Robotics 2

RLT Robotics Learning Track

Module for Level 2
Version 4.3 - Sept 2012

Colophon

This Robotics module is part of the RoboDidactics Robotics Learning Track (RLT). The material presented in this module is based on the Dutch Robotics material developed by the author for the SLO Certified Robotics Module.

All the original and associated material for the RoboDidactics Learning Track may be downloaded from the Phyrtual site or from the RoboPal for NXT site at www.virtualbreadboard.com. It can also be found in the English download section of www.robocupjunior.nl. Teachers are permitted to modify this material for use in their own lessons, provided these changes are reported in the colophon of the modified material. The Phyrtual site can be reached through www.phyrtual.org.

This module was developed and translated by the author (Peter van Lith) as part of a cooperation agreement with the Fondazione Mondo Digitale in Rome, Italy in 2010. The RoboPAL software used in this version has been developed with VirtualBreadBoard by James Caska.

This version is developed for use with the Lego MindStorms NXT and RoboPAL software. A more extensive version is available (currently only in Dutch). It is based on robots that can be programmed in Java, using the Java Simulator and Eclipse.

This NXT version is easy to use because it uses a graphical programming language.

The module consists of three parts. The first is the basic version needed for all further lessons. The second part deals with the RoboCup Junior Rescue challenge. The more demanding third part is optional and deals with a simple simulated organism, based on reactive behaviour.

CONTENTS

Introduction

Intro Part 2

The Robotics module consists of three parts that may be followed individually. In Part One, the emphasis was on getting to know the robot, its development environment and programming the robot with RoboPAL.

In this second part, you will learn how to program a Rescue Mission in which a dangerous container has to be removed from a swamp (the yellow area). This part is intended for middle grade students.

The third part is more difficult and concentrates on developing a small Robotics project that addresses some of the aspects of Artificial Intelligence that are used in Robotics. This part is more suitable for theoretical education, but may also serve as an introduction to information technology.

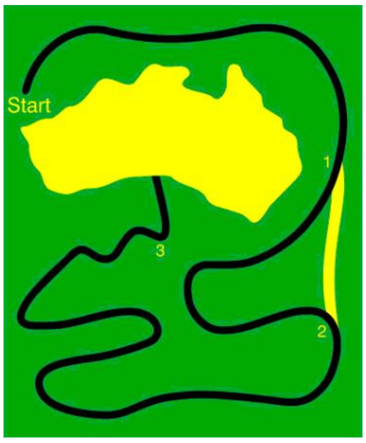


Fig 1: Rescue Field

5. Sensors: the ‘Sense’ Phase

In the first part of this module, you learned that if your robot does not have sufficient information, it is not capable of following a line. This happens because the robot does not know where it is with respect to its desired trajectory and it simply follows the described path. We can use sensors to collect the information necessary to correct its path.
In this chapter, we introduce an important principle, the Sense-Reason-Act *loop*. Using this loop construction, the robot will be able to determine how to behave and adapt to changing conditions.

5.1 You will learn

- how to use sensors
- how to handle sensor data
- what properties sensors have
- what sensors can do

5.2 You will need

- a robot and a computer with RoboPAL
- a rescue field
- the *Calibrate*, *DriveBehavior* and *CuriousBehavior* programs

5.3 You will experiment with

- calibrating the left reflection sensor for yellow, green and black
- making a table with measurement values
- programming the robot to drive up to the black line
- setting a time limit
- adjusting the sensitivity of the sensors
- searching the swamp in the rescue field

5.4 After following this chapter, you will be able

- to calibrate sensors
- to make the robot react to colors and objects
- to use loop constructions
- to draw a graph of the measured values of a sensor and determine their linearity
- to explain how a Sense-Reason-Act loop works.

Type	#	Assignment	Description
	5A	**Discussion**	**How does a robot work?**
	5A.1	Group discussion	Where are the sensors located?
	5B	**CalibrateBehavior**	**Reading sensor values**
	5B.1	Calibration	Reading sensor values in the simulator
	5B.2	Calibration	Reading sensor values on the robot
	5C	**DriveBehavior**	**Driving up to the black line**
	5C.1	DriveBehavior	Stopping on the black line
	5D	**Loops**	**Making a loop**
	5D.1	Loops	Using a loop
	5E	**DriveBehavior**	**Using a time limit**
	5E.1	DriveBehavior	Adding a time limit
	5F	**CuriousBehavior**	**Making a graph**
	5F.1	Measuring	Drawing a graph
	5F.2	CuriousBehavior	Changing CuriousBehavior
	5G	**DriveBehavior**	**Mowing the grass**
	5G.1	Search the Swamp	Mowing the grass
	5G.2	Modification	Using a subroutine
	5H	**Test**	**Test on chapters 4 & 5**

5.5 Explanation

The *Sense-Reason-Act* Loop

Three logical steps are used to control robots. As a matter of fact, you will also find this true of the behavior of animals and possibly humans. First, our senses observe things, the ***Sense*** step. Then, we determine what to do with that information - the ***Reason*** (or thinking) step - and as a result of this, we need to take an action, the ***Act*** step. As these three elements are used in succession, we call this a loop or a repeat construction.

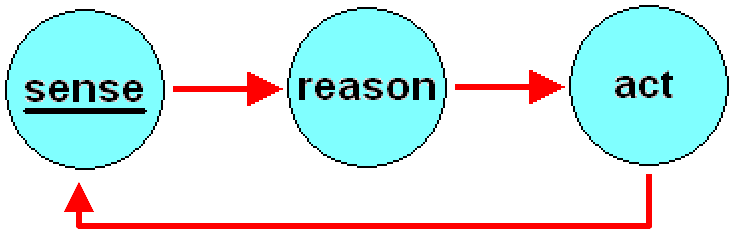


Fig 2: Sense-Reason-Act Loop

Robots use sensors to retrieve information from their environment. In this chapter, you will investigate how robots perform the Sense step and use the information they collect.

