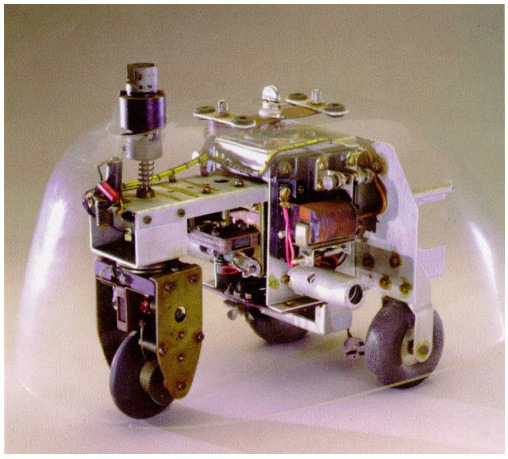
Contents

Why Do We Need Robots?

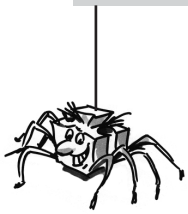
■ Carel Capek coined the term robot in his 1923 novel "Golem". This artificially created figure was designed to take over human labor with his abilities.
In the 30's and 40's of the last century the robot became more of an automaton. Today we can look back and smile at the different attempts to divest it with human characteristics such as a head with blinking lights as eyes, etc. These machines showed little sign of intelligence or even mobility. Since the principle of control has great influence on the robotics, the design of robots became more realistic with the advent of electronic circuits. Even today, the question of the "intelligence" of the robot is subject of much research and fact-finding in many companies, institutes and universities.

■ So called cybernetics offered the first promising approach to the problem. The term "cybernetics" is derived from the Greek word kybernetes. The kybernetes was the navigator on Greek ships. He had to determine the ship's position and chart the course to the destination.
Clearly cybernetics was supposed to make the robot "intelligent". But what would such intelligent behavior look like?
We shall try to illustrate this using a thought experiment. Everybody has probably observed a moth's behavior in the light of a lamp. The moth detects the source of the light, flies toward it, and then avoids hitting the lamp at the last moment. It is clear that in order to exhibit this behavior, the moth has to detect the source of light, plot a course there and then fly toward it. This ability is based on instinctive intelligent behavior patterns of the insect.
Now lets try to apply these abilities to a technical system. We have to detect the source of the light (optical sensors), execute a movement (operate motors) and we have to establish a meaningful connection between the detection and the movement (the program).



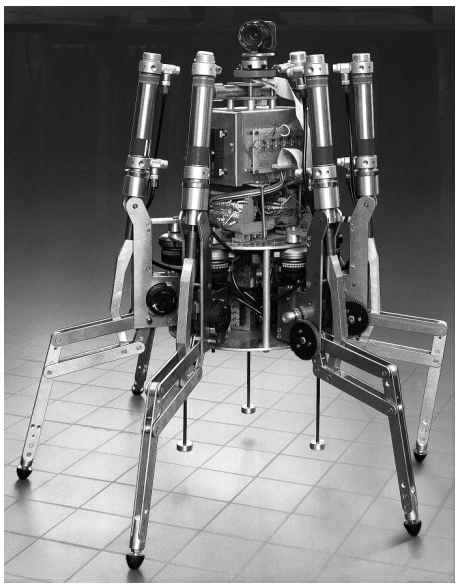
■ It was the Englishman Walter Grey who put the thought experiment described above into practice in the 1950's.
With the help of simple sensors, motors and electronic circuits he created a variety of "cybernetic" animals that displayed the very specific behavior of, lets say, a moth. The photograph shows a replica of the "cybernetic" turtle, exhibited at the Smithsonian Museum in Washington.

Based on these ideas we will create similar "patterns of behavior" for our robots and will try to communicate them to the robot in the form of programs.



■ But why do we need mobile robots? Lets try to apply the behavior of our "imaginary moth" to technical devices. A simple example for this is light-seeking behavior. We modify the light source by attaching a bright strip, a guideline, to the floor and align the sensors to face not forward but downward. With the help of such guidelines a mobile robot can find its way in a warehouse for example. Additional information at specific points along the line, in the form of a bar code for example, direct the robot to perform further actions at these locations such as picking up or dropping a pallet.
In fact such robot systems are already in existence today. In large hospitals it is often necessary to cover long distances for the transportation of consumable supplies such as bed linens. It is time-consuming and expensive to have the nursing staff transport these materials and often requires hard physical labor. Additionally performing such tasks reduces the time available to take care of patients.

■ For the last couple of years scientists have begun to deal with another from of movement that is very common in nature, walking or running. Robots have been developed that have the ability to move about on legs. The electropneumatic walking robot "Achille" that was developed by the royal military academy in Brussels is an example of a six-legged walking robot. Equipped with one camera above and one on each of the six legs, this robot is designed to be able to react mechanically to raised or sunk obstacles (objects or holes).
Such walking machines can be deployed everywhere where wheeled and track vehicles don't have much of a chance, such as rough or soft terrain, for climbing over obstacles, ascending stairs, surmounting ditches or for operation in inaccessible or dangerous areas in nuclear power plants, mining tunnels or during rescue operations.



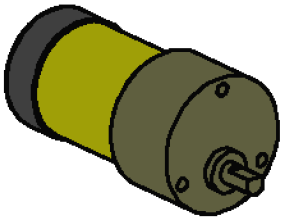
It is easy to recognize that mobile robots can play an important role in modern society.

Robots Built with fischertechnik

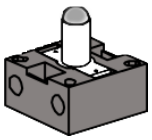
■ So how can we build a robot with our fischertechnik construction kit? To build a robot we need sensors (e.g. pushbutton sensors,) and actuators (e.g. motors) but also many mechanical parts to construct a model. The fischertechnik ROBO Mobile Set offers an ideal foundation for this. The following sensors and actuators are included in this construction set:

Actuators

Power Motor:



Two of these powerful DC motors (9VDC/2,4W) with built-in gearbox and a reduction of 50:1 drive the mobile robots (this means that while the motor performs 50 revolutions the shaft extending from the motor turns only once during the same time.

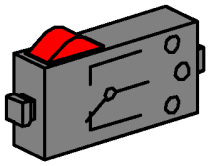


Lens Tip Lamp:

This incandescent bulb (9VDC/150mA) enables the output of simple light signals.
A lens is integrated into the glass bulb of the lamp focusing the emitted light. By directing a beam of light toward a light sensor (phototransistor, see below) you can build a light barrier able to differentiate between light and dark. The lamp can also be used to display certain states or to generate warning messages in the form of a blinking lamp. In this construction kit the lamp is used together with 2 phototransistors as special sensor for line recognition.

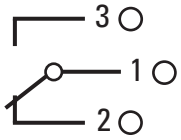
Sensors

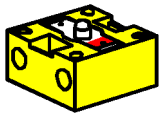
■ The pushbutton sensor is an example of a digital sensor. Digital values can only assume 2 different states. These states are identified with 0 or 1 respectively. For the pushbutton sensor "0" signifies that no current flows between the contacts, "1" signifies that current is flowing.



The fischertechnik **pushbutton** sensor is designed as a three-way switch. For this reason there are three terminals. When the red button is pressed the switch connecting the terminals 1 and 3 is activated mechanically. At the same time the connection between the terminals 1 and 2, which was connected during the quiescent state, is interrupted. This way both possible starting positions can be selected:

Closed in quiescent state (terminal 1 and 2 connected)
Open in quiescent state (terminal 1 and 3 connected).





The **phototransistor** may be used both as digital as well analog sensor. In the first case it serves to recognize clear transitions between light and dark, a marked line for example. But it is also possible to distinguish the amount of light according to its brightness. In this case the phototransistor works as analog sensor. Analog values can vary freely between their extreme values. These values have to be converted into their respective numerical values in order to be processed by the computer.

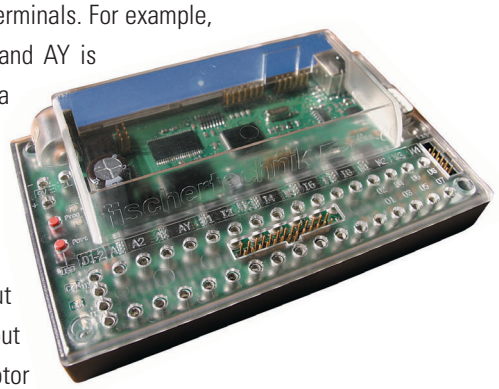
Incidentally phototransistors belong to the so-called semiconductor devices, its electrical characteristics are dependent on the intensity of the light. Everybody knows about solar cells that use sunlight to

generate electricity. A phototransistor can be understood as a combination of a mini solar cell and a transistor. Light impulses (photons) received by the phototransistor generate a very low current that is then amplified by the transistor.

