of the later building blocks.
All the models contain some electrical components or other: switches, motors or sensors. These are connected to the 28-pin connector which you fitted. We have also supplied you with a number of two-wire cables and fischertechnik connectors. The assembly instruction will tell you which of the cable lengths you will need (6 cm, 18 cm or 44 cm; appr. 2½", 7" and 17", resp.). Select the correct colors for the plugs so that the color markings on the ribbon cable and the plug coincide. This will make it easier for you when you check the wiring on larger models.
To mount the connector, strip the insulation off the cable end, twist the stranding a little. Then wind the stranding around the insulation (see Fig. 2.3). Insert the cable end into the connector terminal so that the screw presses onto the insulation when it is tightened. Again: Do not tighten too tightly otherwise you can break the conductor.
Of course, mount the same color connector at either end of a conductor. The color coding in the cable insulation will help you to tell the two conductors apart.
The next chapter details the powering up.

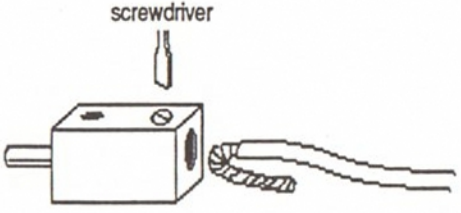


Fig. 2.3: Mounting of the fischertechnik connector.

By the way:
From time to time we give you really useful additional information. For instance, a summary of commands, the explanation of a term, the way a sensor works, etc. You may carry on reading the main body of the text but perhaps you may also want to give a closer look at these tips. They always stand in the margin. Here is the first one:
If you want to know what models belong to what parts of the text, just compare the symbols in the top left corner of the two booklets. Right! The same symbols always indicate the same models and texts.

3 How to start

For the following experiments, you will nearly always need one or more motors to move the WINCH, the RADAR EYE, the WELDING ROBOT arm and the mobile robot named TURTLE. They are connected to the computer via the interface, the 20-pin color-coded cable and the 28-pin connector. Power is supplied by the transformer. But nothing is moving yet. Right! There is no software to put everything in motion.

The first thing you should do is to make a backup copy of the fischertechnik floppy disk. You ask why. Just imagine: What would happen to your fischertechnik COMPUTING EXPERIMENTAL kit, if you were to loose or damage the fischertechnik diskette? It would be as bad as losing the kit itself. The fischertechnik software is not copy-protected, therefore you don't need a sophisticated copying program. You may use the FID program supplied on your DOS 3.3 system disk or any other suitable program. Just follow the instructions of your copying program. Please note that the disk is double-sided, although only one side can be used at a time. The two disk sides have to be copied to two separate disks. One disk will contain the software of the experiments described in chapter 4 through 8, the other disk the programs of chapters 9 and 10.

Maybe you consider to make even two copies of one side. In case you will write a lot of own programs, you will erase all sample programs from the extra copy and only keep the interface driver (more about this technical term on the next pages). So delete all files below the dashed line appearing in the catalog of the disk. This will give you ample space for own programs.

Our suggestions are not an open invitation, however: according to copyright law, you are only allowed to make copies for your own personal use.

Then, only use the copied floppy disks to work with. Store the original floppy disk in a safe place where the natural enemies of disks, such as sand, heat, cats or magnetic fields, cannot reach it. Only use the original fischertechnik floppy disk to make another copy, if necessary.

Some fischertechnik programs store data to disk. Once you are at it, you should also format at least one empty data disk.

Switch off your Apple computer once more. This will clear the computer's memory most reliably. Now insert the copy of the fischertechnik in the disk drive and switch on again. The procedure for Apple //gs will be detailed a few paragraphs later.

After a short period the display will show

you a welcome message of fischertechnik COMPUTING EXPERIMENTAL. Press any key to go on. Now some screen full of instructions follow. We advise you to carefully read the instructions. The instructions will give you many tips which are **not** in this experimental handbook. They mainly refer to a short description of the models. However, watch out! They may also contain last minute information to the hardware and the software of the kit. After the information is displayed, you will be prompted to load the interface driver. You may escape by pressing the ESC-key. When you decide to load the software the computer will again start the disk drive.

By loading the interface driver the computer is being provided with a number of new commands which it previously did not have. These commands will allow you to control fischertechnik components by program via the interface.

Loading the interface driver is concluded by the choice of your printer, which may be used to protocoll your experiments. You are offered six different printer drivers. They fall into two major categories according the type of printer and printer interface. If you use a serial interface, e.g. for an Apple Imagewriter, you will use a choice of the first three drivers. A serial interface is mostly adressed via the command PR#2, i.e. the interface card is located in slot number 2. Parallel interfaces are for EPSON compatible printers. They are mostly adressed by the command PR#1, i.e. the interface card is located in slot number 1. In that case try one of the interface drivers 4 through 6.

Due to the huge amount of different interface cards and printers we advise you just to try the different drivers. Start with the 8-bit version and only if doesn´t work try the slower 4-bit version. Printing of screen graphics will be detailed in chapter 6.3.

When loading is finished, the computer will again report that it is ready to accept commands by displaying its prompt, the bracket sign].

Now about the Apple //gs. Normally your booting diskette will be a 3½" disk and contain PRODOS. We advise you to have an external 5¼" disk drive connected, which will take the fischertechnik disk.

After booting PRODOS, reboot from the fischertechnik diskette, e.g. by entering the command

PR#6

In the above command we have assumed,

that the 5¼" disk drive is connected to slot 6. In case you use a different arrangement, modify this command appropriately.
When executing the command your computer will boot from the 5¼" disk and from that time on emulate the Apple //e. Your operating system will be the good old DOS 3.3. The ongoing procedure as described above will now apply for the Apple //gs as well. This status lasts until you switch off the power of your computer or boot from a different disk.
You can check whether the additional commands run correctly by entering:

&IN

If the prompt appears again on the screen, everything is OK. If there is an error message, start again from the beginning. All interface commands like the one above start with the ampersand symbol &.
You can always test whether the interface is operating correctly by entering at any time:

&IN

The light emitting diode (LED) on the printed circuit board of the interface should light up for a short time. If it doesn't, test whether the interface is correctly connected to the power supply.
On later occasions you may wish to skip the introductionary part of the booting procedure. There is a short cut, which may be installed on your copy disk. Enter the following commands with your copy disk inserted in the disk drive:

LOAD LOADER.BAS
SAVE FISCHER

When booting next time, all the instructions are omitted and the extension routines are loaded right away.
In the following chapters, you will find a lot of experiments all with programs for your Apple computer. The programs are developed in the text step by step. If they work, you should store the programs on a disk (not the fischertechnik disk!). If you really don't get the hang of a program or if you want to quickly demonstrate a program or if you want a bit of help to enhance the program, then use the fischertechnik disk (or rather its copy). There you'll find example programs for all the experiments. The part of the program which is always identified as the main program is almost similar to the programs printed in the follow-

ing chapters. The remainder of the example programs, often the major part, is for operator guidance, screen layout, etc.
In the ongoing text we will assume that you have a basic knowledge of writing programs in the commonly used computer language BASIC. You should be able to type in the commands, correct them if necessary, start a program, list it on the screen and save it on the disk. Although we give you some help and hints, we are not going to present you a BASIC teaching course.
So in case you want to do more experimenting than just using the ready-made programs on the floppy disk and you are not quite sure about your programming skills, you might consider to study the manual of your computer as well as an introductory course on programming BASIC first.
Before starting, here are just a few tips on the instructions. You may page through the instructions browsing here and there. But when it comes down to the experiments, you should go through carefully point for point. You ask why. The instructions develop the programs just as programs are created in reality. After an experiment has been conducted, the next enhancements are considered and additional commands inserted. If you skip something, you won't know where and what to insert.
But this not only goes for the programs: You will learn a lot while working through the experiment manual. And if you skip a section, you may not understand or be able to integrate later instructions.

4 Experiments with pushbuttons and motors

4.1 Motor control using the computer: Output

When everything is functioning correctly, you can start with the first experiments. First, assemble the model WINCH 1 using the instructions in the assembly manual. On the base frame there is a motor and gear unit. A winch drum is mounted on a transverse axle driven by the motor. The motor is connected to the interface by the orange and the yellow wires. If you look at the wiring diagram on the interface, you'll find that the two wires are marked M1 (= motor 1). Insert the model wire into the interface and enter:

&IN
&1R

This procedure is called initialization. It is also very important for computers and is always carried out just after switching on the computer. The initialization even loads a first program in the computer for some computer systems. However, the interface must have its own command. To be sure that the interface is really initialized all other commands like &1R will not be accepted until the command &IN is given.

The motor will give a short turn. Let's explain the new commands. The command &IN (initialize interface) resets the interface to its initial state. You should always do this at the start of a program.

With the command &1R, the motor 1 responds. It should turn to the right (&1R=motor 1 right). If it turns to the right, it can also turn to the left! Just have a go:

&1L

Now the motor will give a short turn to the left. But why doesn't it run all the time? This is because there is a protective circuit in the interface which interrupts the motor control approx. ½ second after the last command obtained from the computer. This can also be observed when watching the lamp in the interface. It goes out when the interface interrupts. The protection circuit prevents damage to the model. Let's imagine that the wiring of the model is faulty and a motor would start to run in the wrong direction. It could demolish that beautiful model you worked so hard to build. So the first thing to do is to jump to the computer keyboard and press the Control-C key. Normally that would stop the program execution only and any peripheral like the motor would keep on running. With the protection circuit, however, the data transmission between computer and interface stops, as the program is stopped. With a short delay the motors are switched off also. The same situation comes up if the program is stopped due to programming errors or if you would switch off your computer. Now, how can you get the motor to run continuously? This is done by setting up a program loop. Enter:

10 &IN

20 &1R
30 GOTO 20

and start the program entering the command

RUN

Now the motor turns continuously. Line 20 starts turning the motor, line 30 jumps the program back to line 20 and the motor continues to turn. The program now runs endlessly in this loop. We can stop it using the Control-C key (hold down the key labeled CTRL while pressing the key for the letter C). This activates the protection circuit and after an afterrun of ½ second the motor will stop. Although the motor stops due to the protective circuit, the interface electronically has stored the information that the motor should run under normal circumstances. So if you enter the command

CONT

(which means CONTINUE) the program and the motor will resume operation. Stop the program once more. The correct way of switching the motor off is by issuing:

&1X

This also clears the storage of the control command in the interface.
You may ask why the protection circuit delays its action by ½ second. Have a look at the program. The command in line 30, which loops back the program execution will take a tiny amount of time. In other applications, there might be more commands necessary before issuing the next interface command. So the protection circuits allows for ½ second of calculation time. If you exchange the command &1R in line 20 by &1L, and issue the command

RUN

the motor will turn continuously to the left. Now let's try to control the motor for a limited amount of time: Try to make it turn for a short time to the left and then stop. This will operate the WINCH model mounted on the edge of the table as shown in Fig. 4.1.
Wind about 30 cm of cord onto the winch drum and hang a weight at the loose end (e.g. a hoisting cage made of building blocks). The command &1R can be made to loop an exact number of times - here 2000

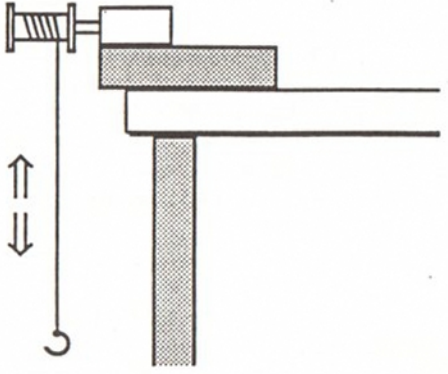


Fig. 4.1: You may install the rope winch at the edge of a table.

times - by putting it into what's known as a FOR...NEXT loop:

```
10 &IN
20 FOR Z=1 TO 2000
30 &1R
40 NEXT Z
50 &1X
```

Enter the program and start it with RUN. The motor will first turn for a time to the right and the rope will lower. Then the motor will stop - the program is finished.
How does the FOR...NEXT loop work? The command in line 20 contains a counter Z. It has a start value of 1 (...Z=1...). Next line 30 is performed: &1R, this means that the motor starts (or continues) to turn to the right. Line 40 will increment the counter by 1. There is also a test of the counter's content. As long as the counter is below its limit, which is 2000 in this case (...TO 2000) program execution loops back to line 30, which is next the FOR-command. When the counter Z arrives at 2000, the loop is interrupted and the next command is performed. Here it is &1X in line 50: The motor is switched off. If you want the loop to repeat itself only 1000 times, just change the upper limit of the loop:

```
20 FOR Z=1 TO 1000
```

Now let's wind the rope up. The motor must turn the same number of times to the left as it did to the right. Just change line 30 to:

```
30 &1L
```

After RUN the rope starts to wind up. You can get the winch to perform both movements one after the other by putting two FOR...NEXT loops one after the other. The first is for the downward direction and the second for the upward direction.

```
10 &IN
20 FOR Z=1 TO 2000
30 &1R
40 NEXT Z
50 FOR Z=1 TO 2000
60 &1L
70 NEXT Z
80 &1X
```

Start the program with RUN. When the rope has reached the top, the program has finished. You can let the rope run up and down continuously by telling the program to start again when it reaches the end:

```
90 FOR Z=1 TO 1000
100 NEXT Z
110 GOTO 20
```

We have added a pause before starting the downward movement. For this purpose we need another FOR...NEXT loop (line 90-100). It does nothing more than count from 1 to 1000 because there is no instruction inside the loop as in the previous experiment. This type of loop is called a "wait loop" which delays the start of the program part by a specific interval of time.
Line 110 makes the program jump back to the start (line 20) and the process starts again.
The command RUN starts the winch and pressing the Contol-C key stops it.
As you can see from the label on the interface, a maximum of 4 motors can be controlled. Each motor is provided with 2 wires, e.g. motor 2 has wire colors green and blue. If you connect the motor to these wires, the previous command &1R doesn't work. However, motor 2 responds with:
&2R or **&2L**.

This also goes for motors 3 and 4:
&3R or **&3L**
&4R or **&4L**

4.2 Switch contacts and pushbuttons: Input

In addition to motors, the models in the kit often use switches or pushbuttons. This section will explain how these components work and how they can be addressed by the computer.
Let's first take a pushbutton out of the kit. It has three terminals and next to each of them is an explanation of the circuit function, as shown in Fig. 4.2.
This pushbutton has two switch contacts. In its OFF position, when the pushbutton is not being pressed, there is a conductive connection between terminals 1 and 2. If the pushbutton is pressed, the switch contact to terminal 1 changes to the other side. Now there is a connection between terminals 1 and 3 - and the path from 1 to 2 is interrupted. This state continues as long as the pushbutton is pressed. If you let go, it goes back to its home position (connection 1 - 2). Now you know the difference between switches and pushbuttons. When you operate a switch - e.g. the light switch in your room - it stays in this position (OFF or ON). A pushbutton has a home position; when it is pressed, it switches over, and when it is released, it goes back to its home position. Now let's use the pushbutton in connection with the interface. Wire it as shown in Fig. 4.3. Terminal 1 is connected to the +5V

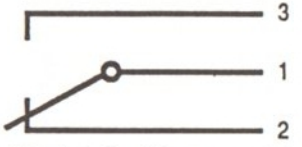


Fig. 4.2: Pushbutton switch markings

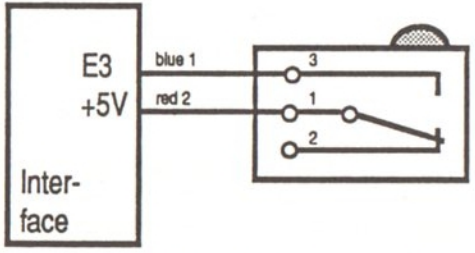


Fig. 4.3: Pushbutton-interface connection

terminal on the interface (red wire 2), and the switch contact 3 is connected to one of the inputs of the interface. We have chosen in the following samples the input E3 (entry 3, wire blue 1). In order to address the input line E3 of the computer, enter:

&IN
&DI

The command &DI (digital input) reads the values of all the input lines of the interface. The value E3 is displayed on the monitor by:

PRINT E3

In this case the value will be "0", i.e. the connection between 1 and 3 is interrupted as you can see in Fig. 4.3. No voltage is applied to E3. Now press the pushbutton and enter the above commands again. The value "1" will now be displayed on the monitor: The switch contact is closed and +5V is applied to E3. Contact 3 is usually called a make contact or a normally open (NO) contact because it closes when the pushbutton is operated.
To avoid entering the commands every time, you can write a short program to read in the switch state. Before entering the program make sure, that you have stored the previous program on disk, if you like to use it later. Use the command

SAVE MOTOR1.BAS

Instead of MOTOR1.BAS you may choose any other name you prefer. The restrictions in characters to be used and in length of name are detailed in your computer manual. Then clear the memory by entering the command

NEW

As a rule of game, we will not remind you to save any programs. It's your decision to preserve what you feel to be worth while. We also will assume that on starting a new chapter, you will start with the computer's memory cleared.
Now, here is the program:

```
10 &IN
20 HOME
30 &DI
40 VTAB 1
50 PRINT E3
60 GOTO 30
```

The command HOME in line 20 will clear the display.
In line 30 the value of E3 is measured (&DI). Before it is displayed, the print position is given at the top left corner of the screen. This is performed by the VTAB (vertical tabulator) command. The number after the command specifies the line number on the screen.
In line 60 there is a jump command to line 30 which makes the program start almost from the beginning again. We already know this type of loop from the previous chapter. The loop is repeated until you interrupt it by pressing the Control-C key.
Start the program with RUN. The screen is cleared and the switch state is then continuously displayed. If you press the pushbutton, the program will recognize this immediately. So that you know what "0" and "1" mean, add the following lines to the program:

```
50 IF E3=0 THEN PRINT "PUSHBUTTON
       OFF"
55 IF E3=1 THEN PRINT "PUSHBUTTON
       ON  "
```

The explanation will then be displayed after issuing RUN. Each text is assigned by inserting an IF...THEN inquiry in lines 50 and 55.
These commands test a condition: If (IF) a certain condition applies (i.e. E3=0 in line 50), then (THEN) perform the following action (PRINT "PUSHBUTTON OFF"). If the condition is not fulfilled (E3 is not equal to 0), then perform the next command. A similar inquiry is also made in line 55.
Check carefully that you entered a space character after the word ON in line 55. The textes in line 50 and 55 have to match in length.
As you know from above, the pushbutton has a second contact, terminal 2. Insert the wire from 3 to 2 and start the program again with RUN. Again the switch state of the pushbutton is displayed with the only difference that at the start it is "ON". Before, when contact 3 was used, it was "OFF". Just test it again to see!
Therefore, the pushbutton is in the OFF position when it is connected to terminal 1 and 2, as you can see in Fig. 4.3. This contact is therefore called a break contact or a normally closed (NC) contact because it opens when the pushbutton is operated. Try the different switch circuits with the program. You will need them quite often later for controls and counts.

Finally, we would like to show you a practical application using the pushbutton: Here, you can test your reactions! Are you quick enough to stand the test? Enter the following program:

```
10 &IN
20 HOME
30 PRINT "IF A FIELD APPEARS,"
35 PRINT "PRESS PUSHBUTTON!"
40 PRINT
45 I=0
50 FOR Z=1 TO 2000
55 NEXT Z
60 INVERSE
65 PRINT " "
70 NORMAL
75 &DI
80 I=I+1
85 IF E3=1 THEN  GOTO 70
90 PRINT
95 PRINT "STOP AFTER: ";I
100 END
```

Contacts 1 and 2 in the pushbutton are connected. Start the program and press the pushbutton when the field appears. If you have a result under 10, then you're really on the ball! You can try again by entering RUN. You can probably understand the meaning of each program line from previous examples. The field is generated in lines 60 through 70. The field itself is an inverted blank space. To generate it, the character color displayed on the screen has to be reversed with the background color. This is done by the command INVERSE. After printing the visible blank space, the screen is switched back to standard display (command NORMAL).
You can now see a practical demonstration of integrating a pushbutton into a program.

4.3 Motor control using pushbuttons: WINCH

So far you have learned to connect and control a motor and operate pushbuttons. The next task is to connect the two components together. Now let's tell the program to inquire about the position of a pushbutton in order to control a motor.
For this experiment, assemble the model WINCH 2 using the instructions in the assembly manual. On the base frame there is a motor with a winch which has a cord approx. 12" long wound on it with a weight attached to its end. Two pushbuttons are mounted on the outer edge of the frame. Terminal 1 of both pushbuttons is connected to +5V (wire red 2 on interface). The right-hand pushbutton is connected to E3 (cable blue 1), and the left-hand pushbutton to E2 (wire red 1). Terminal 3 of the pushbutton is used in each case, i.e. the make contact. When everything is connected, test the motor function by entering:

```
&IN
&1R
&1X
```

The motor will run for a short time after entering the second of the above commands. After entering:

```
&DI
PRINT E2, E3
```

the monitor should display "0 0". Then press the left-hand pushbutton and enter the above lines again. The monitor will display "1 0". If you press the right-hand pushbutton and enter the commands once more, the monitor screen will display "0 1". This proves that the components have been correctly connected up and are operational.
Now let's design the program. In the first task the motor should run until a pushbutton is pressed. Enter:

```
10 &IN
20 &1R
30 &DI
40 IF E3=0 THEN  GOTO 30
50 &1X
```

After issuing RUN the motor will run continuously. If you press the right-hand pushbutton, the motor will stop. Line 40 inquires about the switch state. As long as E3=0, that is, the make contact (1-3) is open, the program will always jump to line 30. There the input is read in again. The motor will continue to turn even if it receives no further

output instruction. The motor may also be kept running by inquiring about the inputs of the interface. Only if there is no data transmission at all, the protection circuit will switch off all the motors after ½ second because it assumes that the program has been stopped.
If the condition E3=0 no longer applies (pushbutton has been pressed), the motor is switched off (line 50). You can also use the other pushbutton for control:

40 IF E2=0 THEN GOTO 30

After having started the program, operate the left-hand pushbutton this time to stop the motor. Both pushbuttons can also be used:

40 IF E3=0 AND E2=0 THEN GOTO 30

In this IF...THEN inquiry, 2 conditions must be fulfilled for the following command (...GOTO 30) to be executed. The two conditions are E3=0 and (AND) E2=0, i.e. the motor will continue to run if neither pushbutton is pressed. This is called a logic AND operation.
The problem can also be solved in another way:

40 IF E3=1 OR E2=1 THEN GOTO 60
50 GOTO 30
60 &1X

Line 40 tests whether pushbutton 1 or (OR) pushbutton 2 has been pressed. If so, the program jumps to line 60 (...THEN GOTO 60), and the motor stops. Otherwise it continues to run (GOTO 30). This is called a logic OR operation. Either one or the other condition must apply for the command to be executed.
After having started the program it doesn't matter which pushbutton you press to stop the motor.
Now we want to give the instructions "winch up" to the left-hand pushbutton and "winch down" to the right-hand pushbutton.

20 HOME
40 IF E3=1 AND E2=0 THEN &1R
50 IF E3=0 AND E2=1 THEN &1L
60 GOTO 30

For safety reasons, both pushbuttons must be inquired for their direction of movement. One must be released if the other is pressed to prevent two different commands from being sent to the interface shortly one after the other if both pushbuttons are

pressed at the same time.
Enter the lines and start the program. Now you can raise or lower the rope as you wish using the two pushbuttons. You can stop the motor using the Control-C key. In lines 40 and 50 there is another AND-operation with two conditions which must both be fulfilled to execute the command.
The pushbuttons can also be used as a start button to completely lower or raise the winch rope. For this, just enter:

```
40 IF E3=1 AND E2=0 THEN GOTO 70
50 IF E3=0 AND E2=0 THEN GOTO 120

70 FOR Z=1 TO 2000
80 &1R
90 NEXT Z
100 &1X
110 GOTO 30
120 FOR Z=1 TO 2000
130 &1L
140 NEXT Z
150 &1X
160 GOTO 30
```

When the rope is completely wound up, start the program with RUN. If you give the right-hand pushbutton a short press, the rope will lower. This is done in the program by lines 70-110 which we learned in the last chapter. The rope can be rewound by giving the left-hand pushbutton a short press. The program steps for this are in lines 120-160. They are called up by the IF...THEN inquiries in lines 40 and 50. If the respective condition applies, the command ...THEN GOTO 70 or ...THEN GOTO 120 , respectively, is executed.
What will happen if the rope is lowered and you press the key for "rope down"? The drum will turn as if you wanted to lower the rope. But then it will wind the rope in the wrong direction. To stop this from happening, note the position of the rope and make the program only execute the correct sequence. If the rope is at the top, it can only go down and vice versa. Stop the current program with the STOP key and enter:

```
25 PO=0

40 IF E3=1 AND E2=0 AND PO=0 THEN
       GOTO 70
50 IF E3=0 AND E2=1 AND PO=1 THEN
       GOTO 120

95 PO=1

145 PO=0
```

At the start, the rope must be at the top. The position is recorded in line 25: The variable PO is set to 0. The IF...THEN inquiry in lines 40 and 50 has been extended by an additional condition. The program can then only jump to line 70 if pushbutton 3 is pressed, pushbutton 2 is released and the rope is at the top (PO=0). Only then can the rope run downwards. The situation is reversed in line 50: Pushbutton 3 is released, pushbutton 2 is pressed and the rope is down (PO=1). Then the rope will be raised. Each of the positions is recorded in lines 95 and 145 after the movement has been executed. The program is again stopped with the STOP key.

Now you can see how the motor can be precisely controlled using the pushbuttons - either by direct run commands (&1R, &1L) or by calling up part of the program for a longer run. The position of the pushbuttons and the motor can also be connected (logic operations).

On the floppy disk you will find a program called SWITCH.BAS. It gives you a detailed introduction to motor control. You can turn the drum to the clockwise or to the counter-clockwise by using the pushbuttons.

4.4 Commands and positions: Step by step

If you were observant in the previous experiments, you will certainly have noticed that the rope did not stop at the same place every time after the winch has stopped. The reason for this is the time control of the motor. The step control principle is more precise. Here, every revolution of the motor is counted. The rope can therefore be lowered precisely by a specified length by giving a number of motor steps.

The following experiment will show you how these steps are counted and how the motor thereby is controlled. First, assemble the model WINCH 3 using the instructions in the assembly manual. On the base frame there is again the motor and the winch. A pushbutton is mounted on the side of the rope drum and this is connected to input E2 (wire red 1) of the interface. It is mounted so that the cam of the drum operates it after every half turn.