5A.1 Assignment: Group Discussion

- What sensors do we know about?
- In what type of equipment can we find them?

Senses versus Sensors

In chapter 4, we saw that controlling the robot’s motors with fixed values directly from the program does now work very well. Driving over a black line using direct instructions is difficult and requires a lot of work. Even small disturbances send a robot off-track, so we must find a different method. By observing the environment with its sensors, a robot can adapt to the circumstances around it and operate more flexibly to achieve its goal.
Living organisms, whether they are single-celled or human beings, understand their environment through their senses. Robots are equipped with sensors for the same reason. There is a large variety of sensors, but in general they serve the same function as human or animal senses.

Some sensors, like infrared or Röntgen rays, even detect information that we cannot gather with our senses. Most sensors, however, are more limited than our senses; indeed, we sometimes need to use several different sensors to match the operation of just one of our senses.
We are so familiar with our senses that we often assume that sensors on a robot work in an identical manner, but this is often not true. The processing of raw data by a sound sensor, for instance, may lead to very different conclusions than those reached by people using their ears. In addition, our experiences are highly influenced by the associations our brains make with these observations (i.e., optical illusions).

Moreover, two sensors that have the same function may provide different results due to small differences in their manufacturing process. These differences make it necessary to match the sensors. We do this on the basis of a standard reference value. This process is called ***tuning***.
When, for example, a sensor is tuned to represent raw data values on a scale from 0 to 100, we call this process ***calibration***. The process of adapting the non-linear behavior of a sensor to linear behavior is also called calibration. Calibration makes it a lot simpler to compare the values of different sensors.

Reading Sensor Values in the Simulator

If you start the simulator with a rescue field and a robot, on the right side, you will see the control panel with four dials. These dials display the sensor values. They represent, from left to right, the values of the following sensors:

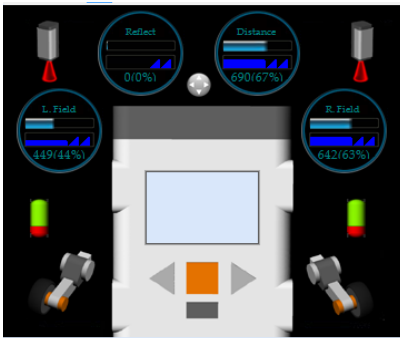
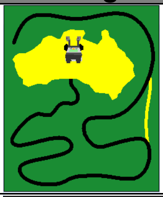
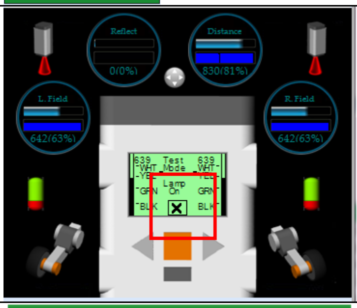
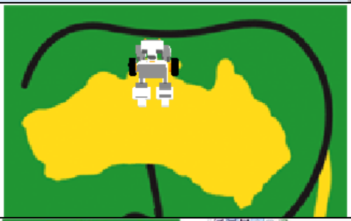


Fig 3: Sensor Values in the Simulator

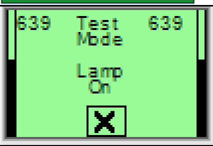
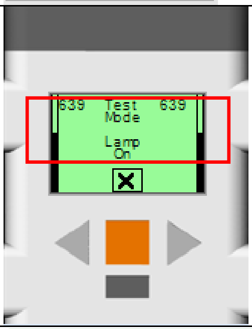
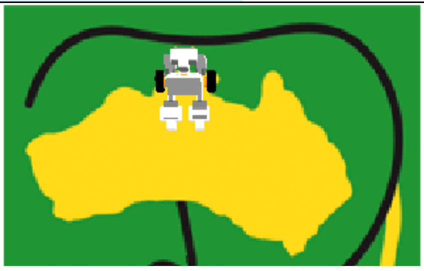
- Left field sensor
- Reflection sensor (if connected).
  We do not use this sensor in these lessons.
- Distance sensor (Ultrasonic sensor)
- Right field sensor

5B.1 Assignment: Reading Sensor Values in the Simulator
Write down the values that you will need for your first calibration.

5B.1	Reading Sensor Values in the Simulator	
1		Select RoboPAL 5B – Calibration from the examples and save it as 5B.1 – Calibration. Start this program in the simulator and watch the control panel carefully. Place the robot on the yellow area as shown on the left.
2		Press the TST button on the control panel to obtain the calibration screen. It is the same as on the real NXT, which you will be using later to collect calibration information. You can see that the Left and the Right Field sensors both have a value of about 640 or 63%. This is the standard value for Yellow.
3		Press the second icon above the Control Panel to view the field from above. This view makes calibration easier because you can judge the position of the robot better. Use the mouse to move the robot to the green area and look at what values the sensors display there.
4		Draw a table on a sheet of paper and write down the sensor value for each color for both the left and right sensors as displayed in the simulator. Then, put the robot exactly on the black line. This is more difficult: watch how the values change. The lowest values are displayed when you place the sensor exactly on the black line. If your robot is only partly on the line, the value will be somewhere between that of green and black.

5B.2 Assignment: Reading Sensor Values on the NXT

- The sensor values displayed in the simulator are just an approximation. So, we will use the NXT to perform the same measurements on the real field and get an idea of the differences.
- There usually also are differences between the Left and Right sensors as well as between sensors on other robots.
- The real values are always slightly different from what you see on the simulator. So, you will have to change your program.

