Mobile Robots II

- Begleitheft
- Activity booklet
- Manuel d'accompagnement
- Cuaderno adjunto
- Folheto

fischertechnik

GB+USA

CONTENTS

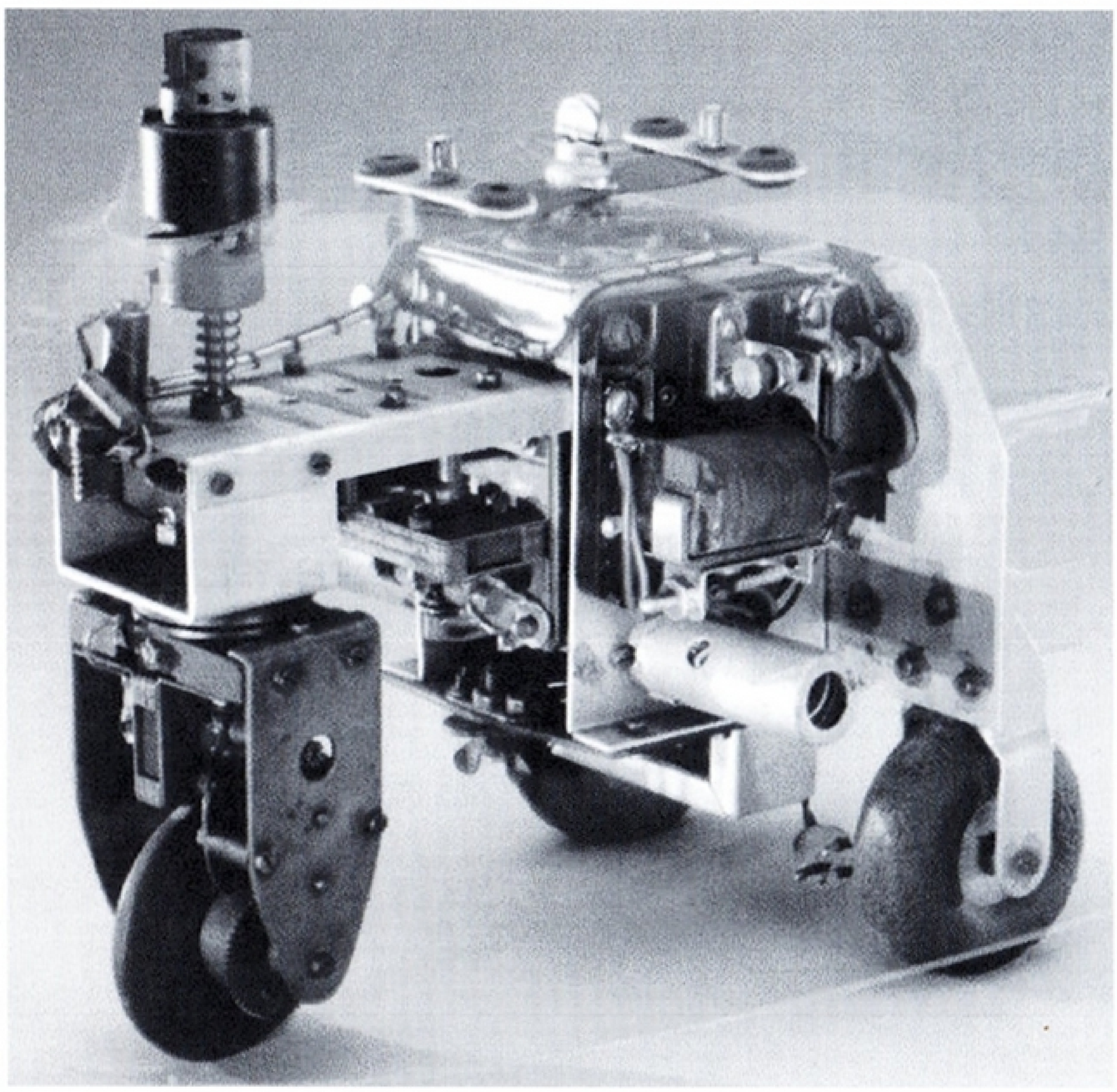
GB+USA

1 What Do We Need Robots for?

Before we deal with the practical aspects of robot technology, we want to try to answer the slightly provocative question in the heading.
The term "Robot" was first used in the novel "Golem" by Carel Capek in 1923. This gloomy, artificially created figure was supposed to replace human work with its capabilities.

As often with literary figures, it is subject to compulsive behavior and we regard the figure with a certain degree of distrust. Robots became more of a kind of automatic device in the 30s and 40s of the twentieth century. Various attempts to give them external human characteristics, e.g., a head with blinking lights as eyes or primitive speech output via a loudspeaker, seem naive from today's point of view. Apparently, the fears of potential domination by robots over people cannot be refuted so easily.

But little mobility, not to mention intelligence, can be seen in these first simple experiments with the constructed machines. The building of robots only became realistic with the invention of electronic circuits.
The problem of required control principles is closely linked with actual robot technology. This question about the "intelligence" of robots is the object of research and investigation of many companies, institutes and universities today.



The first solution strategies were seen in cybernetics. The term "cybernetics" is derived from the Greek word Kybernetes. Kybernetes was the navigator on Greek deep-water ships. He had to determine the ship's location and calculate the necessary course.
Consequently, it is clear that cybernetics should make robots "intelligent." How can we imagine such intelligent behavior in general?

We want to try to make this clear by experimenting with our imagination. Each of us has already observed the behavior of a moth in the light cone of a lamp. The moth detects the light source, flies toward it, and then avoids the lamp shortly before crashing into it. It is clear that the moth has to detect the light source, determine the path to it and then fly to it in order to behave as it does. These abilities are based on the instinctive, intelligent behavior patterns of the insect.

Now let's try to transfer these abilities into a technical system. We have to detect the light source (optic sensors), perform movement (control motors), and we need to establish a meaningful relation between detection and movement (the program).
Our experiment with our imagination then combines an optic sensor with a motor and logic, so that this motor controls the vehicle to always move in the direction of the light source. This vehicle would then act exactly like a moth, wouldn't it?
A British man named Walter Grey conducted the described experiments with such technology in the 50s. Using simple sensors, monitors and electronic circuitry, he created several "cybernetic" animals, which had very specific behavior such as a moth.
These machines represent an important step on the path to modern, mobile robots. The sensors (photoelectric resistance, detectors, etc.) of the machines controlled the actuators (motors, relays, lamps, etc.) using their electronics, so that (apparently?) intelligent behavior resulted. The picture shows a copy of the "cybernetic" turtle, which is displayed in the Smithsonian Museum, Washington, D.C.

Based on these considerations, we are going to create corresponding "behavior patterns" and try to make them comprehensible to a robot in the form of programs.
But how can be answer the question about the benefits of mobile robots asked at the beginning of this text? To answer this question concretely, let's try to apply the previously rather abstract behavior of our "imagined moth" to technical issues. A simple example of this is the search for light. We modify the light source in that we place a bright strip, a guiding line, on the floor and direct the sensors down instead of forward as previously. Using such guidelines, mobile robots can be given a sense of orientation in a warehouse, for example. Additional information, e.g., in the form of a barcode at specific spots of the line cause the robot to perform further operations at such positions, e.g., to pick up or put down pallets.

Such robot systems actually already exist. In large hospitals, there are sometimes very long transport routes for consumables such as sheets. Transport of these materials by nurses or other care-givers takes a lot of time and involves hard manual labor in part. In addition, such work uses up the time available for caring for patients.
Consequently, we can see that mobile robots can take an important place in modern society. But what is the relation of this to fischertechnik construction kits?

In addition to sensors and actuators for a robot, we need many mechanical parts to construct a model. The fischertechnik construction kit Mobile Robots II

is an ideal basis for this. We can combine the mechanical parts in an almost unlimited variety and build sturdy robotic vehicles. Using the associated "Intelligent Interface" (Art. no. 30402, which must be purchased separately), we also have sufficient computing power to design demanding programs. Numerous different sensors and actuators are linked and evaluated via this interface. The sensors convert physical quantities such as light quantity or temperature into electrically recordable values. This includes both analog and digital quantities. Digital quantities are those that can be either logically true or false. These states are identified with 0 or 1. A switch is a good example of a digital sensor.

But many quantities change continually between their maximum and minimum values; these are called analog values. They cannot be displayed simply as 0 or 1. For a computer to be able to process these quantities, they must be converted into corresponding numeric values. The fischertechnik interface provides two analog inputs EX and EY. The resistance value applied to these terminals is stored in a numeric value. The measured values of a temperature sensor, for example 0...5 kΩ, are recorded in a range from 0 to 1024 and are available for subsequent processing. The most important function of the interface is in the logical linkage of input values. The interface requires a program for this. The program decides how the sensor signals, corresponding output data, motor control signals, etc. are created from input data.

A graphic work sheet exists, so that we can write programs necessary for the interface as effectively as possible. There is a software program behind the term "work sheet," which makes it possible for us to create our programs on a very high level. In fact, we can design programs and algorithms using graphic symbols. The computer of the Intelligent Interface can actually only execute commands from its machine command set. These are essentially simple logical or arithmetic control structures, whose use is extremely difficult for beginners. Consequently, the PC software LLWin (Art. no. 30407, not included in the construction kit) provides graphic elements, which can then be translated into a language that the interface can use.

We will proceed step-by-step into the fascinating world of mobile robots. We will begin with a simple test construction to check the basic functions of interface and sensor technology. Then we will start with simple models, to which specific tasks are allocated. Later we will try out increasingly complicated systems. To ensure that any errors, which occur, do not result in permanent dissatisfaction, we will get to know the properties and special characteristics of sensors and actuators in one chapter. There is a "Troubleshooting" section at the end for very "stubborn" cases.

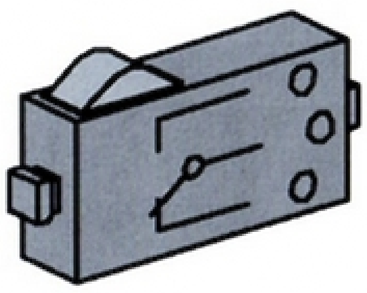
It is very important to take a great deal of care when setting up and starting operation of our robots. We are dealing with complex machines, whose only difference to other robot systems in use throughout the world is their comparatively small size. When we assemble electric components, we should follow the guidelines carefully and check two or three times to make certain that everything is correct. In mechanical constructions, including our own creations, we should pay a lot of attention to softunning and lack of play in gears and attachments. We should never use "force" during assembly.
Writing new programs and consequently defining new "behavior," whose complexity is only limited by the available resources for memory and computing power, depends on our own creativity. The following examples should provide some ideas for this.