Note:
Please make sure the polarity is correct before connecting the phototransistor: Red marking = plus pole.
Maximum voltage: 30V max

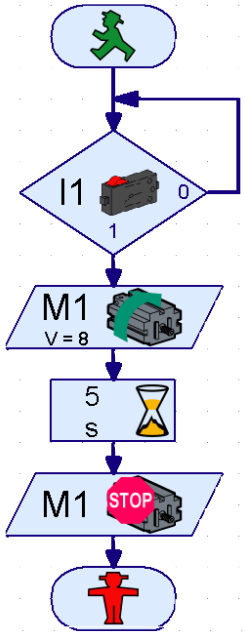
ROBO Interface

■ We can connect different sensors and actuators to the ROBO Interface and interpret them. In addition to 8 digital input terminals the ROBO Interface also offers several analog input terminals. For example, a resistance value between 0 and 5,5 kΩ applied to the input terminals AX and AY is converted into a numerical value between 0 and 1024. The measured values of a brightness sensor, such as a phototransistor, can so be acquired and are available for further processing. Voltages between 0 and 10VDC can be measured at the analog input terminals A1 and A2.



The most important role of the Interface lies in the logical connection of the input values. The Interface needs a program to do this. The program decides in how input data and sensor signals are processed to generate appropriate output data, motor control signals, etc. With the ROBO Interface we have enough computing power at our disposal to design even the most sophisticated programs.

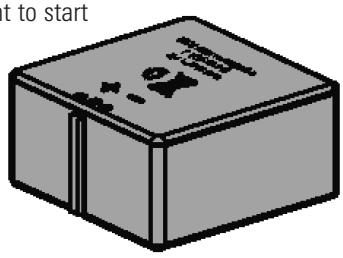
Software ROBO Pro



■ A graphical programming interface provides us with the most effective way to create the necessary programs for the ROBO Interface. The term "programming interface" stands for a software that enables us to create our programs in a very comfortable way, using graphical symbols. Actually, the computer of the ROBO Interface can only execute commands contained in its so-called machine instruction set. These are essentially simple control structures that are extremely difficult to use for beginners. That's why the PC software ROBO Pro uses graphical elements that are later translated into a language that can be executed by the Interface.

Power Supply

■ The only thing you need in addition to the ROBO Mobile Set is the Accu Set. It contains the battery pack serving as mobile power supply for our robot models and a special charger for the battery pack.
It would be best to start charging the battery pack right away using the charger. This way it will be fully charged when we want to start experimenting.



Approach to Experimentation

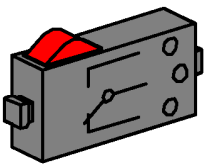
■ We will go step by step in our exploration of the fascinating world of mobile robots. We will start with a simple test setup to check the basic functions of Interface and sensors. Then we will build simple models, which will be assigned specific functions and later attempt more and more complicated systems.
Should you feel at some point that creating your own programs is too complicated or takes too much time you can load the supplied sample programs into the Interface and use them to operate the robots.
At the end of this resource guide is a chapter on troubleshooting so you don't despair should errors occur.

It is very important to take proper care during the construction and initial operation of our robots. When connecting electrical components we'll stick closely to the specifications, double and triple checking to make sure everything is ok. When it comes to the mechanical construction, also for your own creations, we will pay close attention to the smoothness of operation and low play in the gears and fastenings. It is up to you and your creativity to write your own programs defining new "behavior". You are only limited by the amount of memory and computing power of your hardware. The following examples can give you some ideas.

First Steps

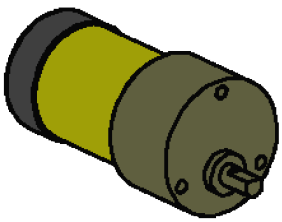
■ Now that we have covered the theoretical considerations we want to start conducting our own experiments. Some of you might want to start right away, maybe even with the big walking robot. This is, of course, possible and if you follow the construction manual closely you will succeed in building the model on the first try.
But what do you do if it isn't working? In this case the cause of the fault must be tracked down systematically. But before dealing with this let's check the interaction between computer and Interface.

Chapter 1 and 2 of the ROBO Pro software manual describes how to install the control software on your PC and how to connect the Interface. With the help of Interface tests we will test the different sensors and actuators.



Pushbutton Sensor

Pushbutton Sensor
We can for example connect a pushbutton sensor to the digital input terminal I1 and observe how the state of the input changes if the pushbutton is pressed.

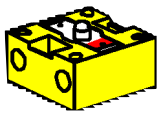


Power Motor

Power Motor
We will test the output terminals by connecting a motor to a motor output terminal, e.g. M1. Using the left mouse button we can start the rotation of the motor and with the slider we can change the speed.

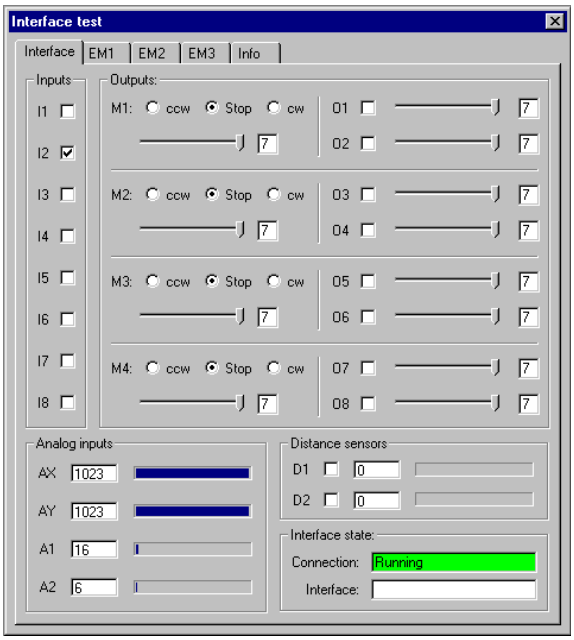
Phototransistor
If we also want to test the analog input terminal AX a phototransistor can be used as analog sensor.

While polarity plays no role in connecting a motor or pushbutton sensor (in the worst case the motor will rotate in the wrong direction) it is vital for the function of the phototransistor to connect it correctly. The contact of the transistor with the red marking should be connected to the red connector and the other contact with the green connector. The second green connector belongs into the socket of the input terminal AX that is located closer to the edge of the Interface. The second red connector fits into the socket of AX that is located further on the inside. (Attention: When connecting the phototransistor to a digital input terminal I1-I8 the red connector needs to go into the socket located closer to the edge of the case.

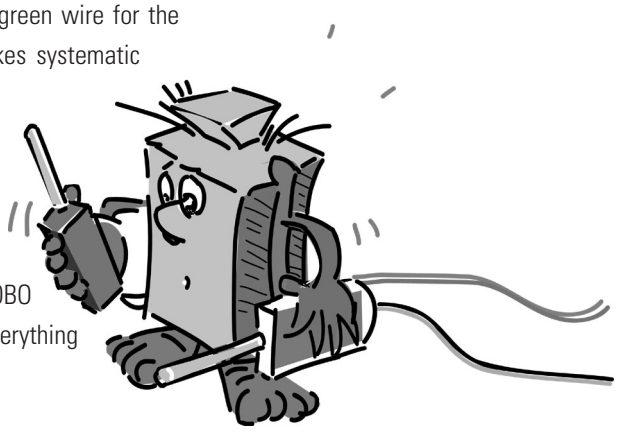


Phototransistor

Now we can vary the intensity of the light of the phototransistor using a flashlight. This will change the reading of the blue bar of AX. If the indicator does not move from its maximum position we should take another look at the connections of the phototransistor. If however the indicator remains at zero even with the flashlight turned off it is possible that the lighting in the room, the ambient light, is too bright. The position of the bar will change when we cover the phototransistor.

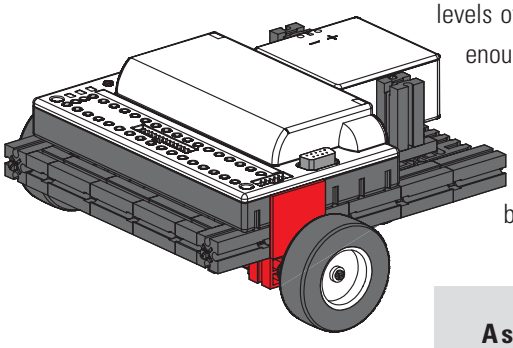


To come back to the color-coding of the connectors again: During assembly we will always be sure to connect the red connector to the red wire and the green connector to the green wire. If the right polarity is important for the circuitry layout we will always choose a red wire for the positive pole and a green wire for the negative pole. While this might seem overly meticulous a clear color-coding makes systematic troubleshooting that much easier.



A simple program will round out our first steps in the area of robotics. The program "Garage Door Control System", explained in chapter 3 of the ROBO Pro manual might have nothing to do with mobile robots but it is an excellent way to get to know the ROBO Pro software. Only the motor and three pushbutton sensors from the ROBO Mobile Set need to be connected to the Interface in order to recreate that program. Everything else is described in detail in the software manual.

The First Simple Robot



■ After Interface test and Garage Door Control System we finally want to put our first robot into operation. We will assemble the model "Simple Robot" with the two drive motors according to the construction manual. This will be pretty quick and easy since this model deliberately only holds what is absolutely necessary for a robot to drive. The motors we'll connect to the output terminals M1 and M2.

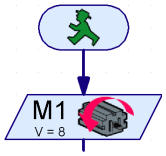
We will open the ROBO Pro software and set up a new program (FILE – NEW). ROBO Pro offers different levels of difficulty to work in. They can be set in the ROBO Pro menu item LEVEL. For now Level 1 is enough for us.

An empty work sheet will appear and on the left side the element window. There you can select the different program elements and place them on the work area using the left mouse button. The right mouse button will let you change the properties.