5B.2	Reading Sensor Values on the Robot	
1		Switch on your NXT. You do not need to load any programs, the calibration program is already installed. Push the TST button and you will see the same screen you saw in the simulator. Place your robot on the Rescue field as shown on the left.
2		The screen on your NXT displays less information than the simulator. The extra information displayed in the simulator is only available on the NXT in Level 1. We are working with Level 2.
3		Press TST on the start-up screen to view the calibration panel and collect the sensor information. Write down the values for Yellow of the Left and Right sensors and compare them to those from the simulator. You will notice that these values are different. Ambient light, light from outside and shadows influence these values. Turn your robot in different directions to see how the values change.
4		Also measure the values for Green and Black and write them down on your sheet of paper. Be extra careful with Black: watch the reflection of the red leds on the surface. They must be exactly above the black line to read the correct values. Also compare these values with those from the simulator. Keep this sheet of paper, because you will need these values again for the programs that you are about to make.

Driving Up to the Black Line

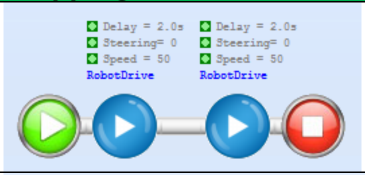
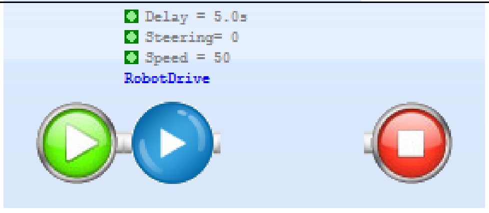
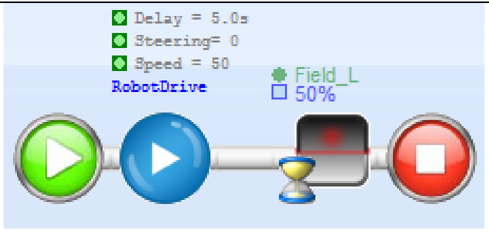
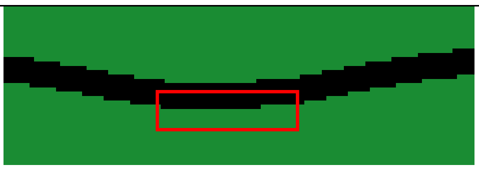
Suppose your robot has to stop on the black line. Each of the two field sensors is able to read enough information to distinguish the green field from the black line. You will make your robot drive forward and stop as soon as, for instance, the left sensor detects a dark surface.

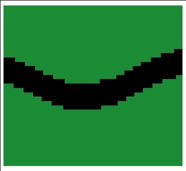
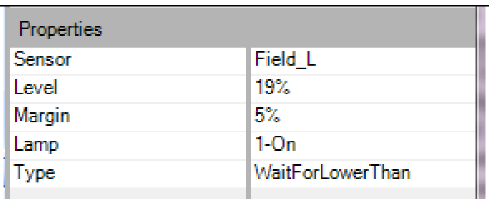
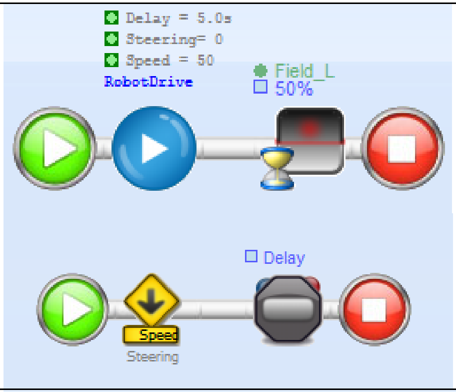
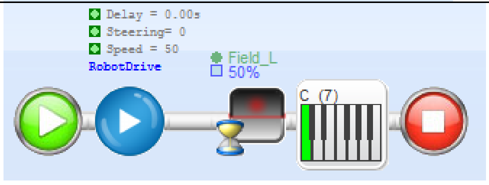
Use the *Driver* icons to make the robot move forward and read the sensor value with the *BranchIfHigher* or *BranchIfLower* icons. As we have several sensors, each sensor has a unique name.

For now, you will only use the Left Field sensor - the *Field-L* - that you will find in every *BranchIf* icon.

5C.1 Assignment: Stopping on the Black Line

- The *RobotDrive* FlowCode must be modified so that, in every iteration of the loop, the left field sensor checks if it is driving over a dark area. It is important to do this repeatedly otherwise the information will only be read once and the robot may miss the black line.
- As black reflects the least amount of light, we need to check for a value that is lower than that of Green or Yellow. You have these values in the calibration data that you collected earlier. (See paragraph 5V - 'Reading Sensor Values in the Simulator'.)
- To prevent the robot from driving off the field if it misses the black line, you also need to keep count of how long the robot has been moving. In chapter 4 of Part 1, you learned how to use a timer to determine how long a robot has been executing a program. In this version, the robot is programmed to stop after two seconds. So, there is a good chance it will stop before the black line has even been detected.
- You need to change the program so that the robot detects the black line, but does not drive off the field. Add additional code to make the robot stop in time.

5C.1	Stopping on the Black Line	
1	 Delay = 2.0s Steering= 0 Speed = 50 RobotDrive; Delay = 2.0s Steering= 0 Speed = 50 RobotDrive	Load program 5B - Calibration and make a safety copy named 5C.1 – DriveBehavior.
2	 Delay = 5.0s Steering= 0 Speed = 50 RobotDrive	Go to the main program and remove all the icons except the ones shown on the left. The program must let the robot drive forward for 5 seconds max., but we also want the robot to stop on the black line.
3	 Delay = 5.0s Steering= 0 Speed = 50 RobotDrive; Field_L 50%	Instead of using a stopwatch, make the robot detect the black line. This can be done with the *WaitForLowerThan* icon. Replace the 50% value with the value you found for Black. Unfortunately, this approach will not work. It has two problems.
4		First of all, the robot will not find any color that is darker than black. However, as it moves over the black line, it will briefly see a color with a value that is in between that of Green and Black, as it is crossing over these two colors.