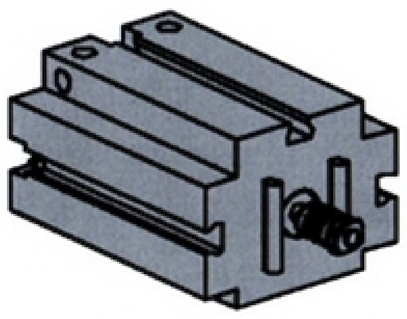
2 First Steps

Following the theoretical considerations, we now want to finally begin our own experiments. Some of you would certainly like to start immediately, maybe even with the complicated automatic forklift. This is of course possible, and if you follow the construction instructions carefully, you can construct the model right away.

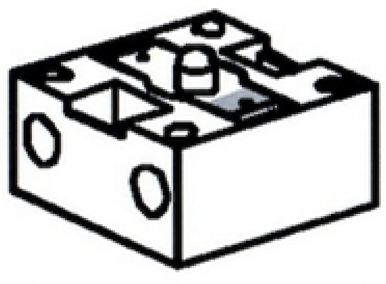
But what can you do if it doesn't work? In such cases, you must search for the cause of the error systematically. When you do this, questions inevitably arise about the mode of operation and the properties of the components used. A certain degree of basic knowledge about sensors and actuators is apparently indispensable. But before we start to learn about these things, we should check the interaction between computer and interface.
Install the control software on your PC in line with the guidelines from the LLWin manual.

Using the interface diagnosis (Check Interface), we can test the different sensors and actuators. For example, we can connect a pushbutton with two lines at input E1 and then see which logical switching state the interface detects. When you press the pushbutton, the state must change at the corresponding input.





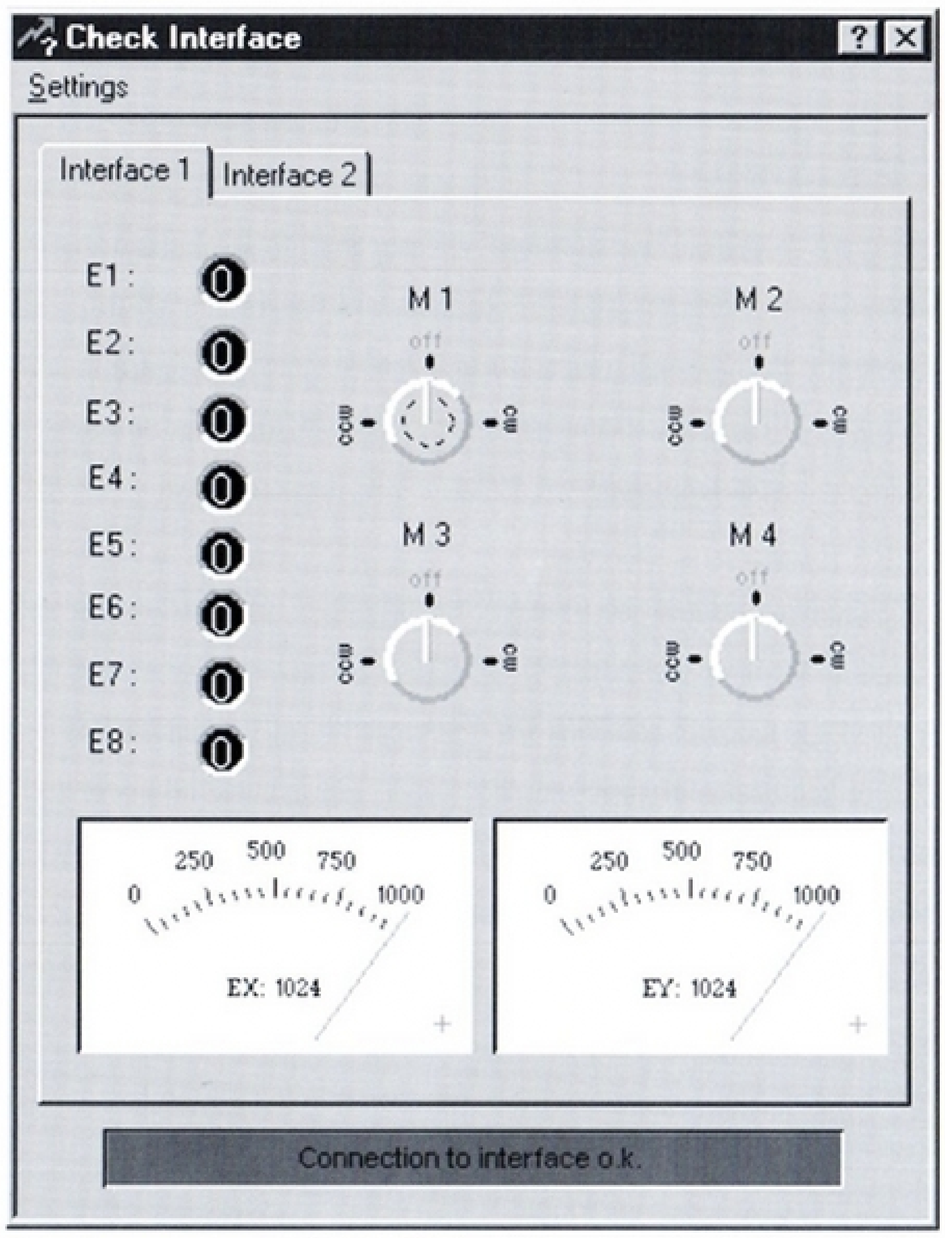
We can check the outputs by connecting a motor with a motor output, e.g., M1. Using the mouse, we can change the rotation of the motor. If we want to test the analog input, we can use a phototransistor as an analog sensor. While the polarity does not play any role for motor or pushbuttons (the motor rotates in the reverse direction in an unfavorable case), the correct connection of the phototransistor is imperative for correct functioning. One contact of the transistor is identified with a red marking; we connect this one with a red connection plug, and the contact not marked with a green plug.



The second red plug is inserted in the socket closer to the edge of the interface of the EX input; the second green plug is inserted in the socket of EX placed further inside. Then we can vary the lighting strength of the phototransistor using a flashlight and consequently change the pointer deflection.

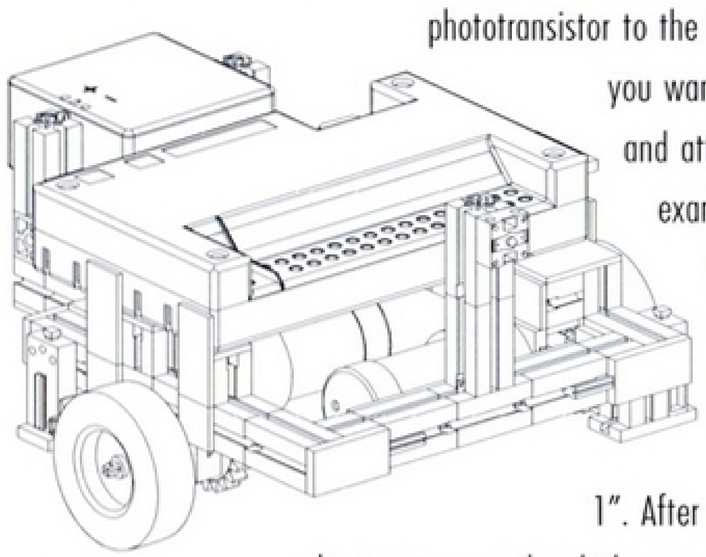
GB+USA

If the pointer does not move away from its maximum deflection, we should check the connections of the phototransistor once again. On the other hand, if the pointer points to zero when the flashlight is switched off, the lighting in the room, i.e., the ambient brightness, can be excessive. The pointer deflection changes when we cover the phototransistor.



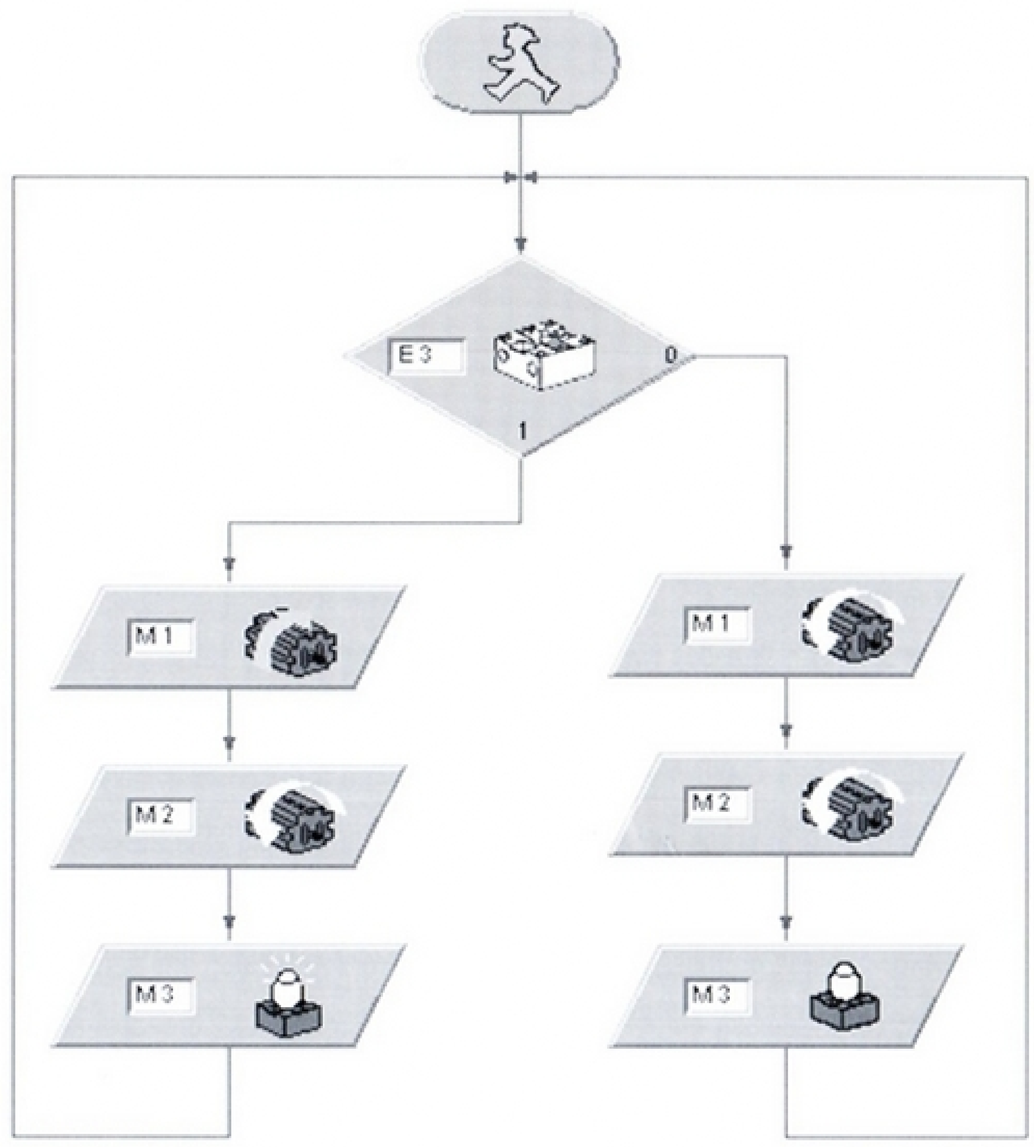
We would like to talk briefly about the color assignment of the plugs again. We should pay very close attention during assembly to ensure that a red plug is connected to a red wire and a green plug to a green wire. If we use polarized signals in circuit design, we always use a red wire for the plus pole and a green wire for the minus pole. This might seem a bit pedantic (and the electric current does not care which color a wire has), but clear and unique color assignment simplifies systematic searches for errors substantially.