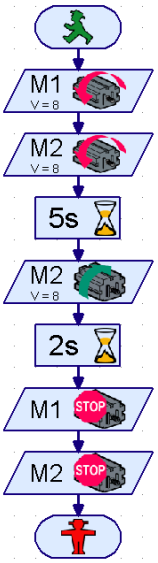
Assignment 1 (Level 1):
Our "simple robot" is should drive straight ahead for 5 seconds, then spin in circles for 2 seconds and after that come to a stop.

Tipps:

Together we will program the first robot step by step:



- We will start with the little green GO man. It represents the start of the program.
- Then we get the motor symbol from the element window and put it below the Start element causing an automatic connection line to be drawn. In the property window we'll set the motor output terminal to "M1" and the rotational direction to "ccw" and confirm with OK.



- By following the same steps to place another motor symbol below the first one we switch on motor 2.
- To wait a certain amount of time we use the Time Delay element placing it below the second motor symbol and set the time to 5 seconds.
- After that we set motor M2 to rotate in the other direction (cw) then wait 2 seconds and finally switch both motors off. Our program concludes with the End symbol, the little red STOP man. The illustration shows the finished program flow chart.

If you are not sure you did everything correctly, you can compare your program with the supplied sample program. To do this you first save your own program and then load the file Simple Robot 1.rpp from the samples directory of ROBO Pro (default setting C:\Program Files\ROBO Pro\Sample Programs\ROBO Mobile Set).

If everything is ok the program can be downloaded into the Interface. After clicking the Download button a pop-up window will appear. There we select the for program to be loaded into FLASH memory 1 and that it should be started as soon as it is downloaded.

Immediately after the download the model will start driving straight ahead, turn for a short time and then stop. If you would like to start the program again press the Prog button on the Interface for a short time. The LED Prog1 will start blinking again for as long as the program is running. After that it will remain on. By the way, the program will remain in the FLASH memory of the Interface even if the power

supply is interrupted. To test this we disconnect the battery pack. Then we reconnect the battery pack and select the saved program by pressing the Prog button until the LED Prog1 lights up. To start the program we simply press the button again.

Our robot can do much so far, right? Why don't we extend the assignment a bit?

Assignment 2 (Level 1):

In order to keep the robot from stopping after just 7 seconds, we will now teach it to dance.

- **Lets have it go straight, turn right, turn left, go backwards for different lengths of time and at various speeds.**
- **This should to continue until the program is stopped, by pressing the Prog button on the Interface.**

Tips:

- Simply keep reversing the polarity of the motors to make the robot go in the desired directions.
- The speed of the motors can be set between 1 and 8 in the property window of each motor symbol. If M1 and M2 rotate in the same direction at different speeds the robot will make a turn.
- Draw a connecting line from the exit of the last program element to the line leading into the first element to make the program repeat continually.
- You can find a finished example under Simple Robot 2.rpp.

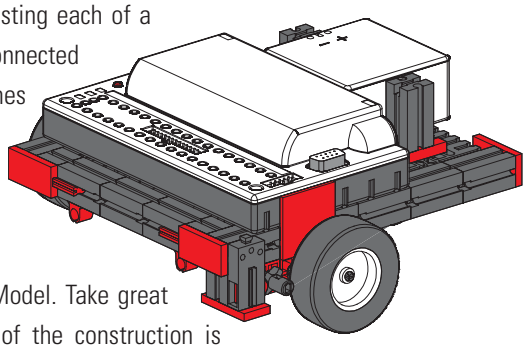
Congratulations, you have now built your first robot and programmed it yourself. It might not be especially intelligent since it is unable to recognize obstacles and would fall of the table if you don't watch out. But this will change during the course of our further experiments.

Intelligent Wheeled Robots

■ Robots require sensors in order to be aware of their environment. The following suggested models introduce a few different mobile robots enabling us to try out the operation of the different sensors. It is thereby imperative that internal states of the robot e.g. measurement of distance traveled using pulse wheels, as well as external signals e.g. light-seeking or tracking are linked. Different assignments have been created for each model. They are designed to give you ideas and to make you familiar with the subject matter. The programs for each assignment can be found in the ROBO Pro directory under \Sample Programs\ROBO Mobile Set\. But feel free to come up with your own assignments for each model. Once you have completed the following examples you're sure to come up with many more ideas.

Basic Model

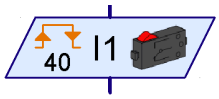
■ Compared to out first "Simple Robot" the basic model is more stabile and robust. In addition it contains 2 sensors to measure the distance traveled, consisting each of a pushbutton sensor and a pulse wheel. The pulse wheel is connected to the motor shaft and activates a pushbutton sensor four times with each rotation of the motor.
This model serves as a foundation for the other mobile robot models.



Follow the construction manual to put together the Basic Model. Take great care during the construction. When the mechanical part of the construction is complete test the smoothness of operation of each motor by connecting it directly to the battery pack without using the Interface.

Assignment 1 (Level 1):

- **Program the Interface so the model drives straight ahead for 40 pulses.**
- **Use the counting sensor at input terminal I1 to measure the pulses.**
- **Measure the distance covered by the model and calculate the distance traveled per pulse.**
- **Repeat this test 3 times and record the variations of the values in a table.**



Tips:

- First switch on both motors (rotational direction left).
- Use the program element **Pulse Counter** to count the pulses at I1.
- Count both pulse edges (0-1 when pressing, 1-0 when letting go of the pushbutton). You can set this under **pulse type** in the property window. This will make the measurement of the distance traveled more exact.
- Then switch off the motors and end the program.
- You will find the finished program under Basic Model1.rpp.

Result:

	Amount of Pulses	Covered Distance	Distance/Pulse
Test 1	**40**		
Test 2	**40**		
Test 3	**40**		

You can say that the model travels a distance of roughly one centimeter (or 0.394 in.) per pulse.

By now you will also know what rotational direction you have to set for each motor in order for the model to drive in a certain direction. Record what you have learned in the table below so you don't have to think about it each time you want to change the driving direction. If you connect the wires exactly as described in the construction manual, a rotational direction to the left causes the wheel to go forward for any motor. That's how the motors are programmed in all sample programs.

Complete the table:

Direction of Model	Rotational Direction M1	Rotational Direction M2
Straight	**ccw**	**ccw**
Backwards		
Left		
Right		
Stop		

Normally you would have to place two motor symbols on the screen for each change of direction. This can be avoided by creating a subprogram for each direction. This will simplify the programming enormously. Chapter 4 of the ROBO Pro software manual describes in detail how to create a subprogram. As soon as you have read this chapter you are ready to tackle the next assignment. You can now switch to **level 2** in ROBO Pro.

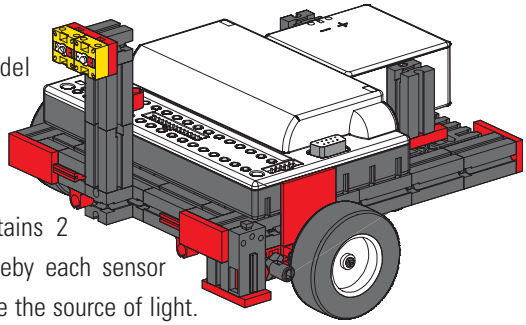
Assignment 2 (Level 2):

- **Create a subprogram for each direction.**
- **Program the model to drive along the path of a square with an edge length of one meter (or 3.28 ft.).**
- **How high is the repeat accuracy?**

Tips:

- First create a subprogram **"Straight"**. Then you can create the other subprograms by copying the first one. The only thing left to do is to adjust the rotational direction of the motors.
- Using a lower speed when turning left and right will increase the accuracy.
- Once again, use the **Pulse Counter** element and the pushbutton sensor at input terminal I1 to count the pulses.
- First load the program into the RAM until you have found out how many pulses are necessary to perform a 90° turn. For one thing loading into the RAM is much faster than loading into the FLASH memory and secondly the Flash memory has a "limited" life span of approx. 100.000 downloads.
- The finished program is called Basic Model 2.rpp.

The Lightseeker



■ Since you have now worked with the Basic Model extensively it is time for your robot to learn to react to signals from its environment. Similar to the moth from our thought experiment of the first chapter it will detect a source of light and follow it. The construction kit contains 2 phototransistors that we will use as light detector. Thereby each sensor affects one motor making it possible for the robot to pursue the source of light.
The program consists of two parts. One part deals with searching for a source of light, the other part implements the pursuit or driving toward the light source. Again we will use subprograms to accomplish this. After switching it on the light-seeking subprogram will be activated. This subprogram will continue until a source of light has been detected. The main program tries to steer the robot toward the source of light. Whenever the direction of the robot greatly deviates from the ideal line, one of the sensors will no longer be illuminated from the light source. This makes the robot change its direction until both sensors can again detect the source of light.

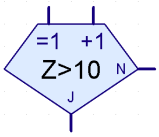
First assemble the light-seeking model according to the construction manual.