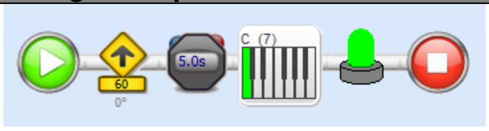
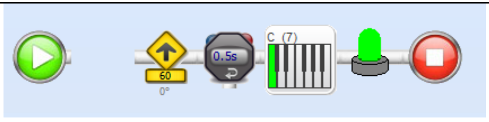
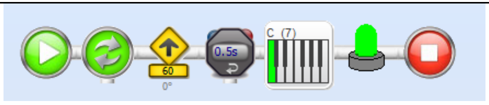
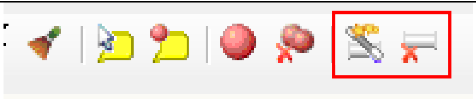
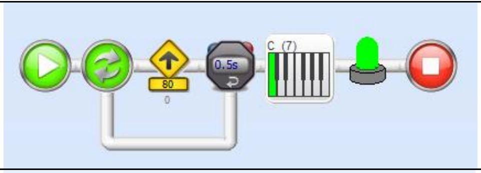
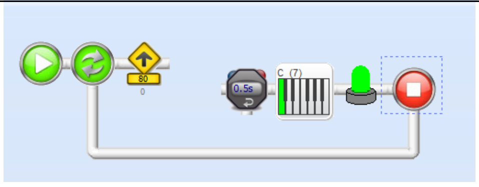
5	 44% 32% 19% 32% 44%	So you need to specify a value that is higher than the measured value for Black, but lower than that of Green. This will ensure that the robot always recognizes the black line. To avoid getting too close to Green, the best option is to use a value that is about 5% higher than that of Black.
6	 Properties Sensor Field_L Level 19% Margin 5% Lamp 1-On Type WaitForLowerThan	In the WaitForLowerThan Properties, there is a field called Margin. Insert 5% for the Margin so that RoboPAL will interpret the value for black as a little higher or lower than the measured value. All icons that use a value have such a Margin. If you do not specify a value, it will take the default value, which also happens to be 5%.
7	 Delay = 5.0s Steering= 0 Speed = 50 RobotDrive Field_L 50% Delay Speed Steering	This will make the robot react properly on the Black line. However, there also is another problem. As soon as the main program calls the subroutine, it makes the robot drive forward and then wait for 5 seconds. Then, the subroutine returns to the main program before checking for the black line. In the meantime, however, the robot will have passed the black line. How do we solve this?
8	 Delay = 0.00s Steering= 0 Speed = 50 RobotDrive Field_L 50% C (7)	If you set the Delay to zero, the subroutine will no longer wait and will return directly to the main program that checks for the black line. An even better idea would be to have the robot emit a signal when it detects the black line. So, test this program in the simulator and save your final version.
9		Change the value in the program, using the values you measured on the rescue field and then try the program on the NXT on the field.

Using Loops

We have already seen some programs that use a loop. Making a loop requires drawing lines, or pipes, between icons. We will first practice making a loop and then use this method in subsequent programs. You already did this in Chapter 2, but let's review it.

5D.1 Assignment: Using a Loop

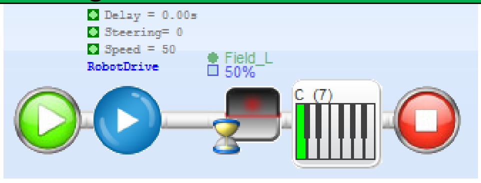
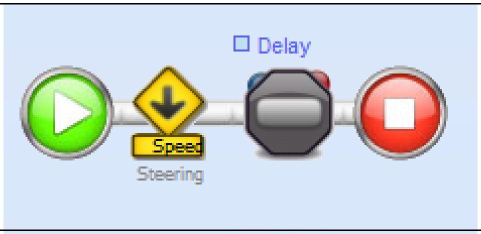
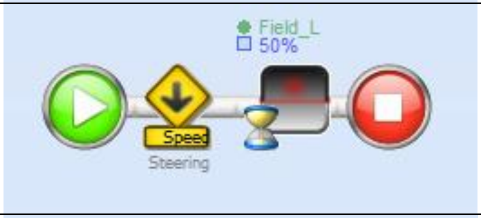
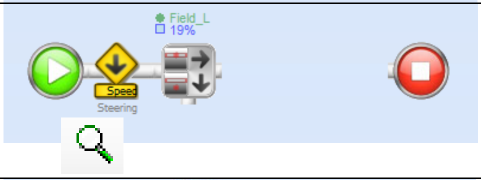
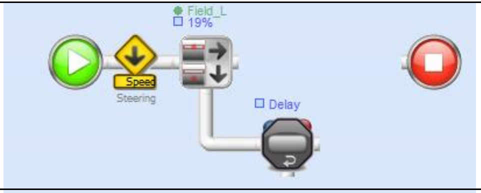
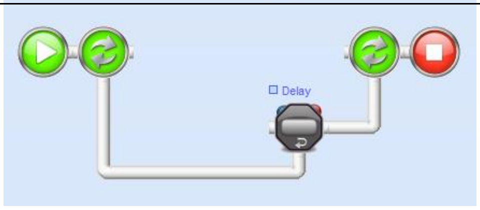
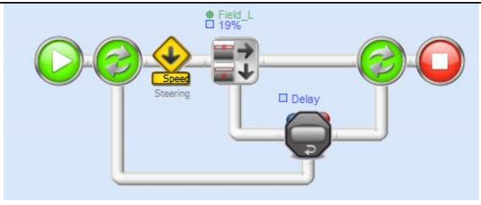
- At the beginning of this lesson, we showed you a picture of the Sense-Reason-Act Loop. A Loop or repeat construction is a program that repeats itself. As we have not used repeats yet, we will now look at how to insert a loop in a program.
- We are going to change *RobotDrive* by inserting a Loop. We will see why that is important in the following lessons.