We want to conclude our first step into the area of robotics with a simple program. Set up the basic model with the two drive motors and the support wheel according to the construction instructions. Only connect the motors to the outputs M1 and M2. Also take the phototransistor and connect it to input E3 (pay attention to the polarity). Before you do this, attach the phototransistor to the basic model, so that it looks "forward." If you want to, you can connect the lens lamp to M3 and attach it next to the battery pack, for example, but this is not absolutely necessary.



Open the LLWin program and create a new project (PROJECT – NEW). LLWin provides us various templates; select "Empty project" and give it a name, e.g., "Step 1". After you press the [OK] button, a blank work sheet appears with a little green man and the block window. The little green man symbolizes the program start.

All our programs begin from this starting point. We retrieve the various program parts from the block window using the mouse. The symbols there stand for the inputs and outputs on the interface. We can place the desired symbol using the mouse button on the left, and change the properties using the mouse button on the right.



The pushbutton symbol identifies an input. Place the pushbutton under the start symbol for our program. Once you release the symbol, a selection dialog appears. Select phototransistor. If you want to make changes later, you can activate this dialog using the mouse button on the right. Assign the outputs to the motors and indicate the desired rotational direction. We want the motors to rotate in the same direction when no light hits the phototransistor, and in the opposite direction when light is detected. Then link the elements using the draw function. The lamp on M3 signals the state of the phototransistor.

The drawing shows the precise link of the program branches. If you are not certain whether everything is correct, compare your program with the Step1.mdl program. Save your own program before you do this, and load the Step1.mdl file from the CD-ROM, which is contained in the construction kit.

If everything is correct, the program is downloaded into the interface and started immediately (RUN - DOWNLOAD).

The first robot then rotates on one spot. It does this until we attract him with a light source. As soon as the phototransistor detects the light, the motors, which previously rotated in opposite directions, are rotated in the same direction and the robot moves straight toward the light source. If it moves away from the light source, we must change the poles of both motors. It will probably not move in a precisely straight direction, so that the phototransistor will lose contact to the light source after part of the way. Then its movement will switch from Move forward to Rotation, and the light search starts anew. Provide sufficient space for the robot, since it can unfortunately not (yet) detect obstacles in its path.

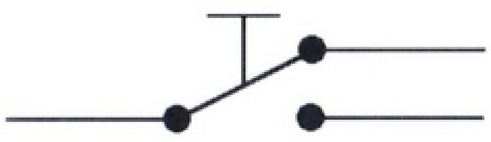
3 Sensors and Actuators

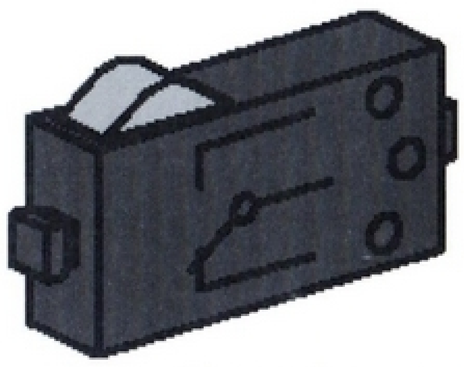
You now know that our most important unit, the interface, "plays" together with the PC. Now it is a question of how the interface can detect the signals from our environment.

Let's start with the inputs. In technical language, input signals are often simply called inputs. There is only one possibility for a computer, and this includes our Intelligent Interface, to detect and process signals. We have to make the environmental stimuli "computer-compatible." Consequently, all sensors are converters, which convert the desired "sense" into an electric signal. Because we do not want to follow the construction instructions "blindly," it makes sense to look into the basic properties of the available sensors.

This is even more important for expanding the interface later to handle new applications that you define yourself.

3.1 Switches as Digital Sensors

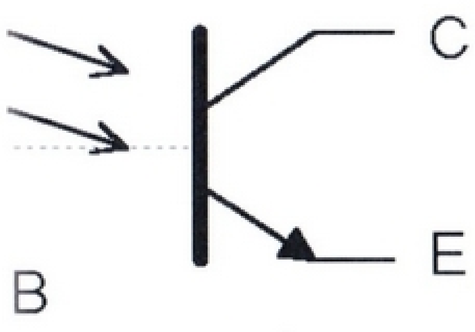


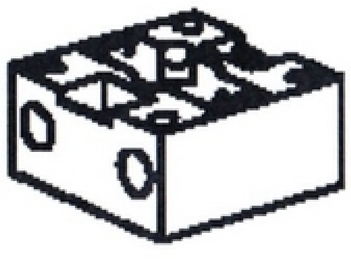


The simple, logical levels "0" and "1" can be depicted using a switch. Precision snap switches are used in the system of fischertechnik construction kits. The special property of snap switches lies in their switching behavior. If you press the red button carefully and slowly, you feel a clear pressure point when the switch contact switches over with a slight clicking sound. If we release the switch lever slowly, we have to let the lever go substantially further than the original switch on point to switch it back. This difference between mechanical switching on and off positions is called hysteresis. The switching hysteresis of contacts or other electronic switches is an important property. If they did not exist, i.e., the switching-on point would be the same as the switching-off point, substantial programs in signal assessment would result. Tiny interferences such as very slight jittering in the switching time point would result in several unintentional contact activations; it would not be possible to count events precisely. The switch is designed as a change over switch. Consequently, you can assess both imaginable starting positions, i.e., closed and open when idle, in your experiments.

3.2 Light Detection using the Photo-transistor

The phototransistor is a semi-conductor element, the electric properties of which are dependent on light. A normal transistor is a component with three connections. These connections are called emitter, base and collector. Its main task is to amplify weak signals. Weak current, which flows from a signal into the base of the transistor, results in much stronger current at the collector of the transistor.





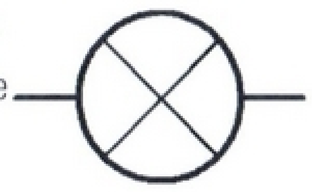
The current amplification can reach factors of more than 1,000. But the phototransistor from the construction kit only has two connections. What happened to the third connection?

We want to detect light with our transistor. Everybody is familiar with solar cells, with which power is generated using sunlight. The phototransistor should be understood as a combination of mini solar cell and transistor. The base connection is not to the outside (consequently, it appears as a dotted line in the drawing). In its place, light pulses (photons) generate a very small photocurrent, which is then available amplified from the transistor at the collector for assessment. In order for this to function as described here, the phototransistor requires additional external wiring. Because this is contained in the interface, this is not important for us.

The phototransistor can be used both as a digital sensor and an analog sensor. In the first case, it serves for detecting clear light-dark transitions, e.g., a marked line. But it can also differentiate the strength of light; then the phototransistor operates as an analog sensor.

3.3 Signal Output using an Incandescent Bulb

An incandescent bulb serves for outputting simple light signals. To put this in technical language, we will call the incandescent bulb an optic actuator. The structure of an incandescent bulb is very simple. A filament made of thin tungsten is mounted between two connection pins in a glass bulb, in which there is a vacuum. If current flows through the filament, the tungsten filament heats until it glows white. Because there is no oxygen in the glass bulb, the filament does not burn and consequently the lamp has a long operating life. Due to the strong thermal stress on the coiled filament, the wire filament expands each time the light is switched on and contracts when it is switched off. These minimal movements due to material wear result in "burning out" of the incandescent bulb at some time.



One possible use of incandescent bulbs is for displaying switching states. Warning messages can also be generated by the programming of a blinking lamp.

We also need the bulb in another case. A special sensor is created together with two phototransistors, with which lines can be detected. The bulb works as a light source, so that the phototransistor can detect color marking based on light reflected with different strengths.

A special feature of the incandescent bulb, which is used in fischertechnik construction kits, is the optic line contained in the glass bulb. This improves focusing of the light rays, and markings are detected more reliably as a result, for example.

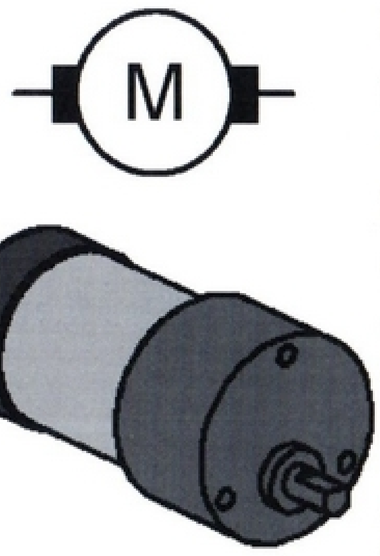
3.4 Direct Current Motors as Power Source

Direct current motors are important actuators for mobile systems. Two different types of motors are contained in the "Mobile Robots II" construction kit. Although they differ greatly in a mechanical sense, their electric structures are identical.

Direct current motors are made of a rotating "rotor" and a fixed "stator." The rotor should be understood as a conductor loop in principle, which is in the magnetic field of the stator. If current flows through the conductor loop,

GB+USA

power is generated, which results in displacement of the conductor in the magnetic field. The rotor moves. The simple conductor loop is designed as a coil for practical applications (with or without iron core for amplifying the magnetic field). Very many direct current motors generate the required magnetic field using permanent magnets, which are stuck into the metal coating of the stator. Current is fed to a rotating rotor using sliding contacts. These contacts provide the current direction reversal in the conductor loop at the same time, which is necessary for uninterrupted rotational movement. The rotational speed of normal motors is in the range of a few thousand revolutions per minute. Gears provide for lower rotational speeds with large torque.