Assignment 1 (Level 2):

- **First program the "light-seeking" function. The robot should turn slowly at least 360°. Should it find a light, the robot will stop. Otherwise it will turn another 360° in the other direction. If it is still unable to detect a light source it will wait for 5 seconds and then begin again with its search.**
- **If it successfully detects a source of light the model should drive toward it. If the light source moves to the left or the right, the robot should follow the movements of the light If the robot looses contact the program should return to the light-seeking routine. See if you can attract the robot with a flashlight and guide it through an obstacle course.**

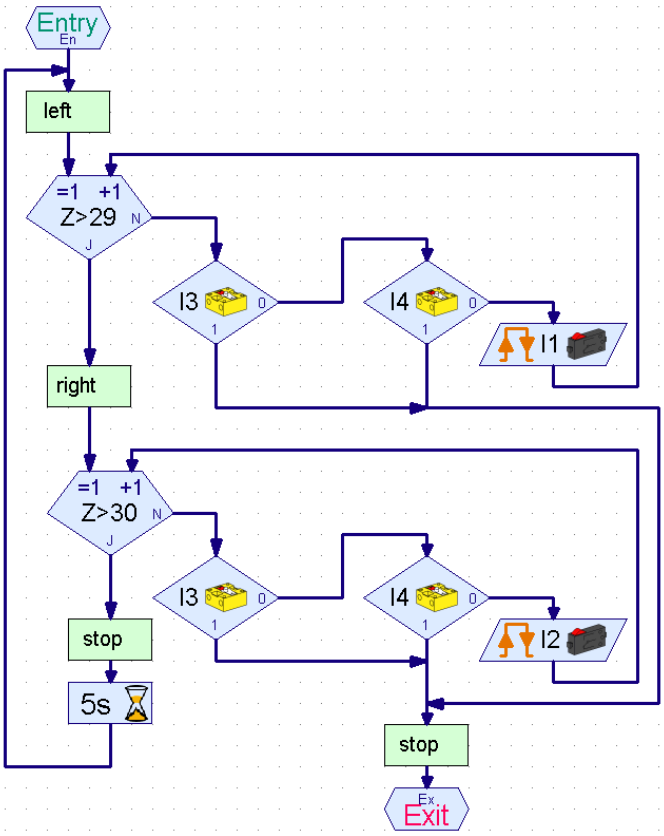
Tips:

- Use the subprograms you already programmed for the Basic Model for the different directions. As soon as the program Basic model 2.rpp has loaded you can find the program Basic model 2 and with it its subprograms in the **element group window** of ROBO Pro under **loaded programs.** You can simply insert these subprograms into your new program.
- For the "light-seeking" subprogram use the **Count Loop** element. (For a description of this element please see the ROBO Pro manual).



Count Loop

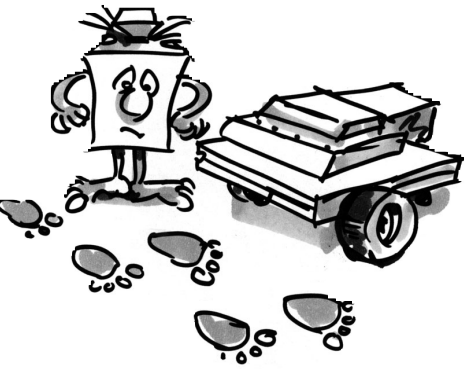
- In the loop between connection "N" and connection "+1" you query the phototransistors and count one pulse at the pulse sensor I1. The loop iterates until the robot has detected light or has rotated 360°. You simply have to try out how many loops it takes until the robot has completed a full rotation. Then enter that value "Z" in the count loop element.
- A second loop is then programmed in exactly the same way for the search in the opposite direction.
- If the robot detects light, it stops, and exits the subprogram.
- Here the complete light-seeking subprogram:



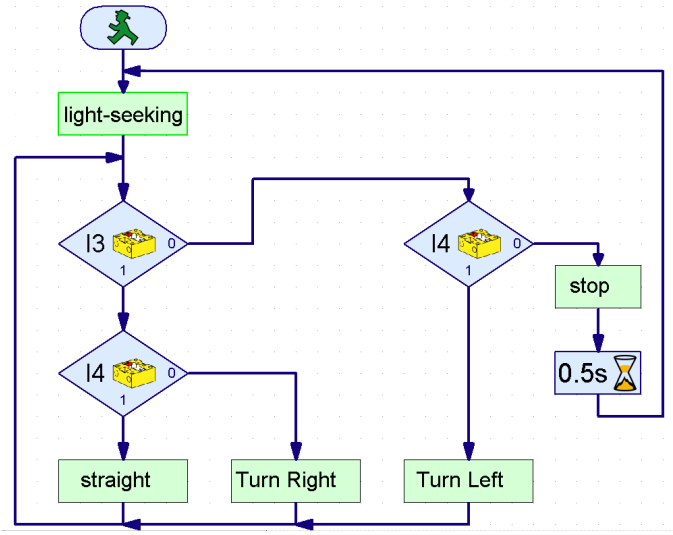
- In the main program you again query the phototransistors and control the motors depending on which phototransistor has detected the light:

Light at I3 and I4	**Straight Ahead**
Light only at I3	**Turn Right**
Light only at I4	**Turn Left**
No light detected	**Stop and return to the light-seeking subprogram**

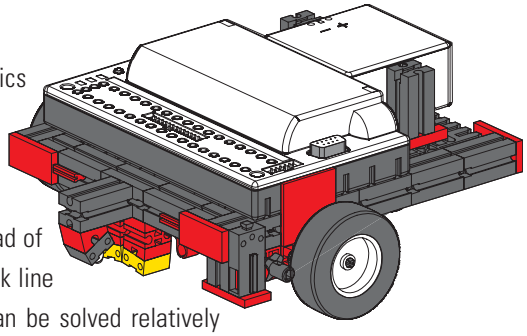
- You can achieve the right and left turns by setting different speeds for M1 and M2 with the same rotational direction. This results in a very smooth driving style.



- The main program finally looks like this:
- You will find the finished program under Lightseeker.rpp.
- Use a flashlight as light source. Take care that the beam of light is not focused too narrowly so that both photosensors are illuminated by the light source. Please note that in a very bright space another source of light such as sunlight through a big window might outshine the flashlight. This might cause the robot to drive right past your lamp toward the brighter light.



The Tracker



■ Search and pursuit are fundamental characteristics inherent to intelligent beings. With the Lightseeker you built and programmed a robot that is able to react to direct signals from its target.
With the Tracker we apply another search principle. Instead of a targeted approach to the source of light we mark a black line that the robot is supposed to follow. This assignment can be solved relatively easily by using the phototransistors. They measure the light reflected by the marking and the motors are corrected accordingly. To make sure this functions accurately the line is illuminated with the lamp. Take care to avoid an unfavorable configuration that causes stray light from the lamp to throw off the photo sensors. The light focusing properties of the incandescent bulb's optical lens is especially helpful in this regard.

Now follow the construction manual to put together the Tracker Model.

Assignment 1

- **First write the subprogram used to find the track. In order to do this the robot should turn around once.**
- **If it is unable to find a track it should drive straight for a short while and then start searching again. The phototransistors are queried for track recognition.**
- **If the robot has detected a track it should follow it.**
- **If the track has run out or the robot lost it, due to a sharp directional change for example, it should start a new search.**

Tips:

- After the lamp has been turned on you have to wait a short while (about one second) before you can query the phototransistors. Otherwise the phototransistor will detect "dark", meaning a track, where there is none, because the reading is done before the lamp is fully lit.
- As track you can use black duct tape that is about 20mm (or 0.787 in.) wide or simply draw a black track with this width on white paper. The turns should not be too tight otherwise the robot will lose sight of the track too frequently. First use the Interface test to make sure that the phototransistors are able to detect your track accurately. Don't forget to switch on the lamp when you do this.
- Adjust the lamp so both phototransistors output the value 1 on a light background, even with motors M1 and M2 turned on. If the battery charge is a little low, the lamp will be somewhat darker when the motors are turned on. If the lamp hasn't been adjusted properly it is possible that the phototransistor will read "dark" even though it hasn't found a track.
- Tracking works in a similar way as light-seeking. You just have to adjust the search so the model drives straight ahead for a bit, after the full rotation, before searching again.
- Please note that the model is supposed to drive straight ahead whenever both phototransistors output the value "dark" (=0).
- You will find the finished program under Tracker.rpp.

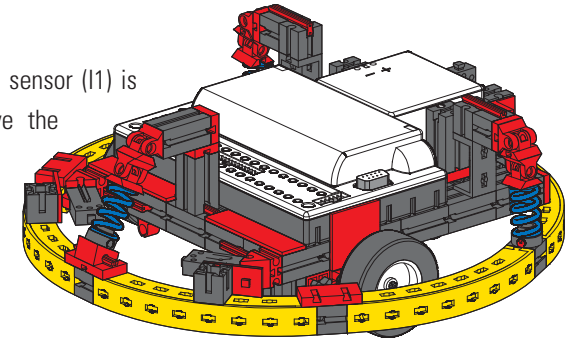
Assignment 2

- **Create a track with curves of varying degrees of tightness. Which is the smallest radius the model can handle?**
- **When correcting the track, experiment with different speeds of M1 and M2. Which combination offers the best result?**
- **Create a round track. Try to optimize the speeds in such a way that the robot achieves the best possible lap time. This assignment lends itself perfectly to a competition between several robots.**

Robot with Obstacle Detection

■ All of the robots we have built so far were able to cover a certain distance as well as follow a light source or a track. But what happens if there is an obstacle in its way? Well, either the obstacle is pushed aside or the robot keeps running against it senselessly until the battery is empty. It would, of course, be much more intelligent if the robot would be able to recognize the obstacle and avoid it accordingly. To accomplish this, the robot is equipped with a flexible circular bumper with three pushbutton sensors. With this bumper it can differentiate if an obstacle is to the left, to the right or behind it.
How it reacts to the obstacle remains only a question of the programming.