The program should now make the motor run and stop after the pushbutton is operated by the switch cam. Enter:

```
10 &IN
20 &1L
50 &DI
60 IF E2=0 THEN GOTO 50
70 &1X
```

Set the rope drum so that the pushbutton is not pressed. After RUN the motor will turn until the pushbutton is switched. When the motor stops, it will have moved the winch drum approximately half a turn. You already know program steps 10 to 70 from the last chapter: There, we operated the pushbuttons manually.
Now look more closely at the pushbutton. Perhaps the pushbutton is released again because the switch cam has overshot the mark. If this happens, you can call up the next step with command RUN. Now position the rope drum so that the cam presses the pushbutton. Then give the command RUN. The motor will give a short, hardly noticeable jerk and it will stop again. The reason is that the pushbutton was pressed and the inquiry in line 40 immediately fulfilled, thus terminating the program right away. Enter the following lines:

```
30 &DI
40 IF E2=1 THEN GOTO 30
```

Line 30 waits until the pushbutton is released. Then it tests in line 60 whether the pushbutton was re-pressed. This program will run no matter what the start position of the rope drum.

Since this program sequence is very often required, it has its own command:

&1F (Motor 1 forwards)

The command also runs much faster than the five lines of the BASIC program. Positioning is therefore more precise with this command.
If you want to let the motor run, for example, 10 turns of the drum, the command must be repeated 20 times (1 command = 1/2 turn):

```
20 FOR Z=1 TO 20
30 &1F
40 NEXT Z
50 END
```

It can also be reversed; change:

30 &1B (Motor 1 backwards)

After RUN, the motor will reverse 10 turns.
In addition to the two commands &1F and &1B, there are also commands for motors 2, 3 and 4:

&2F and **&2B**
&3F and **&3B**
&4F and **&4B**

This step control principle is frequently found in digital technology. In addition to robots, disc drives and printers, everyday objects such as digital watches with hands are operated by a step-controlled motor. If you look carefully, the seconds hand moves in tiny steps.

With these new commands, the winch can be precisely positioned. Enter the following new program for this purpose:

```
NEW
10 &IN
20 HOME
30 PO=0
40 GET A$
60 IF PO=1  THEN GOTO 120
70 FOR Z=1 TO 20
80 &1B
90 NEXT Z
100 PO=1
110 GOTO 40
120 FOR Z=1 TO 20
130 &1F
140 NEXT Z
150 PO=0
160 GOTO 40
```

The rope is at the top of the winch. Start the program with RUN. The motor does not yet move. Now press a key to lower the rope. The keyboard entry is inquired in line 40. The GET command reads the code for the pressed key and stores it in the variable A$. The key read in will not be used later in the program. This type of programming just waits for any key to pressed as a start signal. As soon as a key is pressed, the program finishs the GET-command and the rope lowers. The start position was at the top (PO=0). The program can then only continue to line 70. If the rope is at the bottom, the program waits for another keyboard entry. Press a key and the rope will rise because now PO=1 (line 60). The rope is wound clockwise on the drum.
The program is stopped this time by simultaneously pressing the CTRL and RESET keys. The Control-C keypress will not interrupt the GET command.
The rope can also be stopped half way by entering only 10 loops:

```
70 FOR Z=1 TO 10
120 FOR Z=1 TO 10
```

After entering RUN and pressing a key, the rope will run half way and back.
If we enter the number of steps after program start, the rope can be moved to any position. We therefore provide an INPUT command:

```
35 INPUT "STEPS (1-20):";S

70 FOR Z=1 TO S
120 FOR Z=1 TO S
```

After RUN enter the number of steps; this is stored in S. After pressing a key the rope runs for this number of steps and then reverses. The final value of the FOR...NEXT loops in lines 70 and 120 is the value of S. A nominal/actual value comparison of the number of steps can also be used to check whether the destination has been reached. Enter the step S and start the rope moving by pressing a key. Before, enter the following lines:

```
50 Z=0

70 &1B
80 Z=Z+1
90 IF Z<S THEN GOTO 70

120 &1F
130 Z=Z+1
140 IF Z<S THEN GOTO 120
```

In line 50 the step counter Z is set to 0. When the rope lowers, every step is counted in line 80 (Z=Z+1) and is compared with the nominal value S in line 90. As long as Z is less than S (Z<S), the program will continue with line 70: The rope then travels downwards. If the condition in line 90 is no longer fulfilled - this means that Z has reached the given number of steps - the motor stops. The program then jumps back to the start of line 40. The same thing applies to the upwards direction: The motor continues to turn until the number of steps has been reached.

You will frequently meet again this nominal/actual value comparison with the inquiry "greater than..." or "less than...".

Now tell the rope to drop by 20 steps and stop the program with the STOP key. With these new commands, we want to show you a practical application of an elavator traveling three floors. Make the elavator travel upwards and downwards using the up and down cursor keys. You can use the winch as an elavator. Fig. 4.4 shows an example of an elavator.

For the elevator program clear the memory and enter:

```
NEW
10 &IN
20 HOME
30 PO=0
40 UP$="I"
50 DOWN$="M"
55 VTAB 1
60 PRINT "FLOOR:";PO
70 GET A$
```

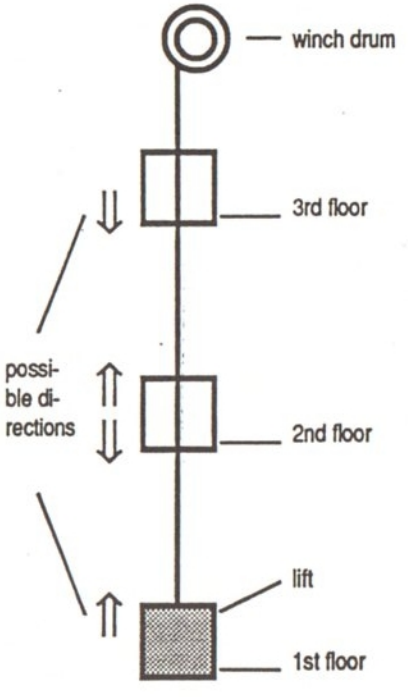


Fig. 4.4: Working principle of an elevator

The codes for going up and down have been chosen to be the characters I and M, respectively. On the Apple][+ there exist no up and down cursor keys, so that the for keys I, J, K, and M have come into use for directional commands. On the Apple //e and Apple //gs you may introduce the up and down cursor keys by modifying the following program lines:

40 UP$=CHR$(11)
50 DOWN$=CHR$(10)

```
80 IF A$=UP$ AND PO<>2 THEN
      GOTO 160
90 IF A$=DOWN$ AND PO<>0 THEN
      GOTO 110
100 GOTO 60
110 FOR Z=1 TO 10
120 &1B
130 NEXT Z
140 PO=PO-1
150 GOTO 60
160 FOR Z=1 TO 10
170 &1F
180 NEXT Z
190 PO=PO+1
200 GOTO 60
```

This time the rope is lowered completely, the elavator is at its home position at the bottom (position PO=0 in line 30). Start the program with RUN. The screen will display the floor which you are on. Enter the direction of the elavator: e.g. press the I-key, if you want go to up. The elavator will travel to floor 1 and this will be displayed. From floor 1 you can continue to go upwards or again go downwards. Going downwards is started by pressing the M-key. From floors 0 and 2 you can only go in the direction indicated in Fig. 4.4.
After setting the travel direction, the appropriate part of the program is selected in lines 80 and 90. The floor position is ANDed in the IF...THEN inquiries by the cursor key code. The codes for the cursor keys are fixed in lines 40 and 50.
If the elavator is standing on floor 2, it cannot travel upwards and from floor 0 it cannot travel downwards.
Stop the program again by pressing the keys Control-Reset.
Try it yourself by extending the program to other floors. Another application of motors with step control is a conveyor belt which runs forward a particular distance step by step. After every step it stops for an interval of time. Try to write a program for this by making the belt move forwards and backwards five steps at a time using the right and left cursor keys.
One variation of step control is positioning the motor by giving the destination position (nominal value). The program then notes the count (actual) position and decides by means of the nominal/actual value comparison what direction the motor needs to turn to reach its destination. The step number is used as position reference point. For this experiment use again the previous model and wind the rope completely onto the drum. It should be in position 0. Then

enter the following program:

```
NEW
10 £IN
20 Z=0
30 HOME
40 INPUT "POSITION (0-20):";S
50 IF Z<S THEN GOTO 80
60 IF Z>S THEN GOTO 130
70 GOTO 30
80 FOR I=Z+1 TO S
90 &1B
100 NEXT
110 Z=S
120 GOTO 30
130 FOR I=Z-1 TO S STEP -1
140 &1F
150 NEXT
160 Z=S
170 GOTO 30
```

The start position 0 is noted in line 20 by Z=0. After starting the program with RUN, give the desired position by entering a number between 0 and 20. The program then checks by comparing line 50 and 60 whether the destination position (nominal value) is greater or less than the start position (actual value). If Z<S, the motor will run the corresponding number of steps (lines 80-100). Don't worry that &1B is included: The rope is wound clockwise and therefore runs down. Then the program notes the new position as actual value in the variable Z (line 110) and jumps back to line 30. Afterwards it waits for an new entry.

If the nominal position is less than the actual value, the motor reverses (line 130-170). In line 130 the loop counts backwards (...STEP -1), that is to say, from the higher value Z to the smaller value S in steps of 1. Here, too, the destination position is fixed in order to determine the direction of rotation of the motor after the next entry.

The program is again stopped by pressing Control-Reset. The Control-C-key also does not interrupt the INPUT command.

As an exercise, you find two programs on the floppy disk: STEP.BAS program turns a motor forwards and backwards in steps; POSITION.BAS moves it to specific positions. The sequence is displayed each time on the monitor screen.

5 Switching with light

5.1 Non-contact switching: Fork Light Barrier

Instead of a mechanical pushbutton, a light barrier can also be used to measure the number of steps and to control a motor. It consists of a light emitter and receiver. When the light beam between the two components is interrupted, a signal is generated. This section explains how the light barrier can operate the motor and control it. First build the model Fork Light Barrier using the instructions in the assembly manual. The motor on the base frame now drives a vertical axle on which a disk with six slots is mounted. At the outer edge of the disk is the light barrier. A lamp connected to output M3 of the interface sends a beam of light to the receiver cell. This component, a light-sensitive resistor (technical term: photoconductive resistor), is connected to input E2. Check the general function by:

```
&IN
&1R
&1A
```

The motor turns the wheel a few centimeters forwards. After entering:

```
&3R
&3A
```

the lamp lights up for a short time. Before testing the function of the photoconductive cell, let's find out how it works.
The photoconductive cell is a resistor which is sensitive to light. It changes its resistance, that is its ability to conduct electricity, depending on the strength of the light falling on it. Turn the wheel on the axle until a slot is opposite the photoconductive cell. Then enter:

```
&DI
PRINT E2
```

The monitor will display "0". If not, the room may be too bright. Make sure that no direct light falls onto the photoconductive cell. Then switch the lamp on the model on:

```
10 &IN
20 HOME
30 &3R
50 &DI
60 PRINT E2;
70 GOTO 50
```

After RUN the function of the photoconductive cell is displayed: The monitor will display several "0" and then changes to "1". What does this mean? The photoconduc-

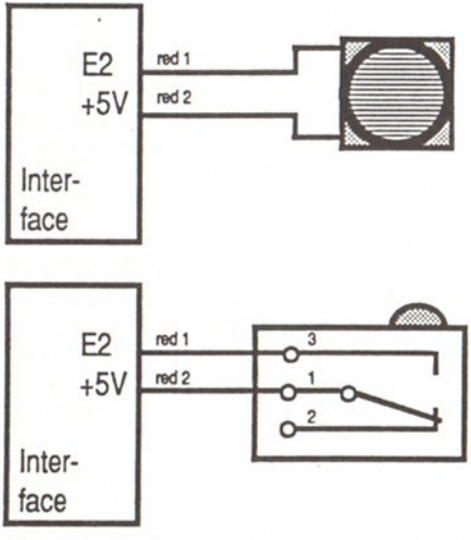


Fig. 5.1: A photoconductive cell may replace the pushbutton as shown in Fig. 4.3.

tive cell is connected to the interface input like a pushbutton as shown in Fig. 5.1.
As we know from the experiments in the previous chapter the program displays a "0", when the pushbutton contact is open. When the pushbutton is pressed, the display changes to a "1". That means that electric current can flow from the +5V terminal of the interface via the closed contact to the input E2. The same happens with the photoconductive cell: When light is received, it conducts the electric current like a closed contact and the display shows a "1". In this state, it is also said to be at low impedance. When there is no light falling onto it, its conductivity is poorer and the display is "0", corresponding to the switch position "open". Here it is said to be at high impedance. This switching effect is used by applying either very bright light to the photoconductive cell or no light at all. Scattered light, even from bright light bulbs, can affect the experiments.
The reason why a couple of "0" were displayed in the beginning of the program is that the lamp needs a short moment after the command &3R to reach its full brightness. This effect will have to be taken into account in the later programs by switching the lamp on for a short time before.

The switch effect of the light barrier can be checked by interrupting the light beam (Fig. 5.2). While the program is running, hold a dark sheet of paper between the lamp and the photoconductive cell. The display should change to "0". The photoconductive cell is cut off - it goes to high impedance.
It behaves like a pushbutton (make contact) with large changes of light. Its advantage is that it operates without contact, it has a fast reaction and has no wear like a pushbutton. The disadvantage is, of course, that it needs an additional light source.
The next experiment shows you how to control the motor with this light barrier. First turn the wheel until the path between the lamp and the LDR is blocked. After starting the program, it should display "0". Now stop the program by pressing Control-C and extend it by:

```
40 &1R
60 IF E2=0 THEN GOTO 50
70 &1X
80 &3X
```

Then start again with RUN. The motor will turn the disk until light falls onto the photoconductive cell through a slot in the wheel. Then both, motor and lamp, are switched

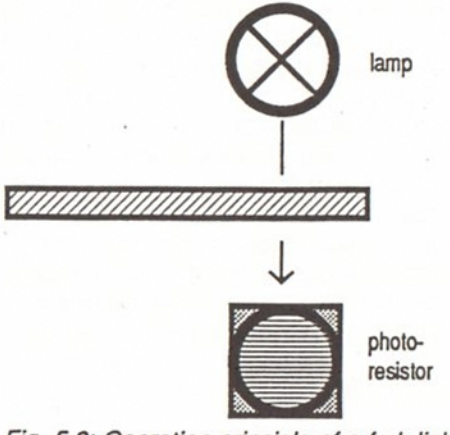


Fig. 5.2: Operation principle of a fork light barrier.

Light sensitive components are widely used in modern electronics and are introduced in every-day-products as well. The component in our experiments is also called a light dependent resistor (LDR). Besides this there exist also photo-diodes, photo-transistors and solar cells, which react on light input as well. The working principle of all those components is based on the physical phenomenon, that the light particles (photons) release electric particles (electrons) in the semiconductor material. You may find light sensitive components in exposure meters, in solar cell watches, in remote control of television receivers, but also in the modern data links using fibre optics.

off. We already met lines 40-70 in the last chapter and inserted a new command for them:

&1B

Then enter the new program below:

```
NEW
10 &IN
20 FOR Z=0 TO 100
30 &3R
40 NEXT Z
60 &1B
80 &3X
```

Start the program with RUN. The lamp is pre-heated for a while in the FOR...NEXT loop. The motor will then run until a slot in the disk passes again in front of the light. Since the disk has six slots, you can make it turn completely once with the following three lines and RUN:

```
50 FOR I=1 TO 6
70 NEXT I
```

The disk will turn in the other direction with the following modification:

60 &1F

On the floppy disk you'll find the program ANGLE.BAS which controls the disk in a similar way to the program POSITION.BAS for the winch. However, the program has a slight refinement. Since the disk reaches its start position after six steps, the program only accepts five different position references (it is not practical to give the starting position as the destination position!). The positions are also given in degrees: 0°, 60°, 120°, 180°, 240° and 300°. 360° no longer exists because this position is now given as 0°. Not only that, the program always searches for the shortest way to the destination. For example, from 60° to 240° it goes backwards over 0°.

5.2 Remote switching:
Reflecting Light Barrier

One experiment variation is the Reflecting Light Barrier. Here the light does not fall directly onto the photoconductive cell but is reflected by a bright surface. Fig. 5.3 shows the difference: The beam is interrupted by removing the reflecting surface. Change the model Fork Light Barrier from the last experiment into the model Reflecting Light Barrier using the instructions in the assembly manual. The lamp is now mounted below the wheel. It shines onto the disk and the light is reflected by bright segments attached onto the disk. These bright surfaces are at exactly the same place where the light shone through the wheel. The program should also run as before. Start it with RUN. The wheel turns once and stops. If this does not happen, maybe the distance between the wheel and the lamp is wrong. It can be adjusted by adding the following part:

```
200 &3R
210 HOME
220 VTAB 1
230 &DI
240 PRINT E2
250 GOTO 120
```

Turn the wheel so that a bright segment is located between the lamp and the photoconductive cell as shown in Fig. 5.3.
Now start the program by entering the command **GOTO 200** (not RUN!). Push the wheel onto the axle either upwards or downwards until the monitor displays "1". When you turn the wheel to the left and to the right now, the display should change between "1" and "0". The distance between the lamp and the wheel is now correct. Stop the test program using the Control-C keys and the wheel can be made to run again using the command RUN as usual.
Various applications can also be tried here as before: you can make the wheel turn alternately to the right and to the left. For this, enter:

```
50 FOR I=1 TO 3
60 &1F
70 NEXT I
80 FOR I=1 TO 6
90 &1B
100 NEXT I
110 FOR I=1 TO 6
120 &1F
130 NEXT I
140 GOTO 80
```

Mark a point on the wheel and start the program with RUN. The point will move back and forth through 360°.

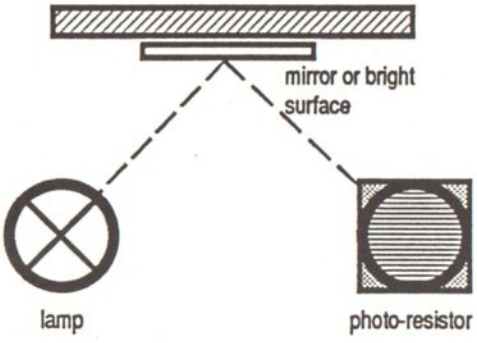


Fig. 5.3: Operation principle of a reflecting light barrier.

In practice, light barriers can be installed at many places: for alarm systems in the home, in car washes or as counters in assembly lines. Light barriers are also found in several floppy disk drives. They detect the start of a data track by a hole in the floppy disk. Turn an old 5¼" floppy disk by hand inside its sleeve. At a particular spot you will find the hole which is used to switch the light barrier.

6 Measurement and evaluation
6.1 Recording analog values: Lightmeter

Input circuits at computers, which are designed to register analog values, are called AD converters (analog to digital converter). The AD converters use widely different measuring principles, which also differ in expense, in precision and in operation speed. The AD converters in the fischertechnik interface use the same operation principles as the paddle inputs of some computers (A paddle is like a joystick which can smoothly be varied from one limit to the other). According the resistance of the external device, a timer unit in the interface generates pulses of corresponding duration which are passed on to an input line of the computer. The computer determines the duration by a counting procedure.

As you can see from the last experiment, working with light barriers for motor control is not as reliable as a pushbutton. Especially the reflecting light barrier was very sensitive to external light sources. Sometimes it failed to switch at all and sometimes it even stopped at undesired places.
In order to measure the exact behavior of the photoconductive cell with changes of light, build the model LIGHTMETER using the instructions in the assembly manual. You don't need to dismantle the base frame and the motor. The remaining components are enough for the LIGHTMETER. The photoconductive cell is located in the holder in front. This time it is connected to the analog input EX (orange wire). This input, as opposed to the digital input which we used in experiments with the light barrier, detects analog readings. These are, for example, voltages which continuously change between a minimum and a maximum. Our interface is designed so that a variable resistor can be switched between the analog input and the + 5 V line. The value of the resistor is converted into a numeric figure which the computer can read and process.
Connect the model to the interface and load the BASIC extension program. In order to read an analog value - i.e. a resistance - via input EX, enter:

```
&IN
&EX
```

With &IN the interface is reset and with &EX the value of the photoconductive cell is stored in variable EX in the computer. It can be displayed by:

```
PRINT EX
```

The following program can find out whether the photoconductive cell reacts to changes in light:

```
10 &IN
20 PRINT "MEASURED VALUE:";
30 &EX
40 PRINT EX
50 GET A$
60 GOTO 20
```

Enter the lines and start the program with RUN. Then swing the photoconductive cell to and fro so that it is exposed to different amounts of light. Again and again press any arbitrary key of the computer's keyboard to trigger the next reading and display. The figures displayed by the monitor will differ. It

can be clearly seen that with every change in brightness the figure becomes larger or smaller. There are no longer two stable states as with the pushbutton (ON and OFF). This explains the different switch behavior of the light barrier with changes in brightness.

The exact relationship between light intensity and resistance (reading) can be obtained by making a series of readings. Hold the photoconductive cell to the brightest source of light in the room (e.g. towards the window) and start the program with RUN. The corresponding reading will be displayed. Now turn the photoconductive cell away from the light a little bit and measure again by pressing a key. This figure will be displayed under the first.

Turn the LIGHTMETER one step further into the darkness and measure the light intensity again. Repeat the whole procedure once more until the photoconductive cell is pointing into the darkest part of the room.

Then stop the program by pressing Control Reset. The table in the monitor should look like the one in Fig. 6.1. The only addition is the luminosity.

Of course, your readings will be somewhat different because each room has a different light intensity. The only important thing is the graduation from bright to dark. The best way to follow the progression of change in resistance is with an X/Y graph (Fig. 6.2). Here, a dot is entered for every light intensity reading in the table. By connecting the dots you get a curve, called the characteristic of the photoconductive cell. You can see that when pointed in the direction of greater luminosity, the curve has a non-linear drop - this special type of curve is called to decay exponentially. In data books you will find characteristics with exact luminosities and resistances for each photoconductive cell (Fig. 6.3). The designer can then select and tune the right photoconductive cell for his/her circuit, e.g. a lightmeter. As a practical application, we want to show you how to provide your LIGHTMETER with an indicator. Measure the light intensity in the room and display it as bars on the monitor screen. To start with, the exact luminosity in lux or candela (these are the units for luminosity) is not important. The program will display a lot of light by a long bar and less light by a short bar. Enter the following program:

```
NEW
10 £IN
20 GET A$
```

MEASURED VALUE:

58	bright
87	
123	
174	
255	dark

Fig. 6.1: Series of light intensity readings

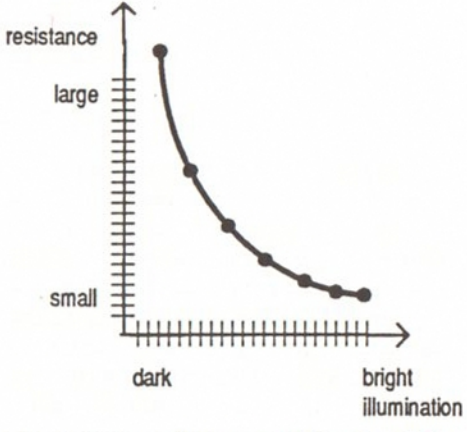


Fig. 6.2: General characteristic of a photoconductive cell

```
30 &EX
40 HM=EX
50 HE=EX
60 &EX
70 IF EX=HE THEN GOTO 60
80 S=40*HM/EX
90 HOME
100 INVERSE
110 FOR I=1 TO S
120 PRINT " ";
130 NEXT I
140 NORMAL
150 GOTO 50
```

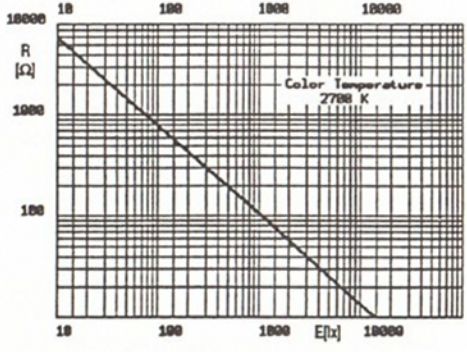


Fig. 6.3: Data sheet of a photoconductive cell

In order to make the brightest part in the room record a bar which covers the entire screen width, it must first be measured. Turn the LIGHTMETER into direction of the brightest light and start the program with RUN. In line 20 there begins the part of the program for measuring the brightest spot in the room. Start the process by pressing a key.
On line 30 the luminosity is measured and stored in the variable HM. You will need this value later as a reference for every reading. It also stores the measured light intensity in the variable HE.
Then measure the luminosity in the direction in which the photoconductive cell is pointing and compare this reading with the previous one in line 70. The last reading is always contained in the variable HE. At the start it was HM (line 50). Only if the actual reading contained in EX differs from the last reading, there is the necessity to display the bar. Calculate the bar length from the reading. Let´s assume that the actual measurement reports half luminosity compared to the brightest spot, so HE is approximately double the size of HM. This results in

S = 40 * 1 / 2 = 20

The bar has half the length compared to maximum.
Line 90 clears the monitor screen. Then the program displays S spaces (FOR I=1 TO S) in sequence. This bar consists of inverse blank spaces because of the INVERSE command in line 100.
After this display there is a new reading (GOTO 50 in line 150).
Now turn the LIGHTMETER into different directions with the program running. Each luminosity will be displayed as a bar. Of course, this takes time and you shouldn't turn the photoconductive cell too fast.
Try yourself to display the bars line by line thus obtaining a record of change of light

intensity. If you want to display the actual luminosity, you have to tune the photoconductive cell or the measuring system. For this you need to know the characteristic of the photoconductive cell. Using comparison resistors instead of the photoconductive cell, you can find out what figure corresponds to what resistance. You can see that there is a lot to experiment with here!

So far the cap with the tube was always mounted in front of the photoconductive cell. This arrangement limits the light acceptance angle of the LIGHTMETER so that it can be pointed directly at an object. This arrangement is also called a collimator. Sometimes a different measuring problem arises. It is not the luminosity of an object that needs to be measured but the light which illuminates an object. In this case the LIGHTMETER becomes the object and is directed at the light source(s). Where there are several light sources, the LIGHTMETER must measure the incidence of light from all sides. Therefore the collimator is replaced by a diffusion screen which is shaped like a semi-transparent white cap. Such an arrangement is also called diffusor.

Now you can use the software developed above or the program EXPOSURE.BAS on the floppy disk to verify whether the LIGHTMETER still reacts as sensitively when pointed at an object.

In the measurement of light there exist a lot of different physical units. Each unit correlates to a different questioning or measuring principle.

The light flux is denoted in lumen (lm). The light intensity, that is the light flux radiated in a particuliar direction, will be quoted in units of candela (cd).

The density of light, that is the light flux incident onto a certain area, will use the unit lux (lx, 1 lx = 1 lm/m²).