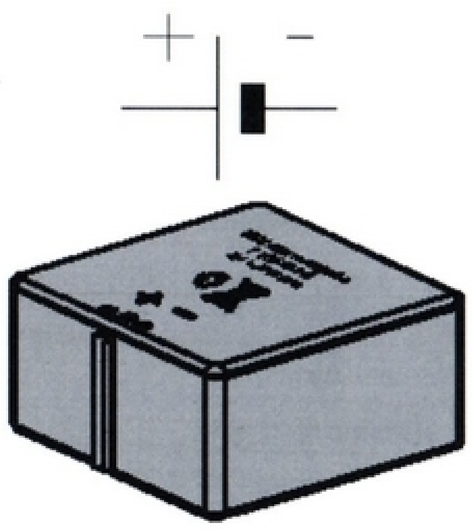


The construction kit contains two different motor types: the mini motor and the power motor. The small, compact mini motor with worm is designed for auxiliary drive or special purposes with small power requirements. It always requires gears for reducing rotational speed.

The power motor provides much larger torque. It has fixed, flanged gears with a reduction ratio of 50:1. Consequently, it is ideal for the drive requirements for our robot. But it also exists in a variant with a reduction ratio of 8:1 (not contained in this construction kit). But the rotational speed would be too great on the drive shaft for the robot drive.

3.5 Power Supply

Mobile systems require an autonomous power supply. All energy users are supplied from this energy source. The power supply requirements differ. While drive motors are content with destabilized voltages, many sensors require stable voltages to provide precise results.



For economic reasons, the use of batteries or accumulators is the only meaningful way to supply robots with current. Solar cells or fuel cells are unfortunately not sufficiently powerful to provide practical results with a reasonable amount of expense.

Accumulators are preferable to normal batteries, because they can be recharged many times. The fischertechnik Accu Set provides a good compromise between energy supply and size. The Accu Set is not a component of this construction kit. It can be purchased together with a special charger as an "Accu Set" with article number 34969. The switching symbol and the accumulator are displayed in the drawing. In a normal case, the polarity is not shown in a switching symbol. You can remember more easily which connection is plus using this simple example: "You can cut through the long line and make it a plus."

It is very important to pay attention to correct polarity when you connect voltage sources.

3.6 Additional Sensors

The fischertechnik system can be expanded relatively easily with additional sensors. In the simplest case, we can use sensors from other kits, e.g., the thermal sensor or the magnetic sensor from the "Profi Sensoric" construction kit, article no. 30491.

But we can also use completely different sensors. Very different kits and components are for sale in specialized stores. Even exotic sensors such as gas detectors or radar sensors can be used. But because we do not want to destroy the Intelligent Interface with high input voltages or incorrect loads, only experienced do-it-yourselfers should create their own solutions. The most reliable way to connect additional sensors is to separate sensor and interface galvanically. A number of sensors have a relay, which is well suited for this. The switching contacts of the relay are connected like a customary fischertechnik switch and then signal the occurrence of new environmental stimuli. Tip: Such expanded experiments are published by enthusiastic "fischer technicians" in the Internet.

4 Robot Models

A few variants of mobile, autonomous robots are presented in the following construction suggestions. We'll start with a simple model. Based on that, you can use your imagination and try using different sensors. Here is it a question of linking both the internal states of the robot, e.g., distance measurement by impetus wheels, and the external environmental signals such as light or searching for lanes. Then specific tasks are set for each model. They are designed to stimulate your imagination and make you familiar with the material. The LLWin programs for the individual tasks are on the CD-ROM, which is included in the construction kit. But try to think up your own tasks for the models. The simplest model is the basic model. The drive motors are assembled with the interface to create a compact unit with it. Two motors provide the drive power of the robot. They are arranged opposite each other, so that each motor runs a drive wheel. A support wheel provides stability, so that this robot does not tip over. Such an arrangement of the motors is called differential drive. It provides the highest degree of mobility with the minimally required movement space. Turning in place is even possible. But the center point of both motors is the center of rotation around which the robot moves. In this way, it is able to navigate in the most difficult situations with little computing work.

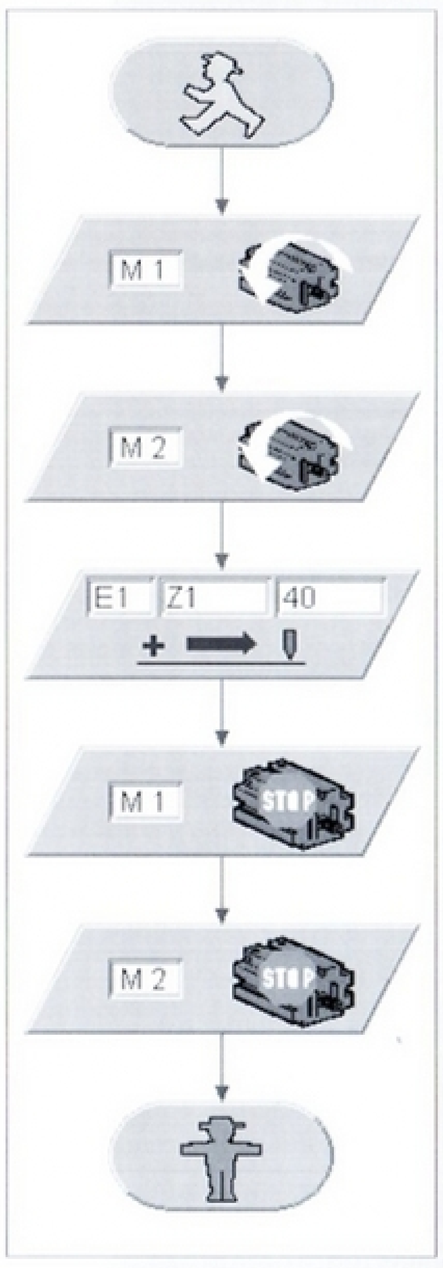
The motors can drive the wheels via two different gear reductions (slow 100:1 or fast 50:1). For the slower variant, the drive is reduced additionally at a ratio of 2:1 using fischertechnik toothed gears. Which gear reduction is used is indicated for the models.

4.1 The Basic Model

Let's build the basic model (gear reduction 100:1) first in accordance with the construction instructions. Because this model serves as a basis for many experiments, proceed very carefully when assembling it. After all mechanical components have been assembled, check the soft running of the motors. Each motor is connected directly with the accumulator without the interface for this.

Task 1:

Program the interface, so that the 40 Pulse model moves straight ahead. Use the counter button E1 to measure the pulses; use the E8 button as reset switch. Then modify the program, so that the model moves differently long distances, e.g., 80 cm. How high is the repetition accuracy?



Task 2:

The model should turn 180° after 80 pulses of moving straight ahead. Note the different rotational directions of the drive motors during the straight-ahead movement and the turn.

Solution:

The model moves an average of 1 cm on its path per counted pulse. The repetition accuracy is in the same range, approximately 1 cm with 80 pulses. It fluctuates dependent on the surface on which the robot moves. Thick or fluffy floor coverings are especially unfavorable for accurate measurements.

Before we deal with this task, we want to clarify two things. First, we are using a new function block in our program, the POSITION block. This is a block, which remains active until the set number of pulses is detected at a fixed input (E1 here). From the viewpoint of the program, this means that we are using a defined delay condition here. In the first experiment, we used this function as distance measurement for moving straight ahead.
If the robot should turn, practically the same procedure is used; we only need to change the rotational direction of the motors. Now you only need to enter the number of pulses, and the robot rotates on the spot.
And now to the second point. You don't want to simply try out values until the robot turns 180°, but instead you want to calculate this value in advance.

The drive motors are configured as differential drive, i.e., the wheels of the robot move around the circumference of a circle, the diameter of which is determined by the distance between the wheels. Consequently, each wheel must travel the distance of exactly half of this circumference for a turn of 180°.

Calculate the circumference u first:

$$u = \pi \cdot d = 630mm$$

d: diameter (wheel distance approx. 200 mm)

We previously determined a distance of approx. 1 cm/pulse. As a result, we need 305 pulses for the 314 mm distance (half of the circumference). Because we can only calculate whole number values, we have to select either 30 or 31 pulses. Test which value provides greater accuracy.

Conclusion:

You see that the result of our measurements with the pulse wheel does not provide a very high degree of accuracy. The absolute measurement error becomes greater especially when several distances are traveled one after another or repeatedly. The error caused by the clock pulse, which was not entered precisely, also results in problems.
We only have limited possibilities for minimizing these errors. On one hand, you can increase the path pulses per unit of distance. The counter would ideally be mounted directly on the motor shaft. In addition to the fact that we cannot access this shaft, there is also the problem of the limited sampling rate of the interface. If too many pulses are received within a time unit, the interface might "forget" a few. Then precise path calculation becomes an illusion.
We cannot register other errors at all, such as the slippage of the wheels on different surfaces or deviating wheel diameter. We can console ourselves with the thought that these problems have in part not been solved satisfactorily by substantially more complex and expensive commercial systems either.