First assemble the model "Robot with Obstacle Detection". Only one pushbutton sensor (I1) is necessary to measure the distance traveled. For this reason we can remove the pushbutton sensor I2 from the Basic Model and use it for the obstacle detection.



Assignment 1 (Level 2):

- **First the robot should drive straight ahead. If it encounters an obstacle (I4) to the left, it should back up a bit and then move to the right.**
- **If it encounters an obstacle (I3) to the right, it should back up a bit and then move to the left.**

Tips:

- Obstacle recognition when going backwards will be set aside until later.
- The main program queries the pushbutton sensors. Depending on which pushbutton sensor is activated the model evades to the left or to the right. In each instance this is done using a subprogram.
- The pulse count when turning right should be different from the pulse count when turning left (e.g. 3 pulses to the right, 5 pulses to the left). Otherwise it can happen that the model drives into a corner and gets struck because it turns to the left and right in equal amounts.
- The finished program is called Obstacle 1.rpp.

There are two things the obstacle recognition model does not yet know how to do: It cannot recognize obstacles when going backwards. Equally it does not yet recognize when there is an obstacle directly in front of it. But it could be able to recognize both. If I5 is pressed while going backwards an obstacle is behind the model. Are I3 and I4 activated at the same time while going forward an obstacle is located directly in front of the model. In this case the robot could turn 90° immediately. In all we now have the following possibilities the robot should be reacting to:

Obstacle	Pushbutton Sensor	Reaction
right	**only I3**	**move to the left (turn approx. 30°)**
left	**only I4**	**move to the right (turn approx. 45°)**
in front	**I3 and I4**	**move to the left (turn approx. 90°)**
behind	**I5**	**Only queried when going backwards. Stop, then continue evading as planned.**

Some new program elements such as Operators (e.g. AND, OR) from ROBO Pro Level 3 can help you to solve this problem in an elegant way. Level 3 offers you the possibility to exchange data between different elements by using orange arrows. Switch to that level in the software to take full advantage of these options. It would now be a good idea to take out the ROBO Pro manual and read chapter 5 carefully. After that you'll be ready for the next assignment.

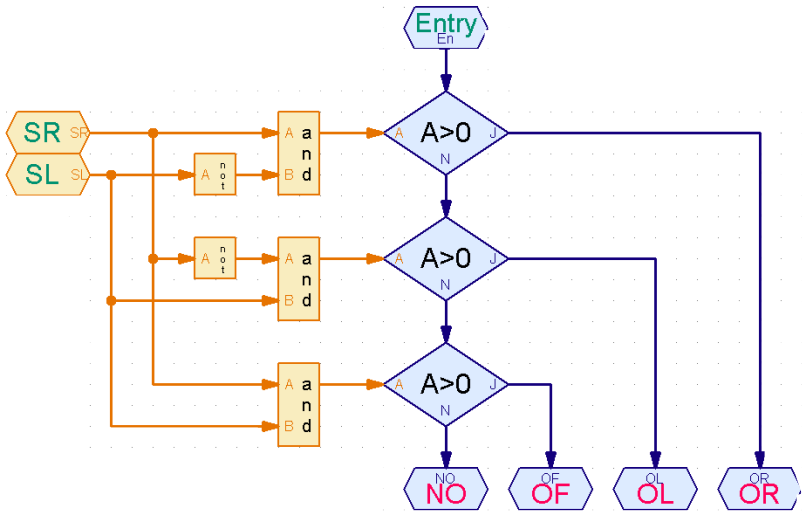
Assignment 2 (Level 3):

- **Modify your obstacle recognition program in such a way that the model will react as described in the table above.**
- **Take advantage of the possibilities offered by ROBO Pro level 3.**

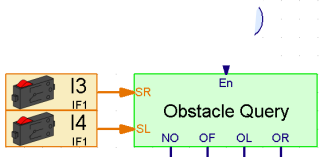
Tips:

- In the "Obstacle Query" subprogram the different sensor combination possibilities are queried using operators. The subprogram has a separate exit for each of the possibilities.

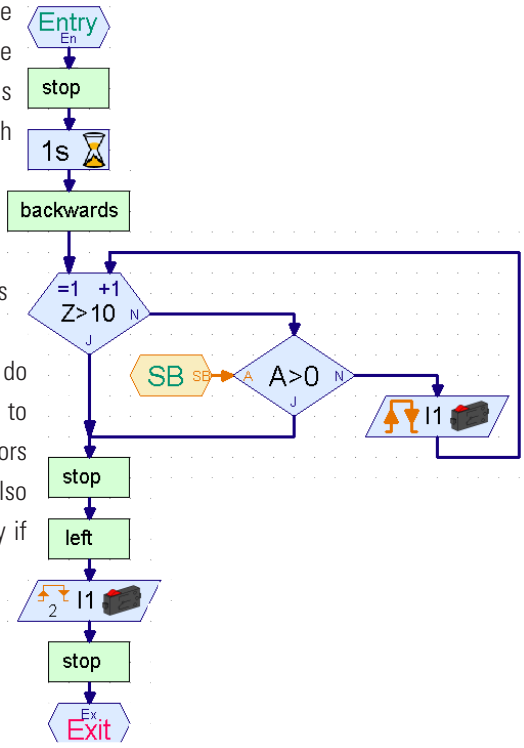
Data Input SR = sensor right
Data Input SL= sensor left
Exit NO = no obstacle
Exit OF = obstacle in front
Exit OL = obstacle left
Exit OR = obstacle right



- Place the orange Sensor elements into the main program and connect them to the subprogram using data inputs, so you can see right away which sensor is being queried.



- In each of the different evasion subprograms, I5 is queried when the robot is going backwards. The model will go backwards until either the set pulse count has been reached or until I5 is pressed. Again, I5 is placed into the main program, so you can see right away in which subprograms the sensor is queried.
- You will find the complete program under Obstacle 2.rpp.



The advantage of the programming technique used in this assignment is that you can see directly in the main program which sensor is being queried in the subprogram. If you would like to change the input you can do so at one single point without having to search through all subprograms to find out where the sensor might be hiding. Additionally, using operators enables you to create very clear logic operations. In principal this is also possible using branch elements. But this becomes confusing very quickly if several cases are queried.

Lightseeker with Obstacle Detection

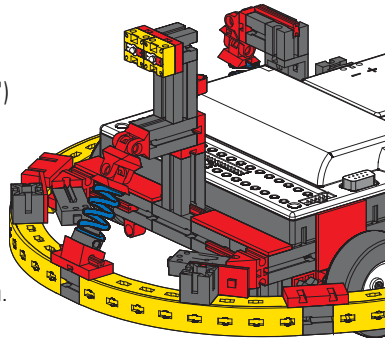
■ We are nowhere near the end of the possibilities offered by the ROBO Mobile Set.
For this reason we will now combine the two functions of light-seeking and obstacle detection. From a scientific point of view the robot will then be equipped with two behaviors. Since both behavior patterns can't be active at the same time, they will receive different priorities. As a rule the robot performs its light-seeking behavior. Should it detect an obstacle, a hazard to the robot, the obstacle recognition behavior becomes active. If everything is clear the robot can continue its light-seeking.

Professional software developers, when faced with such a demanding task won't simply plough ahead with programming. No, they use a specific strategy to develop the program. One of these methods is called "top-down design". With this approach the system is defined as a whole from the top down without dealing with the details at the beginning. It is this method we will use to work out this problem.

Assignment 1 (Level 3):
Teach the robot the following behavior patterns:
- **Search for a source of light.**
- **As soon as you have found it, follow it.**
- **Should an obstacle appear on the way, evade it.**
- **Then begin searching for a source of light again.**

Use the program elements of ROBO Pro level 3 to find the solution.
Apply the "top-down" approach when working on the assignment.

Tips:
First divide the assignment into three parts:
- Query if the robot sees a source of light (subprogram "Light")
- Query if it encounters an obstacle (subprogram "Obstacle")
- Tell the robot what to do depending on these results (subprogram "Driving")



Now consider the different situations the robot is able to perceive to create the subprograms "Light" and "Obstacle". Assign a numerical value to each situation. This value is saved as variable using a Command element. Each situation will result in a reaction that is executed in the "driving" subprogram.