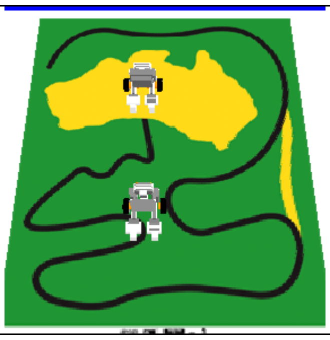
5D.1	Using a Loop	
1		First load 5D – Loops and save it as 5D.1 – Loops. In this program, the subroutine uses a stopwatch to wait for a given amount of time to pass. However, we have changed the stopwatch to wait for the black line.
2		A wait command, however, only makes the program wait. We would like to use this time to do something: for instance, to read the sensor values. Therefore, we will replace the StopWatch with a LoopTimer icon.
3		A LoopTimer icon allows the program to continue when the set amount of time has passed, but it also has an exit at the bottom that the program uses during the wait. We need to connect this exit to the beginning of the program. We use the Merge Icon from Program Flow to do this.
4		We have to connect the LoopTimer to the Merge Icon. The simplest way of doing this is to use the *auto-magic* Wiring Wizard. Use the Wiring Wizard and watch how the connection is made. The second icon in the icon bar lets you remove the automatic connection.
5		Wiring is inserted from left to right and from top to bottom. The Wizard looks for empty connections at the same level and connects these automatically.
6		If the icons are not at the same level, things can go wrong. Sometimes a program is just too complicated for the Wizard and you will have to make the connections manually with the Pipe cursor. This was discussed in Chapter 2 of Part 1.

Using a Time Limit

Suppose that the robot starts moving, but misses the black line. It will then drive off the field and bump into things or drive off the table. It is better to insert a time limit that prevents the robot from driving off the field. This is more difficult, however, because the program needs to consider two situations at the same time.
First of all, we need to check for the black line and at the same time we need to check that the robot is not driving longer than the set time. The robot needs to stop when one of these two situations occurs.
This kind of situation is found in almost all robotic programs, so we need to find a way to solve this problem.

5E.1 Assignment: Adding a Time Limit
We will extend the *DriveBehavior* program with a check for the maximum time that the robot moves forward to prevent it from driving off the field if it misses the black line.

5E.1	Adding a Time Limit	
1	 Delay = 0.00s Steering= 0 Speed = 50 RobotDrive Field_L 50% C (7)	Load program 5E – Time Limit and save it as 5E.1 – Time Limit. As the subroutine is going to check for the black line, we first need to remove this part from the Main program.
2		We avoided this problem in the previous assignment by eliminating one of the two conditions: the wait command. However, this is not a good solution if we want to include a maximum time limit.
3		While the robot moves, RobotDrive must continually check for the black line. The way we did that was to wait for the black line, so the program just stopped when it detected the black line.
4		Now, instead of waiting, we will check for the black line and if we do NOT see it, we will continue doing something else. We use the BranchOn icon to do this. Specify the value for Black and also insert a value in Margin.
5		If the BranchOn does NOT detect Black, we want to keep checking how much time has passed. We cannot do this with a StopWatch, since that would wait and block the other actions. Instead, we will use the LoopTimer from Program Flow.
6		If a given amount of time has not passed, the robot continues to look for the Black line and to do this it must return to the beginning of the program. We do this by inserting a Merge icon after the Start icon and connecting it with a LoopTimer.
7		We now have a program that executes commands within a loop, until a maximum time limit has been reached. As long as this loop executes, the program will continue to check for the black line. Once the black line is detected, the program exits the loop and stops. So, we are checking for two conditions at the same time. Try the program on the simulator.

8		First, put the robot on the yellow field and then start the simulator. It should now stop on the black line, when the stop condition becomes true. Now, put the robot further back on the field and start the program again. The robot will stop in front of the black line, because its time limit has expired. Check if this works fine, if not, change your program until it does.

Drawing a Graph

In the photo below, someone is making a measurement on the computer screen. We are not going to do this on the screen. We will measure the distance with the NXT on the field.

In the Rescue challenge, we use a can that is officially called “the Victim”. In the original Rescue game, we used a puppet that was supposedly drowning in the swamp. However, a reflecting object is easier to detect, so we now use a can (wrapped in aluminum foil) and continue to call it the victim.

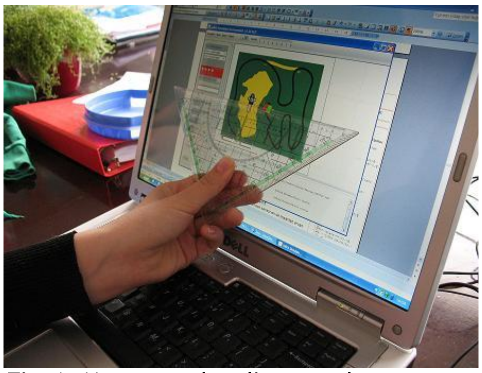


Fig 4: Measure the distance between the victim and the robot.