In order to judge, how bright our eye or the photoconductive cell will register an object not only depends on the illumination of that object but also on our distance from the object and on the aperture of our eye or photoconductive cell. This will be measured using units of stilb (sb, 1 sb = 1 cd/cm²). One stilb corresponds to the visual impressions at bright daylight; the human eye, however, still can record stimuli of one millionth of a stilb.

6.2 Automatic light measurement: COMPUTER EYE

With the LIGHTMETER in the last chapter you were able to measure the brightness in every direction in your room and to display the result on the monitor screen. The photoconductive cell was turned by hand and the result was displayed on the screen in the form of bars of different lengths.
Now we want to show you how to perform this measurement automatically. Assemble the model COMPUTER EYE shown in your assembly manual. On the base frame there is a motor which turns a vertical axle via a worm drive. The photoconductive cell is mounted on top of the axle and the drive allows it to turn in all directions. The motor operates according to the step control principle which can be seen by the pushbutton at the side of the base frame.
First test the functions of the motor and the photoconductive cell before starting with automatic light measurement. Enter:

```
10 &IN
20 FOR I=1 TO 10
30 &1F
40 NEXT I
```

After RUN the vertical axle should turn through 90°. Make sure that the cable to the photoconductive cell doesn't jam. Now you know the ratio between motor steps and turn angle of the meter: 10 steps (FOR I=1 TO 10) turn the cell to 90°. Therefore, one step means a turn of 9°. This angle results from the gear ratio. The command £1F turns the worm wheel through 1/2 revolution. One revolution of the worm wheel turns the gear wheel by one gear tooth. The gear tooth has 20 teeth. Therefore:

360°/20/2 = 9°

One full revolution of 360° can therefore be obtained by :

```
20 FOR I=1 TO 40
```

and RUN. Don't forget the cable to the photoelectric cell! In an emergency, stop the program with the Control-C key. The COMPUTER EYE can be turned back by:

```
30 &1B
```

and RUN. The light sensor can be turned to a particular position by:

```
NEW
10 &IN
20 R$=CHR$(21)
```

```
30 L$=CHR$(8)
40 GET A$
50 IF A$=R$ THEN &1B
60 IF A$=L$ THEN &1F
100 GOTO 40
```

In lines 20 and 30 the keyboard codes for the left and right cursors are assigned to the variables R$ and L$. The function CHR$(...) is used, as the codes for cursor operation cannot be entered directly in between quotation marks like other characters.
Start the program with RUN. Now you can operate the cursor keys "right" or "left". In lines 40 through 60 the pressed cursor key is detected and the associated step is executed. The program runs in the loop (GOTO 40) until you press the Control-Reset key.
The photoconductive cell is connected as before to the analog input EX. It is inquired by:

```
&EX
PRINT EX
```

After you enter this line, press RETURN and a number will appear on the screen corresponding to the brightness which the COMPUTER EYE sees. If the cell is turned, the display should change depending on the luminosity. Add the following lines to the program:

```
70 &EX
80 HOME
90 PRINT EX
```

and start with RUN. You can change the direction of view of the cell using the two cursor keys. The luminosity measured is displayed each time. Even a full automatic rotation through 360° with measurement and display is possible. After stopping using Control-C, enter:

```
NEW
10 &IN
20 R=0
30 GET A$
40 HOME
50 IF R=1 THEN GOTO 130
60 FOR I=1 TO 40
70 &1F
80 &EX
90 PRINT EX,
100 NEXT I
110 R=1
120 GOTO 30
130 FOR I=1 TO 40
```

```
140 &1B
150 &EX
160 PRINT EX,
170 NEXT I
180 R=0
190 GOTO 30
```

After starting the program with RUN the program waits for you to press a key. Then the program continues to line 50. Here it checks the direction in which the cell should turn. If it turned to the right the first time, it will now turn to the left and vice versa. There is included what is known as a "flag": the variable R. At the start it is set to 0 (line 20). Therefore, the first turn is to the right (lines 60 - 120). After, the flag R is given the value 1 (line 110). At the next inquiry in line 50, the program jumps to line 130. The cell turns back. Then it turns to the right again because R is set to 0 (line 180).
During the turn the readings are written next to each other on the screen - one reading for each step (9°). The smallest number corresponds to the greatest brightness as you saw in the last chapter.
The figures will be more understandable if each brightness reading is given a corresponding direction. The best way to do this is with an appropriate graphics display. For example, the brightness could be shown in each direction (starting with 0° at the top of the screen). The readings for greater luminosity should be displayed by marks which are further away from the center of the screen. This requires the use of high-resolution graphics. A detailed explanation of the programming would take up too much space in this booklet. Therefore we have provided you with a simple painting tool.
On the floppy disk there is a complete program called SCANGRA.BAS, which generates a graphics display of the above light measurement. It uses the painting tool and the next chapter will tell you how it functions.

In addition to the standard text monitor screen with 25 lines and 40 characters per line your computer is also equipped with a graphics screen capability. For this, it divides the screen into 280 x 160 picture elements (pixels) and each is individually addressable. This allows quite good pictures and graphics. On a color display you even may generate colored pictures.
The graphics tools are activated by entering:

&GON

This clears the screen and in the middle appears a small triangle which we call the graphics turtle. It points upwards showing the next direction of movement.
The bottom four lines of the screen continue to be used for text entry and display. Just imagine that this turtle is your hand and the screen is a sheet of drawing paper. In your hand you have a pencil which you can draw with. You can move the pencil anywhere on the paper and draw dots or lines. This is exactly what the graphics turtle can do. First let's draw a vertical line. For this you need to enter:

10 &IN
20 &GON
30 &GF,20
40 &G0

After RUN the turtle moves forwards (up) and leaves a line on the screen. In line 30 the turtle moves 20 steps forwards and draws a line of 20 dots. Then the graphics pen is switched off - you lift the pencil off the paper. Please note: The command &G0 ends with a zero, not the letter O.
If you move the turtle 10 steps forwards with

&GF,10

it leaves no mark (the graphics pen was switched off). In this way the pen can be placed in any position on the screen to draw dots and lines. If you want to continue drawing, you must switch the graphics pen on again, in other words place the pencil on the paper:

&G1

You can also turn the graphics turtle with:

&GR,90

Now it turns by 90° to the right. If you enter:

If you are familiar with the programming language LOGO, you'll certainly recognize this symbol. The turtle was first introduced in this powerful programming language. Turtle graphics was later on implemented in couple of other computer languages like PASCAL, COMAL, and now also in BASIC since it allows the construction of graphic displays without too much detail knowledge of mathematics. If you want to experiment more using turtle graphics you should study the literature related to LOGO.

Don´t worry that some graphic commands appear to be torn into parts, when you list the program. The reason is that letter combinations like ON, GR and END have a special meaning for the BASIC interpreter and it wants to format them using leading and trailing blanks. Nevertheless the graphic commands are recognized and executed correctly.

Summary of all graphics commands:

&GON : Switch on/clear graphics
&GEND : Switch off graphics, return to text screen
&GF,S : Move graphics turtle by S steps forwards
&GB,S : Move graphics turtle by S steps backwards
&GR,D : Turn graphics turtle by D degrees to the right
&GL,D : Turn graphics turtle by D degrees to the left
&G1 : Switch on graphics pen
&G0 : Switch off graphics pen
&GC : Copy graphics (background graphics is displayed)
&GP,C<,P> : Select color C (0-3) and palette P(1,2) of graphics pen. Palette needs not to be given; default is palette 1.
&GS,C : Select color C (0-3) of background; becomes effective with next &GON

(continued overleaf)

&GF,20

it will draw a line to the right. If you enter:

&GL,45

it will turn through 45° to the left. Change the program as follows:

30 &GB,30

Now start with RUN. The graphics turtle turns back and draws a line of 30 dots.
You can also draw a square. Please enter:

```
NEW
10 &IN
20 &GON
30 FOR I=1 TO 4
40 &GF,30
50 &GR,90
60 NEXT I
70 &G0
```

The last two lines of what you enter are always displayed. After issuing RUN the graphics turtle draws a square on the screen. You can also draw a larger one:

40 &GF,40

After starting the program with RUN, a larger square appears on the screen. Enter:

```
25 FOR K=1 TO 9
70 &GR,40
80 NEXT K
```

After RUN, several squares will be drawn on the screen each turned through 40° (line 70). Now you can see how easy it is to draw with the graphics turtle.
You can select four different colors of one of the palettes below for the pen and the background. On the Apple II they have the following codes:

	Palette 1	Palette 2
0:	black	black
1:	green	brown
2:	magenta	blue
3:	white	white

(The colors may differ depending on your monitor´s or TV adjustment.)
With the following lines you can color the background green:

```
&GS,1
&GON
```

The color pen is changed to black before moving the graphics turtle by:

&GP,0
&GF,30

Be careful not to give the turtle the same color as the background - the tracks left by the turtle will then be invisible in spite of the command &G1!
Changing the color palette is performed by adding a the pallette number to a &GP command, i.e.:

&GP,0,2

This will immediatly flip all colors from the first palette to the second one.
You can also inquire the actual position of the graphics turtle by:

&GH
&GX
&GY

The actual heading of the graphics turtle is fixed by the variable GH in degrees to the angle of the start direction. GX and GY contain the X- and Y-coordinates of the graphics turtle position. For this enter:

&GON
&GF,20
&GR,90
&GF,10

The graphics turtle has now moved forwards and is pointing to the right. The actual direction is displayed as 90 (°) by:

&GH
PRINT GH

The following lines show the position:

&GX
&GY
PRINT GX,GY

With the command

&GT

the graphics turtle senses its way along. The variable GT contains the code for the color value to which the turtle was last moved. It can therefore recognize marks drawn previously.
Now restart the program with RUN. The screen now displays again the star made up of squares. Enter the command &GSAVE together with a file name in quotation marks, seperated by a comma, e.g.:

&GH : Store actual course of graphics turtle in GH
&GX : Store X-coordinate of graphics turtle in GX
&GY : Store Y-coordinate of graphics turtle in GY
&GT : Inquiry dot under graphics turtle; dot color stored in GT
&GSAVE : Store actual graphics picture on floppy disk
&GLOAD : Load graphics picture from floppy disk
&GPRINT :Print out graphics picture on printer

If you are observant, you will recognize that the turtle leaves traces that are two pixels wide in X-direction. This is due to color control which needs two neighbouring pixels to generate the colors. On a monochrome monitor you will of course see no colors, but find out that "white" triggers two pixels, "green" the leftmost pixel and "magenta" the rightmost pixel.

&GSAVE,"STAR"

The graphics picture is now stored on the floppy disk under the filename STAR.PIC. The file name extension ".PIC" denotes the file as a picture file. The extension ".PIC" will be appended automatically, you don't need to worry about it. Then enter:

&GON

to clear the screen and you can now continue to experiment with the graphics turtle. If you want to load the stored picture again, enter the command &GLOAD together with the file name. Again you don't type in the ".PIC" extension.

&GLOAD,"STAR"

You won't see the loaded picture right away. It is, so to speak, behind the scenes. It only becomes visible when you enter:

&GC

In actual fact, it is copied from the invisible loader memory into the visible screen memory. This in fact saves you a lot of trouble. Just imagine that you had to load a graphic such as the screen of the computer eye. Readings are now entered onto the screen. If you want to clear these readings you don't need to clear the whole screen and reload the picture from the floppy disk. The command &GC copies a fresh radar screen from the loader memory and you can start plotting the next readings.
You can also copy the screen contents onto your printer. Specialists normally talk about a "hard copy". Our hard copy even shows you the color of the lines by means of various print patterns in gray tones. Just enter the command:

&GPRINT

In this way you can print out the readings of your experiments on paper. The screen graphics is switched off by:

&GEND

Then you'll find yourself back in the standard text screen.

So far we have used external light sources for measuring light. The light was sampled from objects illuminated by the sun or the room lamp, and its luminosity was displayed. Now if light is beamed from the model onto an object and the reflected light is measured, it could be possible to use this measurement to discover how far the object is from the model. If the distance from the photoconductive cell is increased, the luminosity will decrease. Let's see if we can develop a radar with the following experiment.

First assemble the RADAR model in the assembly manual. It looks similar to the model COMPUTER EYE but it has an extra lamp above the photoconductive cell. It is connected to terminal M3 of the interface and lights up when you enter:

```
&IN
&3R
```

Don't forget to switch it off with:

```
&3X
```

Now take a bright object, e.g. a white shoe box, and place it 10 cm in front of the model as shown in Fig. 6.4.

After entering RUN the luminosity reflected by the bright box can be measured using the following program:

```
10 &IN
20 HOME
30 &3R
40 &EX
50 PRINT EX
60 &3R
70 IF PEEK(-16384) < 128 THEN GOTO 60
80 POKE -16368,0
90 GOTO 20
```

The light is beamed out by the lamp on the model. This means that no external light should fall on the box. You must therefore darken the room a little. The display now corresponds to a distance of 10 cm. Be careful: The first reading does not give you the correct result. The measuring equipment must first stabilize, as you can see in the program change:

```
70 GOTO 40
```

Due to the inertia of the lamp and the photoconductive cell (see also Chapter 5), the display only becomes stable after 10 to 15 readings. Restore line 70:

The function PEEK(...) is used to find out the contents of a specific memory location. In the present context it tests if a key has been pressed at the keyboard.
The POKE command resets the keyboard register so that it is ready to record the next keypress.
You should avoid using PEEK and POKE in your programs, they are resources if something cannot be programmed in another way.
In the present case the program has to loop through the &3R command while testing the keyboard; therefore the GET command was not useable.

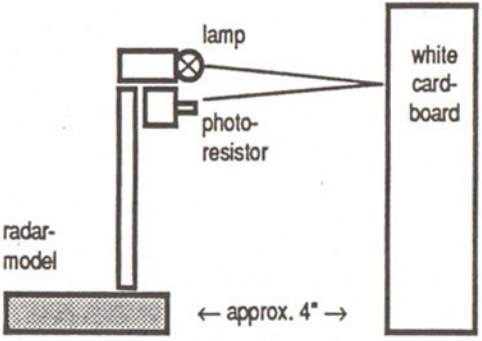


Fig. 6.4: Distance measurement with radar model

```
70 IF PEEK(-16384) < 128 THEN GOTO 60
```

Note down the value which the program now displays after the second keypress and increase the distance between the model and the box to 20 cm. A value is then displayed which is approx. double the previous value. If we approach the box to 5 cm, the value displayed will only be approx. half the first reading when it was at 10 cm. What is important here is not the exact figure but the principle itself. However you should make sure that the numbers are in the usual range of analog values of the interface, which is 20 to 255. If you obtain 255 all the time, you should remove the tube from the stray light cap on top of the photoconductive cell. Thereafter the readings will lower, since the photoconductive cell receives more light.

To summarize, we can say that by means of the luminosity reflected off an object, a very rough estimation can be made of its distance from the point of measurement. At the same time, a known distance and the associated luminosity can be used as a comparison. When measuring several positions, it is important that they all have the same gray tone and that no external light falls onto them. Just try this out with two boxes set up as in Fig. 6.5. At the same time, turn the RADAR system around in the same way as you did with the COMPUTER EYE. In fact, it is the same program and only the evaluation has been changed:

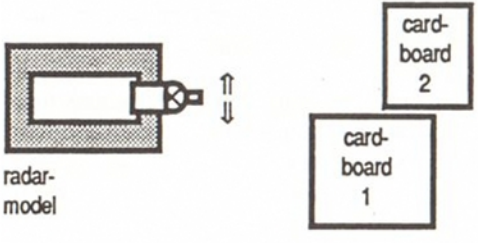


Fig. 6.5: Radar scanning of several objects

```
NEW

10 &IN
20 &GON
30 FOR I=1 TO 30
40 &3R
50 NEXT I
60 FOR I=1 TO 40
70 &1F
80 &GL,9
90 &EX
100 &G0
110 &GF,EX/4
120 &G1
130 &GR,90
140 &GF,5
150 &GB,5
160 &G0
170 &GL,90
180 &GB,EX/4
190 NEXT I
200 FOR I=1 TO 40
210 &1B
220 NEXT I
```

```
230 PRINT " EXIT PROGRAM HITTING
      ANY KEY"
240 GET A$
250 &GEND
260 &3X
270 END
```

The luminosity measured is converted in this program into a distance of travel for the turtle. As you already saw previously, the less light is reflected back to the photoconductive cell, the further the object is away, and the larger the analog value of EX. Therefore, you can make the turtle move further away from the screen center, the greater the analog value of EX. This is done with the pen switched off. Finally, the graphics urtle draws a small bar on the screen and jumps back to its start position. For the next measuring point, the RADAR system turns through 9°. The turtle is also turned through the same angle.

Enter RUN to try out the program. As the RADAR model turns, you will clearly see the difference in distance.

You can load a similar program called RADAR.BAS which is on your floppy disk.

This experiment very clearly shows the basic principle of a radar device. The only thing is that these devices don't work with light but with radio waves. These are somewhat longer electromagnetic waves but in principle the same waves like light waves. Also the method of measuring differs. While we use in our experiment the decay of light intensity with distance, a radar station measures the time lag from sending the waves till detecting the echo. Radio waves, like light waves, propagate at a velocity of 300 000 km per second. Though the time lag is rather short due the high propagation velocity, however it is still precisely measureable using specific electronic circuits.

7 Measurement and control

7.1 Measuring temperatures: THERMOMETER

Now we come to another unit of physics: temperature. In this section you will learn how to measure temperature and convert it into an electrical value so that the computer can understand it. And then you will deal with temperature control. You meet it everywhere today: e.g. in the refrigerator which maintains a preset temperature or in the heater which guarantees a constant room temperature or the cooling fan in the car which ensures that the engine doesn't overheat.

To measure the temperature, a temperature-dependent resistor is used - specialists call it an NTC resistor or thermistor.

Its value changes when the ambient temperature rises or drops. If a constant voltage is applied to this resistor, the current will be different depending on the temperature.

We want to show you in the following experiment how easy temperature measurement is with a thermistor. First assemble the model THERMOMETER using the instructions in the assembly manual. The resistor is connected to the interface between +5V (green wire) and the analog input EY (yellow wire). The analog input line EX, which could have been used as well, is disabled to avoid any disturbance. Disabling is achieved by connecting the input line EX with the reference voltage +5V. If you have connected everything up correctly, you can start with the first measurement. Enter the following in the computer:

&IN
&EY
PRINT EY

A number between 20 and 200 should now appear on the screen. If this happens, you have measured a temperature for the first time electronically by computer.

If an error message (SYNTAX ERROR or similar) appears after entering &IN, check whether the machine program for the BASIC command extension is loaded in your computer. After the command &IN, the prompt] should appear. If not, reload the program as described at the beginning.

Now to the reading. The number you can see on the screen is the ambient temperature. The following test will show that the thermistor really changes with temperature.

Enter the following program lines:

5 &IN
10 HOME

NTC means negative temperature coefficient. The NTC resistor or thermistor conducts electricity better, the hotter it is. A thermistor consists of a ceramic material, the components of which are the oxides of manganese, iron, cobalt, nickel, copper and zinc, which show a strong dependence of their electrical conductivity on the ambient temperature.

```
20 VTAB 1
30 &EY
40 PRINT EY;" "
50 GOTO 20
```

and start the program with RUN.
Line 10 clears the screen. Line 20 sets the print-out position to the top left corner of the screen. Line 30 reads the value at the analog input EY - i.e. the resistance of the thermistor - and line 40 displays this value on the screen. Then the program jumps back to line 20. The program is prevented from jumping back to the command to clear the screen. This would have made the screen display flicker. Instead the print-out position is set to overwrite the previous result. However, remember that the display can sometimes have two digits or even three digits. With the extra space character in line 40 the new print-out always covers the previous one.
After RUN the current temperature reading will be continuously measured and displayed. As you can see, the reading doesn't change very fast. This is understandable because the room temperature is more or less constant. Now simply hold the thermistor between your thumb and first finger so that it warms up to your body temperature. After a short while, the reading changes and becomes smaller. If you let the thermistor go, the reading will slowly change back to the old value which was room temperature. Then stop the program using the Control-C key.
This experiment shows that the thermistor reacts to ambient temperature. The only thing is that the reading doesn't indicate the true temperature in degrees Fahrenheit or Celsius. How come?
The interface between the thermistor and the computer which converts the resistance value into understandable digital signals for the computer is not calibrated. The problem is that calibrating the interface is a very complicated process requiring the skills of an electronics expert. So just let the computer do the converting using a special program. This will convert the original values into degrees Fahrenheit. In the ongoing we will show the conversion to the Celsius (or centigrade) scale as well. The expert here talks about a software solution. If the interface was modified, this would have been a hardware solution.
Now let's go back to the calibration. Here you need a beaker filled with water. Immerse the thermistor into the beaker together with a household thermometer. Run

Temperature (°F)	Value EY
32	148
40	131
50	116
60	102
70	90
80	79
90	70
100	62
110	55
120	48

Fig. 7.1: Table of temperature readings measured using thermometer and thermistor. The actual readings may differ in your experiment (see text).

the above program and note down the reading on the screen and the temperature reading on the thermometer.

Before starting with the experiment, pack the thermistor in a plastic bag so that no short-circuit occurs through the water. Then put the thermometer into the plastic bag as well so that you have the same measuring conditions. The photo in the assembly manual shows you the setup of the tuning configuration. Be very careful during the experiment that not a drop of water goes onto your computer - it could trigger a short-circuit. For this reason set up the beaker as far away from your computer as possible. Put the beaker also in a bowl to catch the water if the beaker tips over. Have towels and kitchen paper at the ready.

To start with, fill the beaker half full with water and put a few ice cubes into it from the refrigerator. This cools the water down almost to the 32°F threshold. Start the program with RUN and note down the thermometer and screen readings on a bit of paper. Now warm the water up by pouring a little bit of hot water in at a time (be careful not to spill any water). When the temperature has stabilized, measure and note down the new reading. Keep on performing the measurements until the water has reached about 120°F.

Now stop the program with the STOP key. You should have a series of readings showing the value of the thermistor as a function of temperature. It should look like the list in Fig. 7.1. Slight deviations are allowed because the conversion of the resistance value into a figure depends on the scatter of a number of electronic components in the interface.

Take the thermistor out of the beaker and let it cool down to room temperature. Restart the program (RUN). The display will show a figure which the computer will convert into the correct temperature value. For this, use the table of readings which you have just made. A reading of 90 is equal to 70°F, a reading of 55 equals 110°F and a reading of 148 equals 32°F. As you can see, the two lists of numbers run in opposite directions with the highest temperature allocated to the lowest EY value and vice versa.

In addition, in the jargon of the expert, the characteristic of the thermistor is non-linear. You will see this if you plot the values on an X/Y graph and join the dots. Have a go! You'll see that the characteristic is not a straight line.

By making several experiments, you could

try to find out the correct equation for the conversion - but we'll give you a rather simple approximate solution. Enter:

```
15 DEF FNT (X) = 427-79*LOG (X)
40 PRINT "TEMPERATURE= ";FNT
      (EY);" DEGREES F "
```

and start the program with RUN. In line 15 which we have added, there is a conversion formula from analog values to degrees Fahrenheit (DEF FNT...). It also contains the function LOG which stands for natural logarithms. Line 40 then prints out the converted temperature value instead of the EY value. Now the display looks a bit better.
But even this formula is still not precise. Because of the scatter of the thermistors and the interface components you should work out the exact conversion function for your computer. Do this by using the program CALIBR.BAS on the floppy disk. Enter the actual table value (degrees Fahrenheit and analogue value EY) which you obtained during your experiment. The program calculates the most suitable conversion functions for the Fahrenheit and the Celsius scale, respectively and displays them on the screen. Make a note of the functions. You can use them anywhere for the experiments described in this handbook. The programs on the floppy disk use the Celsius scale instead of the Fahrenheit scale. By replacing the line:

```
DEF FNT (X) = 200-40*LOG (X)
```

with the appropriate Celsius formula displayed by CALIBR.BAS you will provide a more accurate reading for those programs as well.
Now you have built an electronic thermometer with a computer readout indicating the ambient temperature. You could even use this model as a thermometer in a weather station. And if you manage to indicate the time for each reading and print the whole thing out continuously, you would then have a temperature plotter with which you could perhaps control your heating. But we don't want to go that far here. Instead, we want to show you how to indicate temperature graphically on the screen. You already learned the necessary graphics commands in Chapter 6. For this you need to append two subprograms which the main program can call up:

```
17 GOSUB 300
20 VTAB 21
```

In line 15 we have taken into consideration that the temperature coefficient of the thermistor is negative. Remember - the higher the temperature, the lower the reading. The logarithm has to be used to compensate the exponential decay of the resistance of the thermistor:

$R = R_N * e^{B/T}$

R is the resistance and T is the absolute temperature in Kelvin (see below); B and R_N in the above formula are constants of the material. The symbol e stands for the exponential function.
So if you are an expert in mathematics you will find out that the calibration formulas are not exact, but an approximate solution.

The Celsius scale of temperature (°C) is adjusted according the melting point and the boiling point of water. The temperature range in between has been divided into hundred evenly spaced steps, each representing one degree Celsius (1°C). The Fahrenheit scale uses different fixed temperatures. 100°F was defined as to be the human blood temperature; 0°F as the lowest temperature to be achieved at that time using a solution of some salt in water. The Kelvin scale (K - without a degree sign) is of later origin. When scientists detected that there exists an absolutly lowest temperature, -273.15°C, this temperature was defined as to be zero Kelvin. The spacing of the Kelvin scale is the same as for the Celsius scale.
The conversion factors for each of the temperature scales are:
0 K = -273.15 °C
32...212 °F = 0...100 °C
or - in precise mathematical terms -
C = 5/9 (F - 32)
where C = temperature in °C and F = temperature in °F.

```
48 GOSUB 200
```

Subprograms of this kind are always used when certain program sequences are required several times. Subprograms are also of special advantage, when you can identify them by a specific action like in the present case. Subprograms are called in using the GOSUB command.
Now add the subprograms themselves, each of which is concluded by the RETURN command.

```
200 &GX
210 IF GX=139 THEN GOSUB 300
220 &GF,FNT(EY)
230 &G1
240 &GF,1
250 &G0
260 &GB,FNT(EY)+1
270 &GR,90
280 &GF,1
290 &GL,90
295 RETURN

300 &GON
310 &G0
320 &GB,79
330 &GL,90
340 &GF,139
350 &GR,90
360 RETURN
```

The first subprogram from line 200 makes the graphics turtle move upwards by the temperature value while the pen is switched off. Then it switches the pen on and moves the turtle one step forwards. This marks a dot on the screen which is so many pixels from the bottom edge as corresponds to the temperature. The graphics turtle is then returned to the bottom screen edge, turned to the right and placed in the next colomn. Then it is again turned to the left. Finally, the graphics turtle is ready to accept the next task.
When the screen page is full or before initial use, the screen must be cleared. This is performed by the subprogram as from line 300. After switching on the graphics, the position of the turtle is detected. It can then be placed in the bottom left corner of the screen. By the way, the figures 79 and 139 refer to the screen measurements of the Apple II.
The numeric temperature display has been shifted to screen line 21, which is the first of the remaining four text lines (see program line 20).
The temperature reading can also be made

even clearer. For example, you can certainly omit the temperature range below 60°F and spread the range between 60°F and 100°F over the entire screen height. In order to do this, change the following lines:

```
220 &GF, (FNT(EY)-60)*4
260 &GB, (FNT(EY)-60)*4+1
```

We would now like to show you how to adapt the program to international requirements and indicate the temperature not only in Fahrenheit but also in Celsius. For those interested in science you can also give the temperature in Kelvin.
This extends the program so that after each reading the temperature is displayed on the screen simultaneously in degrees Celsius, Kelvin and Fahrenheit:

```
42 TC=(FNT(EY)-32)*5/9
44 PRINT "TEMPERATURE= ";TC;
       " DEGREES C  "
46 PRINT "TEMPERATURE= "; TC+
       273.15;" KELVIN"
```

After starting with RUN the current temperature is displayed in all three units.