Subprogram light:

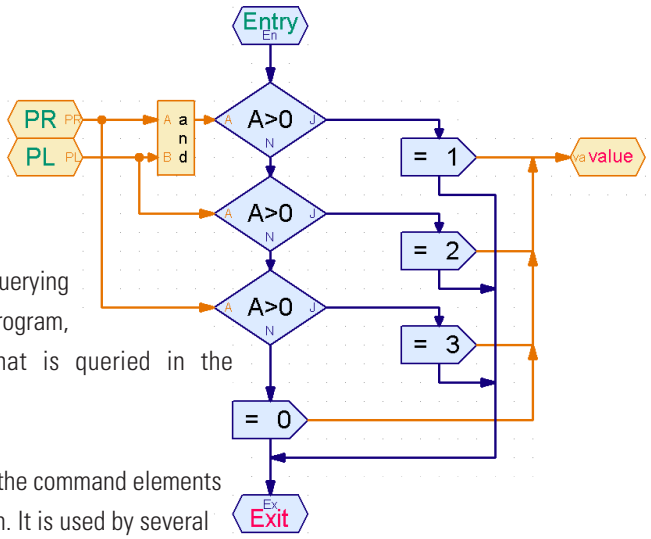
No	Situation	State of the Sensors	Reaction
0	No light source present	I6=0; I7=0	seek light
1	Light source directly in front of robot	I6=1; I7=1	drive straight ahead
2	Light source to the left of robot	I7=1	execute a left turn
3	Light source to the right of robot	I6=1	execute a right turn

Subprogram obstacle:

No	Situation	State of the Sensors	Reaction
4	Obstacle directly in front of robot	I3=1; I4=1	evade 90°
5	Obstacle to the right of robot	I3=1	evade to the left
6	Obstacle to the left of robot	I4=1	evade to the right

Now you simply have to reproduce these conclusions in ROBO Pro using program elements.

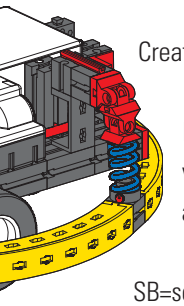
Subprogram light:



PR=phototransistor right
PL=phototransistor left

As before, place the elements for querying the phototransistors into the main program, so you can see right away what is queried in the subprogram.

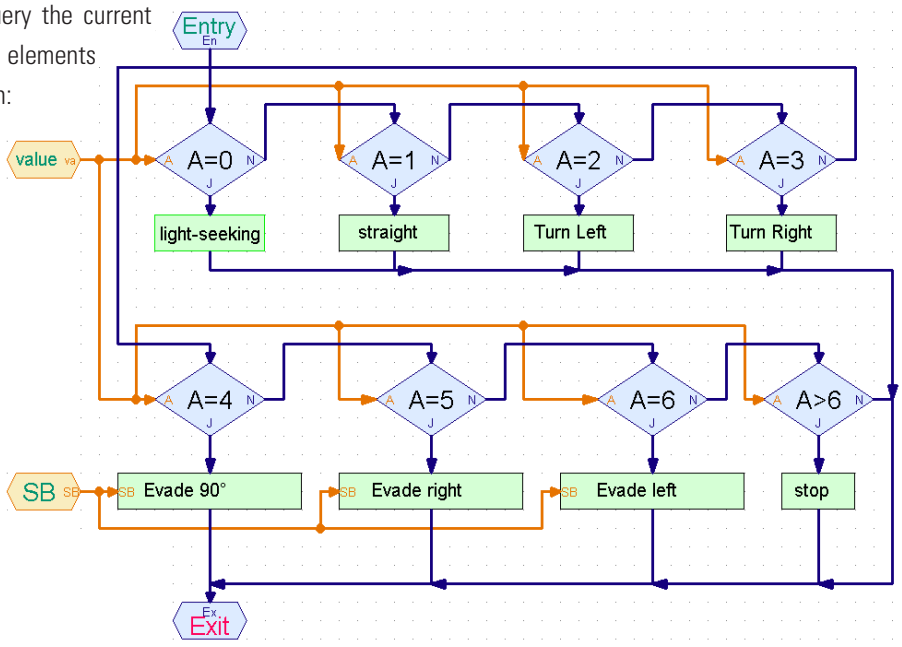
The variable storing the value from the command elements is also placed into the main program. It is used by several subprograms. You connect it to the subprogram using a data output.



Create the subprogram "Obstacle" according to the same principles as the subprogram "Light".

In the subprogram "Driving" you query the current value of the variables using branch elements and program the appropriate reaction:

SB=sensor behind



As a final detail you now have to create the subprograms used in this subprogram.
But wait a minute! Most of them already exist. The light-seeking subprogram for example can be copied from the program for the Lightseeker Model. If you don't remember how to do this, please read chapter 4 of the ROBO Pro manual.

But watch out:
For the Lightseeker Model the phototransistors were connected to input terminal I3 and I4. But now they are on I6 and I7. Additionally, we queried pushbutton sensor I1 to count the pulses when turning left, and I2 when turning right. Now there is only I1 to count the pulses, which works just as well by the way. This means that you will have to adapt the light-seeking subprogram after you have copied it. Since the sensor query is hidden in the subprogram it is easy to miss. This won't happen if you place the inputs into the main program and connect them to the subprogram using data inputs. But you weren't aware of this yet when we worked on the Lightseeker.

The subprograms for evasion also already exist, we wrote them for the Obstacle Recognition Model. Here the pushbutton sensor I5, queried additionally when going backwards, has already been placed into the main program

You can take a look at the finished program under Obstacle-Light.rpp.
At first glance the main program looks very clear and simple. But there is a lot of mental elbow grease behind the subprograms. Still, by using the step-by-step approach of the top-down method you too would be able to tackle such a complex program.

By the way, if you have a friend who owns a ROBO Mobile Set as well, you can go even further with these experiments. Simply mount a source of light on each of the robots. And both robots will be seeking each other.

Robot with Edge Detection

■ We have just seen, in the previous example, how to approach the programming of a more complex program. Now you can turn to another very important behavior of a mobile robot. It is supposed to learn not to fall off the table. In most cases driving against an obstacle won't hurt the robot. But if it falls off a table that is almost three feet high it might be damaged in one way or the other, even if the fischertechnik building blocks are very robust. For this reason the robot will be equipped with sensors that enable it to recognize edges. These edge detectors each consist of a pushbutton sensor that is activated by a rotating wheel. This wheel is also able to move up and down. As soon as the wheel moves over the edge of the table it drops, the pushbutton sensor is no longer activated, the program realizes that the model has reached a precipice and reacts accordingly. The robot has 4 edge detectors in total, enabling it to feel for a precipice on each side while going forward or backwards. As a result this model does not have a pulse sensor to measure the distance traveled. The covered distance will be controlled using the on-time of the motors.

First assemble the model according to the construction manual.

Check carefully if the edge detectors respond correctly:

- when the model reaches the edge of the table and the pushbutton is pressed precisely again
- when the wheel is on the table again.

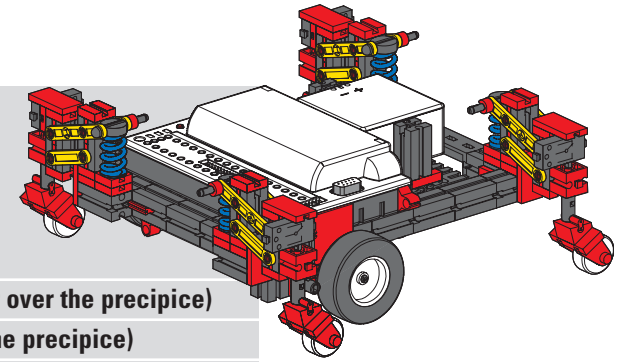
If necessary, one or the other pushbutton sensors might have to be adjusted up or down a bit.

Assignment 1 (Level 3):

- **First consider how the robot should react when reaching a precipice.**
- **Upon closer examination you will realize that there are many possible combinations of sensors located over the precipice.**
- **One of the 4 detectors could be activated, or 2 or 3 different ones at the same time, or even all four sensors.**
- **How should the robot react to each of these possibilities?**

Tips:

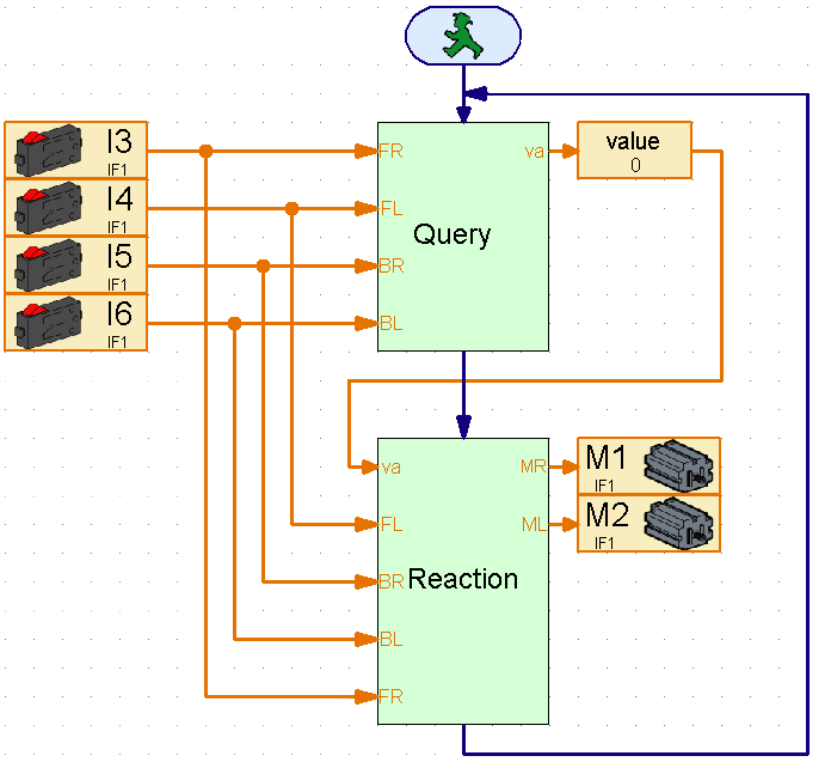
You will find the solution in the following table. Those sensors located over a precipice (pushbutton sensor=0) are marked with an ●. Each combination receives a number. In the program that will be created later each option is assigned the corresponding number. The robot will react to the current situation on the basis of that number. But more about this later. First, only think about how the robot must stand for a certain combination to occur and if it reacts correctly.