We will measure the values of the Distance Sensor at various distances from the robot.

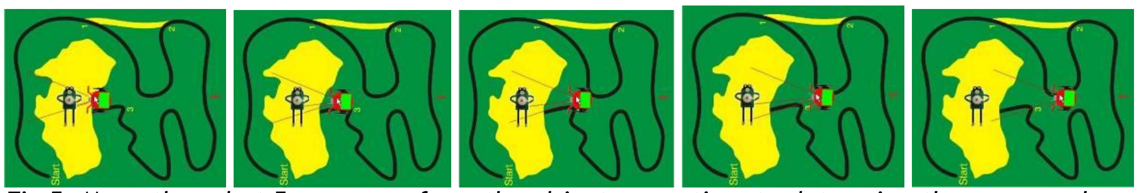
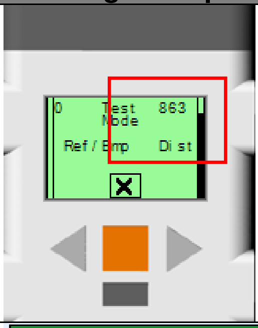
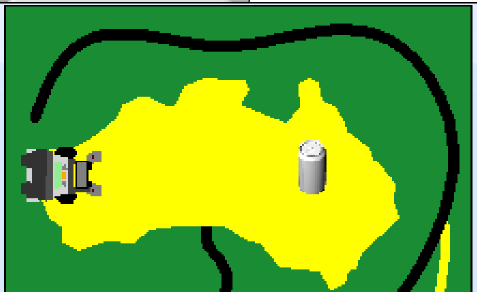
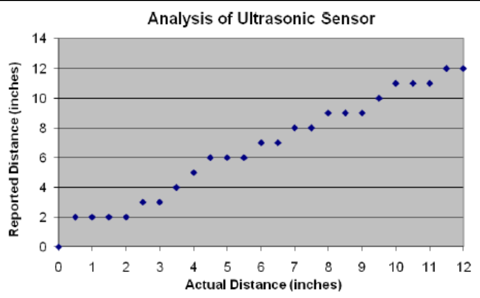
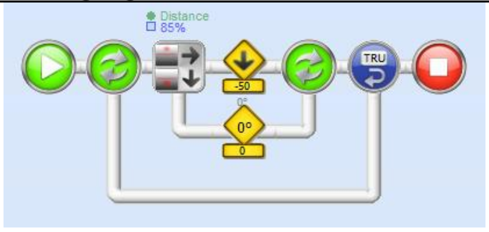


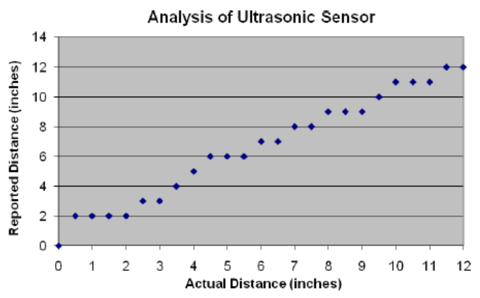
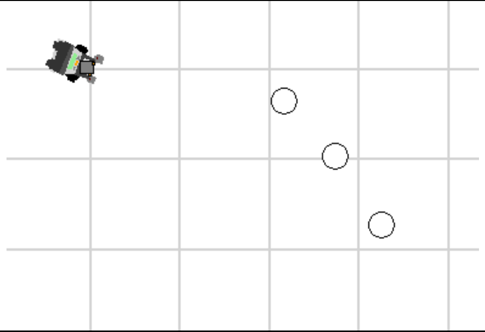
Fig 5: Move the robot 5 cm. away from the object every time to determine the sensor value.

5F.1 Assignment: Drawing a Graph
The sensor values on the simulator are only an approximation, so we will take measurements with the NXT to get an idea of the differences.

5F.1	Drawing a Graph	
1		Switch on the NXT. As we are going to use the distance sensor, press TST three times to obtain the screen that displays the Distance (Dist) sensor value. Of course your robot needs to have a distance sensor connected to port 3.
2		Place a can or some other object in front of the robot. Start at a distance of 5 cm and write down the sensor value. Then move the object another 5cm away and take note of the value again. Continue to do this until you reach the end of the yellow field (or a distance of about 50 cm if you are not using the rescue field).
3		Make a graph and plot the distance in centimeters on the X axis and the measured values on the Y axis. The graph will show you how to calculate the value given the distance and vice versa. You can also determine if the sensor characteristic is linear (in a straight line). If you see a nice straight line, it is easy to calculate the distance from the value. If the graph is not a straight line, the sensor is not linear and the formula will be much more complicated.

5F.2 Assignment: Changing CuriousBehavior
Once you have written down the sensor values, try to modify *CuriousBehavior* (the behavior that makes the robot drive forward when it detects an object)..

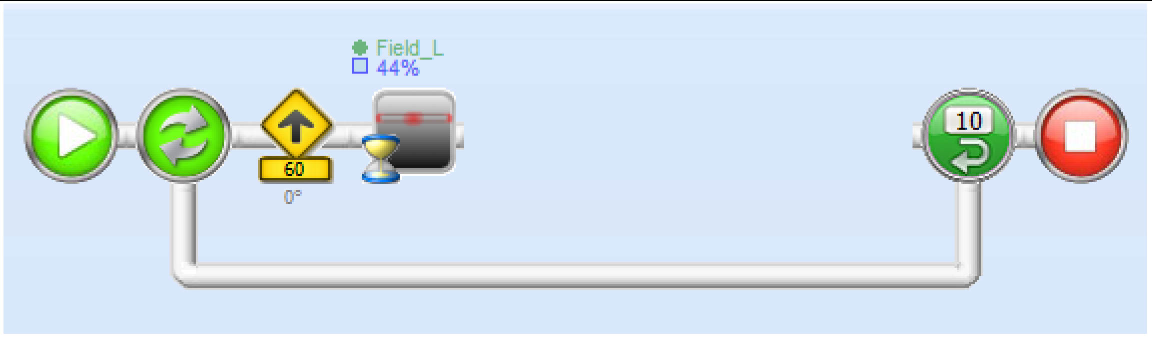
5F.2	Changing CuriousBehavior	
1		Load the program 5F – CuriousBehavior that you used before and save it as 5F.2. You have already used the distance sensor in this program. You can now use your table and graph to easily determine what value to insert.