4.2 Robot with Edge Detection

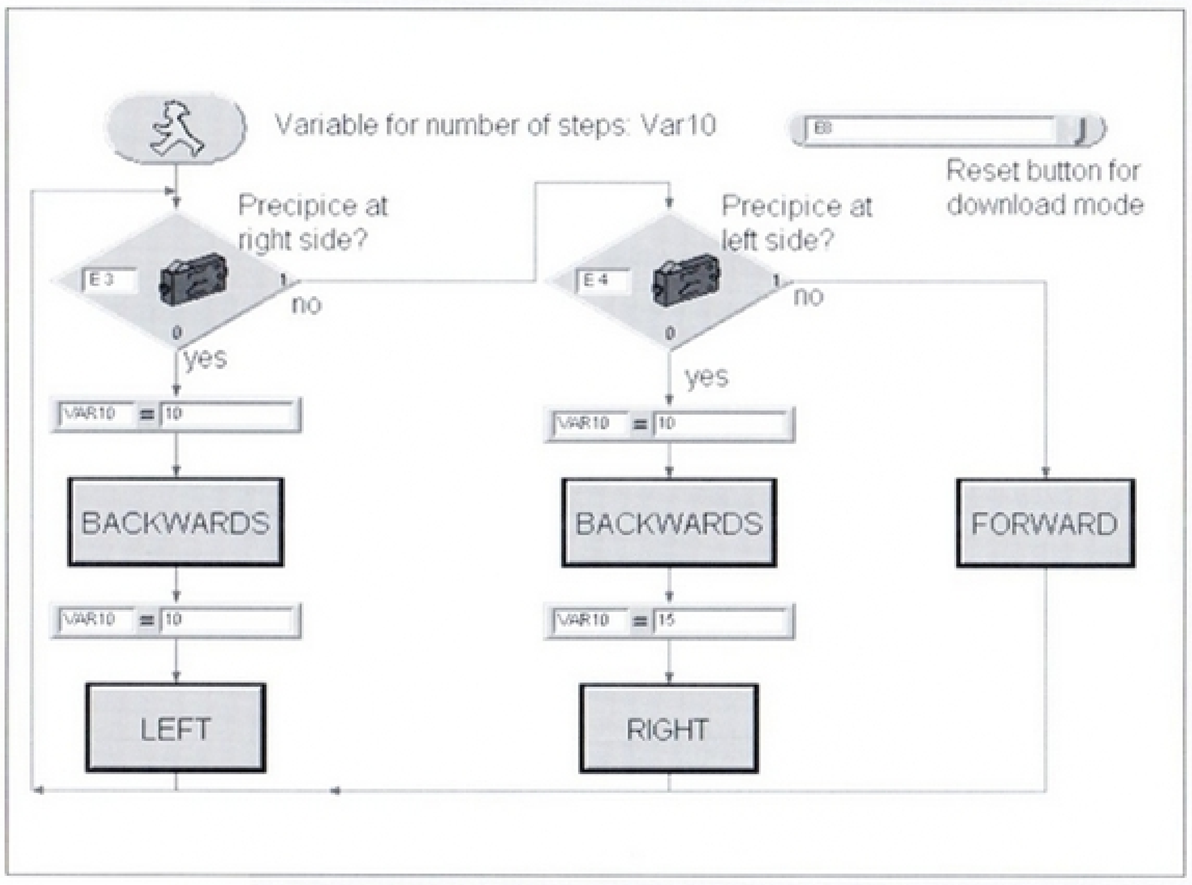
Now that we have experimented extensively with our basic model, we want to now try to teach our robot "fear" of sheer drops. We previously watched the robot like a hawk as it scurried around a tabletop, so that it did not plunge off the edge. This really does not seem to represent especially intelligent behavior. Consequently, we want to change it.
The robot needs an edge detector for this. Two auxiliary wheels provide a simple and useful procedure. The wheels are equipped with a switch, similar to a sensor, in front of the robot's direction of movement. The wheels are designed, so that they can move vertically. An edge lets the auxiliary wheel fall downward and consequently triggers the sensor.

Task 3:

Build the "Robot with Edge Detection" model in accordance with the construction instructions (gear reduction 50:1). The model should move straight ahead. As soon as it reaches a drop on the left, it should avoid this by moving to the right; if there is a drop on the right, it should move to the left. To make this clearer, specific movements are programmed as subprograms (forward, left and right).
The number of steps is recorded by the counter button. The number of steps is set in a variable VAR10. This value differs for the left and right subprograms.

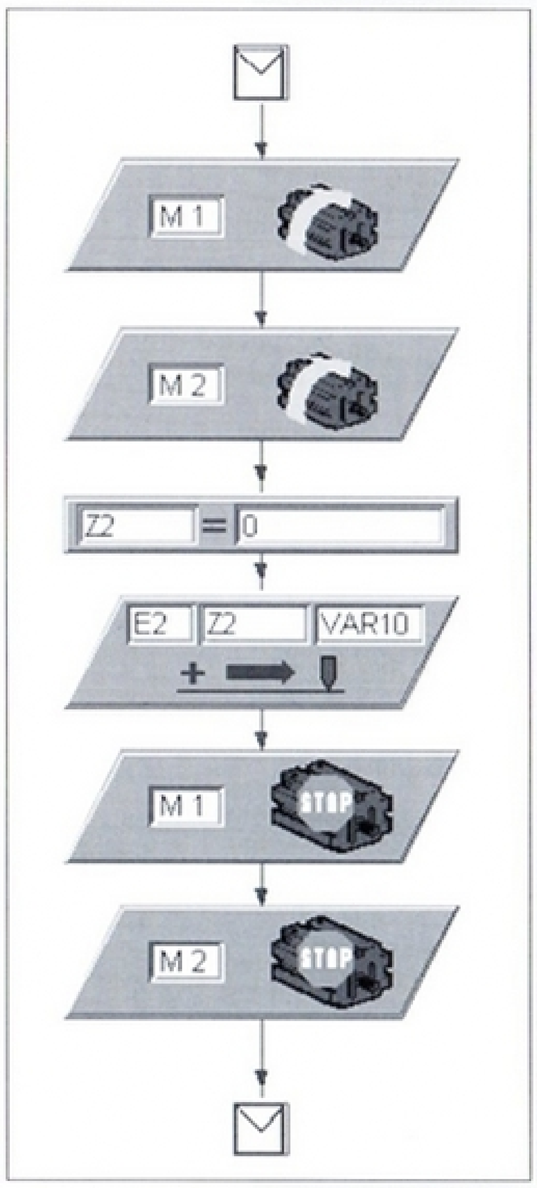
GB+USA

The different values reduce the risk of getting stuck in a corner.



Solution:

We can deduce from the task that we have to control the robot dependent on the edge detectors. Consequently, we should first split up the task into smaller parts. We first query the sensors (edge detectors). If no sensor is active, the robot moves forward. This is the "Forward" block in the diagram. A subprogram is behind such a block, which is new to us. Subprograms improve the clarity of complex systems, and they can also take better advantage of your computer's performance with multiple use. Our first subprogram is very simple; it only triggers movement of the motors M1 and M2.



But we need additional subprograms. The robot should move left, right or backward to avoid falling depending on the situation. In this case, it no longer suffices to simply switch on the motors. A specific sequence of movements must be programmed. Let's take a look at another subprogram. This should cause the robot to move backward when it detects an edge.

We want to switch the motors to backward movement for this. Then our pulse wheel should detect a defined distance. Set the distance with the variable VAR10. The assignment block Z2 = 0 is new. After you think about it for a while, you will understand the sense of this. If you do not set the count variable to zero, the mechanism only works once, because when Z2 reaches the value of the variable VAR10, our distance counter would fail us in each subsequent run. From the viewpoint of professional programmers, we are dealing with local and global variables here. The local variable Z2 is initialized before each use in the subprogram.

Conclusion:

Subprogram calls increase the clarity of programs. We use variables to measure various values, in this case path distances. We need the different path distances, so that our robot can "free" itself from corners. If the paths were absolutely identical, a robot could always move back and forth in a corner.

When you use variables, pay attention to their validity range. We initialize local variables, i.e., variables that are only used within a subprogram, before their use.

We have also seen that our robot needs a certain amount of movement leeway for it to function properly. If it runs into an edge during a movement to avoid another edge, the robot cannot react to it. Those of you who really enjoy solving such puzzles can try to find a solution for this.

4.3 Robot with Obstacle Detection

Our robots can now detect edges very well. But there is still a problem with ordinary obstacles. We have to modify the principle of edge detection. A corresponding crash contact replaces the auxiliary wheels. At this time, you should also think about one shortcoming of the edge detector; there are dangerous situations when the robot can "blindly" plunge into the abyss during backward movement without a sensor in the back.

A third sensor can compensate for this shortcoming. In the meantime, we have become experienced programmers. Fortunately, because the task is more complicated this time, as a look at the program shows.

The program contains additional, new function blocks. We add a WAIT block at specific program positions. This is easy to understand: the program waits until the time entered here has finished before the next program function is processed.

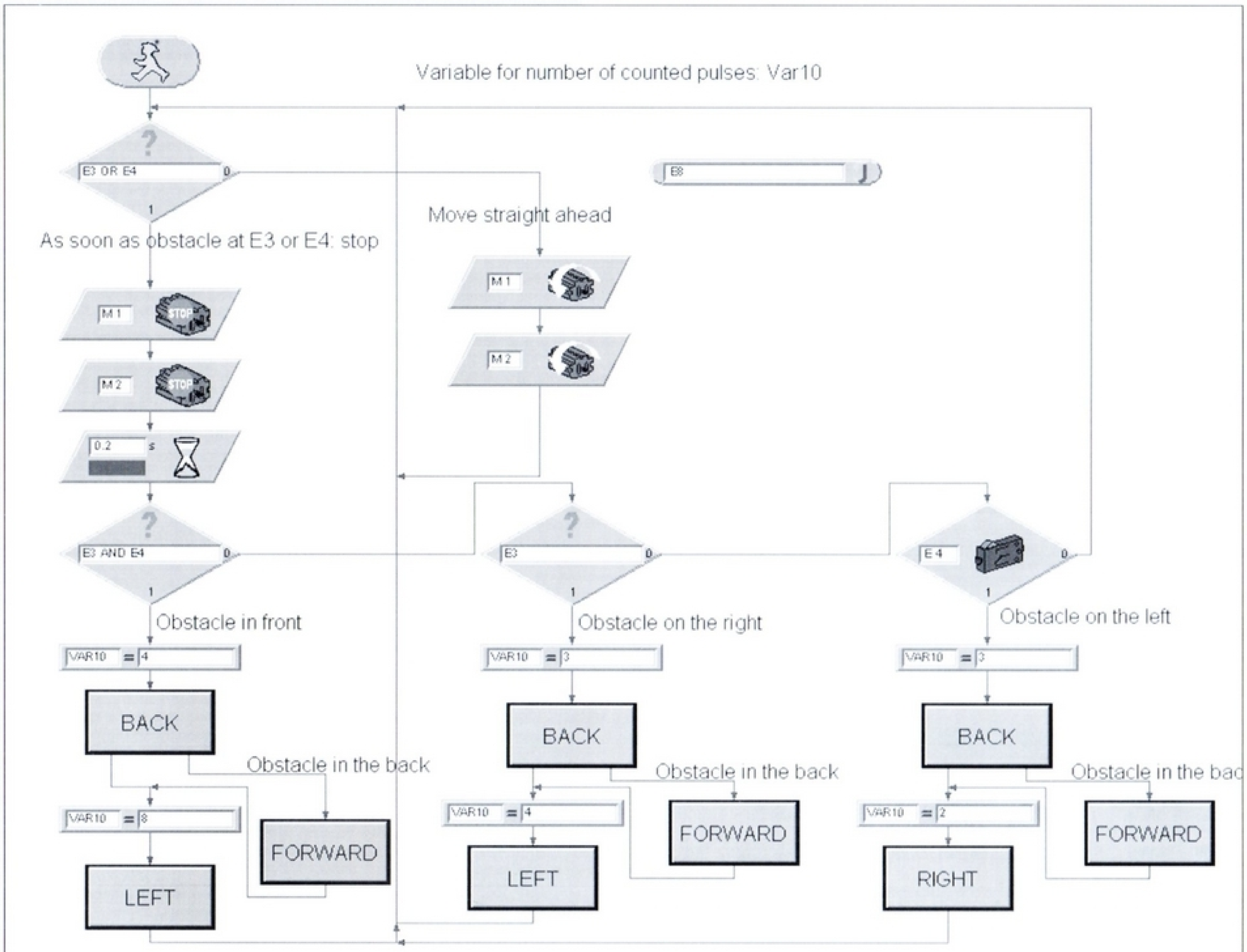
Logical comparisons are really new. Up to now, we have always made a comparison with the sensor queries and put a corresponding branch in our program. A similar comparison exists for pulse counting: comparison with a specific number of pulses. The logical comparison with several expressions is also new in a COMPARE block

Task 4:

Assembly the robot as described in the construction instructions with the faster gear reduction 50:1. The model should move straight ahead. As soon as an obstacle is detected at one of the front sensors (E3 or E4), the robot stops. If an obstacle is detected on the right, the robot moves back and then to the left to avoid it (approx. 30°). If there is an obstacle on the left, the robot moves back and then to the right (approx. 45°). The unequal number of degrees is necessary, so that the robot can find its way out of a corner.