No.	Front Right (I3)	Front Left (I4)	Back Right (I5)	Back Left (I6)	Reaction
0					Straight ahead (no sensor over the precipice)
1	●	●	●	●	Stop (all 4 sensors over the precipice)
2	●	●	●		Turn a bit to the right
3	●	●		●	Turn a bit to the left
4	●		●	●	Turn a bit to the left
5		●	●	●	Turn a bit to the right
6	●	●			First back up, then turn to the right
7	●		●		Turn a bit to the left
8	●			●	Turn a bit to the left
9		●	●		Turn a bit to the right
10		●		●	Turn a bit to the right
11			●	●	Drive forward a bit
12	●				First back up, then turn to left right
13		●			First back up, then turn to the right
14			●		Drive forward a bit
15				●	Drive forward a bit

Pretty intense, right? But don't worry there is a finished program for this model using all the advantages ROBO Pro has to offer. It is called Edges.rpp.

The most important elements are located in the main program making it easy for you to understand the sequence as a whole. The complex query of the sensors and the control of the motors are hidden in subprograms. First the main program:



The sequence starts with the query of the 4 pushbutton sensors. All the way to the left you can see which pushbutton sensor is being queried. They are connected to the subprogram via data inputs. The "Query" subprogram determines which pushbutton is pressed and assigns the value detailed in the table above. This value is assigned to a variable with the same name that you'll be able to recognize in the main program. The value of the variable is then sent to the "Reaction" subprogram that controls both motors depending on this value. The pushbutton sensors are read again in the "Reaction" subprogram since the edge sensors are still queried while the model is avoiding the precipice.

If need be, you are now able to change the assignment of the pushbutton sensors as well as the motor outputs on the Interface, without having to dig through the subprograms to see where an input element or a motor symbol might be hiding. Each input and each output appears only once.

This programming technique is especially useful when you would like to use a subprogram for many different models and don't know yet which inputs and outputs to you'll use on the Interface in each case.

If your curiosity has been piqued simply take a look at the subprograms and try to understand them. The programming principle is similar to the "Lightseeker with Obstacle Detection" Model.

Assignment 2 (Level 3):

Load the program into the Interface and let the model drive around a table.

- **Does the model always react in the right way?**
- **Should it behave differently with certain sensor combinations?**
- **Tweak the program as required.**

The Walking Robot

■ After concentrating on wheeled robots we now turn to another method of locomotion that we can use for mobile robots, walking.
The gait of insects lends itself perfectly as model for the movement of "mechanical six-legged walkers". During the so-called tripod walk three of the six legs always lift off the floor simultaneously. The front and back legs on one side lift together with the middle leg of the other side.

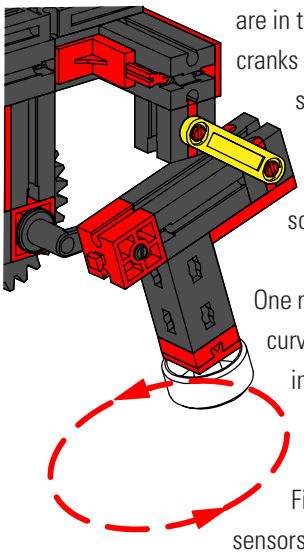
Tripod Walk

Die Beine, die auf dem Boden stehen (schwarz dargestellt), bilden ein stabiles Dreibein, so dass das Modell immer sicher steht und beim Laufen nicht umkippt.

The legs that remain on the floor (shown in black), form a stable tripod, so the model always has a solid stance and does not tip over when walking.

The legs of the fischertechnik Walking Robot are constructed as so-called four-joint gears. The design of the four-joints used here is called a "crank-rocker mechanism". Driven by a crank, the floating members of the gears make oscillating movements. The distance between the individual joints and the position of the nadir (this is the bottom of the leg), are arranged in such a way that the foot performs an elliptical movement when the drive crank is turning. This results in a movement resembling a walking step.

The 6 cranks driving the legs have to be adjusted exactly as shown in the construction manual. The three legs that touch the floor at the same time have the same crank position. The cranks of the 3 legs that are in the air at that time are rotated 180° against the other three. The correct position of the cranks in relation to each other ensures that the model is able to walk with the correct sequence of steps, the tripod walk.



The hub nuts used to secure the gears on the shafts have to be screwed in tightly, so the cranks won't become misaligned during walking motion.

One motor each drives the right and the left side of the model (this is necessary for walking curves). For this reason you have to make sure that the middle leg on one side is always in the same position as the two outer legs on the other side. The software controls this synchronization via the pushbutton sensors I1 and I2.

First assemble the model according to the construction manual. Double-check all sensors and motors using the Interface test, to make sure they are connected accurately. Rotational direction of the motors: rotational direction left = straight ahead.

Assignment 1 (Level 1):
Teach the robot how to walk.

- **Program the model to walk straight ahead using the tripod walk.**
- **Use the pushbutton sensors I1 and I2 to synchronize the left and right legs.**
- **When doing so make sure that the two outer legs on one side and the middle leg on the other side are in the same position.**

Tips:

- First bring the legs on the left and the right side into the starting position. Switching on both motors will allow you do this (rotational direction left).
- The sequence should continue only when both pushbutton sensors I1 and I2 are not depressed (This query is necessary as soon as the model is supposed to take its second step).
- Let the motors run until the appropriate pushbutton sensor (I1 for M1, I2 for M2) is depressed again. It is very important here that the model doesn't begin the next step until both pushbutton sensors are depressed. Because only then the legs are in the right position to each other. Provided, of course, that the cranks driving the legs are adjusted correctly as shown in the construction manual.
- Now the sequence can start over and the robot will take its second step. The model will now walk straight ahead until you stop the program.
- You will find the finished program under Walking Robot 1.rpp.

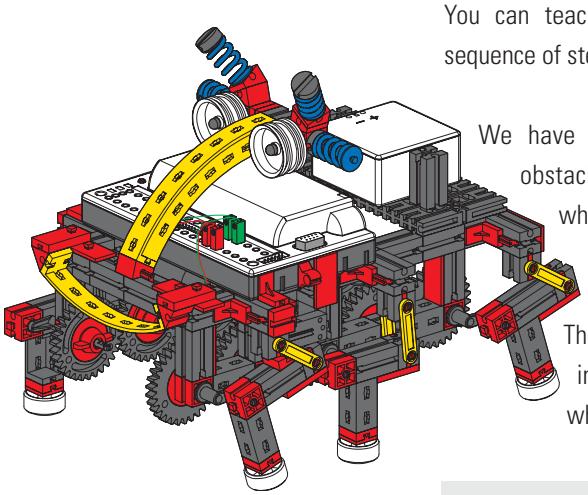
Similar to what we did with the wheeled basic model you can now make the model walk to the left, to the right or backwards by changing the rotational direction of the motors. You can use I1 or I2 to count the steps.

Assignment 2 (Level 2):

- **Program your model to take 10 steps forward, 3 steps to the left, 3 steps to the right and 10 steps back again.**
- **Create an individual subroutine for each direction.**
- **Use the Count Loop element to count the steps.**

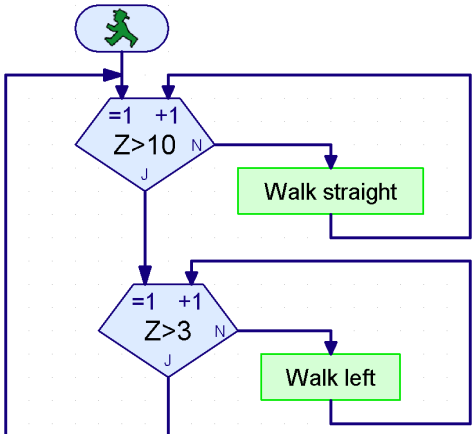
Tips:

- Simply copy the program Walking Robot 1.rpp into a subprogram.
- Copy this subprogram as often as necessary for each of the different walking directions. Change the rotational directions of the motors in each subprogram to make the model move into the desired direction.
- Use the Count Loop element to count the amount of steps for each rotational direction. With each cycle of the subprogram the model takes one step. If the program cycles through the loop with the subprogram 10 times, the model takes 10 steps.



You can teach your walking robot any desired sequence of steps in this way (Walking Robot 2.rpp).

We have already discussed the subject of obstacle recognition in detail for the wheeled robots. So we won't repeat it here. But why don't you try to apply this behavior to the walking robot. The necessary sensors are all included in the construction kit. You can use the wheeled robot as example during the programming. Best of Luck!