2		In chapter 2, you experimented with distance and sensor sensitivity. You checked the field of view of the sensor to understand how far and how wide the sensor can detect objects. Now, combine both graphs so that you can see the distance vs. the sensor value in one graph and the sensor value vs. the width of the field of view in the other one. This will be a useful tool for the lessons that follow.
3		Change your program based on what you have found out about the sensor. Try to make your program do the following: use the right sensor value and to drive towards the first ball; however, when it gets close, it should go towards the other two balls, because they enter into its field of view. Save your program as 5F.2 – CuriousBehavior.

Mowing the Grass in the Swamp Area

5G.1 Assignment: in practice: Mowing the Grass

- Make a program that searches the swamp area in a back-and-forth manner by first driving forward to the green border, then making a slight turn backward and then forward again, so that each cycle covers a different part of the swamp.
- Use 5G - Search the Swamp as the basis for your program.
- Use a Loop. Use *WaitForMatch* to recognize the color of the field (leave the Margin at its default value) and the *LoopCounter* to create the loop. Look at the examples below to see what this looks like.
- Test your program in the simulator.



FlowCode 1

The pseudo code could look like FlowCode 2.

```
Turn off the leds
Repeat the following:
 Drive forward
 Until the left sensor sees a green area
 Turn on the green led

 Drive backward
 Until you are on the yellow field again
 Turn off the green led

 Drive backward while making a slight turn
 Do this until the robot points in a different direction

Repeat this procedure 10 times
Sound a beep
Stop the program
```

Code 2: Pseudo code searching the field.

5G.2 Assignment: Using a Subroutine

- Check your program to see if there are any parts that you can move to a subroutine. In general, you do this for parts that have a similar structure and only use different values for the parameters.
- Make the subroutine and pass the variables to it as parameters.
- Make sure that the Loop is created in the main program and that the subroutine only contains commands to make the robot move. You can copy and paste the program from *RobotDrive* to *Main* by selecting all the icons and then pasting them on the clipboard with Edit Copy. Then, go to *Main* and select Edit Paste and your program will now appear in both flow code sheets. Delete the parts that you do not need.
- Do not forget to specify that the starting program is *Main* in the robot properties or only the subroutine will be executed.
- Test your program in the simulator. If it works ok then test it on the NXT on the rescue field.

5.6 In Practice

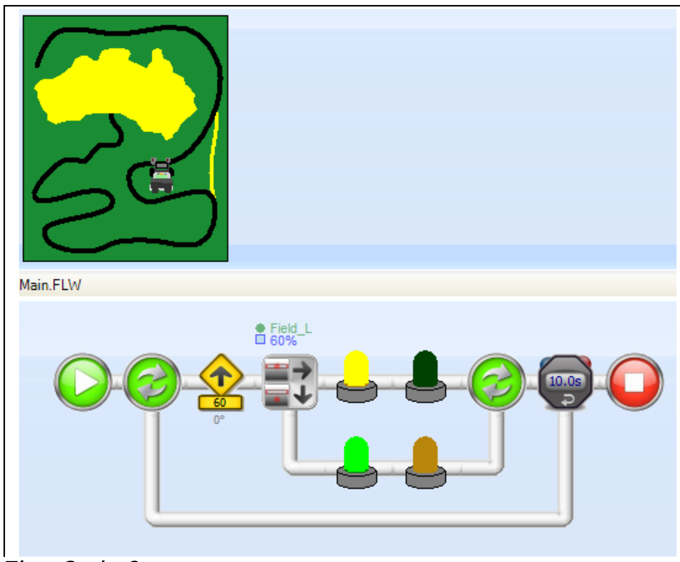
Fig 6: iRobot. Source: http://www.irobot.com.

In practice, we have robots that function in the same manner. You have probably heard about lawnmowers that operate autonomously while you comfortably lounge in a chair. The robot recognizes the difference between green (grass) and black (earth). There are also vacuum cleaners that clean the living room in the same way. They do not look at the floor, but look around using distance sensors to prevent them from bumping into the objects around them.

SO 5H: Test on Chapters 4 and 5
These chapters are completed by a written examination.

5.7 Test

1. The senses of living organisms look a lot like the sensors of robots, but there also are big differences. Provide some examples.
2. When do you have to calibrate sensors?
3. What kind of information is collected by robot sensors? Explain, for every sensor, what physical property is recorded and what values will be presented.
4. Look at the program below and explain what happens when you run this program and the robot is in the indicated position.



FlowCode 3

5. What happens to the program if you lower the value of the sensor?
6. And if you increase the value ?

6. Processing: the ‘Reason’ Phase

Once a robot has used its sensors, it needs to do something with these values. In the previous chapter, we made the robot stop, but of course a robot can do more than this with its sensors. In this chapter, you will learn how a robot can follow a line by using a sensor.