If an obstacle is detected directly in front, the robot should move back and turn 90°. If the robot detects an obstacle when moving back, it should move a short distance forward and then avoid the obstacle as in the other cases.

Solution:



The main program tries to control the robot to move toward the light source. Whenever the direction of the robot deviates greatly from the ideal line, one of the light sensors is no longer receives light from the light source. Then the corresponding motor is stopped for a brief time, so that the two sensors can again detect the light source. This gives us our task.

Conclusion:

The complexity of the programs is increasing. We'll try to deal with this increased difficulty with improved programming techniques. Subprograms are a good means for doing this. We are now familiar with logical comparisons with subsequent program branching as a new control structure. We also now know that an increasing number of sensors are required for new properties or improved functioning of experiments, which we have already conducted.

Task 5:

Construct the Light Finder model with the slower gear reduction 100:1. First program the "Find light" function (preferably as a subprogram from the start). The robot rotates in this by at least 360° in one direction. Then it rotates by at least 360° in the other direction. If a light is found during the search, the robot stops. If no light is found after the rotations, the search is aborted and the program ends.

If the search for light is successful, the model should move toward the light source. If the light source moves to the left or right, the robot follows the movement of the light. If it loses contact to the light source, the program starts to search for light again.
Try to attract the robot with a flashlight and see if you can guide it through an obstacle course.

4.4 The Light Finder

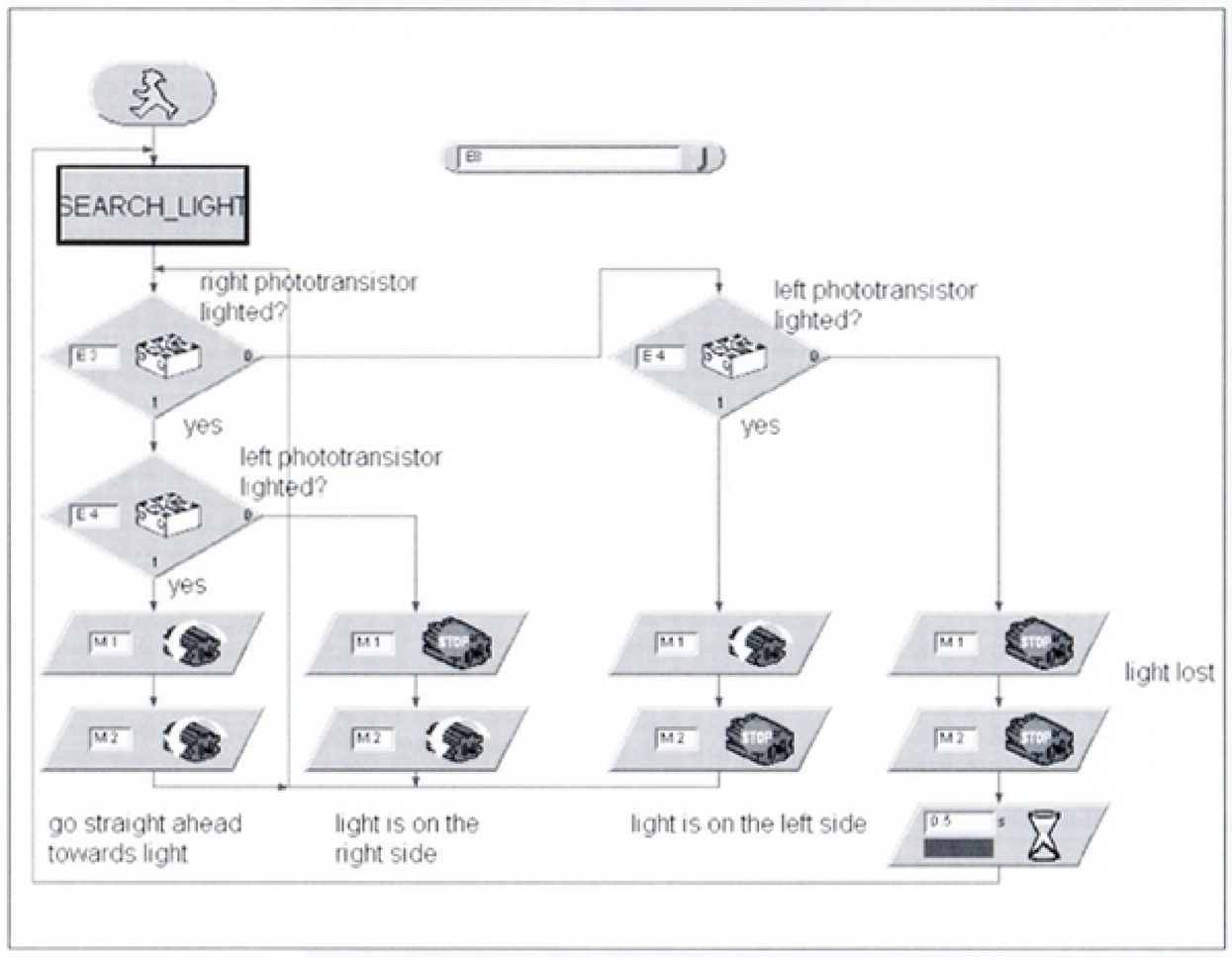
Till now we have let the robot react rather passively to environmental signals. Now we want to send the robot out to find things actively. The construction kit contains two phototransistors, which we can use as light detectors. Each sensor acts on one motor in order for "light ray tracking" to be possible.
The program is composed of two parts. One part contains the search for a light source, and the other part handles the tracking or controls movement in the direction of the light source.
Of course, we again take advantage of the possibilities of the subprogram technique. The LIGHT SEARCH subprogram is activated after the program is started. This subprogram is only exited after a light source has been located.

GB+USA

Tip:

Use a flashlight as a light source. Try not to focus the light beam too small, so that both photosensors can receive the light source. Note that other light sources in bright rooms, e.g., sunlight from a window, can be stronger than the light from your flashlight. The robot might move past your flashlight and toward brighter light.

Solution:



Conclusion:

We have now constructed a robot, which detects its environment actively. Its sensors are programmed to locate a light source and – if it succeeds – to move toward or follow this source.

We saw that the program switched off the robot if no light source was found. For example, if only a small obstacle interrupts the direct line of view to the robot, it does not detect the light source and cannot find it, although it is there. It would apparently make sense to have the robot move more or less randomly to another spot after an unsuccessful light search, so that it could search for light again there.

But what happens if an obstacle, which prevents light from reaching the robot, is precisely in the robot's movement of direction? This question seems sufficiently interesting for us to examine it more closely.

4.5 The Tracker

Searching and following are two essential characteristics, which intelligent beings have. We built and programmed a robot, which reacted to direct signals from its target or potential victim.

We use a different search principle with the tracker. Instead of targeted, precise movement to a light source, we mark a line that the robot should follow. This task can be solved relatively easily using optic sensors.

We measure the reflected light of the marking and then adjust the motors. You should also light the line with your flashlight, so that this functions precisely. Make certain that the photosensors are not dazzled by stray light due to unfavorable arrangement of lamps and photosensors. Concentration of light by the optical lens of our incandescent bulb works very favorably in this context.

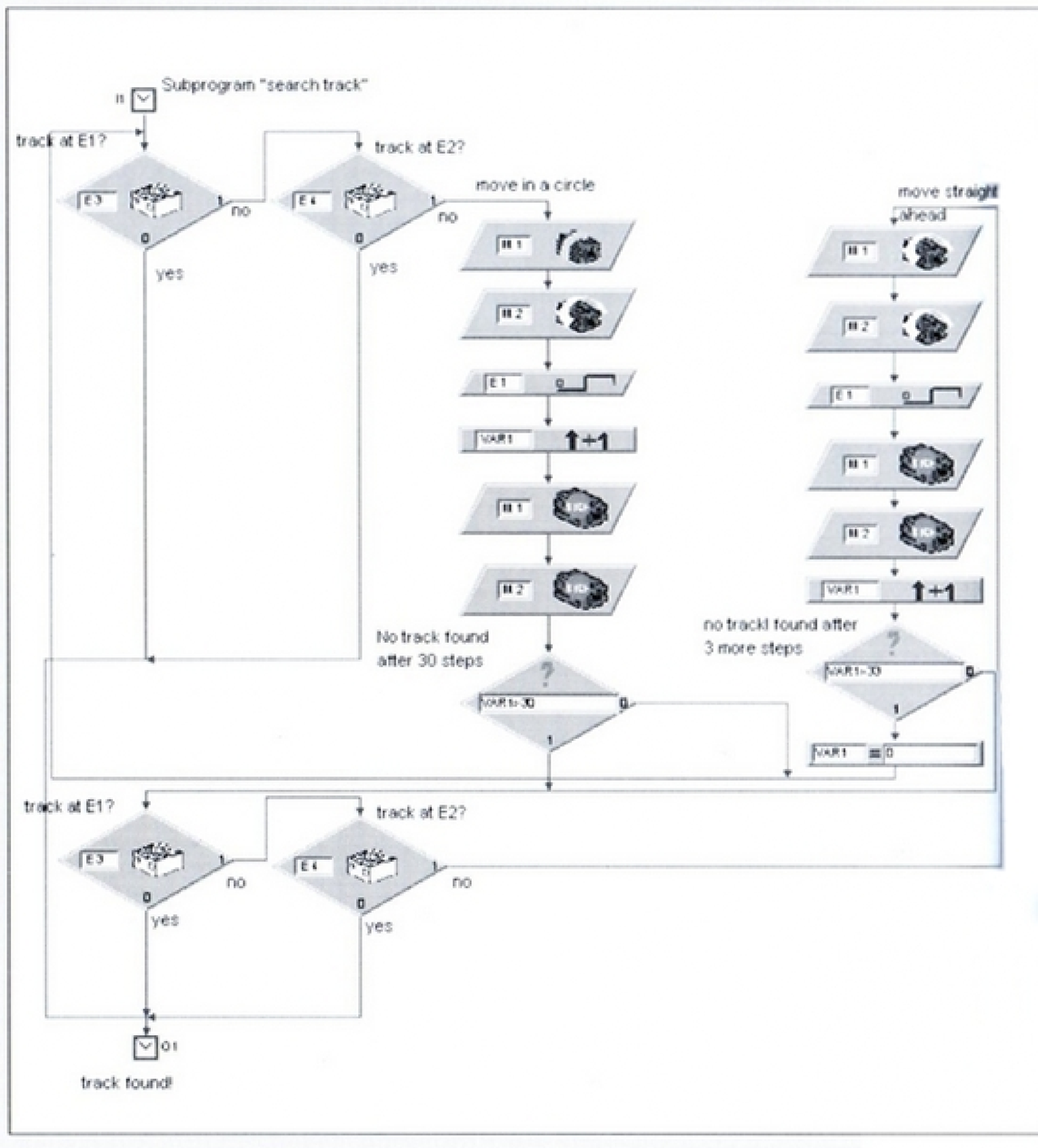
Task 6:

Build the Tracker model (gear reduction 100:1). First write a subprogram, with which the track is sought. The robot rotates 360° in a circle for this. If no track is found, the robot moves a bit straight ahead and searches again. The photosensors are queried about track detection. If the robot finds the track, it follows it. If the track ends or the robot loses it, e.g., if the track changes direction too suddenly, the search begins anew.

Tip:

After the lamp is switched on, there must be a short delay time (approx. one second) before the phototransistors are queried. Otherwise, the phototransistor detects "dark," i.e., a track where there is none. Use an approx. 20 mm wide strip of black insulating tape for the track or draw a black line in this width on a sheet of white paper using a felt pen.

Solution:

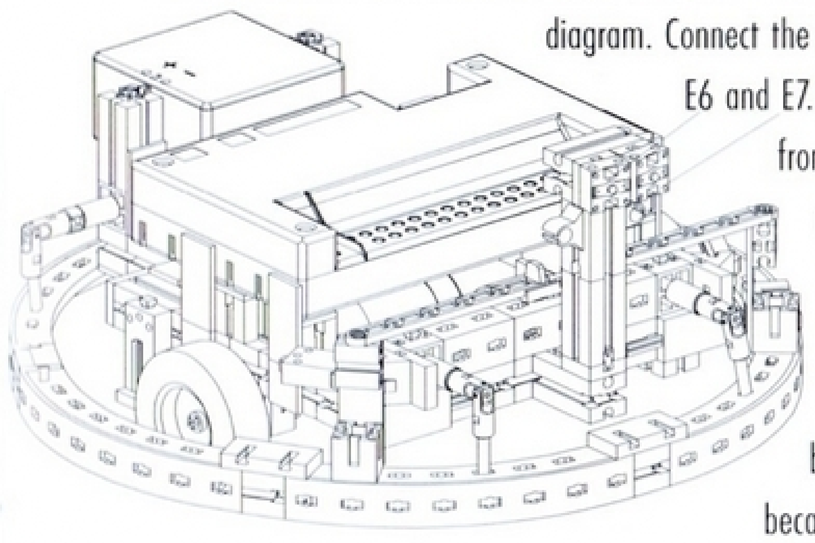


Conclusion:

If the robot finds the track, it follows it tirelessly. If we create the track as a closed loop, the robot goes round it. But you must avoid too sharp curves, where the photosensors could lose contact to the line. Although the robot tries to locate the track again, we have to admit that this is only possible where there are no obstacles.

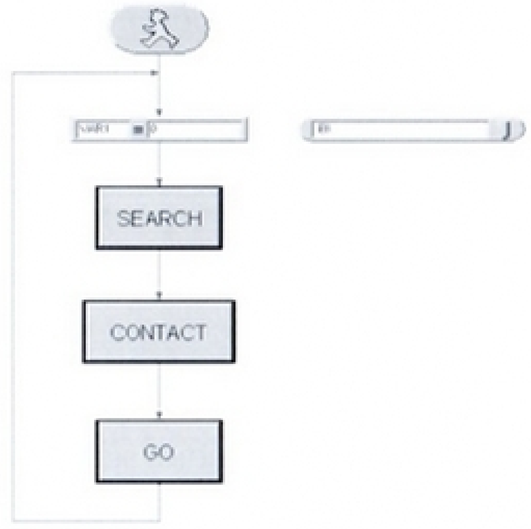
4.6 The Electronic Moth

The problem of linking different modes of behavior such as searching, following and avoiding has already arisen several times. Consequently, we want to investigate the interaction of such behavior in another experiment.
Let's summarize our previous results. The simple light finder follows a flashlight held in front of it more or less blindly. The robot assumes that no obstacle can exist where the flashlight previously was. It also assumes that it does not really reach the light. But these assumptions only correspond to reality in exceptional cases. We are forced to guide the robot with the flashlight around each obstacle. We actually would need a robot, which can avoid obstacles on its own. To do this, we can supplement the "Robot with Obstacle Detection" model with the property of light search, as shown in the diagram. Connect the phototransistors to E6 and E7. We use everything else from the obstacle detection model. From a scientific point of view, we have equipped the robot with two behavior modes. But because both behavior patterns cannot be active simultaneously, they are assigned different priorities. In this case, the robot is normally searching for light. If an obstacle is detected, which is more or less a danger for the robot, the behavior mode of obstacle avoidance becomes active. Once there is no longer an obstacle, the robot can again search for the light source.



As a result, we have provided the robot with the essential capability to navigate independently and avoid dangers. If you have a friend, who also owns a fischertechnik construction kit, you can carry the experiment even further. A light source is mounted an each of the two robots, and then the robots search for each other.
We have worked according to the principle of "trial and error" until now. You probably "simply" started with your programs and have discovered during the course of experiments how the robot did what it was supposed to (or how it didn't do that).
Of course, professional software developers cannot use such design methods for their products. In such cases, a precise design strategy must be defined before the first program line is written. That is not something that you can simply do, but instead you use fixed guidelines. Procedures have become standard, which use such strange names as "top-down design." An attempt is made to define the overall system from the top down, without dealing with all details from the start. Let's try to program our moth according to the top-down design. To do this, we have to begin with the main program, called the "main loop."

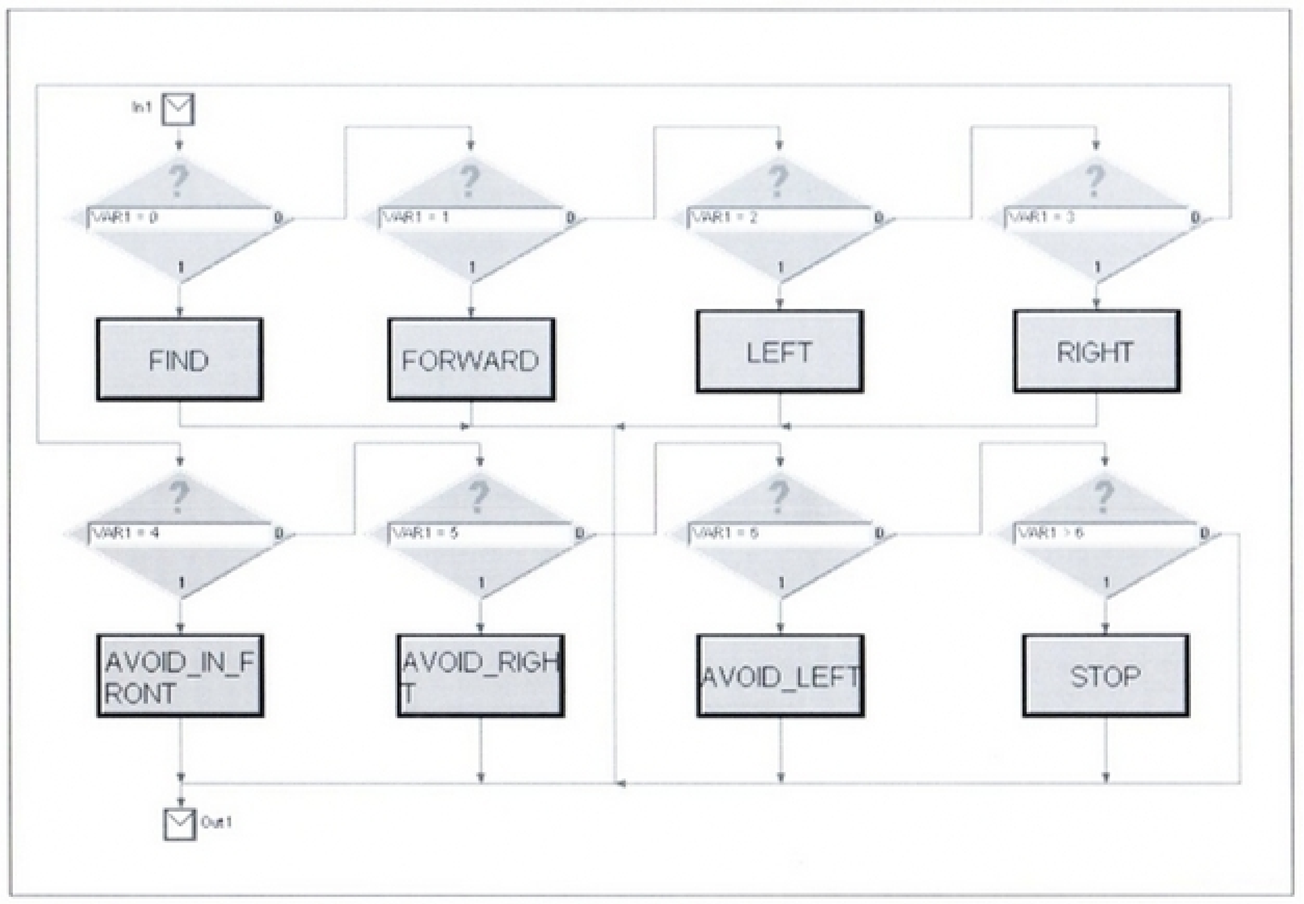
The diagram shows the simple design. The main loop is composed simply of the behavior modes necessary for the robot: SEARCH, CONTACT and GO. The variable VAR1 seems strange to us, and how can be give priority to the behavior modes of the robot with such a simple loop?
Well, priority is assigned by the sequence of the subprograms, and the variable VAR1 serves for assigning the movement orders. If we think this over very carefully, we can reduce the necessary drive movements of the robot to a clearly defined number of maneuvers. The robot needs a search movement, an avoidance movement and a correction movement. Avoidance and correction movements must be distinguished according to directions: right or left. This makes it clear that the SEARCH subprogram can calculate one of n (n = total number of movements) possible movements. The same applies to the CONTACT subprogram. Because CONTACT is executed after SEARCH, it overwrites the instructions of SEARCH. The GO subprogram only executes what it is commanded to do. The top-down design procedure really seems to be useful. But we don't know what the subprograms look like. Let's start with the first subprogram, SEARCH. This subprogram is responsible for querying the photosensors. Depending on the two sensors, four possible variants exist: left sensor, right sensor, left and right sensors, and no sensor. These define exactly four possible drive maneuvers: FIND, FORWARD, LEFT and RIGHT: These drive maneuvers are also assigned to subprograms; we are proceeding strictly according to top-down design criteria.