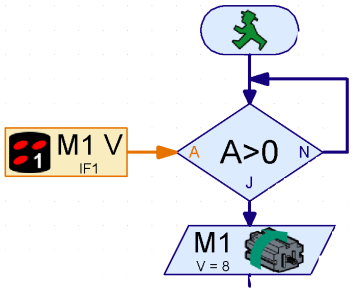


Expansion Possibilities

■ The ROBO Interface offers many more functions than discussed previously with the mobile robots. But you will need additional components that are not included in the construction kit to take full advantage of them. But since they offer very interesting options for the robots we now want to introduce a few of them here.

Handheld Infrared Transmitter

■ The ROBO Interface has an infrared receiver diode for the handheld transmitter included in the IR Control Set Art No. 30344. This enables you to query the buttons on the transmitter in the ROBO Pro software as digital inputs and so for example turn motors on and off.



We have programmed a remote control for the walking robot as a sample program. With the 4 oval arrow buttons on the remote control you can make the model go forward, backwards, to the left and to the right. You only have to load the program Walking Robot IR.rpp into the Interface to get started.

Another ingenious program in connection with the remote control is the program Mobile-Teach-IR.rpp. Using this teach-in program you can control a wheeled robot, e.g. the Simple Robot or the Basic Model. The model will remember the path it traveled and will then be able to repeat it as often as you want it to. But once the program is stopped this saved path will be erased.
It is the "List" program element in ROBO Pro that makes such a program possible. Many values can be stored in this element and then called up again (please also see the ROBO Pro manual). The program itself might be pretty complex but using it is very simple:

1. Load the program Mobile-Teach-IR.rpp into the Flash memory of the ROBO Interface and start it.
2. Press the **M1** ▸/▸▸ button on the remote control. This starts the "learning process".
3. Use the oval arrow buttons to steer the model into the desired direction.
4. Press the **M2** ▸/▸▸ button. This will save the path the robot traveled.
5. Press the **M3** ▸/▸▸ button. This will make the robot travel along the saved path.

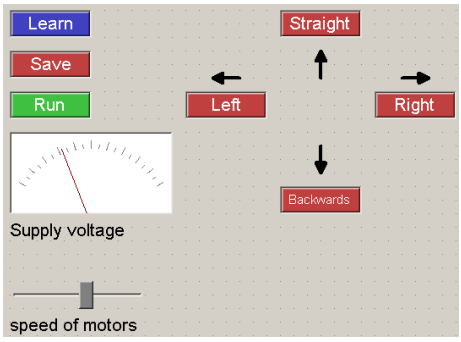
With an application like this, the programming of robots is a breeze! You should note that the saved path will be erased as soon as you stop the program with the Prog button on the Interface.

ROBO RF Data Link

■ The radio interface ROBO RF Data Link Art No. 93295 replaces the interface cable between PC and the Interface with radio data transmission. This is a fine thing indeed. First of all you won't have to keep connecting and disconnecting the cable each time you load a program into the Interface. Secondly, you will be able to run programs wirelessly in online mode. This will make it much easier to find errors than with the regular download operation. And finally you will be able to control the mobile robots in online mode on your screen using a panel in ROBO Pro, similarly to using the IR remote control. But differently from the remote control, the screen additionally displays the data provided by the interface such as values of variables or analog inputs, supply voltage from the battery pack and speed of the motors.

As an example we have modified the teach-in program and are now able to control the wheeled robot using a software panel. The program is called Mobile-Teach-RF.rpp. Of course you can also try it out using the interface cable. But that might be pretty uncomfortable. The operating range of the model is limited, the cable gets tangled and the robot no longer turns correctly. After trying this, you will probably run out right away and get yourself the RF Data Link.

Load the program Mobile-Teach-RF.rpp.
Switch to Panel in the Function Bar of the main program . Then start the program in online mode. Now you'll be able to control and program the model using the buttons in the panel.



1. Press the "Learn" button. This starts the "learning process".
2. Use arrow buttons to steer the model into the desired direction.
3. Press the "Save" button. This will save the path the robot traveled.
4. Press the "Run" button. This will make the robot travel along the saved path

Here as well, the saved path will be lost once the program is terminated.

Please refer to the ROBO Pro manual to learn more about creating panels.

ROBO I/O-Extension

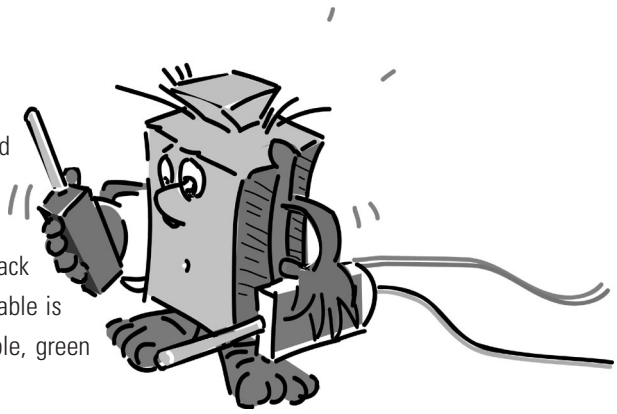
■ Should you build a model with so many sensors and motors that the input and output terminals of the ROBO Interface are not enough, you can connect a ROBO I/O-Extension Art No. 93294 to the Interface. This will provide you with an additional 8 digital inputs, 4 motor outputs and one analog resistance input. A second and a third module can be connected to this I/O-Extension, all controlled using a ROBO Interface. This will provide you with a total of 16 motor outputs, 32 digital inputs, 5 analog resistance inputs, 2 analog voltage inputs as well as 2 inputs for distance sensors.

If this is still not enough for you, you can even control several Interfaces from your PC in online mode. For example, one connected to a serial COM interface, one to the USB port, or 2 Interfaces connected to the USB port and each with up to 3 ROBO I/O-Extensions! Dizzying, isn't it? Chapter 6 of the ROBO Pro manual also explains how the whole thing works.

Trouble Shooting

■ Experimenting is fun. That is, as long as everything is working. Most of the time this will be the case, but unfortunately not always.
It is only when a model is not working right, that you will find out if you truly understand the mechanism and are able to find the fault right away.
With mechanical faults at least there is something to see (assembled incorrectly) or feel (stiff to move). But if electrical problems also arise it becomes much more difficult.
The pros use a series of different measuring instruments to troubleshoot, such as voltmeter or oscillograph. But not everyone has such devices at hand. For this reason we want to try to zero in on a fault with simple means and fix it.

Assembly of Cables



Before we begin with our experiments, we first have to get some of the components from the Fischertechnik construction kit ready. The supplied connectors, for example, are clamped to the individual cable segments.
First we cut the cables to size. We measure the specified lengths and cut the segments accordingly. Each cable is tested after assembly using the battery pack and the lamp. If the lamp lights up after it is connected to the battery, the cable is ok. We will also check if the color-coding is correct, red connector red cable, green connector green cable.

Interface Test

If a program (even a supplied one) does not work in connection with our model we start the Interface test. This utility program enables us to test each input and output separately. Are the sensors working? Are the motors rotating in the right direction? For all our mobile robots the motors are connected in such a way that the wheel or the leg will move forward if the rotational direction=ccw. If everything is ok here as well we'll start looking for a mechanical cause.

Loose Connections

Loose connections are a nasty fault. On one hand it is possible that the way the connectors fit into the sockets is too loose. In this case you can adjust the contact springs a bit using a small screwdriver. But be careful, if you bend them too much the contacts might break or the fit might become too tight.
Another cause of loose connections are the clamp locations where the connector is secured to the cable with a screw. Please tighten the screws carefully! This is also an excellent opportunity to check if any of the thin copper wires might have broken off.

Short-Circuits

On occasion you might also create a short-circuit by connecting cables incorrectly. In that case, nothing will work as it should. The battery pack has a built-in fuse that will interrupt the current when the temperature or the current is too high. In case of overheating the outputs terminals of the Interface will also be shut down.
There can also be a short circuit if you fail to tighten the little screw properly that secures the electrical connector to the cable. The screw might then stick out over the edge of the connector. If you plug two connectors into two adjacent sockets on the Interface and their screws come into contact a short-circuit

will result. For this reason the little screws should always be tightened properly. Always make sure that the screws don't come into contact with each other when plugging in the connectors.

Power Supply

If there are inexplicable interruptions during operation an almost empty battery pack might be the cause. The voltage falls for a shot time when connecting a load (motor on). This causes a reset of the processor on the Interface. When the red LED on the ROBO Interface lights up, the voltage of the power supply is too low. This means the battery pack needs to be charged.

Programming Error

If errors occur in a program you have written yourself and you cannot find a way to explain them, it might be a good idea to be on the safe side and load one of the supplied programs that comes closest to yours. This way you will be able to rule out electrical or mechanical defects. During online mode you can follow the program flow on your screen. If the program gets stuck at a certain point, then this is the place to look for the cause. You might have selected an incorrect input or motor for example or maybe an incorrect value is queried at a branch or a Y/N connection has been switched.

If none of this is successful, you can always contact fischertechnik service (e-mail: info@fischertechnik.de).

Or visit our website at www.fischertechnik.de. There you will find a Forum, Chat, Market place, Gallery and you can join the Fischertechnik Fan-Club for free.

We hope you'll have many hours of fun with the ROBO Mobile Set with plenty of surprises and sudden insights.