In the last chapter you saw how to measure temperatures with a thermistor and how to display them on the computer. The computer can do much more: You also make it control heat supply. The temperature will then change only minimally. In practice, this is none other than a heating control. You will also learn what the experts call a control circuit.
For this experiment assemble the model FURNACE in the assembly instructions. Connect up the model to the interface. Before starting, make a careful check that everything is correctly wired up and that there is no short-circuit. After loading the basic extension program, enter:

```
&IN
```

The computer will reply with its prompt].
If the computer displays an error message, reload the program as described and start again with the command &IN.
Let's have a closer look at the model. It consists of a lamp above which is the thermistor. The bulb is used here as the "burner" for the heater. This is no wonder because even a small lamp like this not only radiates light but also heat. If you don't believe this, just touch the light bulb of a

floor lamp which has been on for some time. But be careful, it's hot!
The thermistor acts as the temperature sensor for the heat produced by the burner. The task is to maintain the temperature at a specific constant value. This is done by setting up a control circuit: Enter a nominal temperature and measure the actual temperature of the burner. If the burner temperature attains the nominal temperature - this is reported by the thermistor - the burner should be switched off. If the temperature is undershot after cooling down, the burner should be switched on again.
Let's just try this out. The lamp is connected to the interface at the motor terminal M3. It is switched on - as you already learned in a previous experiment - by the command &3L or &3R and it is switched off by &3X. The thermistor is connected to the analog input EY and is therefore inquired by &EY. Enter the following:

```
5 &IN
10 HOME
15 DEF FNT (X)=380-70*LOG(X)
30 VTAB 21
40 &EY
50 TE=FNT(EY)
60 PRINT TE
```

You are already familiar with these program lines: line 5 sets the interface to its initial state. The conversion factor into °F, which of course you should change into your personal conversion factor, is defined in line 15. Lines 40 through 60 indicate the temperature of the thermistor which is stored in the variable TE.
Now enter a temperature at which the burner should switch off (nominal value), for example 90°F:

```
20 TS=90
```

Of course the actual temperature value must be compared with the nominal value after every reading. For this, enter:

```
70 IF TE<TS THEN &3R
80 IF TE>TS THEN &3X
90 GOTO 30
```

If the actual temperature TE is less than the nominal temperature TS, the burner will switch on (&3R). This takes place in line 70. If the actual value attains the nominal temperature, the burner will switch off with &3X (see line 80).
After, the program jumps back to line 30 and there is another temperature reading.

And now to practice: Start the program with RUN. The lamp switches on, which is correct, because to start with the temperature at the thermistor is less than 90°F (nominal value). Then it seems that nothing else happens. In some cases the lamp may blink, the reason for that being the long calculation time of the logarithms. Due to the protection circuit (see the description in in chapter 4) the interface disables the output lines in the meanwhile. The burner must first heat up the thermistor and that takes time. After a while the lamp will switch off: The nominal temperature has been attained. (If still nothing happens, perhaps the thermistor is too far from the lamp.)
Now the burner is not heating any more and the temperature sensor is cooling down. After a short time the burner switches back on and the process starts again. This switching on and off process repeats itself continuously: The controller switches the burner on between two points - it switches it off when the nominal temperature is exceeded and it switches it back on when this temperature is undershot. This is what is called an on-off controller. Its characteristic can best be understood in the control curve in Fig. 7.2. The control curves can also be displayed on the screen by adding the subprogram from chapter 7.1 to display temperature (from line 200) and to clear the screen (from line 300). The subprograms are called up as follows:

```
28 GOSUB 300

65 GOSUB 200
```

The actual temperature fluctuates in a zigzag above and below the nominal temperature. The time between "burner on" and "burner off" is called a hysteresis interval. Now let's look at this effect a little closer. After stopping the program with the Control-C key, change the following lines:

```
25 UT=TS+3
26 LT=TS-3
70 IF TE<LT THEN &3R
80 IF TE>UT THEN &3X
```

Restart the program with RUN. What has happened? Right! The time between "ON" and "OFF", the hysteresis interval, has got bigger because the burner is only switched off above 93°F (UT) and is switched on again below 87°F (LT).
Now find out the number of switch intervals over a period of, say, 5 minutes. Then raise

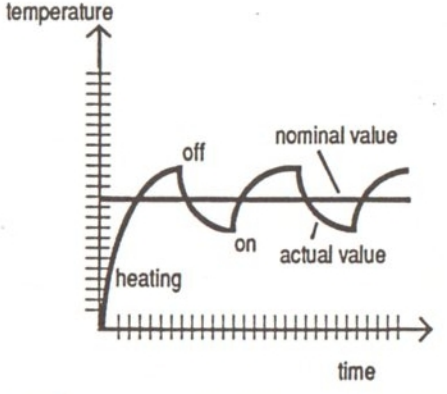


Fig. 7.2 characteristic of an on-off controller.

Controllers, as we have become acquainted with during this experiment, are used in a lot of technical processes. But not only temperature is concerned. It could be as well the output voltage of a power suplly, the gas supply of the carburettor of a car engine, the pressure in a steam engine (by the way the first technical controller) or the activity in a nuclear power plant. But still more: also biological processes are subject to control mechanisms, so that the "trees don't grow up in the sky". Even for social processes the schemes of control theory may be applied.

the nominal temperature to 100°F:

20 TS=100

Restart with RUN. How many switch pulses can you now count in the same time period? There should be more than before, but why? The thermistor cools down faster at the same ambient temperature than before. Try it again with 50°F nominal temperature (20 TS=50). But don't wait too long. Or are you sitting in a very cool room? You can see that there is still a lot of experimenting to do here. The model has shown you how to control temperature by controlling heat generation. Now you understand the function of the room thermostat in the central heating system and why the heating switches on more often in cold weather.
There is also a ready-made program for this model on the floppy disc and this will explain to you the control process in more detail. It's called FURNACE.BAS. Compare here also the lines of the main program with the program lines which you have entered.

Instead of controlling heat supply you can of course control temperature by cooling. What you do depends on the nominal and actual temperatures and normally on the air temperature. This means, however, that there are controllers which are able to heat as well as cool. For example: air-conditioning plant. In winter it works as a heater, and in summer it works as a cooler.
Now we want to show you how to control temperature by controlling a cooling process. So let's put the heating on, that is, the light bulb. Cooling can be easily performed - when you are dealing with high temperatures - with just regular air. And to make it function faster and more effectively, you can assist the air supply by what is known as a cooling fan.
This principle is frequently used when a source of heat cannot be switched off. For example, a car engine cannot simply be switched off while traveling if it overheats. Or would you like to stop every few miles and wait for a few minutes until the engine cools down enough before traveling on?
A fan can control the temperature while the motor is running. How this works will be shown in the following experiment. First assemble the model FAN using the instructions in the assembly manual. The thermi-

stor for measuring temperature is again mounted above the lamp which acts as a heat source. In front of it is a cooling fan. Connect the model to the interface and first test whether everything is correctly wired up by performing short tests. Enter:

```
&IN
&1R
```

The fan must then run for approx. ½ second. Enter:

```
&1X
&3R
```

Now the lamp should light up for a short period of time and with:

```
&3X
&EY
PRINT EY
```

A value between 20 and 200 will be displayed on the screen. If everything is in order, you can then start with the experiment.
The control circuit runs as follows: The lamp burns continuously as a heat source. The thermistor measures the temperature and switches the fan on when the nominal temperature is exceeded. First, enter:

```
10 &IN
15 DEF FNT(X)=427-79*LOG(EX)
40 &3L
50 &EY
60 TE=FNT(EY)
90 GOTO 40
```

This fulfills the first two conditions. The lamp burns continuously (line 40) and the temperature is measured (line 50-60). It is stored in the variable TE. Then there is a nominal/actual comparison for the control circuit:

```
20 UT=90
30 LT=80
70 IF TE>UT THEN &1R
80 IF TE<LT THEN &1X
```

Start the program with RUN. The lamp burns and heats the thermistor. This needs a bit of time. When the upper cut-off temperature (UT, 90°F) is exceeded, the fan switches on (line 70). It cools the thermistor down until the lower cut-off temperature (LT, 80°F) is undershot. Then the fan is switched off and the circuit begins again:

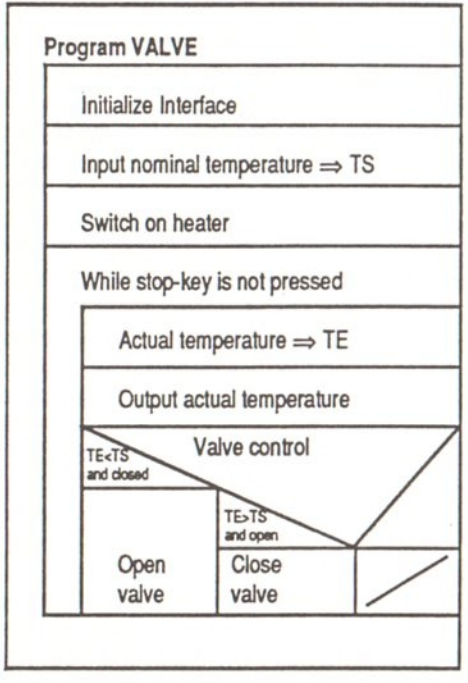


Fig. 7.3: structured flow diagram of throttle valve.

heat up - fan on - cool down - fan off - heat up, etc.
The temperature at which the fan is switched on and off can be displayed by:

```
35 HOME

84 VTAB 21
85 PRINT "ACTUAL TEMPERATURE="
       ;TE;" DEGREES F  "
```

The hysteresis interval and the switching frequency at different nominal temperatures for this experiment can also be graphically displayed as for the model FURNACE. Try doing the program changes on your own this time.
Now we want to show you the other effect which happens in the car. You have probably already seen the sign: "Caution: Fan may run even when ignition switched off". The car engine is therefore cooled when it is switched off and if its temperature is too high. We want your model to do the same thing. Therefore, enter:

```
90 IF PEEK(-16384)<128THEN GOTO 50
100 POKE -16368,0
110 &3X
120 &1R
130 &EY
140 TE=FNT(EY)
150 VTAB 21
160 PRINT "ACTUAL TEMPERATURE=";
       TE;" DEGREES F  "
170 IF TE>70 THEN GOTO 120
180 &1X
190 END
```

After starting with RUN the program runs as before. The "engine" (lamp) can be switched off by pressing any key (lines 90-100). When the lamp goes out, the fan runs on until the temperature drops below 70°F. Then the program ends. If you want to log the temperature curve with the turtle, don't forget to switch off the graphics:

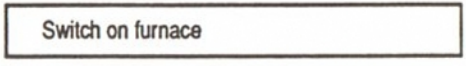
```
185 &GEND
```

The temperature inquiry and the nominal/actual value comparison in lines 40 - 80 was not modified.
This should be enough for your experiment. Of course, the program can be improved with many enhancements. For example, the heat generator could be heated up in steps, etc. The floppy disk contains the ready-made program FAN.BAS to display temperature control through cooling.

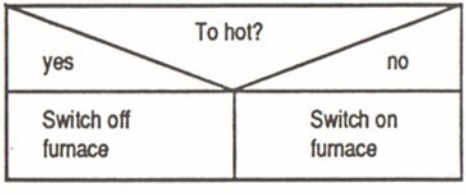
Temperature can also be controlled by regulating the flow of heat. What this means can best be explained by a thermostat valve on a radiator. The radiator is integrated in a heat circuit through which warm water is continuously pumped. How much water (quantity of heat) should flow though the radiator is determined by the thermostat valve. Here is where the desired temperature is adjusted. The valve is open until this temperature is attained. It closes then and no more water can flow until the temperature again drops under the nominal value.
We want to demonstrate this principle to you by another experiment. For this, assemble the model THROTTLE VALVE using the assembly instructions in the assembly manual. We have replaced water by air and the valve by a flap.
The temperature sensor is again mounted above the heating element (the lamp) and the lamp is connected to output M3 and the thermistor to input EY of the interface. What is new is the flap which acts as a valve and interrupts the heat flow from the lamp to the thermistor. It is made by a disk which can be turned by a motor (output M1). As the motor is driven using the step control principle, a switch (E2) is mounted on its axle.
Now we want to show you how to make a control program for this model. We have given you a structured flow diagram in Fig. 7.3. Before making the program for complicated tasks, programmers always start with a structured flow diagram. It is then much easier to find any errors after or build in additions.
The program runs as follows: The nominal temperature which the model must attain and maintain is given (e.g. 90°F). The burner is switched on, the flap is closed and the actual temperature is measured. If it is less than the nominal temperature, the flap is opened. It stays open until the nominal temperature is exceeded. Then it closes.
Before you start keying in the following program, try to understand the structured flow diagram of the program yourself. You have now gained a lot of experience through previous experiments.
Did it work out? Great! Now let's have a look at the program:

```
10 &IN
15 DEF FNT(X)=427-79*LOG(X)
20 INPUT "NOMINAL TEMPERATURE:"
       ;TS
30 UT=TS+1
40 LT=TS-1
50 S=0
```

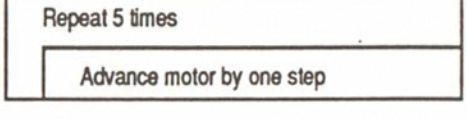
A structured flow diagram is a graphical representation of a program. You should read the structured flow diagram from top to bottom. Whenever you meet a box, the program should perform the action described in the box.



A branch is described by a triangle sitting on its corner. The decision to be taken is described in the triangle; the ongoing actions in the boxes below.



Loops are indicated by a bar on the left-hand side. Either at the begin, at the end or even somewhere in the middle is noted the condition for repeating the loop.



```
60 &3R
70 HOME
80 &EY
90 TE=FNT(EY)
100 VTAB 21
110 PRINT "TEMPERATURE=";TE;
        " DEGREES F  "
120 IF (TE<LT) AND (S=0) THEN &1F:S=1
130 IF (TE>UT) AND (S=1) THEN &1F:S=0
140 GOTO 80
```

In lines 90 and 100 we have written two BASIC statements in a single line. The statements are seperated by a colon. Up to now we have not yet used this option in order to avoid the BASIC-typical unreadable programs (usually called spaghetti code). The case here is a special case. If the condition of the IF...THEN-statement is true, both commands are executed. If the condition is not true, program execution continues on the next line, so ***both*** *commands i.e. &1F:S=1 are skipped.*
Not using this option of several statements on a single line would have let to sequences of GOTO-statements hard to follow up.

The nominal temperature is entered in line 20. Don't select too low a value because the lamp is quite hot (e.g. 90).
Line 30 and 40 give the cut-off values for "open flap" and "close flap". The variable S in line 50 stores the flap position (0=closed). After measuring the temperature, there is a nominal/actual comparison. If the actual temperature TE is less than the nominal temperature and (AND) the flap is closed, the flap will be opened (line 120). This is noted by S=1. In line 130 this is reversed: If TE>TS and (AND) the flap is open (S=1), it will be closed. The program runs in a loop, i.e. it starts again at line 80.
There is also a ready-made program called VALVE.BAS on the floppy disk for this experiment.

In technical terms, a robot is "a universally applicable moving automaton with several axes, whose movement is freely programmable with respect to sequence, path and angle, and may be sensor-guided".

Industrial robots normally have six axes. Three main movement axes move the gripper arm into the correct position. The reason why three axes are necessary is because space is three-dimensional (length, height and width). The robot has three orientation axes in the "wrist". Their functions are to correctly position the tool or the workpiece (roll, tilt, yaw). Operating the tool (welding tongs, screwdriver etc.) or the gripper does not count as an axis.

8 Robotics

8.1 Robot geometry: Operating areas

So far you have performed a number of experiments which really had only one objective: the computer had to be capable of recognizing its environment and using its knowledge to control it. For instance, the computer learned to see - by means of the photoconductive cell - and sense - by means of the NTC resistor, and it learned to control a movement - by means of a motor. With these skills, your computer has already a great similarity with the brain of a robot. But what is exactly a robot?

A robot is a machine or an automaton capable of moving almost like a human arm and performing work such as gripping, stacking, welding, etc. What it does and the sequence it does it in, is taught by a program. And the program also contains measurement data which it must take into account, such as luminosity and heat.

We want to show you what they mean by axes, paths and angles of movement with the help of our model. First build the robot model using the assembly instructions. The model is what is termed a WELDING ROBOT. We have replaced the welding tongs at the front by a lamp. Of course, you don't need to provide bits of iron because you aren't going to do any real welding. However, the welding robot will teach you the possible movements of a robot and how to program them. We thought of including the WELDING ROBOT because they play such an important role in automobile production.

Once you have connected the ROBOT to the computer via the interface, we will show you how to make the ROBOT move before going on to talk about the axes of movement. First, the ROBOT should be in its home position: welding arm retracted and pointing forwards. To do this, slightly remove the motor from the gear axle and put the robot arm in the correct position. Slide the motor again to its proper position. Then enter:

&IN
&1L

The entire robot structure will turn through a few degrees. Then enter:

&1R

It will then return to its home position. Switch off the motor by entering £1X. You can make the ROBOT turn precisely by using the step commands:

&1F or **&1B**

If you enter:

&2F or &2B

the robot arm spindle will turn for a short time. Return the spindle to its home position (welding arm fully retracted). When you enter:

```
10 &IN
20 FOR I=1 TO 12
30 &2F
40 NEXT
```

and RUN, the arm travels fully out. Change the command in line 30 to &2B, and the arm will travel back. But be careful: if the spindle wasn't correctly in its home position at the start, the mechanism may jam slightly at its final position. Just have a few more tries to find out the "reach" of the ROBOT. If you enter:

```
10 &IN
20 FOR I=1 TO 10
30 &1F
40 NEXT
```

and RUN it will turn through 90°. One step therefore corresponds to 9°. Just make sure that the connecting cable of the ROBOT does not jam or get tangled. By entering &1B in line 30 the ROBOT will turn back again.

The ROBOT has therefore two axes of movement: rotation about the vertical axis and a forward/backward movement of the arm. If you compare this with your body, it is equivalent to turning the hips and stretching out the arm forwards.

Of course, modern robots have many more axes. For example, they can raise and lower the arm, or turn the gripper to reach every corner of the car body in a car assembly line.

Schematically represented, the ROBOT can move as shown in Fig. 8.1.

We now want to show you how to derive the maximum reach of the ROBOT. Here we assume a direct movement without negotiating any obstacles. Try to sketch the area on a sheet of paper. If something is unclear to you, move like a robot as described above.

As you can see, the ROBOT can reach a ring-shaped area - similar to a large washer. It can reach every point with just two axes of movement (Fig. 8.2).

It is also said that the reach is two-dimensional (width x depth). For industrial appli-

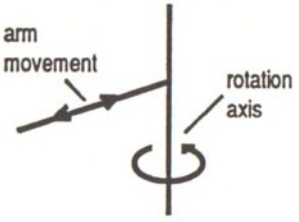


Fig. 8.1: Axes of movement

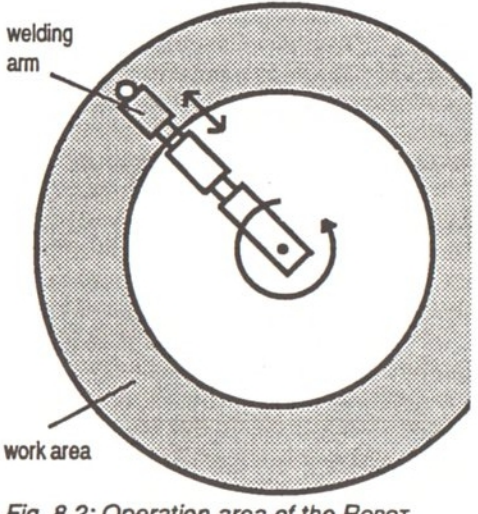


Fig. 8.2: Operation area of the ROBOT

cations, however, all three spatial dimensions are necessary, this also means height. You can also move in three axes with another fischertechnik model, the training robot.
But here we want to concentrate on programming the ROBOT after showing you something about robot geometry. Each of the movement steps can be compiled in a program. The ROBOT can then perform a job of work to schedule - in other words, it will do exactly what you tell it.

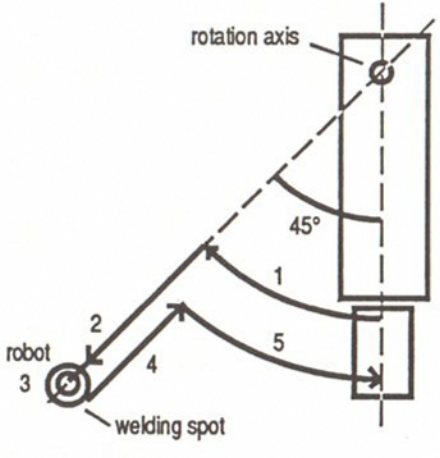


Fig. 8.3: Sequence of welding movements

We now want to show you how to write a control program for the WELDING ROBOT. First put it into its home position, that means with the welding arm retracted and the structure pointing forwards lengthwise along the frame. It can be precisely positioned as explained in the previous section by the commands:

&1F or **&1B** for the axis of rotation and

&2F or **&2B** for the arm movement.

The ROBOT should weld at a point A and then return to its home position where it will wait for a while, and return to point A, where it will repeat its welding operation. This movement process is depicted in Fig. 8.3:
The ROBOT must therefore perform the following sequence of movements:

1. Turn 45° to the right
2. Extend arm
3. Weld
4. Retract arm
5. Turn 45° to the left
6. Pause

The rotational direction "to the right" corresponds to a clockwise turn viewed from

the top of the model (test: &1R). These six steps are realized as follows: First make a structured flow diagram for the program: (Fig. 8.4).
You can see that the program sequence from top to bottom is linear. The procedure then starts again from the beginning. A program section is responsible for each part of the movement. This method is also called "linear programming"of a robot. The program is then compiled from the flow diagram. First try to do it yourself before you continue reading.

```
10 &IN
20 REM ARM RIGHT
30 FOR I=1 TO 5
40 &1B
50 NEXT I
60 REM EXTEND ARM
70 FOR I=1 TO 12
80 &2F
90 NEXT I
100 REM WELD
110 FOR I=1 TO 500
120 &3R
130 NEXT I
140 &3X
150 REM RETRACT ARM
160 FOR I=1 TO 12
170 &2B
180 NEXT I
190 REM ARM LEFT
200 FOR I=1 TO 5
210 &1F
220 NEXT I
230 REM PAUSE
240 FOR I=1 TO 2000
250 NEXT I
260 GOTO 30
```

Did you manage? This is how the program should look like. You can see that it has a lot of FOR...NEXT loops: Each loop is necessary for an arm movement because the movements each contain single steps.
For instance, the first loop to turn the arm to the right by 45° has five cycles, that means the command &1B is repeated five times. The arm then turns through 9° each time. The welding and waiting periods at the end of the cycle are also composed of FOR...NEXT loops. But here they are used to mark a specific period of time.
When you have entered the program and started it with RUN, the ROBOT will perform its work precisely according to the program which you have written for it. It can be stopped by means of the STOP key.
Although the program runs without a hitch,

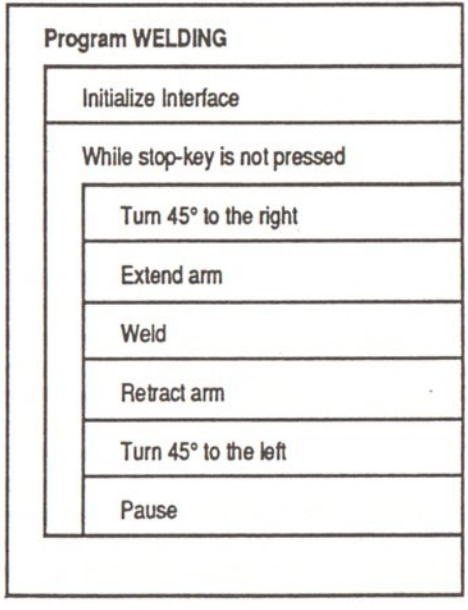


Fig. 8.4: Flow diagram for WELDING

it has a disadvantage: it can only be used for this one work process. If there is any change, e.g. turn to the left or weld at two points, it must be completely re-written. With short programs this is not too difficult, but with long programs it would mean a lot of work. The next chapter will show you how to do this differently.

8.3 Table programming: Custom movements

Every plant which uses a robot for a particular job of work must surely change the robot programs from time to time if a new workpiece needs to be fabricated or if the work process changes. Let's take the example of the auto industry: every year a new model comes onto the market and they all use the same welding robots. If a program costing several hundred thousand dollars has to be purchased or developed again, it will naturally be very expensive. But there is a cheap solution for this: table programming for robots.

The commands for the movement sequence of the robot are stored in a table in the computer memory and are called up one after the other. Different jobs mean that only the table needs to be rewritten.

We want to show you how to try this with your WELDING ROBOT. Take the same work process as in the previous chapter and write a table program. First let's consider how to design the table. Why not give every action the ROBOT makes a code letter?