The design of the first subprogram, SEARCH, is now clear. SEARCH provides a parameter, which is converted into GO.

The actual subprogram is very simple. The output parameter is set via several comparisons. A check is conducted in the CONTACT subprogram in the finished program to determine whether obstacles triggered the sensor on the robot. For simplicity's sake, we will ignore this program now and take a look at how the motor movements are activated in GO.

Surprisingly, this subprogram calls additional subprograms. Doesn't this ever



GB+USA

end? We are still in the design phase (next level of top-down). GO sets when which drive maneuver is started.
"Real" program code is only hidden behind the various subprograms such as FIND and FORWARD. We have finally reached the bottom. Now the motors are switched on and off. Each of these subprograms has a clearly defined task, which we can program with more of an overview.

Conclusion:

Complex programs require completely new types of solution strategies. It makes sense to aim for an overall solution strategy. We wrote a program according to the strategy of top-down design. The required solution approaches are (first) formulated in formal structures, e.g., SEARCH or GO. These approaches are only refined in further steps and filled with the actual program code at the end. As a result, we separate program structure from the actual program code. We can consider both separately and analyze errors separately too, which is very important.

4.7 FTS – Driverless Transport System

Let's leave the area of scientific experiments now, which malicious gossip refers to as playing around, and deal with the area of practical applications. We now want to build and program a robot not just for its own sake, but instead it should handle a task. The tracker is equipped with a mobile transport fork for this. This creates a forklift robot, which can transport material stored on a pallet.

industrial application. And such transport systems are already in use in part today. In the meantime, you have had good training in programming, so that you certainly are willing to take on this complex programming task in good spirits.

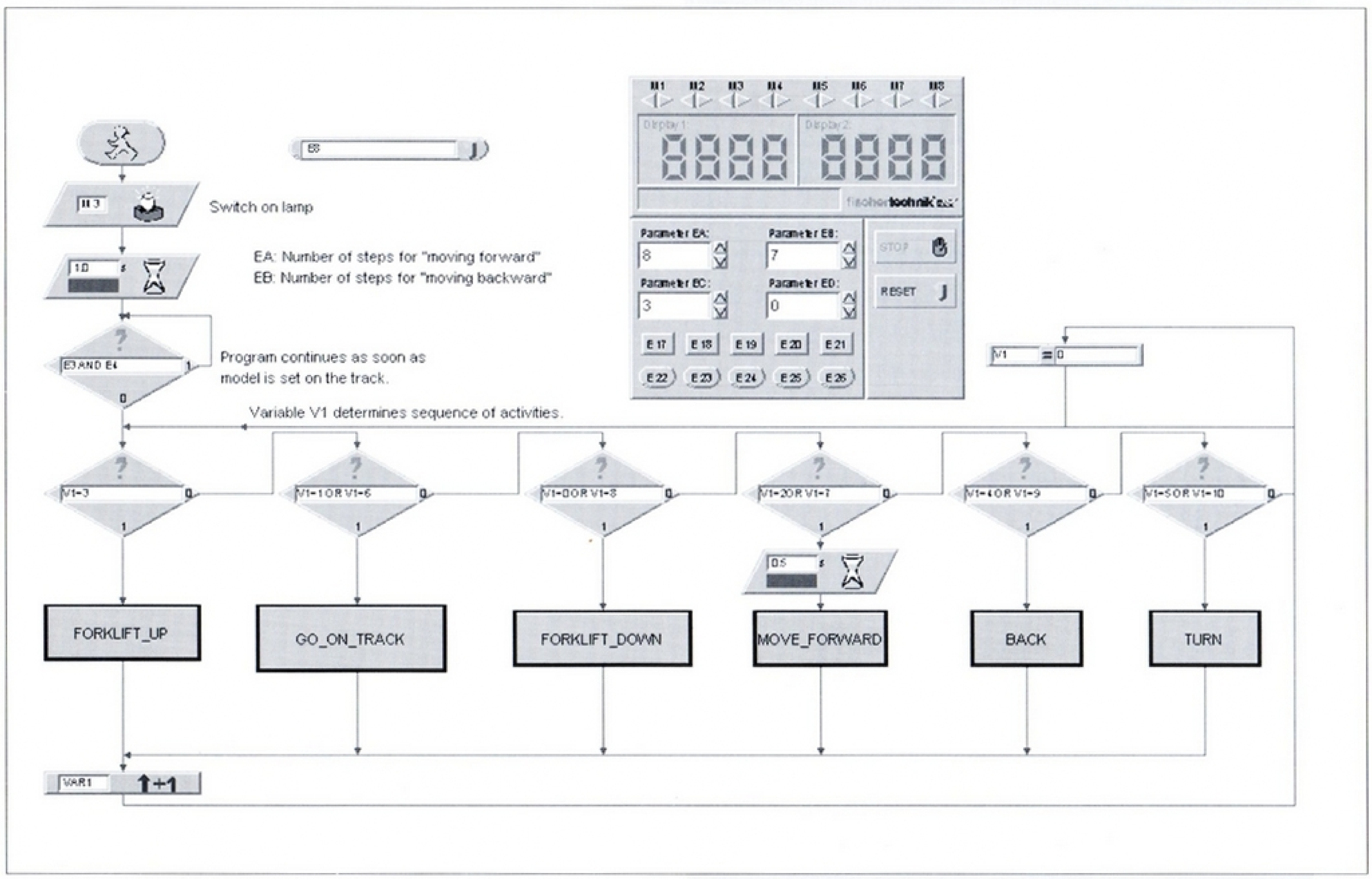
Task 7:

The driverless transport system should drive along a black track. At the end of the track, it picks up a pallet, turns 180° and drives back along the track. When the end of the track is detected, the robot puts the pallet down, turns again and gets the next pallet.

Task 8:

As an extension to Task 7, supplement the program so that the pallet is only put down for a short time at the end points of the track. In other words, the robot searches for the pallet, drives to the end of the track, puts the pallet down for a short time, picks it up again, drives to the other end, etc.

Solution:



As simple as this experiment seems, it contains all the components of an

Conclusion:

As soon as we take at look at the program, we immediately recognize the already known program strategy. A state variable controls the behavior of our driverless transporter.

Using the refined programming techniques you have acquired, you can implement multifaceted controls. The driverless transport system demonstrates the possibilities, which the combination of fischertechnik construction kit with the Intelligent Interface provides. Starting with simple mobile robots, you have reached a state, which hardly differs from industrial control engineering.

5 Troubleshooting

Experimenting is fun, but only as long as everything functions well. The best would be if everything succeeded at first try. Unfortunately, this is usually not the case.

You only know whether you have understood a mechanism precisely if a model does not work and you find the error immediately.

With mechanical errors, you can still see (assembled incorrectly) or feel (runs hard) something. When there are electric problems in addition, it becomes more difficult.

Professionals use a number of very different measurement instruments for troubleshooting (searching for errors), for example, voltmeters or oscillographs. Not everyone has such equipment on hand. Consequently, we want to try to pinpoint errors and correct them using simple means.

Before we start with the experiment, you have to put together a few components from the fischertechnik construction kit. These are essentially cable connections. The supplied plugs are connected to the individual cable sections here.

First, you cut the cable. Measure the given lengths and cut the sections. Before you cut, check very carefully to make sure the lengths are correct.

Then you need to test each cable. You need the accumulator and the lamp for this. If the lamp lights after it is connected with the accumulator, the cable is okay. If the color assignment is also correct – red plug with red cable and green plug with green cable – put the cable aside and check the next one.

If the program (including the one supplied) does not work with our model, start "Check Interface". This auxiliary program lets us test inputs and outputs separately. Each sensor must trigger the corresponding action on the interface here.

After you check that, you know that all the electric connections should be okay. Switch the actuators on and off individually via the motor control buttons. If everything works here too, look for the mechanical cause.

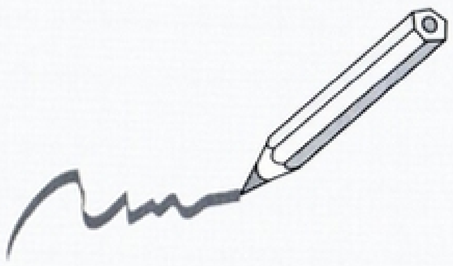
Loose connections represent bad errors. On one hand, connection plugs can be loose in the sockets. If this is the case, the contact springs of the plug are widened a bit using a small clockmaker screwdriver. Be careful, because widening them too much can result in breaking the contacts or make it very difficult to plug them in.

Another cause of loose connections is created by loose terminal connections on the screw-in spots of the plug. Screw them tight carefully! You can use this occasion to check whether any of the thin cooper wires have broken off.

If there is an unexplainable outage during operation, an almost empty accumulator can be the cause. The voltage decreases briefly when a load is switched (motor on), and then a reset is triggered for the computer on the interface. Such an error is difficult to find, because the program always functions correctly at first.

If errors occur in programs you have written yourself, which you cannot explain, install a supplied program, which is as similar as possible to yours, to rule out electric or mechanical defects.

If you are still not successful in finding the error, you can always contact the fischertechnik service department.



Mobile Robots II

- Begleitheft
- Activity booklet
- Manuel d'accompagnement
- Cuaderno adjunto
- Folheto

fischerwerke
Artur Fischer GmbH & Co. KG
Weinhalde 14–18
D-72178 Waldachtal
Telefon: 0 74 43/12-43 69
Fax: 0 74 43/12-45 91
email: info@fischertechnik.de
http://www.fischertechnik.de

78 897 • ZE • 10/02 • Printed in Germany • Technische Änderungen vorbehalten • Subject to technical modifications

fischertechnik