F - Move welding arm forwards
B - Move welding arm backwards
R - Turn ROBOT to the right
L - Turn ROBOT to the left
W - Weld

P - Pause
E - End of program

Then a figure must be given for all these actions except for the program end, e.g. how many steps the welding arm must move forwards or how long the welding operation should take. For the sake of convenience, we have put the figure before the action code. This makes the table easier to evaluate afterwards for the program. The movement process of the previous chapter then looks like this:

5R	(Move robot five steps to the right)
12F	(Move welding arm twelve steps forwards)
500W	(Weld for 500 loop cycles)
12B	(Retract welding arm by twelve steps)
5L	(Turn robot five steps to the left)
2000P	(Wait for 2000 loop cycles)
E	(End of program)

The table in the computer will be in the form of DATA lines. Please enter:

```
900 DATA 5R,12F,500W,12B,5L,2000P,E
```

The table values, e.g. 5R, should not be confused with commands. The program must first determine what commands are necessary from the table values. The table values "5R", "12F", etc., can be read and executed. This is done as follows:

```
20 RESTORE
30 READ A$
```

The command which we want to process further is then in the variable A$. In the first step, the figure is separated. This is performed by the function VAL(...). The action is determined as the letter standing on the far right of the command with the function RIGHT$(...). Just try the whole thing out. Enter:

```
10 &IN
20 RESTORE
30 READ A$
40 W=VAL(A$)
50 K$=RIGHT$(A$,1)
60 IF K$="F" THEN GOTO 200
70 IF K$="B" THEN GOTO 300
80 IF K$="R" THEN GOTO 400
90 IF K$="L" THEN GOTO 500
100 IF K$="W" THEN GOTO 600
110 IF K$="P" THEN GOTO 700
120 IF K$="E" THEN END
```

The BASIC function VAL(...) is quite useful to extract numeric values from a string of letters. It starts at the begin of the string and collects all characters contributing to a numeric value (like digits and decimal points)and it stops on the first non-numeric character.
The BASIC-function RIGHT$(...) cuts out of a string as many characters as are denoted by the second argument.
So in our case the ROBOT´s action is derived by analyzing the right end, the action´s parameter by analyzing the left end of the table entry. This gives an interesting opportunity: You may comment the ROBOT´s action in the middle part of the string (no spaces and commas are allowed, however):
12steps_down_to_the_object_F

```
130 GOTO 30
```

This first part of the program breaks the command into its components and branches off into the following program parts depending on the desired action.

```
200 REM WELDARM FORWARD
210 FOR I=1 TO W
220 &2F
230 NEXT I
240 GOTO 30
300 REM WELDARM BACKWARD
310 FOR I=1 TO W
320 &2B
330 NEXT I
340 GOTO 30
400 REM TURN ROBOT RIGHT
410 FOR I=1 TO W
420 &1B
430 NEXT I
440 GOTO 30
500 REM TURN ROBOT LEFT
510 FOR I=1 TO W
520 &1F
530 NEXT I
540 GOTO 30
600 REM SWITCH WELDTONGS ON
610 FOR I=1 TO W
620 &3R
630 NEXT I
640 &3X
650 GOTO 30
700 REM PAUSE
710 FOR I=1 TO W
720 NEXT I
730 GOTO 30
```

Start the program with RUN. The ROBOT performs the previous movement sequence once through.
Try a few work processes out - you only need to change the table accordingly. For instance, enter:

```
900 DATA 6R,12F,200W,12B,12L
910 DATA 1000P,12F,200W,12B,6R,E
```

The ROBOT does (almost) everything you tell it to. We recommend to terminate the movement sequence always in the home position. This saves you from placing the robot into the home position yourself for every program start. If you follow this rule when compiling the DATA line(s), you can repeat the movement sequence using the program as often as you want:

```
120 IF K$="E" THEN GOTO 20
```

This shows the benefit of the RESTORE instruction. It causes the next READ command to go to the first table value in the first DATA line.
We advise you to keep this program loaded or to store it on disk, as we will continue to enhance it in the next chapter.
Programming with movement tables is also used by professional robots. It makes the robot universally applicable. On the disk you will again find a somewhat extended program on this topic: ROBOTTAB.BAS.

8.4 Guiding the robot by sensors: Using its own senses

If you know something about welding, you will know that different materials require different welding methods. For instance, the temperature of the welding seam determines the durability of the joint. Let's try to get the ROBOT to do this test. This means that it must determine when the welding point has reached the correct temperature before it goes on to the next weld. This is exactly what a modern industrial robot does.
First build the variation WELDING ROBOT WITH THERMISTOR of the ROBOT as described in the assembly instructions. In principle, the ROBOT looks exactly like it did before except that it now has a temperature probe installed at the "welding tongs". You have already used the thermistor in previous experiments - the ROBOT will now learn to sense heat and include the temperature reading in the welding process. A probe is also generally termed a sensor so that, without any exaggeration, you can say that the WELDING ROBOT is sensor-guided.
But first, the temperature has to be built into the program because the weld duration is now dependent on the signal received from the thermistor. When it reports the correct temperature, it interrupts the welding process.

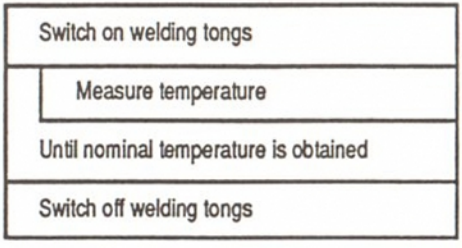


Fig. 8.5: Flow diagram to program WELDING WITH SENSOR

In addition to temperature sensors, optical sensors are also used in robotics to find specific points or probes for testing material thickness, etc. The robot arm can therefore be positioned with millimeter precision and work processes can be exactly adjusted. Up-to-date robots are even equipped with a video camera. Picture processing lets the robot grab with great certainty even if parts are placed in any position on a conveyor belt. The video camera can also be fitted with an infrared filter to detect heat radiation. The robot can then "see" the quality of its welding seam.

The temperature figure which the thermistor should later attain, e.g. 90W, is then entered into the movement table. Naturally, this temperature does not correspond to the welding temperature of real welding tongs - that is much higher, over 5000°F.
If the robot is to execute a welding process, the welding tongs must first be switched on. The temperature at the welding tongs will now be continuously measured. When the nominal value has been reached, the welding process will be terminated.
First start with the flow diagram (Fig. 8.5). If you compare it with the program in the previous chapter, you will see that only the program section which controls the welding process has been changed.
Try to write this program again yourself before it is described.

```
15 DEF FNT(X)=427-79*LOG(X)
```

This is the well-known conversion of the analog value into degrees Fahrenheit, see Chapter 7. There you will also find a description of how to calibrate the thermistor exactly.

```
600 REM WELD (WITH SENSOR)
610 &3R
620 &EY
630 T=FNT(EY)
640 IF T<=W THEN GOTO 620
650 &3X
660 GOTO 30
```

Of course the DATA line has to be changed, too. Instead of the number of loops of the welding procedure now the temperature has to be given, i.e.:

```
900 DATA 5R,12F,90W,12B,5L,2000P,E
```

Enter the lines and start the program with RUN. The ROBOT will now make the weld duration dependent on temperature. Just try it out at other temperatures.
In this experiment, you learned how a sensor-controlled robot can react to heat.
You can study robot movement with sensor guidance in a sample program on the disk. Its name is ROBOTSNS.BAS. Another variation of the above program is ROBOTGRA.BAS. This program uses the graphics turtle to display the movements of the ROBOT on the screen. The background shows a car body. Study this program. It will show you how simple accompanying graphics is integrated.

9 The TURTLE

9.1 Moving the TURTLE: Two to the right, two to the left

So far when we talked about robots, we meant stationary work automatons. They stand on their frames and have only one moving arm with which they can reach their surroundings. Now we are going to show you something new: the mobile robot.
As its name already says, it can move around and work at different places. Typical mobile robots are what are known as aisle conveyors which transport loads to and fro without a driver. How these vehicles - or rather robots - are controlled and what they can do will be demonstrated with the next model which you can assemble using the instructions.
Before putting this vehicle into operation, have a closer look at the illustrations. On the left and right sides, it has independently driven wheels. They are driven by two step-control motors, which you can see by the two microswitches at the rear of the model. Balance is provided by a third wheel at the rear which can move freely and is not driven. At the front, the vehicle is equipped with a bumper behind which is located a switch. When the bumper is pressed it makes an audible click. Above the axle, another optical sensor is mounted.
To make the TURTLE run precisely in the following experiments, try to make a special effort when building the mechanical part. The TURTLE is always best adjusted if it travels fast during the commands below.
Connect the cable of the turtle to the interface. After loading the BASIC expansion program, enter the following into the computer:

```
&IN
&1F
```

The right-hand motor (viewed in travel direction) turns for a short time and the TURTLE turns a few degrees counterclockwise (viewed from above). By entering:

```
&2F
```

the same will happen with the left-hand motor. The TURTLE will turn clockwise. These commands will make it turn in a circle:

```
10 &IN
20 FOR I=1 TO 150
30 &2F
40 NEXT I
```

Enter the lines and start the program with RUN. The TURTLE will turn clockwise around

We have called the vehicle the TURTLE. There are also other names, for example, "the buggy" - but its meaning is always the same. The name "Turtle" was chosen because many model robots are provided with a recording stylus and draw a line along its track - similar to the tail of a turtle in the sand. The programming language LOGO and the graphics turtle also stem from a mobile model robot.

the right-hand wheel. Make sure that the connecting cable does not jam. After each turn, place the TURTLE back into its home position. Otherwise the cable to the interface would be completely twisted after a few attempts. You will have to pay attention to this point in later programming.
Straight-line movements are only possible when the two motors are driven at the same time. Enter:

&1F : &2F

You will see that first the right and then the left wheel turn. The TURTLE then travels forwards in a zig-zag pattern. The next chapter will show you how to make it move in a more elegant way.

The first attempts at moving the TURTLE demonstrated that the two independently driven wheels can move the TURTLE forwards, backwards and in circles. It therefore moves like a tracked vehicle and this of course also applies to straight-line motion. This requires the two motors to be driven at the same time. However, since this is not possible with the present commands, we will give you new commands. Enter:

&TI
&TF,1

The TURTLE will travel forwards. The first command, &TI, is to initialize the TURTLE. You will find out what initializing the TURTLE does in Chapter 10. However, first we assume that initialization is necessary before each of the new TURTLE commands. The next command will move the TURTLE one step forwards with both motors simultaneously. If you enter:

&TB,1

it will move back. By entering:

&TF,20
it will move for a longer distance. It will move

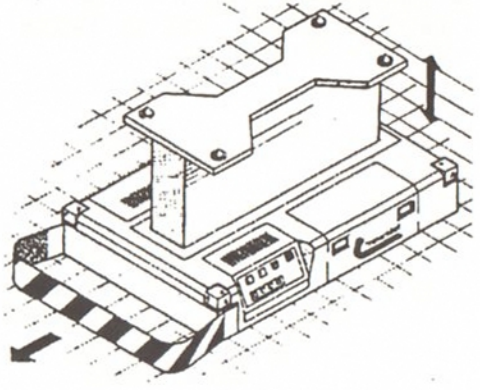


Fig. 9.1: An automatically guided vehicle with a lifting platform.

Mobile robots are termed "automatically guided vehicles"(AGV). In production, they offer much greater flexibility than conveyor belts. When they transport loads, they only need to travel to stations where they either pick up or deposit loads. They receive their orders by radio or other methods from a process computer. Many AGVs look like elevating platforms or fork-lift trucks. Many pull a line of trailers like a locomotive. But they don't look like robots from the science-fiction stories.

Here is a summary of the TURTLE commands:

&TI : the TURTLE is initialized
*&TF,ssss : the TURTLE moves by ssss*5 mm forwards*
*&TB,ssss : the TURTLE moves by ssss*5 mm backwards (ssss = 0 .... 32767)*
&TR,dddd : the TURTLE turns through dddd degrees to the right
&TL,dddd : the TURTLE turns through dddd degrees to the left (dddd = 0 .... 355, divisible by 5)

precisely 10 cm. This shows you the step distance for one forwards command £TF: each of the two motors executes 20 steps. A distance of 10 cm divided by 20 gives a travel distance of 5 mm per step. Of course the same applies to reverse movements. Enter:

&TB,20

and the TURTLE will travel 10 cm backwards. By driving the two motors simultaneously, the TURTLE can also be made to turn on its own axis. One wheel must turn forwards and the other backwards. This requires two new commands. Enter:

&TR,5

The TURTLE will turn a few degrees clockwise (viewed from above the turtle). If you enter:

&TL,5

it will move back again. Now we want to show you how many degrees each command contains. This is obtained by enlarging the turning angle:

&TR,90

Before you press the RETURN key at the end of the line, just mark the line of axis (lift the cable if necessary !). The TURTLE will turn through a right angle. The assumption was right: The degree of rotation of the TURTLE is directly entered in angular degrees. Just try this one out:

&TR,37

The TURTLE doesn't react at all and the screen displays an error message which says that the TURTLE is not capable of executing the steps. The reason is: If the right-hand motor turns back one step and the left-hand motor turns forward one step, the TURTLE will turn through 5°. With a greater number of turning steps, the number of degrees will always be a multiple of 5°. The command therefore tests whether the number can be executed. Here is a similar case:

&TR,400

This command will also lead to an error message because a rotation through 360° (full circle) will lead to nothing else than a

twisted connecting cable. The command permits a maximum turning angle of 355°. In the previous case, it would therefore have been better to enter:

&TR,40

To test this out, try to make the TURTLE move in the following sequence: 5 cm forwards, 90° right turn, 8 cm back. Compare your program with this one:

```
10 &IN
20 &TI
30 &TF,10
40 &TR,90
50 &TB,16
```

Continue to familiarize yourself with the TURTLE control - you will see that the most complicated of movements can be described using the four elementary commands. However, if the path gets too big, that means if it needs many turns and short straights, the present programming method is not recommendable. We want to show you in the next chapter another possibility of writing control programs for the turtle. You are also familiar with it from programming the WELDING ROBOT.

9.3 Route planning with the TURTLE: Planned games

A TURTLE is a mobile robot and it should therefore be capable of traveling over long distances with several changes in direction. Just imagine an aisle conveyor which transports materials in a large hall from one corner to the other and having to crisscross the entire hall. The program is of course just as complicated.
This is exactly the program that you are now going to write for the TURTLE and for this you need to know how numerical route planning works. We already know this type of programming from the WELDING ROBOT: each of the movement steps which the TURTLE must perform afterwards is compiled in a table. The advantage of this method is again the universal applicability of the program. Only the table needs to be changed to make the TURTLE travel a different path. It's a good thing that you already had a look at the method when you were doing the WELDING ROBOT. The new program is in fact simpler than the one for the WELDING ROBOT. We have chosen the four TURTLE movements "to the right", "to the left", "forwards" and "backwards" as actions. Just check it out.

```
10 &IN
20 &TI
30 RESTORE
```

```
40 READ A$
50 W=VAL(A$)
60 K$=RIGHT$(A$,1)
70 IF K$="F" THEN GOTO 200
80 IF K$="B" THEN GOTO 300
90 IF K$="R" THEN GOTO 400
100 IF K$="L" THEN GOTO 500
110 IF K$="E" THEN END
120 GOTO 40
200 REM TURTLE FORWARDS
220 &TF,W
230 GOTO 40
300 REM TURTLE BACKWARDS
320 &TB,W
330 GOTO 40
400 REM TURTLE TURN RIGHT
420 &TR,W
430 GOTO 40
500 REM TURTLE TURN LEFT
520 &TL,W
530 GOTO 40
```

The path of the TURTLE is again coded in DATA lines, e.g.:

```
900 DATA 20F,60R,20F,60R,20F,60R,
      20F,60R,20F,60R,20F,300L,E
```

Just try the program out. What geometric figure does the TURTLE describe? Why is the command 300L written in the last-but-one position in the table and not 60R? You can modify the DATA lines and code your own paths. You may give free rein to your imagination. There is a similar program on the floppy disk called "ROUTENUM.BAS".

How about displaying the TURTLE's route on the screen? Otherwise it wouldn't be any use having a graphics turtle which obeys almost the same commands. If the command &TF is accompanied by the command &GF, the real TURTLE moves in the forward direction and the graphics turtle makes a similar movement on the screen. The graphics turtle also leaves its tracks on the screen if the graphics pen was switched on. The reverse and turn commands are exactly the same, too. Therefore, if all the commands equally apply to the real and to the graphics turtles, it is possible to display the TURTLE's route on the screen. Add the following lines to the program:

```
25 &GON
210 &GF,W
310 &GB,W
410 &GR,W
510 &GL,W
```

Line 110 must be changed to switch off the

graphics at the end of the program:

110 IF K$="E" THEN &GEND : END

The programs on disk use background pictures which are loaded with &GLOAD (see Chapter 6). This is also free for your use:

26 F$="TURTLE"
27 &GLOAD,F$
28 &GC

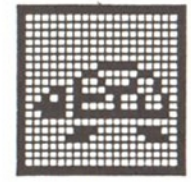
After inserting these lines, the graphics turtle will move on a grid pattern which makes orientation easier for you. Don't forget to insert the fischertechnik floppy disk into the disk drive when you execute this program in order to load the picture.

Modern robots are taught their sequence of movements by putting them through their paces. They are taken step by step from position to position. The robot (or rather its control computer) remembers the sequence and compiles its own movement table which it then uses to work from.
Now let's try to do the same with your TURTLE. First take the TURTLE and connect it to the interface. To check whether everything is plugged in and that the BASIC expansion is loaded and started, enter:

&IN
&TI
&TF,1

The TURTLE must then move one step forwards. If it doesn't, check everything again. Before starting with the route planning according to the teach-in method, organize a solid travel surface over which the TURTLE can easily move. Then use a pencil or adhesive tape to make paths and markings which will later be the actual route.
Cut out a large piece of plywood or thick cardboard about 50cm x 50cm, approx. 20" x 20". Then draw the planned route on it with a soft pencil - don't press too hard because you may have to rub it out again.

This is called the teach-in method because the robot more or less learns its own sequence of movements. The big advantage of the teach-in method is that the robot instructor immediately sees what the robot is doing and does not have to use guess work from tables. Or do you think you could see what the robot was going to do from the DATA lines in the previous chapter?



First the TURTLE will travel in a square from the starting point: So draw a square with an edge length of 10 cm in the middle of the board and place the TURTLE at one of the corners. The corner should be below the axle of the TURTLE. Fig. 9.2 shows the exact starting position.
Now you need a program which will teach the TURTLE the route and which it will later travel. Traveling the route with the TURTLE will be effected by pressing a button for the desired direction on the computer. For this you need the cursor keys which will have the following function:

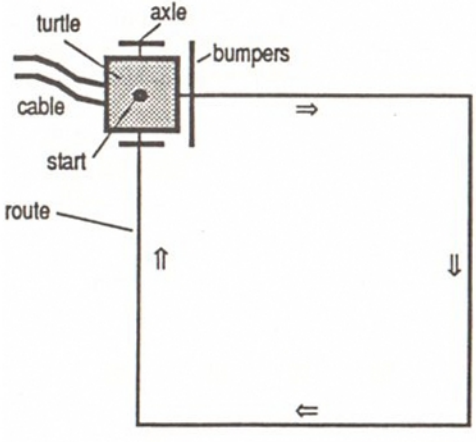


Fig. 9.2: TURTLE route

Cursor up : TURTLE 1 step forwards
Cursor down : TURTLE 1 step backwards
Cursor right : Turn through 5° to the right
Cursor left : Turn through 5° to the left

The program for cursor key inquiry looks like this:

```
10 &IN
20 &TI
40 UP$="I"
50 DOWN$="M"
60 RI$="K"
70 LE$="J"
100 GET K$
130 IF K$<>UP$ THEN GOTO 170
140 &TF,1
160 GOTO 100
170 IF K$<>DOWN$ THEN GOTO 210
180 &TB,1
200 GOTO 100
210 IF K$<>RI$ THEN GOTO 250
220 &TR,5
240 GOTO 100
250 IF K$<>LE$ THEN GOTO 290
260 &TL,5
280 GOTO 100
290 IF K$<>"E" THEN GOTO 100
320 END
```

The lines 40 through 70 determine the character codes for the cursor keys. The characters between the braces again mean that the cursor key is pressed. You are already familiar with the key inquiry using the GET command in line 100: the program waits until a key is pressed. The key code is then contained in the variable K$. Lines 130 through 290 contain the actions for which key is pressed. The program part which is executed depends on which cursor key (direction) was pressed. The program is interrupted by pressing "E". Otherwise it runs in a loop: after each action it jumps back to the start.

Start the program with RUN. When you press a cursor key, the TURTLE will move in the corresponding direction, all other keys except "E" have no function. The TURTLE will move until you release the cursor key. Just play with the program until you can move the TURTLE safely over the board.
Thus the first step in the teach-in programming is completed: the step-by-step movement of the TURTLE over the desired route. Now the computer must remember the route. For this, a table has to be compiled in the same way as for programming the WELDING ROBOT. Insert a field A$(...) into which the movement commands are written. Enter the following:

```
30 DIM A$(500)
80 I=1
```

I is the counter for the positions in the field A$. A command is entered into each position of the field A$ in exactly the same way as you have done so far in the DATA lines. The command must be composed in accordance with the control movements of the turtle. Let's consider an example:
Let's assume that you move the TURTLE forwards, for example you press five times on the cursor up key. From the program point of view, it would be the simplest thing now to write "1F" in five successive positions in the field A$. However, this would be wasting space and would slow the program down. Therefore, collect all the movements that are the same and then, in the example we have chosen, write the action "5F" in one of the positions in field A$. This makes the program a little more complicated, but the effort is worth it.
From line 90, the program is now expanded. The variable STEP counts the number of identical steps. In addition, a copy of the cursor code is made in L$. If ever the updated cursor code no longer coincides with this copy, the sequence of identical commands is ended and must be written into the table A$. This is performed by the subroutine from line 900. There the agreed commands are composed of the number of steps and the code letter. The step counter is then reset and a new copy of the cursor code is made. The pointer in the field A$ is also increased.

```
90 S=0
110 IF K$<>UP$ AND K$<>DOWN$ AND
      K$<>RI$ AND K$<>LE$ AND
      K$<>"E" THEN GOTO 100
120 IF L$<>K$ THEN GOSUB 900
```

The program makes use of the characters I, J, K, and M for direction control. On an Apple //e or // gs you could have used also the following cursor codes:

```
40 UP$=CHR$(11)
50 DOWN$=CHR$(10)
60 RI$=CHR$(21)
70 LE$=CHR$(8)
```

You could also have started numbering the fields with I=0. This was not done because one of the programs on the floppy disk uses I=0 as a special case and we wanted to keep a uniform field numbering system.

```
150 S=S+1
190 S=S+1
230 S=S+5
270 S=S+5
300 GOSUB 900
310 A$(I)="E"
900 IF L$="" THEN L$=K$ : RETURN
910 IF L$=UP$ THEN CODE$="F"
920 IF L$=DOWN$ THEN CODE$="B"
930 IF L$=RI$ THEN CODE$="R"
940 IF L$=LE$ THEN CODE$="L"
950 A$(I)=STR$(S)+CODE$
960 I=I+1
970 S=0
980 L$=K$
990 RETURN
```

In line 950 appears the expression STR$(S).
The use of this function is to convert the internal representation of the number of Steps from numeric (as in the variable S) to character, i.e. a string of decimal digits.

Now start the program with RUN and make the TURTLE move forwards five steps. So far you notice no difference with the previous program. After "E", enter the following in direct mode:

```
PRINT A$(1)
```

On the screen appears

```
5F
```

which is the first command compiled. Now try and make the TURTLE travel on its own along the route it has learned. For this you need to extend the program by a similar section as for the execution of the DATA lines (see previous Chapter). Reading the DATA lines is now replaced by reading field A$. Moreover, the program section has a somewhat shorter formulation. The additional lines are inserted instead of the END command:

```
320 REM EXECUTE PART
330 I=1
340 W=VAL(A$(I))
350 K$=RIGHT$(A$(I),1)
360 IF K$="F" THEN &TF,W
370 IF K$="B" THEN &TB,W
380 IF K$="R" THEN &TR,W
390 IF K$="L" THEN &TL,W
400 IF K$="E" THEN END
410 I=I+1
420 GOTO 340
```

Enter the program lines. In line 330 the counter I is set again to the start of the field A$. After, each command is read and executed (lines 340-420).
Now start the program with RUN. You are now in the enter or learning part. Travel the route (the square) with the TURTLE by press-

ing the corresponding cursor keys. If you come off the path and you want to start again, just press the Control-C key, replace the TURTLE at the starting point and start the program again with RUN.
At the end the TURTLE will stand on the starting point again. Then just enter "E". This tells the TURTLE to execute what it has learned and off it goes - following exactly the route which you have taught it. If you want to see the same route, enter:

GOTO 320

and not RUN because that would clear the field A$. The more the TURTLE travels round the route, the more the connecting cable gets twisted up. Before every start, therefore, turn the TURTLE round so that the cable can move freely.
Why don't you try new routes or extend the program? For instance, you can undo incorrect entries if you come off the path, travel backwards along the path or display the route at the same time on the screen, etc. However, first save the actual program as we are going to enhance it in the next chapter.
You will find a convenient program with screen display on the floppy disk under the name ROUTEACH.BAS. It can also store routes on disk, load routes from disk, produce a hard copy on the printer or display them on the screen.

9.5 Route planning on the screen: Prospects

You have already met computer graphics in Chapter 6.3. With a graphics pen - the graphics turtle - you can draw lines and dots or whole figures on the screen. We now want to show you how to use computer graphics for planning the TURTLE route. The desired route will be generated on the screen using the graphics turtle and this route will then be traveled by the TURTLE. What are the benefits of this? Well, none really. You could take the time during experimenting and use the TURTLE even in the teach-in phase. But it's a totally different ball game in industry. Current production with robots would have to be interrupted for the teach-in phase. Therefore, it is an enormous benefit if the movement of the robots can be studied on screen. This system is known in industry by the name of CAD (computer-aided design) programming. By the way, this is not a simple task for the computer system: the robot programmer must have a realistic impression as if he or she was looking at a telerecording so that the screen robot can be guided reliably. For detail work, the operator must be able to approach with the screen display. The screen must also display the robot's surroundings. It would be disastrous if a robot was to brush against another machine just because it wasn't seen during programming work on the screen.

The task is a little simpler with the TURTLE. A few changes in the teach-in program of the last chapter are enough to make the transition from a teach-in program to a CAD program. All you need to do is adapt the TURTLE commands during the teach-in phase of the real TURTLE to the graphics turtle. Of course you have to switch on the screen graphics first. Enter the following changes into the program:

```
72 &GON
140 &GF,1
180 &GB,1
220 &GR,5
260 &GL,5
400 IF K$="E" THEN &GEND : END
```

Then just try the program out. The route teach-in goes a little faster on the screen. The real TURTLE starts moving when the E key is pressed.

Enhance your program by calling up the grid pattern of the graphics turtle environment onto the screen. Now you can estimate the room for manoeuver much more easily.

```
73 F$="TURTLE"
74 &GLOAD,F$
75 &GC
```

If you want, you can set the graphics turtle to the starting position before the cycle is repeated, change over the pen color and paint the route on the screen in addition to moving the TURTLE. Then you can follow where the TURTLE is at any time. Another suggestion: Extend the program by recording the position of obstacles on the screen first. Then using the CAD method, plan a path between the obstacles. If you have made no mistakes, the real TURTLE should be able to negotiate its path between the real obstacles.

In this chapter you have learned how to use the computer to make drawings on the screen - here the route - which is then executed by a robot. On the floppy you'll find a complete program on this topic under the name ROUTEDIT.BAS. By the way, it is very different to the program developed here. After constructing the route on the screen, you will not be able to transfer to the execute mode immediately. First, the route has to be stored on the floppy disk and then executed with the program ROUTEACH.BAS which can load routes from disk. However, ROUTEDIT.BAS has the trade-off that it is provided with a large number of capabilities. Not only can you construct the route, you can also add changes afterwards, insert route sections from disk at any point, erase commands, etc. At the same time, with this program you can learn how a program editor functions.

10 The TURTLE grows feelers

10.1 Sensor for detecting obstacles: Bumper

Industrial mobile robots are fitted with enormous Emergency Stop bumpers which are primarily provided for protecting people. For orientation they use different sensors, e.g. an echo sounder based on ultrasonics.

The reason why 0 and 1 are changed over in comparison with earlier experiments lies in the button wiring. Just have a closer look: as opposed to other applications, contacts 1 and 2 are used this time. Obviously, with this assembly of the bumper a connector could not have been inserted into contact 3. Another reason is more complex. In industry, all safety screens are connected to the break contact of a pushbutton. If the wire to the pushbutton is damaged or torn out by accident, the electronic circuitry reacts exactly as if the button had been pushed, that means it switches the system off.

So far programming the TURTLE has had the following format: route prescription by table or through the teach-in method followed by route execution according to the table. Just imagine a mobile robot in a large hall, it could be transporting packages to and fro. Its route is of course prescribed. Suddenly a package falls out of the shelf right into its path. What now? The robot can ram it, shove it aside or run over it. Of course it would be better if it could detect the package somehow and avoid the obstacle. To do that it needs a sensor, or a feeler, which reacts to pressure.

The TURTLE is equipped with a sensor, too: a bumper. It is mounted in front of the wheels and operates a microswitch if it is pressed inwards. The switch is connected to the interface input E5. Its function can be checked by using the following program:

```
10 &IN
20 HOME
30 VTAB 1
40 &DI
50 PRINT E5
60 GOTO 30
```

Enter the lines and start the program with RUN. The screen will first display a "1". If you press on the bumper the display will change to "0". "Bumper free" therefore means: E5=1, and "bumper ahead of obstacle": E5=0.

The digital input to which the switch is connected is interrogated by &DI (line 40). You can stop the program with the Contol-C key. Now we want to show you the function of the bumper with the TURTLE in motion. Its task is to move forwards until it hits an obstacle.

Place a book about 10 cm (approx. 4") distance away from the TURTLE to represent an obstacle. Enter:

```
&TI
&TF,200
```

The TURTLE will move forwards and then stop when the bumper hits the book. The whole thing occurs without additional inquiry because the command &TF checks itself whether the button has been pressed. The 0 and 1 of the button are also continuously written into the variable E5 by the command &TF. Now try the following program:

```
NEW
10 &IN
20 &TI
```

```
30 &TF,200
40 PRINT E5
```

Start the program with RUN. The TURTLE will move off. As soon as the TURTLE hits an obstacle, it will stop and the screen will display a 0. However, if the TURTLE travels unhindered, it will stop after one meter and the screen will display a 1. By interrogating E5, you can find out whether the desired route was traveled according to plan or whether there was an obstacle in the way. Of course the bumper can only be interrogated by the command &TF because the turtle only has a bumper at the front. If the bumper is pressed, the TURTLE should be capable of backing off from the obstacle so &TB operates even if the bumper is pressed.

How far did the TURTLE travel before it hit the obstacle? This is shown by the variable TS. It receives the actual number of steps traveled after the command &TF ends. If the complete route was traveled, then it would naturally contain the number given in the &TF command. This is also an aid to establish whether the TURTLE hit an obstacle or not.

So the TURTLE has now become more independent: it is learning how to recognize its environment. Let's leave it to feel its way around the world which is its area of maneuver on the board on which it is traveling. You can build a "wall" of books around it so that the area in the middle is about 40cm x 40cm (approx. 16"x16"). The TURTLE will now travel to the wall in all directions and stop. So that you don't have to turn it every time it stops, it will also turn into the next direction.

```
NEW
10 &IN
20 &TI
30 &TF,200
40 IF E5=1 THEN GOTO 30
50 &TB,10
60 &TR,90
70 GOTO 30
```

Place the TURTLE parallel to a wall and start the program with RUN. It will now travel to each wall, stop, turn through 90° and travel on. Before each turn, it has to move back so that the wheels don't hit the wall. It can be stopped by using the Control-C key. If the number of steps between two opposing walls is known, the size of the area can also be determined. Enter:

```
25 FOR K=1 TO 4
```

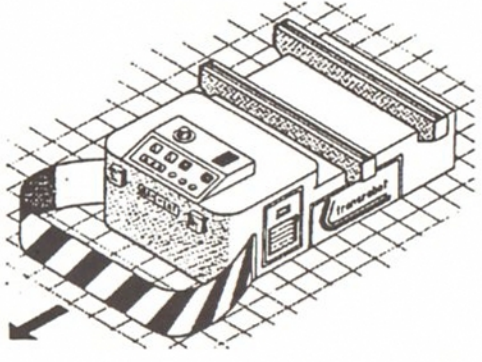


Fig. 10.1: Industrial robot with Emergency Stop bumper

Let's summarize the feed-back commands of the TURTLE:

&TX : Deposits in the variable TX the actual X-Position of the TURTLE.

&TY : Deposits in the variable TY the actual Y-Position of the TURTLE.

&TH : Deposits in the variable TH the actual heading of the TURTLE.

&TF : Besides advancing the TURTLE, the command deposits in the variable TS the number of steps the TURTLE executed sucessfully. Furthermore the switch state of the bumper contact is deposited in the variable E5.

```
45 IF K=3 THEN SB=TS
46 IF K=4 THEN SL=TS
70 NEXT K
80 PRINT "WIDTH:";SB*5+80;" MM"
90 PRINT "LENGTH:";SL*5+80;" MM"
```

Place the TURTLE parallel to the the sidewall pointing to the right and start with RUN. It will now go up to all four walls and stop. This is performed by the FOR...NEXT loop in line 25 (four cycles). The transition of the number of steps denoting the width is obtained by the movement from right to left, i.e. when K=3 (line 70). The length of travel is measured from front to back, i.e. at K=4. The two values are converted into millimeters and the distance between wheel axle and bumper (40 mm) is added twice and then displayed on the screen.
You will notice that the precision with which the TURTLE sounds out its world very heavily depends on whether it travels exactly parallel to the limits of its world. This can be improved by recording the position of the turtle at the first and second corners. There are further TURTLE commands available for this:

&TX

This command stores the current X position of the TURTLE in the variable TX. The same goes for:

&TY

This command sets the Y position in the variable TY. The current course of the TURTLE can also be determined:

&TH

This sets the course in the variable TH. The position data always refer to the position and the course of the TURTLE at the point in time when the &TI command was given. It could therefore be very practical to use the &TI command more than once in the program if you want to select a new starting position.
Change the above program by using the new commands:

```
43 IF K=1 THEN &TY : UP=TY
44 IF K=2 THEN &TX : RIGHT=TX
45 IF K=3 THEN &TY : DOWN=TY
46 IF K=4 THEN &TX : LEFT=TX
80 PRINT "WIDTH:";(RIGHT-LEFT)*5
      +80;" MM"
```

```
90 PRINT "LENGTH:";(UP-DOWN)*5
      +80;" MM"
```

By means of this program and the "bumper" sensor, the TURTLE can already investigate its world quite well. You can expand the program even further by displaying the TURTLE's world graphically on the screen.
There is a demonstration program on the floppy disk called WORLD.BAS. You can also discover the TURTLE's world with it.

In the last chapter you learned about the TURTLE's sense of touch (tactile sensor). Using the bumper as a sensor, it discovered its environment and recognized the limits of its area of maneuver. Here we want to show you how the TURTLE can use its tactile sense to recognize an obstacle and avoid it.
Use the TURTLE with the bumper again. Place the TURTLE on the plywood or cardboard and place an obstacle (book) about 10 cm in front of it. For the first experiment it shouldn't be wider than the TURTLE itself. Now look at the task in Fig. 10.2.
On its trip from A to B, the TURTLE meets an obstacle. It will then drive round it to the right and return to its original path on the other side.
The program begins with the drive around the obstacle:

```
10 &IN
20 &TI
30 &TF,200
40 IF E5=1 THEN GOTO 30
```

Enter the lines and start the program with RUN. The TURTLE stops at the obstacle (E5=0). To avoid the obstacle, it must back off slightly and turn through 90° to the right. Then it will travel at least 12 cm forwards

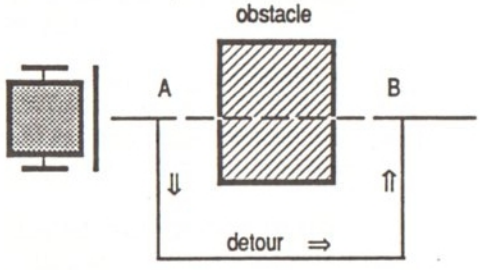


Fig. 10.2: Driving round an obstacle

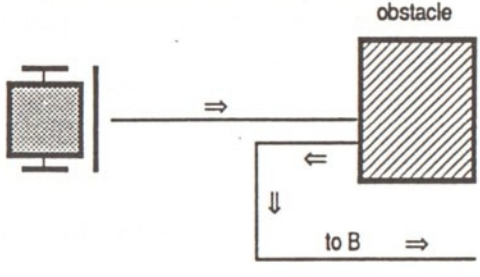


Fig. 10.3: Driving round the obstacle to the right

(one TURTLE width) and turn through 90° to the left. Then it will continue its route to reach its destination B. Fig. 10.3 shows the avoidance maneuver.
Now you can add to the program:

```
200 REM AVOID OBSTACLE
210 &TB,10
220 &TR,90
230 &TF,24
240 &TL,90
250 &TF,34
```

Place the TURTLE back to its starting point and start the program with RUN. After line 240, the TURTLE will be standing next to the obstacle. The TURTLE will then continue its route forwards. The TURTLE will move ten steps forwards which it traveled before turning and will then move one TURTLE width forwards. The final distance is given by:
10 + 24 = 34.
If the obstacle is wider than one TURTLE width (12 cm), the TURTLE will find itself in front of the obstacle and will not be able to continue. The program should be written so that the TURTLE can feel the width of any obstacle, get round it and continue its route. Since you will make frequent use of this task, we will show you how to make a subroutine out of it. The line numbers have been selected with this in mind.

```
50 GOSUB 200

190 END

260 IF TS<34 THEN GOTO 210
270 RETURN
```

What is missing now is an automatic sensing of the obstacle length. For this you will of course need the bumper again and the same subroutine.

```
60 &TL,90
70 &TF,34
80 IF E5=0 THEN GOSUB 200
```

Finally, the TURTLE should be able to find its original route after passing the obstacle. Fig. 10.4 shows this process.
Now supplement the program:

```
 90 &TX
100 IF TX<0 THEN &TB,ABS(TX)
110 IF TX>0 THEN &TF,TX
120 &TR,90
130 &TF,10
140 PRINT "OBSTACLE NEGOTIATED."
```

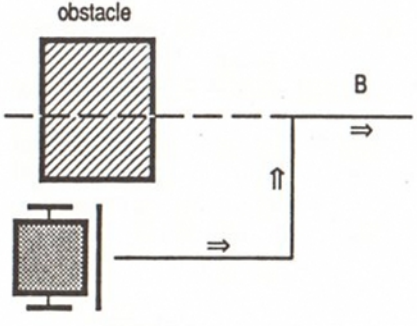


Fig. 10.4: Return to the left

In lines 90 through 110, the command &TX is used to determine the position and to find the starting position in any case X=0. Then the TURTLE will turn back onto its original course and travel straight on.
Now place the TURTLE back in front of the obstacle and start the program with RUN. It will now be capable of passing any obstacle, no matter how long it is. Of course the program can still be worked on - you could expand it to allow the TURTLE to negotiate an obstacle to the left. Also consider another effect: When the Turtle returns to its original path, it might be so close to the backside of the obstacle that it cannot turn unhindered. Improve the program to account for this, too.
On the floppy disk you will find a complete program called OBSTACLE.BAS with user guidance.

For the following experiment you will again need the TURTLE with the bumper. We want to show you how to exploit the capability of the TURTLE to recognize and negotiate obstacles to find its way around a maze.
Before building a maze on the board, you have to find out the necessary track width for the TURTLE. Place the TURTLE at the center of the board and let it turn on its own axis:

```
&TR,180
&TR,180
```

Then mark with a pencil the circumference of the turn which can be negotiated in the maze later. You will find that you will need a track width of about 16 cm (approx. 6½").
Now built the first maze as shown in Fig. 10.5.
It consists of a straight section with a left turn after about 20 cm. The path is flanked by sidewalls which you can make of wooden strips (cross section 10mm x 10mm, approx. ½" x ½"). Attach them with dual-sided adhesive strips onto the board along the edge of the road marked. Leave the entrance and exit clear.
The program should be capable of guiding the TURTLE through the maze to the exit.

In the U.S.A, Japan and Europe, there are whole competitions organized for home-built turtles. The task is to find the fastest mobile robot to negotiate a maze.

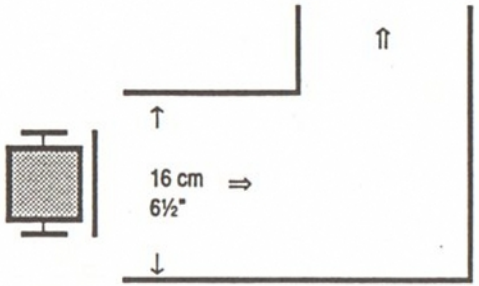


Fig. 10.5: Maze with left turn

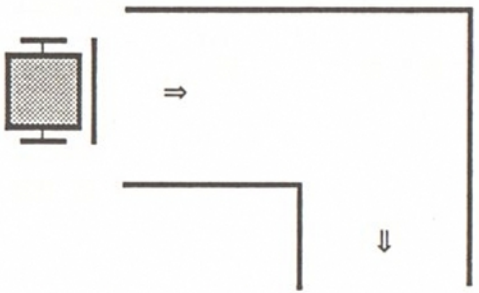


Fig. 10.6: Maze with turn-off right

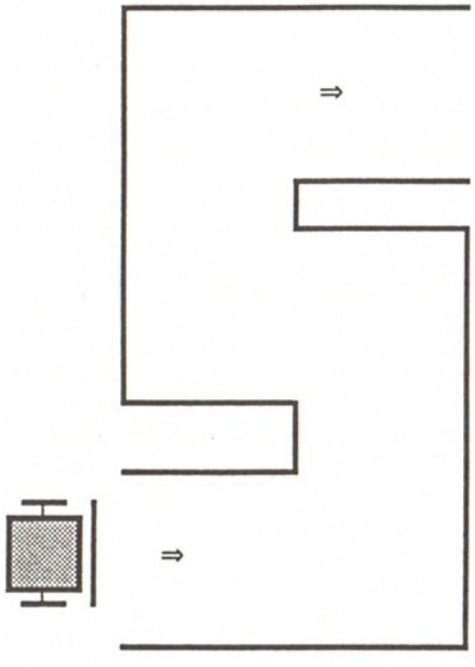


Fig. 10.7: Simple maze

First let it move forwards from the entrance until it reaches an obstacle: the sidewall. Enter:

```
10 &IN
20 &TI
30 GOSUB 200
190 END

200 REM FORWARDS TO WALL
220 &TF,200
240 IF E5=1 THEN GOTO 220
250 &TB,8
260 RETURN
```

The forward motion with obstacle detection has now been placed into a subroutine (lines 200-260) because this movement will be required frequently later. Line 30 addresses the subroutine. When the TURTLE reaches an obstacle, it immediately moves eight steps back (line 250) so that it has room to turn.
Start the program with RUN. The TURTLE must drive right up to the wall and move back. Then it should move left or right. It is assumed that it does not know the direction yet. So let it test whether the path is free to the right and then to the left. You can of course change the sequence. Now expand your program:

```
40 &TR,90
60 GOSUB 200
80 &TL,180
90 GOSUB 200
```

Place the TURTLE back at the entrance and run the program. At the sidewall it will turn through 90° to the right (line 40) and move forwards. This is where the subprogram is required from line 200. If the path is free, it will travel forwards unchecked. If there is an obstacle, it will stop and move back. Then it will try another direction. It therefore turns through 180° (line 80) and will travel forwards (GOSUB 200). Since here is the exit of the maze it will travel on unchecked. It will only stop if you press Contol-C.
Check whether the TURTLE will also find the exit to the right. Change the route according to Fig. 10.6:
Place the TURTLE at the entrance and start with RUN.
The next problem is to see whether the TURTLE will travel through a longer maze with several consecutive turn-offs with this small program. Build the route as shown in Fig. 10.7.
To get the program to start from the begin-

ning after every turn-off, it has to be supplemented:

70 IF S>20 THEN GOTO 40
100 IF S>20 THEN GOTO 40
210 S=0
230 S=S+TS

In the subroutine, the forward steps are added up using the counter TS in S. If the TURTLE travels on unchecked after a turn-off to the right (line 60) and hits a sidewall at the next turn-off, it will turn through 180° (line 80). But because it will have made more than 20 steps, the program jumps back to the start (line 70). The same interrogation also occurs after the second subroutine invocation (line 100).
Enter the lines and start the program with RUN. At each turn-off, the TURTLE checks both directions and always decides for the one that is free. It will therefore find the exit in a short period of time. Stop it by hitting the Contrl-C key.
But do you want to stop with this simple maze? In addition to the correct route, there is always one that is a blind alley. Your TURTLE should naturally be capable of finding its way out of a blind alley. First build the route shown in Fig. 10.8.

After running the program the TURTLE will move to the sidewall at the end and turn right. Here it will find no exit, so it will turn to the other side. It can't get out here either: it's trapped! Expand the program as follows:

40 W=S-8
50 &TR,90
110 REM BACK
120 &TR,90
130 &TB,W

The TURTLE will now travel to the left wall (line 90). If it hits the wall, it will have traveled less than 20 steps. The program continues at line 110. Here the TURTLE again turns into the direction of travel and travels back out of the blind alley. The number of steps it made previously is noted in W (line 40).
Let's go one step further: the exit from the blind alley is somewhere on the side as shown in Fig. 10.9.
The TURTLE should be capable of finding its way out of this one, too. First have a look where the TURTLE is standing at the end of the blind alley (Fig. 10.10) before it travels out (line 130).
It should not travel back directly but test

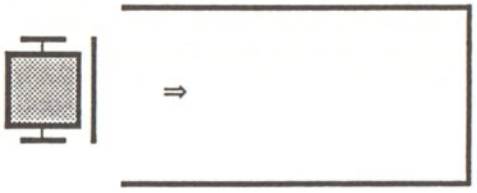


Fig. 10.8: Maze with blind alley

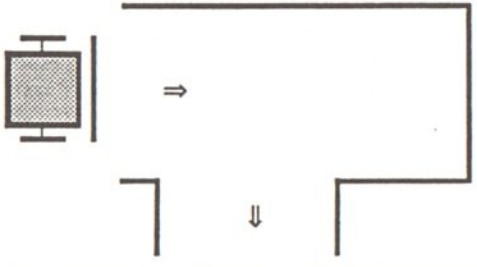


Fig. 10.9: Maze with blind alley and side exit

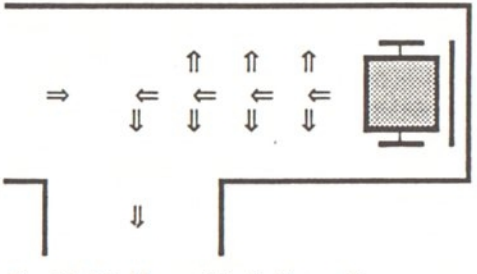


Fig. 10.10: Maze, blind alley, exit

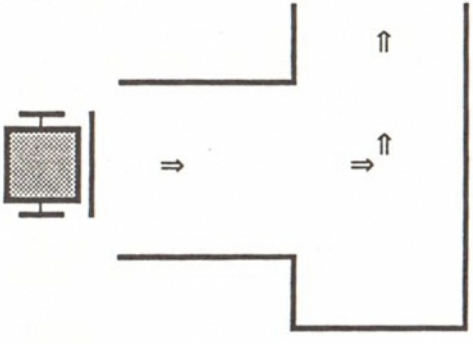


Fig. 10.11: Maze, fork, blind alley

alternately to the left and right whether there is an exit. It will do this right up to the blind alley entrance - it therefore travels max. W steps back. Enter the following program lines:

```
130 &TB,10
140 W=W-10
150 IF W>20 THEN GOTO 50
160 &TB,W
```

The TURTLE will travel 10 steps back and subtract this figure from the total number of steps W (line 140). Then it will test whether there is an exit there - but only if it has not found the entrance to the blind alley (W<20). You already know the attempt to find an exit: just do as if the TURTLE was standing ahead of a turn-off (program from line 50). If the two sides are closed, it will return as if it was in a blind alley (from line 110). Now build the route shown in Fig. 10.10 and start the TURTLE at the entrance. It will find the exit out of the blind alley. If it collides into the walls, widen the path somewhat or reduce the number of back steps to, for instance, 5 (line 140). Then the TURTLE will sense the walls at shorter intervals.
There is only one last maze feature that is missing: when one of the routes at a fork leads into a blind alley and the other goes on. Fig. 10.11 shows the pattern.
If the TURTLE comes from the left, it will travel up to the sidewall (line 30) and move back. If the exit is to the right, it will find this right away because it will test this one first (line 50). If the blind alley is on the right, the TURTLE will search for an exit backwards (lines 130-160) until it is at the entrance of the blind alley again. Then it will be given simply a command to about turn through 180° so that it can continue traveling on the free route (line 30).

```
170 &TL,180
180 GOTO 30
```

Enter the lines and test whether the TURTLE obeys correctly. Then build the route using the wooden strips and test all the possibilities.
Finally, you can build a real maze with all search possibilities which you have learned for the TURTLE to execute. Build up a maze like the one in Fig. 10.12 which has quite a few difficulties.
Send the TURTLE into it. If nothing goes wrong, it will some time arrive at the exit. You will find a complete program for performing a maze on the floppy disk

(MAZE.BAS).
Try out all the other routes. Maybe - or probably - there will be some problems which the TURTLE will not be able to solve using the program. For instance, it may happen that the TURTLE finds its way out of the maze but not at the exit which was intended. Fig. 10.13 shows this type of maze where the exit to the top was overlooked because with the present program the TURTLE only looks for the exit if it cannot travel straight on any more.
Even worse: just have a look how the TURTLE acts in the maze depicted in Fig. 10.14. Right - the TURTLE is trapped in this maze and will continue to turn in circles. The TURTLE never tests the exit because it always prefers to turn to the right. If you change the preferred direction of turn from "right" to "left", the problem will not be solved completely. The TURTLE would still be trapped in a maze which is the mirror image, see Fig. 10.13.
Success is only possible with what is known as the "right-hand rule". This says that you will always find a way out of the maze if you always try to turn right at every possible turning. Example: When you enter a maze, you always touch with your right hand the wall to your right and run along this wall.

Sooner or later you will find your way out of the maze, maybe not at the expected exit or even by the shortest way, but you will get out. This strategy can also be programmed into the TURTLE. But because the TURTLE has no sensor on its right flank, it has to turn to the right at regular intervals and sense whether the wall is still there. If it is, it will return to its route by turning left and travel on. When there is an opening to the right, it will go in. Although continuous sensing makes the movement of the TURTLE slower, it can now find its way out of the maze. And another thing: the program is surprisingly short. Because the previous program can no longer be used, save it (if you like it) and erase it and enter from scratch:

```
NEW
10 &IN
20 &TI
30 &TF,10
40 IF E5=0 THEN GOTO 70
50 &TR,90
60 GOTO 30
70 REM HIT
80 &TB,TS
90 &TL,90
100 GOTO 30
```

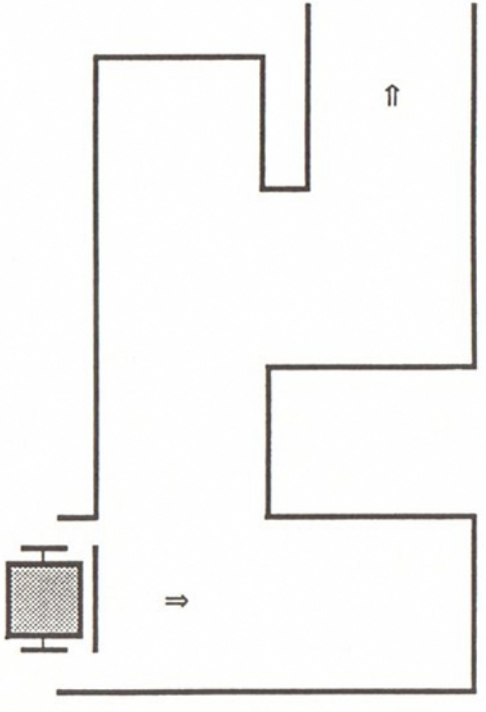


Fig. 10.12: Difficult maze

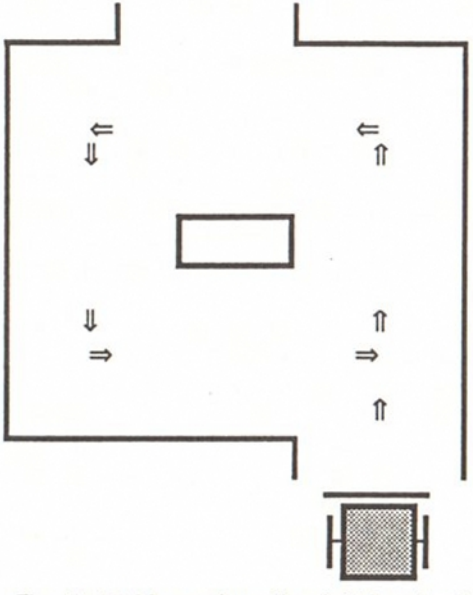


Fig. 10.13: Maze where the right-hand exit is not detected

The figure 10 in line 30 may have to be adapted a little. If it is too large, the TURTLE may not always hit every turn-off to the right. You've already met this problem when backing out of a blind alley. But the figure should not be too small either. If the turtle finds itself in a straight corridor, it must be certain that it can really find the right-hand wall within the number of steps given in line 30.

You are free to provide the program with a display of the TURTLE's route on the screen. This is also done in the program STRATEGY.BAS on the floppy disk.

The question still remains how the TURTLE can find the shortest way through the maze. In the competitions mentioned at the start, the problem is the travel time of the TURTLE. Of course, no progress can be made with simple programs. But we just want to give you an idea of the thought process. During the first run through the maze, the TURTLE tries as many passages as possible. From this it constructs a "map". Using the mathematical methods of graph theory, the program then searches for the shortest route between start and destination on the map. During the second passage, the TURTLE will then find the fastest route through the maze. The turtles in the competitions are also equipped with non-contact proximity sensors, a type of echo sounder, with which they can stay on course at high speeds, detect obstacles in advance and work out negotiating movements without having to travel round them. It would also be practical to fit side sensors to your TURTLE. Perhaps you can expand your TURTLE using pushbuttons and other components from your other fischertechnik kits.

This type of TURTLE is no longer just a game. Corporations and universities have already built and investigated turtles. The objective of their research is the development of autonomous vehicles for industry and for use in fire-fighting or rescue work. Maybe one day a home robot will emerge from this work capable of vacuuming, mowing the lawn and doing other work.

In addition to the tactile sensor "Bumper", the TURTLE also has an optical sensor. This eye, a photoconductive cell, is mounted in line with the turtle axis. You will find the photoconductive cell hidden behind a cover which is intended to limit the field of vision of the optical sensor. The TURTLE can therefore just see as much as it needs and is not affected by lateral differences in luminosity. The photoconductive cell is connected to the analog input EX of the interface (orange wire); you are also familiar with the measurement principle from earlier experiments. You can check whether the light measurement functions by giving the commands:

```
&IN
&EX
PRINT EX
```

When light falls onto the photoconductive cell, a low number (30 - 100) is displayed on the screen. If you close the sensor opening to prevent light from entering, the value will be around 255.
We now want you to combine optical measurement with TURTLE movement: when the TURTLE receives light, it should start and when it is dark, it should stop. The program looks like this:

```
10 &IN
20 &TI
30 &EX
40 IF EX<128 THEN &TF,1
50 GOTO 30
```

Enter the lines and start the program with RUN. The TURTLE will start when the room is lit up brightly. If you hold an object or your finger in front of the light sensor, the TURTLE will stop. In the program, line 40 evaluates the difference in luminosity. The switch threshold is set to a value of 128. By adjusting this figure, you may be able to make the TURTLE start by using the light switch in your room. However, make sure that not too much daylight falls onto the sensor.
You can also change the travel direction of the TURTLE using light. First let's make it shy of light, that means it should turn away from light. Change the program like this:

```
40 IF EX<128 THEN &TR,5
```

Now place the TURTLE so that it is pointing into the light, e.g. towards the window, and start with RUN. It will turn until it finds enough darkness. The opposite will also go.

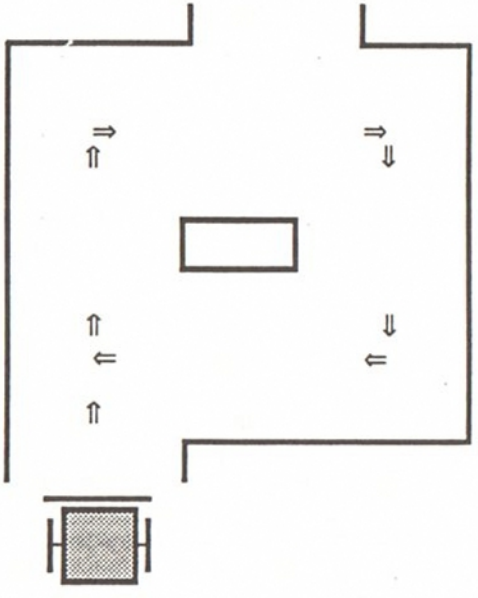


Fig. 10.14: The maze which the turtle can't find its way out of

In practice, light is also used to control mobile robots. Light beams can mark routes or detect obstacles. But to prevent robots from being oversensitive to daylight and other sources of light, invisible infrared light is used which the receiver filters out from all other light.
Infrared light follows visible light at the long-wave end of the light spectrum.

Press the Control-C key and enter:

```
40 IF EX>128 THEN &TL,5
```

After RUN it will turn away from darkness towards light. You can control the TURTLE with a movable light source. You can make it follow a flashlight moved in front of it. Supplement the program as follows:

```
40 IF EX<128 THEN GOTO 80
50 &TF,1
70 GOTO 30
80 L=EX
90 &TR,5
100 &EX
120 IF EX<L THEN GOTO 30
130 &TL,10
140 &EX
160 IF EX<L THEN GOTO 30
170 &TR,10
180 GOTO 120
```

Start with RUN. Direct the light beam from a distance of approx. 20 cm directly onto the sensor. The TURTLE will now search out the lamp. It will turn to the left and to the right. If it detects the light beam, it will travel forwards towards it (line 50). The switch threshold is now set at 128 (value at the given conditions), so that control is not too sensitive. The TURTLE will move forwards until the light measurement gives a value of 128, that means it is darker. The program will make a note of the last light strength (line 80) and will first check to the right (lines 90-120) whether it is brighter there. If it is, that is, the flashlight is pointing from this direction, it will travel towards it. Otherwise it will turn back and check the light strength in the other direction (lines 130-160). The TURTLE will remain in this search loop until enough light falls onto the sensor again so that it can move forwards. You can stop it with the Control-C key.
So that the TURTLE can also detect when the flashlight is switched off, supplement the program with:

```
60 N=0
110 N=N+1
150 N=N+1
180 IF N<5 THEN GOTO 120
```

The variable N counts the search processes for light. If the search in both directions was unsuccessful and counts up to 5, the program is terminated.
More controlling of the TURTLE by means of its light sensor follows in the next chapters.

10.5 Searching for the light: Keep your eyes peeled!

In the last chapter you learned how to use the optical sensor to control the TURTLE. With a simple program it was possible to make the TURTLE follow a moving source of light. But only the difference between light and dark was detected.
Now we want to show you how the light measurement can be made more precisely and how to make the TURTLE search for the light. It will also be taught how to make a picture of the light distribution in its environment and pick out the brightest source of light towards which it will then travel. Fig. 10.15 shows the procedure.
First the TURTLE will measure the brightness in its environment. It will turn through 360° and measure the brightness after every step turned. Then it will print it out. Enter the following:

```
10 &IN
20 &TI
30 H=255
40 R=0
50 FOR W=0 TO 355 STEP 5
60 &EX
70 IF EX<H THEN LET H=EX : R=W
80 PRINT "ANGLE ";W;"  BRIGHTNESS
       ";EX
90 &TR,5
100 NEXT W
```

The TURTLE stores the angle with the greatest brightness so far in the variable R and the figure for the light strength in H.
Now start the program with RUN. The TURTLE will turn to the right and will measure the light strength at each step of rotation (line 60). It will also display the angle W and the value EX on the screen (line 80). Once the TURTLE has turned through 360°, the program will terminate.
Now you have a "light picture" of the TURTLE's environment. The smallest value corresponds to the greatest light strength. The TURTLE will then travel in this direction. But first turn it back to its home position (Figure 10.15) with:

```
110 &TL,180 : &TL,180
```

and enter

```
GOTO 110    (not RUN!)
```

Later it will turn back automatically after each measurement.
The direction with the greatest light intensity is in the variable R. Enter the line:

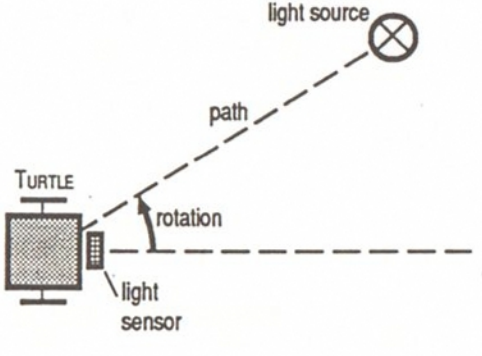


Fig. 10.15: TURTLE searches for light

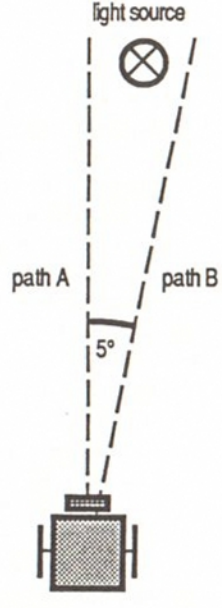


Fig. 10.16: Angular resolution

120 PRINT "GREATEST BRIGHTNESS IN DIRECTION ";R

and start the program with RUN. The TURTLE will again turn through 360° and measure the light strengths. After it has returned to its home position, the direction with the greatest brightness will be displayed on the screen.
Now turn the TURTLE into the direction detected with:

130 &TR,R

After entering RUN, it will execute the command after it has re-measured the brightness. The TURTLE will then turn into its home position (0°) and then travel in the brightest direction. There it will remain standing after turning round. Erase line 110 and change line 130 to:

130 &TL,360-R

After RUN the TURTLE will find the direction faster.
If you try the experiment at home on your table, you will notice that the TURTLE always turns towards the window because that is the source of the most light (during the day).

In addition, it is immaterial in which direction the TURTLE was pointing before the program run. With the line:

140 &TF,200

the TURTLE can be made to travel to the source of the light. Shine your flashlight from a short distance onto the TURTLE and run the program. It will travel towards it.
As you learned in a previous chapter, the smallest step of rotation for the TURTLE is 5°. If the light source is at some distance away from the TURTLE, it may be that it will travel past it with this program. The resolution is too coarse, as Fig. 10.16 illustrates:
On the two paths (A and B) it will not reach the light source. One or several corrections in direction must then be made along the path to the light source. The TURTLE will then travel a short distance forward. At a certain angle, it will re-measure the direction with the greatest brightness and continue its course in that direction. Fig. 10.17 shows this procedure.
Expand the program like this:

140 &TF,20
150 &TL,15
160 H=255

```
170 R=0
180 FOR W=0 TO 30 STEP 5
190 &EX
200 IF EX<H THEN H=EX:R=W
210 &TR,5
220 NEXT W
230 &TL,30-R
240 GOTO 140
```

After RUN the TURTLE will execute a course correction in the direction of the lamp. Each of the steps is represented in Fig. 10.18.

1: 20 steps forwards
2: Turn 15° to the left (start of measurement)
3: Turn 30° to the right and note the direction with the greatest brightness
4: Turn back to this direction
5: Continue traveling

The procedure for detecting the direction with the greatest brightness is contained in lines 30 through 100. You can test this subroutine separately with:

```
GOTO 150
```

Direct your flashlight from a greater distance onto the TURTLE and place the turtle not quite in the direction of the lamp. After a few measurements and corrections, it will travel precisely in the direction of the light source. To make the TURTLE stop at the lamp placed on the travel board, the sensor "bumper" must again be used. With the command:

```
145 IF E5=0 THEN END
```

it will travel only up to the "obstacle", the lamp. If the bumper is touched (E5=0), the program will end in line 145.
This is where we want to finish the programming for this experiment. Of course there are still some enhancements possible: for instance, the path of the TURTLE can be displayed on the screen or the brightness picture can be displayed on a radar scope which you met during the first experiments with the photoconductive cell.
There is again a complete program on the floppy disk for this experiment. It's called FINDER.BAS. The screen will demonstrate the setup and the procedure of the experiment "TURTLE searches for the light".

Fig. 10.17: Direction correction

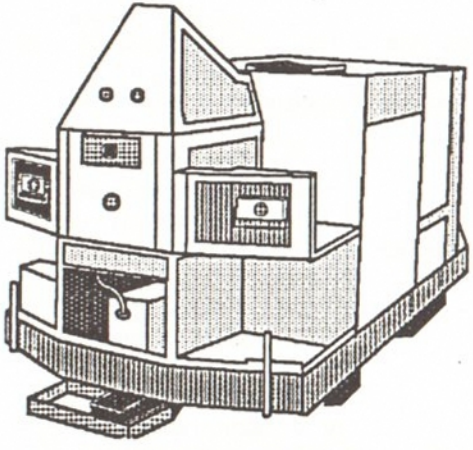


Fig. 10.18: Industrial mobile robot with optical and ultrasonic sensors

Fig. 10.19: Track for the TURTLE

So far the optical sensor was mounted on the TURTLE so that it could detect light beams from the front. Now it will be used as a read head to optically sense an area on the track ahead of the TURTLE.
First modify the model according to the assembly instructions. TheTURTLE no longer has a wide bumper but a read head on the front in which the photoconductive cell is mounted. It senses the reflected light of the lamp from the track. It should have a minimum clearance from the track surface of 3mm (approx. 1/8") so that light can fall under the sensor. Adjust the read head by moving the building blocks accordingly. The pushbutton which was originally behind the bumper is now mounted in front of the read head and can be used temporarily to detect obstacles.
First check again whether the model functions correctly. For the following experiments you need a white surface (e.g. cardboard) for the TURTLE to travel on. A dark surface does not reflect enough light and will lead to incorrect measurements. Then place the TURTLE onto the surface and connect it to the interface. Don't connect the lamp of the read head to the 28-pin connector yet. Take a 44 cm long cable and connect the lamp directly to the plug of the power supply at the interface (insert it into the sideholes). The lamp will now light up permanently. Enter:

```
&IN
&EX.
PRINT EX
```

After RETURN a number will appear on the screen corresponding to the brightness of the track ahead of theTURTLE. &EX reads the value which is displayed with PRINT EX. Place the TURTLE onto a dark surface. If you now enter the last two commands again, the value displayed will be much larger. Thus, a light surface will give a smaller figure than a dark one.
This is what we will now use to control the TURTLE. It is supposed to follow the dark line on the track. To obtain a good contrast, use a matt-black adhesive tape (e.g. PVC insulation tape) for the dark line. Use this to mark the route that the TURTLE must travel. This principle is also used with industrial mobile robots, however induction cables are buried in the floor instead. The mobile robot is then electro-magnetically steered. Make a straight path of about 20 cm length on the surface. The TURTLE will now follow this path (Fig. 10.19).

The program looks like this:

```
10 &IN
20 FOR I=1 TO 60
30 &3R
40 NEXT I
50 &TI
70 &EX
80 H=EX
90 &TF,1
100 &EX
110 IF EX>H-20 THEN GOTO 90
```

Enter the lines. Now connect the lamp of the read head to output M3 of the 28-pin connector. Place the TURTLE on the left of the track so that the read head is pointing to the adhesive strip and start the program with RUN.
The TURTLE will travel forwards until the strip ends. The track line is detected by continuously measuring the reflected light from the surface. At the start, it measured the light strength of the surface (line 70) and noted it, storing it in H. After each step forward it takes another measurement (line 100). This value EX is compared in line 110 with the previous one in H. As long as EX is not less than H, the TURTLE is still on the black line. It will therefore continue to travel.

When the adhesive strip ends, the reflected light intensity from the bright background is greater than before - EX is smaller than H - and the TURTLE will stop. The program will also end.
Between EX and H there is a tolerance range of 20 (H-20) because the readings taken from the track may deviate somewhat. EX must therefore be at least 20 less than H before the TURTLE stops. Without this security it would keep on stopping on the track - check it out!
If the TURTLE does not travel straight on, it may leave the track and stop. To prevent this from happening, add a correction to the program to keep the TURTLE on the right course. Enter:

```
120 &TR,5
130 &EX
140 IF EX>H-20 THEN GOTO 90
150 &TL,10
160 &EX
170 IF EX>H-20 THEN GOTO 90
```

Now place the TURTLE a little skew on the track at the start of the route as shown in Fig. 10.20, and start with RUN.
After a short distance the TURTLE will leave the track. Then it will turn to the right (line

Fig. 10.20: Correcting the direction on track

Fig. 10.21: Turn-off to the left (or right)

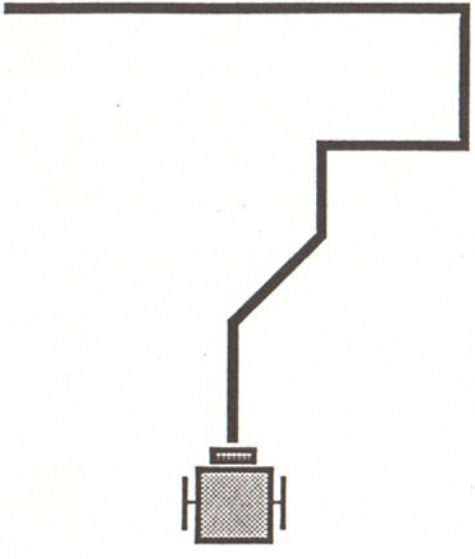


Fig. 10.22: Track with turn-offs

120) and measure the track brightness. If this value is greater than the previous one - if the track is there - the TURTLE will continue. Otherwise it will turn left and, using the same method, it will check whether the track is there (lines 150 through 170). If not, the program will end because in this case, the end of the track has been reached. You will see that the TURTLE will always succeed in remaining on the track with this program addition. It will even detect and follow slight bends in the track.

Now build a turn-off at the end of the track. For the sake of simplicity, make it a 90° turn to the right or to the left. The TURTLE now must find the right way even here. If it travels over the end of the track, it will first assume that it has come off the track. The TURTLE will then check to the right and to the left (one step in each direction) whether the route continues in that direction. However, according to Fig. 10.21 it is situated at a turn-off.

The TURTLE will then check in both directions to find where the track is. It therefore travels thirteen steps forwards so that its axis of rotation is above the turn-off of the track:

```
180 &TR,5
190 &TF,13
```

Before, it will turn back again in the original travel direction (&TR,5). Then it will turn through 90° to the right:

```
200 &TR,90
```

and check whether the track is located there as can be told from the difference in brightness.

```
210 &EX
220 IF EX>H-20 THEN GOTO 90
```

If it is, it will travel on forwards (line 90). Otherwise it will turn into the other direction:

```
230 &TL,180
```

and check whether the track continues there.

```
240 &EX
250 IF EX>H+20 THEN GOTO 90
```

If it finds no path, the track has terminated.

```
260 PRINT "TRACK END"
270 END
```

Enter the lines, place the TURTLE again at the

start of the track and start with RUN. It will detect whether the turn-off is to the left or to the right. If you design a longer section, make sure that the sections between the turn-offs are at least as long as the TURTLE itself. Otherwise it won't know where it is after a 90° turn.
Of course you can also detect turn-offs with a smaller angle. The TURTLE not only checks after the 90° turn whether it is on the track but also during the turning steps. Change the program as follows:

```
210 W=0
220 &TL,5
230 W=W+5
240 &EX
250 IF EX>H-20 THEN GOTO 90
260 IF W<180 THEN GOTO 220
270 PRINT "TRACK END!"
280 END
```

and make a track as shown in Fig. 10.22 (or similar):
After running the program, the TURTLE should be capable of traveling along the track from start to end and detecting each turn. Make sure that the brightness of the light is constant all the time. Even a shadow can affect measurement which will make the TURTLE leave the track in an uncontrolled movement.
Guided tracks like this are very frequently found on aisle conveyors. But due to the greater protection against interference, they are mainly produced through underfloor cables. The robots are then controlled inductively via cable instead of being guided by light.
Programs which control vehicles in this way are often given the term "Artificial Intelligence". This term is causing a lot of excitement at the moment. Can the computer assume the role of a human being by reacting intelligently? Probably not, because human thought has much more than just intelligence - imagination, intuition, creativity, etc. The term "Artificial Intelligence" and its meaning is even contested among specialists. Here we want to give you a purely practical description: programs using artificial intelligence methods can solve problems much quicker than by systematic testing because they store previous experiences in the memory and use them for solving the problem. We now want to help you give the track-following program a touch of artificial intelligence. First make a track in the form of a figure of eight (Fig. 10.23).

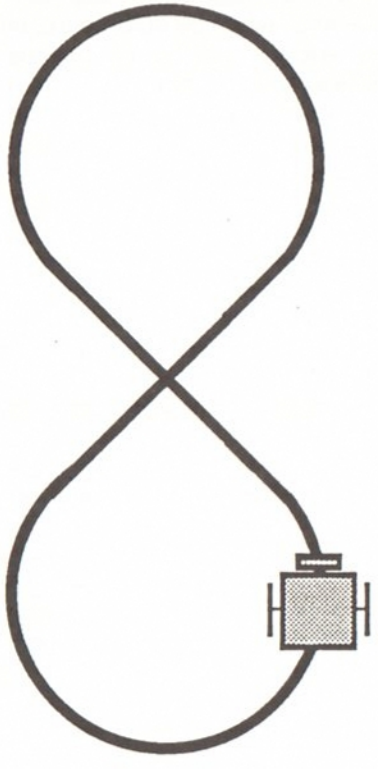


Fig. 10.23: Figure-of-8 for the TURTLE

Volkswagen has also made experiments to make a vehicle travel automatically with a camera. The picture processing system is guided by the boundary and center markings on the road. The experiments really worked. And the car was traveling at 75 mph - without an accident! Now ,a mobile robot manufacturer is offering a mass-produced device for in-house transport which is also guided by track markings and boundaries, even if it doesn't travel so fast.

With the present program, the TURTLE should be capable of following the track easily enough. It will not be deranged by the cross-over if this is more or less at right angles and if the TURTLE has a long enough straight section ahead of the cross-over area. It should be capable of following this route at maximum speed along the straight by using only a few corrections. This will be totally different in the right-hand bend because course correction steps will be continuously needed, thus dropping the speed. It will even be worse in the left-hand bend. Since the TURTLE always searches first to the right, this bend will take even more time. Change of the search sequence in the program will not solve the problem because the same difficulties will then arise in the right-hand bend.

Here is where artificial intelligence can come to the rescue. Let's enter a memory variable RI. If the last course correction was to the right, then RI=0, and to the left RI=1. If another course correction is necessary, the program at RI=0 will first try to the right and at RI=1 will first try to the left.

First erase the lines 140 through 180 and enter the following new lines:

```
60 RI=0
120 IF RI=0 THEN GOTO 300
130 IF RI=1 THEN GOTO 400
300 &TR,5
310 &EX
320 IF EX>H-20 THEN GOTO 90
330 &TL,10
340 &EX
350 IF EX>H-20 THEN RI=1 : GOTO 90
360 &TR,5
370 GOTO 190
400 &TL,5
410 &EX
420 IF EX>H-20 THEN GOTO 90
430 &TR,10
440 &EX
450 IF EX>H-20 THEN RI=0 : GOTO 90
460 &TL,5
470 GOTO 190
```

Place the TURTLE with the enhanced program onto the figure-of-8 course. You'll see that, although at the start of the bend the turtle will begin its course correction on the wrong side due to the previous knowledge, it will learn the new situation immediatly and then travel through the bend at a fair speed. You can, of course, build in more experience rules into the program. To prevent searching to the wrong side when transferring from one bend to the other, you could

include the following rule:
If a number of steps is executed without course correction, then RI will be reversed, hence:

RI=1-RI

With this strategy figures-of-8 and wavy lines can be traveled much easier.
On the floppy disk you will again find a complete program with which you will be able to practise the optical control of the TURTLE on a track. It is called PATH.BAS and contains also an enhanced luminosity adjustment.

10.7 Traffic management systems: On the right course

In the last experiment the TURTLE was equipped with a read head with which it could follow a line on the track. The sensor, a photoconductive cell, received the reflection of the flashlight from the track and could therefore detect whether the TURTLE was on the track (dark) or to one side (light). Depending on the reading, the course of the TURTLE was corrected.
After controlling the TURTLE, we now want to show you how to use the read head to pick up information from the track. You all know this principle from shopping: food and other goods in stores have labels which have a number of straight lines next to each other. At the cash desk the label is read by a lightpen which then indicates the price, the name of the goods, etc. This information is contained in the lines, also called a bar code or product information code.
For the following experiment you need the TURTLE model with the read head as before. The read head is mounted so that it is about 3mm (approx. 1/8") above the track. After connecting the turtle to the interface and loading the BASIC expansion program, place the TURTLE onto the white surface (cardboard). The surface must be bright so that enough light is reflected into the photoconductive cell.

Check the function of the light strength, track brightness and sensor clearance using the following program:

```
10 &IN
15 &TI
20 FOR I=0 TO 60
30 &3R
40 NEXT I
50 &EX
60 PRINT EX
```

A figure corresponding to the brightness of the track then appears on the screen. Stick a piece of black PVC insulation tape approx. 15 mm long in the center of the track and place the TURTLE so that the read head is pointing at the strip. Start the program again. The figure on the screen should be much higher because the track is darker than the white background.
Without you noticing it, you have already read an information from the track: the states "light" and "dark" displayed by different figures. This is called binary information because it only consists of two states. We will identify the two states by figures: light = 1, dark = 0. With this notation - also called logic values - it is much easier to work. Now enter the program for the experiment:

```
70 HE=EX
80 &EX
90 Z$="LOGIC VALUE 1"
100 IF EX>HE+20 THEN Z$="LOGIC
      VALUE 0"
110 PRINT Z$
120 &3R
130 IF PEEK(-16384)<128 THEN
      GOTO 120
140 POKE -16368,0
150 GOTO 80
```

Place the TURTLE with the read head onto a bright spot and enter RUN. "Light" will be detected as "LOGIC VALUE 1". This reading is stored in the variable HE (line 70). The information will appear on the screen through line 110. Then the program waits for you to press a key (lines 120 through 140) and then jumps to line 80. There a reading is taken again and the value (EX) is compared in line 100 with the first (HE). If the reading is at least 20 higher (HE+20) than the second value, the read head is on a dark surface. This corresponds to the state "LOGIC VALUE 0". Place the turtle onto different surfaces and measure the state or the logic value of the measured point in each case by pressing a key.
The information will now consist of

several consecutive bars just like the label with the bar code on the can of milk. Each bar is called a "bit". The TURTLE then reads the bits one after the other whenever it travels over one from left to right.
Before sticking the information bits on the track, something must be said about the line width. The TURTLE can only do one thing at a time: travel or measure. A travel step is 5 mm, as you saw in Chapter 9. This means that each of the bit bars in the form of light and dark surfaces must be spaced at least at intervals of 5 mm. Since the starting point for measurement can also differ by at least ±2.5 mm, you must select a minimum line width of 10 mm so that the photoconductive cell is definitely pointing to the line. We advise to use a linewidth of 15mm, however. The TURTLE will only point reliably to the center of each line when the width is equal to three turtle steps. Therefore cut 15 mm long strips of the black insulation tape and stick on the track as shown in Fig. 10.24.
Now change the program slightly. The printing expression is shortened and the key inquiry at the end is replaced. Enter:

```
80 PRINT "BIT PATTERN :";
90 &EX
100 Z$=" 1"
110 IF EX>HE+20 THEN Z$=" 0"
120 PRINT Z$;
130 &TF,3
140 GOTO 90
```

Line 150 is erased. The TURTLE is first with the read head at the start position (marked by * in Fig. 10.24). After starting the program with RUN, the binary value 1 is displayed, which means light. Then the TURTLE travels three steps (15 mm) forwards and stops on bit 1. A zero is displayed because this field is dark. Then it continues. After the last strip has been passed, the following should be displayed on the screen:

```
BIT PATTERN 1 0 1 0 0 1 0 .....
```

Stop the TURTLE pressing Control-C because otherwise the TURTLE will continue to look for bits and print them out.
Stick the insulation strips in a different sequence and read the information. Is it still correct? If not, the track maybe has different intensities of illumination or the read head is too far from the surface.
What is not satisfactory is that the TURTLE must initially be at the start position. It should also be capable of reading the infor-

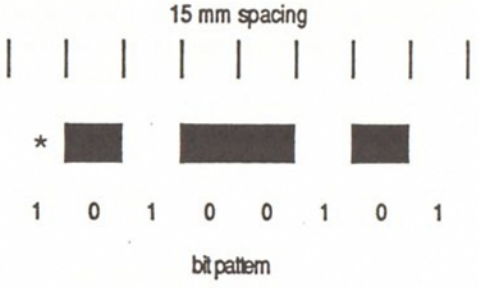


Fig. 10.24: Information code on the track

This type of identification is called a start bit. In data transmission over telephone or other lines, a start bit is always transmitted before each information block. There, only one bit is needed. Here, for reliability reasons, we have used two start bits. After transmitting the information, a delay must follow before the next transmission starts. This delay containing "nothing" is called a stop bit. As you can easily see, the stop bit must be at least one bit long. In data transmission, stop bits are used with a width of one, one-and-one-half and two bits.

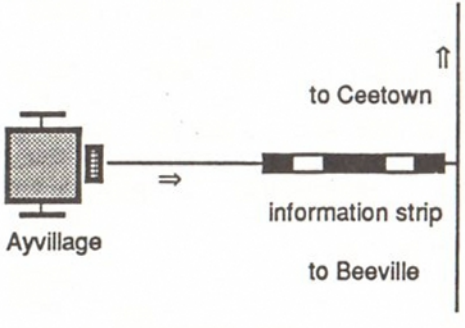


Fig. 10. 25: Freeway network

mation when it is traveling at full speed - just like the lightpen at the cash desk. In order to do this, use the first two strips, a dark one followed by a light one, as an identification ahead of the information field.
This will also allow the sensitivity of the read head to be adjusted ahead of each information block, similar to an automatic level control in a cassette recorder.
Expand the program like this:

```
80 REM SEARCH START BIT COMBI-
       NATION
90 MI=HE+15
100 &TF,1
110 &EX
120 IF EX<MI THEN GOTO 100
130 &TF,1
140 &EX
150 HA=EX
160 MI=(HA+HE)/2
170 &TF,1
180 &EX
190 IF EX<MI THEN GOTO 210
200 GOTO 170
210 &TF,4
220 HA=EX
230 C$=""
240 FOR I=0 TO 3
250 MI=(HA+HE)/2
260 &EX
270 IF EX<MI THEN C$=C$+" 1":
       HE=EX : GOTO 300
280 C$=C$+" 0"
290 HA=EX
300 &TF,3
310 NEXT I
320 PRINT " CODE: ";C$
```

Reading the data bits is now contained in an automatic FOR...NEXT loop (lines 240 through 310). Each of the 4 bits is compared with the start bit, the pattern ahead of bit 0. Place the TURTLE at the start of the track and start the program with RUN. At the end the screen should display:

CODE: 0 0 1 0

Again, try different information codes here. Reliability should be very much better through the two start bits and the automatic threshold value.
What can you do with the information read in? The TURTLE is a mobile robot and during its trip it should be supplied with important information.
We now want to show you how to set up a traffic control system for the TURTLE and to use data transmission from the track to the

read head. First build a small freeway network as shown in Fig. 10.25.
The TURTLE is in Ayvillage and wants to go to Ceetown. It doesn't know the way there. The traffic computer knows the map and the traffic situation and informs the TURTLE before the next turn-off in which direction it should travel. In reality, data would be sent through an induction loop in the street, but here they are being transmitted optically through the information strips.
Now you need to give the bits in the information code a meaning. Fig. 10.26 shows you how to do this.
With 4 bits you can count up to 15. Each bit position has a value. If all the bits are set, you get the number 15. If, for example, only bit 2 is set, the code is equal to the number 4. Of the 16 possible codes, we have selected only four:

Start Bit bit pattern	Code	Meaning
1 0 0 0 0	0	straight on
1 0 0 0 1	1	turn right
1 0 0 1 0	2	turn left
1 0 1 0 0	4	end of trip

Now supplement the program with the different cases for the four codes above.

Also enter these lines and add the information strips into the route as shown in the example in Fig. 10.25.

```
320 IF C$=" 0 0 0 0" THEN GOTO 100
330 IF C$=" 0 0 0 1" THEN
      &TF,13 : &TR,90 : GOTO 100
340 IF C$=" 0 0 1 0" THEN
      &TF,13 : &TL,90 : GOTO 100
350 IF C$=" 0 1 0 0" THEN
      PRINT " PROGRAM END" : END
360 PRINT " INVALID CODE FOUND."
370 GOTO 100
```

After it detects a code for turning, the TURTLE will be allowed to move forwards for a short distance so that it turns behind the code strips. After turning, the program comes back to the start so that it can look for the next code strip. In case of the code for "straight on" the program jumps back right away. The code for "end of trip" causes the program to end after a message is displayed on the screen. All other codes are displayed on the screen as invalid and the turtle will continue its path straight on.
Now place the TURTLE at the route start in Ayvillage and start the program with RUN. If there are no data transmission errors, the TURTLE will end up in Ceetown. Remember

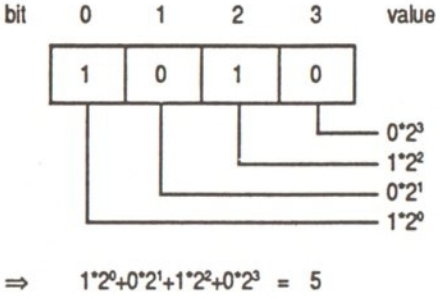


Fig. 10.26: Bits in information code

This type of system is called a traffic management or traffic control system. This is a computer network which collects information from road users (starting point, destination, etc.) and the route (traffic density, congestions, road works, etc.). From this information the driver is informed of the best route. The connection between the traffic computer and the on-board computer in the car is made by an induction loop in the road and sensors on board of the car.

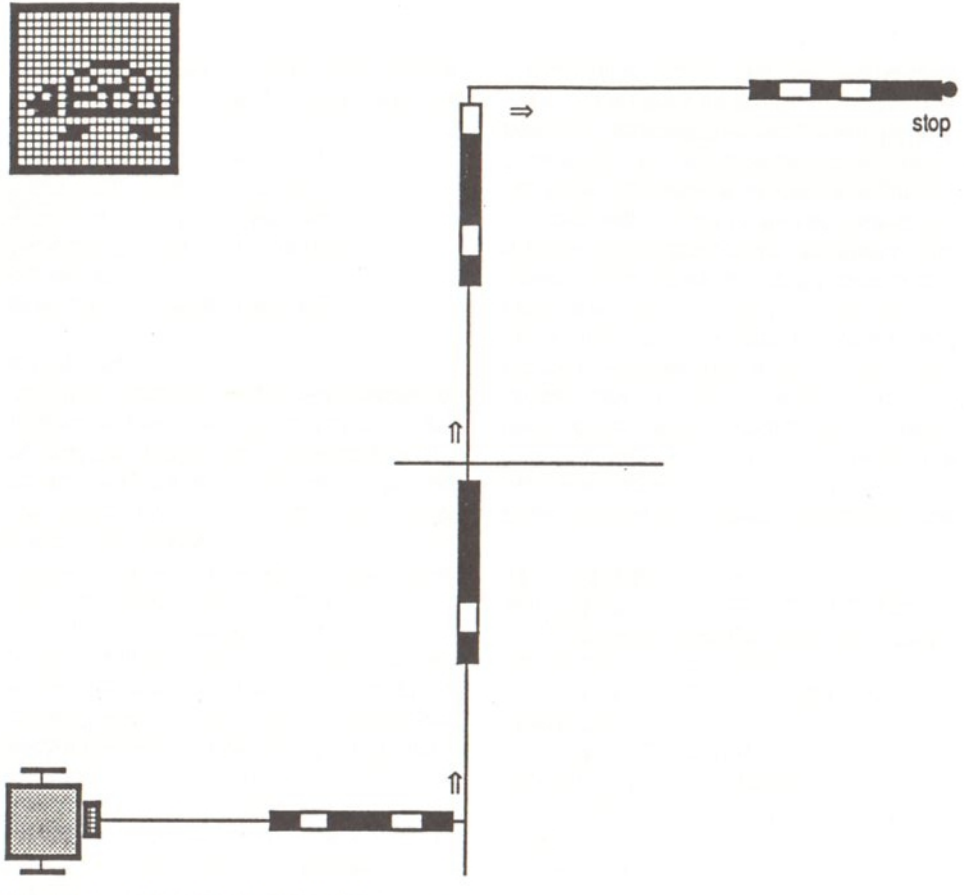


Fig. 10.27: The turtle should also manage this longer route

to provide uniform lighting and the correct clearance of the read head from the road surface!

You are free to assign meanings to the unassigned codes. You can write your own subprograms. Here are a few ideas: build in a different turn-off angle, for instance 45° to the right and to the left. Put on the flank of the main code strip other information codes which signal to the program that the turtle has left the track. If this code is found, a course correction can be performed. You can also encode the place-name signs:

Bit pattern	Code	Place-name
1 0 1 0 0 0	8	Ayvillage
1 0 1 0 0 1	9	Beeville
1 0 1 0 1 0	10	Ceetown
1 0 1 0 1 1	11	Deefort, etc.

You will find a complete program on the subject of "Information codes" under the name CODE.BAS on the floppy disk. It will show you how to handle the turtle when reading and processing route data, e.g. in traffic control systems.

11 Further experiments

Now you have worked through the entire instruction book and you have performed a number of very interesting experiments. But don't think that your fischertechnik COMPUTING EXPERIMENTAL is now worthless. Quite the contrary: you now know enough about computing and robotics to go on experimenting on your own. And you will see that you will be successful. But we would just like to give you a few more tips to help you on your way.

First, you can build a Morse trainer. When it functions, expand it using the lamp and the photoconductive cell to make an optical data transmission system. With the push-button you can then send Morse signals.

Then you can build a scale with the lamp and the photoconductive cell. A rotatable disk mounted on a slide can cover the photoconductive cell more or less as a function of weight.

With the same setup, you can build a control mechanism to maintain a platform in motion vertically.

A propeller and a light barrier are the basic devices to make a wind speed indicator. Just try to develop this device. The propeller can turn freely and is driven by the wind. On its axis is a wing which interrupts the light beam when it turns. The pulses are then counted and the computer can then calculate the wind speed.

Of course you can also expand the existing models: For example, you could build an overhead moving COMPUTER EYE. With that you could design a tracking device for solar energy panels.

You could even give the WELDING ROBOT a third axis to move the arm up and down. The welding tongs can also be replaced by an electric magnet working as gripper for metallic objects.

To improve obstacle detection, the TURTLE could be equipped with additional lateral sensors. If an arrangement of two photoconductive cells next to each other is used as sensor, deviation from route could be more easily and quickly detected by differential measurement.

The TURTLE could also be equipped with a moving arm. If you then attach a marker pen to the arm, it could even plot drawings.

You see that your fischertechnik COMPUTING EXPERIMENTAL offers you an almost limitless scope of possibilities for further highly interesting experiments.

This chapter is mainly intended to help you modify the programs presented in the text and on the disk. For this purpose we will summarize a couple of hints and technical details. This chapter does not profess to be complete. For sure you will discover a lot more tricks when delving into measurement and control, robotics, screen graphics and computer vision.
On the other hand this chapter is also not meant to contain necessary information. What was presented in the previous chapters will be perfectly sufficient for performing the experiments. However you might be interested to know how and why....

Annexe 1 Technical information

A 1.1 BASIC language extension

The interface commands start with the ampersand sign & followed by the command name. The ampersand sign might appear a little bit strange in the context of BASIC. However, it is the standard method provided by Applesoft BASIC to install language extensions.
The interface driver makes use of the routines supplied in the ROM. So if you have installed a modified ROM, you may run into conflicts. In that case return to the original set of ROMs.
The language extension resides in low memory range (addresses to). Then follows the high resolution screen bit map. The start address of BASIC-programs is shifted to16385 (hex $4001) thus providing more space for programs (approx. 21K).
The software requires a minimum of 48K of memory. It is recommended that the computer is equipped with 64 K, however. The language card memory space is used by the &GLOAD and &GC commands for storing the loaded picture.
In order to print graphics with the command &GPRINT on the printer, a printer driver is installed in the program FISCHER.BAS or LOADER.BAS. You have the choice of a variety of Apple- and EPSON-compatible printers via a serial or parallel printer interface, respectively.

A 1.2 Screen output and keyboard input

The fischertechnik COMPUTING EXPERIMENTAL software operates on the standard 40 column display. On Apple][+ do not use an 80 column display card, as it may conflict with the interface.
In order to display texts at a particular point on the monitor, the VTAB and HTAB commands are used. The screen is erased by the special command HOME. Switching over to inverse video is performed by the commands INVERSE and NORMAL, resp.
The cursor keys on the keyboard provide the following codes:

Cursor down	key ↓	10
Cursor up	key ↑	11
Cursor right	key →	21
Cursor left	key ←	8

The software on disk also scans the characters I, J, K, and M as alternative "cursor keys" on Apple][+'es, if necessary.
In cases a single character keyboard input is expected the GET command is used. In special occasions, however, scanning of the keyboard and scanning of other events (like an interface input) may alternate. The programs then directly test the keyboard input register. If the most significant bit is set, a character is available. Its code is given by the register's content minus the most significant bit. After input the keyboard register has to be reset.

A 1.3 High resolution screen graphics

Applesoft BASIC provides a high resolution screen mode for displaying graphics. The interface driver makes use of this facility when the command &GON is executed.
The mixed display of graphics and text screen is just the same as if you would have used the command HGR (280 pixels wide and 160 pixels high). Color control allows only for 140 different turtle traces in vertical direction, however. In order to bring well-defined colors on the screen, every turtle trace is two pixels wide. So vertical turtle traces belonging to the same pair of odd and even X-coordinates will coincide.
In order to avoid conflicts with the interface driver's memory usage, do **not** use the commands HGR, HGR2 and TEXT in own programs and only the commands &GON and &GEND. Also you must not modify the status of your display system by POKEing the flags described in the BASIC reference manual.
If your favorite paint program uses the same address range for pictures as the fischertechnik interface driver, it might be possible to catch pictures using the &GSAVE-command and use them for your robotics projects.

A 1.4 File types

With the Apple II DOS 3.3 there is normally no file type identification on the floppy disk. In order not to confuse you through the various types of files, extensions were supplemented to the file names similar to the MS-DOS names, which is the operating system of the IBM-PC. All BASIC programs have the extension ".BAS". Therefore, when loading the program SWITCH, enter:

LOAD SWITCH.BAS

The pictures on the floppy disk have the extension ".PIC". However, do not enter this extension when using the commands &GLOAD and &GSAVE because the command does it on its own. On the floppy disk you will also find some files with the extension ".OBJ". These contain the BASIC language extension routines. As they are written in the microprocessors native language you cannot load and read them like BASIC programs.
The file TURTLE.SHAPE contains the respective shapes of the graphics turtle as it points to sixteen different directions.
The file LOADER.EXEC is used during installation of the interface driver.

A 1.5 Different generations of Apple II

Apple II computers have been produced in various generations, starting from the predecessor Apple][and ending by the time of printing with the new Apple //gs. In addition there exists a number of compatible computers or even Apple II clones. Any software to operate on such a wide span of computers must rely on some minimum requirements, eventually old computers have to be upgrated to. On the other hand the software cannot make full use of the features of the new computers.
What concerns the fischertechnik COMPUTING EXPERIMENTAL software, it requires a minimum of 48K of main memory; 64K are recommended, however.
The software will use upper case textes only, as the lower case characters may lead to confusing results on the Apple][+. Of course you may improve the programs.
Also 40 column mode is used, as switching to the 80 column mode will lead to conflicts with the interface control on Apple][+ video interface cards.
DOS 3.3 has been chosen as operating system despite its known disadvantages, as it runs on all generations of Apple II computers.

&1X Switch off motor 1
This command switches motor 1 off. It corresponds to the command CALL M1,OFF in the interface manual.

&2X Switch off motor 2
This command switches motor 2 off. It corresponds to the command CALL M2,OFF in the interface manual.

&1B Motor 1 reverse
This command switches motor 1 to run clockwise. Then the command checks whether input E2 is released (0) and whether it is pressed again (1). The motor is then switched off.

&2B Motor 2 reverse
This command switches motor 2 to run clockwise. Then the command checks whether input E4 is released (0) and whether it is pressed again (1). The motor is then switched off.

&3B Motor 3 reverse
This command switches motor 3 to run clockwise. Then the command checks whether input E6 is released (0) and whether it is pressed again (1). The motor is then switched off.

&1F Motor 1 forwards
This command switches motor 1 to run counterclockwise. Then the command checks whether input E2 is released (0) and whether it is pressed again (1). The motor is then switched off.

&2F Motor 2 forwards
This command switches motor 2 to run counterclockwise. Then the command checks whether input E4 is released (0) and whether it is pressed again (1). The motor is then switched off.

&3F Motor 3 forwards
This command switches motor 3 to run counterclockwise. Then the command checks whether input E6 is released (0) and whether it is operated again (1). The motor is then switched off.

&1L Motor 1 counterclockwise
This command switches motor 1 to run counterclockwise. It corresponds to the command CALL M1,CCW in the interface manual.

&2L Motor 2 counterclockwise
This command switches motor 2 to run counterclockwise. It corresponds to the command CALL M2,CCW in the interface manual.

&3L Motor 3 counterclockwise
This command switches motor 3 to run counterclockwise. It corresponds to the command CALL M3,CCW in the interface manual.

&1R Motor 1 clockwise
This command switches motor 1 to run clockwise. It corresponds to the command CALL M1,CW in the interface manual.

&2R Motor 2 clockwise
This command switches motor 2 to run clockwise. It corresponds to the command CALL M2,CW in the interface manual.

&3R Motor 3 clockwise
This command switches motor 3 to run clockwise. It corresponds to the command CALL M3,CW in the interface manual.

&3X Switch off motor 3
This command switches motor 3 off. It corresponds to the command CALL M3,OFF in the interface manual.

&4B Motor 4 reverse
This command switches motor 4 to run clockwise. Then the command checks whether input E8 is released (0) and whether it is pressed again (1). The motor is then switched off.

&4F Motor 4 forwards
This command switches motor 4 to run counterclockwise. Then the command checks whether input E8 is released (0) and whether it is pressed again (1). The motor is then switched off.

&4L Motor 4 counterclockwise
This command switches motor 4 to run counterclockwise. It corresponds to the command CALL M4,CCW in the interface manual.

&4R Motor 4 clockwise
This command switches motor 4 to run clockwise. It corresponds to the command CALL M4,CW in the interface manual.

&4X Switch off motor 4
This command switches motor 4 off. It corresponds to the command CALL M4,OFF in the interface manual.

&DI Digital input
This command reads the digital inputs E1 through E8. The value (0 for open, 1 for +5V applied) is stored in the variables E1 through E8.

&EX Analog input EX
This command measures the resistance applied to input EX. A resistance of 0 ohms (directly connected to +5V) gives a small value (approx. 20), and a resistance of 5 kohms gives a large figure (approx. 200).

&EY Analog input EY
This command measures the resistance value applied to input EX, see also command &EX.

&G0 Graphics pen off
This command switches the graphics turtle pen off. With the following commands &GF or &GB, no line is drawn.

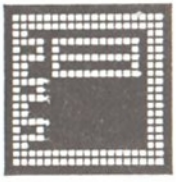
&G1 Graphics pen on
This command switches the graphics turtle pen on. With the following commands &GF or &GB, a line is drawn in the pen color selected. This is the initial state after the first command &GON.

&GB,S Graphics turtle backwards
This command moves the graphics turtle by S steps backwards. For further details see the command £GF.

&GC Copy graphics
This command copies the picture from the invisible graphics screen to the visible graphics screen. The invisible graphics screen is either cleared (background color 0) or receives a picture from the floppy disk through the command &GLOAD. In this way a "clean" background can always be generated. The command sets the graphics turtle to its starting point and the state of the graphics pen is not changed.

&GEND Graphics off
This command switches back to text display only. The screen is cleared.

&GF,S Graphics turtle forwards
This command moves the graphics turtle by S steps forwards. The expression S must be a value between 0 and 32767. Steps



greater than approx. 100 make the graphics turtle normally leave the screen. However, it is not lost. Its position is tracked and can be brought back to the screen by entering the suitable commands.
If the graphics pen was switched on, a line in the selected color will be drawn from the previous position to the destination position (see commands &GON and &GP).

&GH Graphics heading
The command &GH determines the current heading of the graphics turtle and stores it in the variable GH. The heading is 0° when pointing upwards, 90° when pointing right, 180° when pointing downwards and 270° when pointing left.

&GL,D Graphics turtle left
This command turns the graphics turtle through D degrees to the left. The value contained in the expression D must be an integer and must be between 0 and 359.

&GLOAD,F$ Load graphics
This command loads the graphics background from disk . The picture is stored with the name F$, extended by the file identification ".PIC". The string F$ can be a string variable or a string expression. If no file name is given, the user is requested for it in dialog.

&GON Switch on graphics
This command splits the screen into a graphics screen and a text screen. The two screens are cleared and the graphics screen has the background color 0. The graphics turtle is set to its starting point and the graphics pen is switched on in color 1. Each successive call by &GON without an intermediate command &GEND clears the graphics screen with the background color 0 and switches the graphics pen on.
All further graphic commands (with the exception of &GLOAD, &GSAVE and &GPRINT) must be preceded by the command &GON.

&GP,C,P Graphics pen
This command selects the pen color of the graphics turtle. The expression C must be one of the values 0, 1, 2 or 3, which correspond to black, green, magenta and white or black, brown, blue and yellow, depending on the palette number P, 1 or 2, resp.

&GPRINT Print graphics
This command prints out the graphics on the printer attached.
One of the printer drivers supplied must be installed to execute the command.

&GR,D Graphics turtle right
The command turns the graphics turtle through D degrees to the right. The value contained in the expression D must be an integer and must be between 0 and 359.

&GS,C Graphics background
This command sets the background color C of the graphics screen. The new background color is only executed after the next command &GON.

&GSAVE,F$ Store graphics
This command stores the visible graphics screen to disk. The picture is stored under the name F$ and extended by the file extension ".PIC". The string F$ can be a string variable or a string expression. If no file name is given, the user is requested for it in dialog.

&GT Graphics sensor
This command tests the last color pixel which the graphics turtle was on. This can be a background pixel or of a the previously drawn line. The pixel color (0 through 3) is stored in the variable GT.

&GX Graphics turtle X position
This command determines the X position of the graphics turtle and stores it in the variable GX. The coordinate X is 0 at the starting point of the graphics turtle, and positive to the right, negative to the left.

&GY Graphics turtle Y position
This command determines the Y position of the graphics turtle and stores it in the variable GY. The coordinate Y is 0 at the starting point of the graphics turtle, and is positive upwards and negative downwards.

&IN Initialization
This command initializes the interface. All other commands must be preceded by the &IN command because the &IN command installs the system variables and performs other preparatory work.

&TB,S TURTLE backwards
This command moves the TURTLE by S steps backwards. For more details see command &TF. The command sets the number of executed steps in the variable TS. The input E5 is not checked - as opposed to the command &TF.

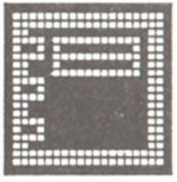
&TF,S TURTLE forwards
This command moves the TURTLE by S steps forwards. One step is 5 mm long. The expression S must have a value between 0 and 32767. The command &TF checks the input E5 (the bumper on the TURTLE). If E5 is open (0), the command is interrupted. When the command in terminated, the variables E5 and TS are set. The variable E5 contains the state of the variable E5 and the variable TS contains the number of successfully executed steps. The number contained in TS can be smaller than S if E5 is open before the end of the track. By checking E5 and TS, therefore, it can be determined whether and when a collision took place.

&TH TURTLE heading
The command &TH determines the current heading of the TURTLE and stores it in the variable TH. The heading is 0° when pointing in the starting direction, 90° when pointing to the right, 180° when pointing away from the starting position and 270° when pointing left.

&TI TURTLE initialization
The command &TI is required before using any other TURTLE commands. It sets the current position and heading of the TURTLE (X and Y positions, heading) to 0. The command can also be used at any time later to set a new position as starting point.

&TL,D TURTLE left
This command turns the TURTLE through D degrees to the left. The value contained in the expression D must be an integer, divisible by 5 and must be between 0 and 355. The command stores the number of steps executed (not in angular degrees!) in the variable TS. The input E5 is not checked - as opposed to the command &TF.

&TR,D TURTLE right
This command turns the TURTLE through D degrees to the right. The value contained in the expression D must be an integer, it must be divisible by 5 and be between 0 and 355. The command sets the number of steps executed (not in angular degrees!) in the variable TS. The input E5 is not checked - as opposed to the command &TF.



&TVAL,B TURTLE brake
The command &TVAL controls the countercurrent brake. With all step commands &1F, &1B,...,&TF, &TB, &TR, &TL, the flow of current through the motor is reversed for a short time after completion of the step and is then switched off. The motor thus stops suddenly. The length of the countercurrent pulse (1 through 255) is set by the command &TVAL. You will not normally require the command &TVAL because the default value has already been optimally set.

&TX TURTLE X position
This command determines the X position of the TURTLE and stores it in the variable TX. The coordinate X is 0 at the starting point of the TURTLE, and is positive to the right, negative to the left.

&TY TURTLE Y position
This command determines the Y position of the TURTLE and stores it in the variable TY. The coordinate Y is 0 at the starting point of the TURTLE, and is positive in the starting direction, negative in the opposite direction.

Acknowledgements

We would like to acknowledge the help of the following companies by supplying pictures for the illustrations of this manual:

Bleichert Förderanlagen GmbH, Osterburken FRG (pages 73 and 85)

Valvo Unternehmensbereich Bauelemente der Philips GmbH, Hamburg FRG (page 36)

Wagner Fördertechnik GmbH&Co. KG, Reutlingen FRG (page 100)

Art.-Nr. 39504 · 03.89 ZE