6.1 You will learn

- to read the values of robot sensors
- how to make the robot remember its calibration data
- how to control its motors
- how a line-follower works

6.2 You will need

- a NXT robot
- fully charged batteries
- a computer with RoboPAL
- a rescue field
- a *LineFollower* program

6.3 You will experiment with

- Using the calibration values of the robot in a program
- making a program that follows a line using a single sensor

Type	#	Assignment	Description
	6A	**Calibrate**	**Automatic calibration**
	6A.1	Calibrate	Automatic calibration
	6A.2	Calibrate	Using the calibration data
	6B	**Line-Follower**	**The first line-follower**
	6B.1	Line- Follower	A line-follower using a single sensor
	6B.2	Experimenting	Experimenting with settings
	6B.3	Testing	Testing on the robot
	6C	**Line-Follower**	**On the other side of the line**
	6C.1	Testing	What needs to be changed?

6.4 After following this chapter, you will be able

- to use a simple line-follower
- to read the sensor values of a robot
- to calibrate the field sensors of a robot

6.5 Explanation

The Sense-*Reason*-Act Loop

In the *Sense* step, the robot observes its environment. Then, it needs to determine what to do with that information. This is called the *Reason* or processing step and it is followed by an appropriate action, the *Act* step.

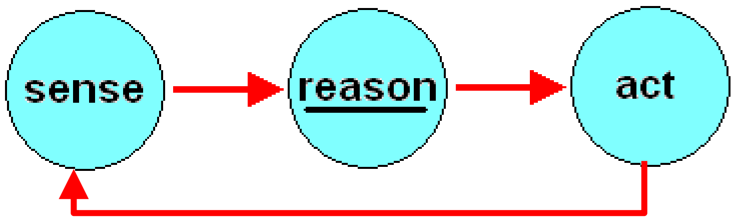


Fig 7: Sense, Reason, Act Loop.

In the *Reason* step, the information from the environment is processed and used to perform a sensible action.

Reading the Robot Sensors

You have already read the sensor values several times and written down their values. However, it is much simpler to let the program store these values by itself, so that you do not have to make changes to your program every time a reading changes. We will do this through a new calibration subroutine.

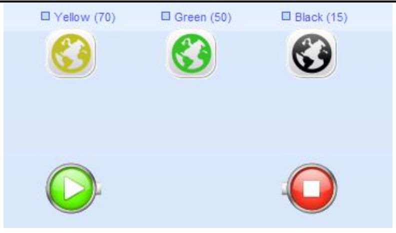
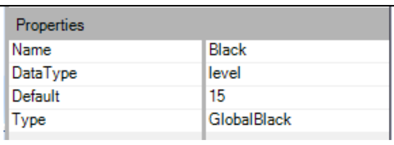
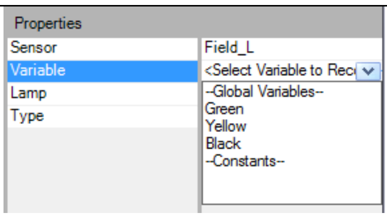
Automatic Calibration

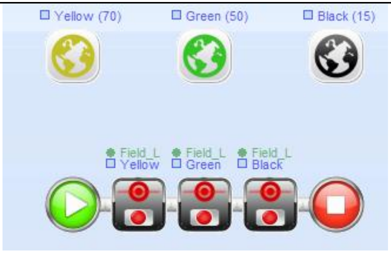
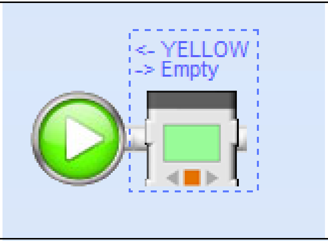
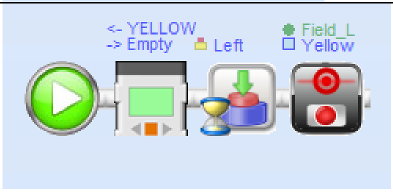
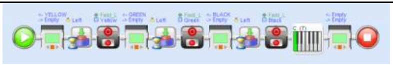
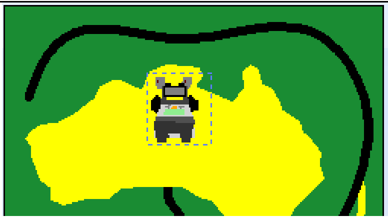
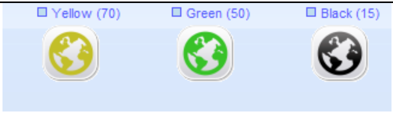
Your robot recognizes the color black by making a comparison. It checks if the value is much lower than the value for green. You should never try to make the robot recognize the exact value you measured for black. Even the smallest fluctuations caused by shadows will prevent the robot from reacting as expected. Keep in mind that when a sensor is halfway between green and black, the sensor registers a value that is in between these two values.
If you want to detect the black line reliably, you should use a value that is lower than green, but higher than black. The same applies for yellow. In this case, however, we will use values that are higher than green and lower than the measured value for yellow to keep a safety margin.
Safety margins are very important and determine both the precision and the speed at which a robot reacts. RoboPAL keeps a standard margin of 5%,

but you can change this value by selecting the Perspective tab in the startup menu (See Chapter 3A, Part 1).
You can also change the margin for every icon that reads a sensor value. If you specify Default in the Margin field, it will use the value that you inserted in the Perspective, but you can override this if you temporarily want to use a different value.

It is also important to take the ambient lighting condition into account. If, for instance, your robot is directly under a lamp or close to a window, the light that shines on the field and/or shadows can influence the sensors. Sensors detect shadows as darker areas than the parts of the field that are well lit. So, it is important to let your robot measure the sensor values from different angles.
Nonetheless, repeatedly measuring values and changing your program takes a lot of time and if you happen to forget to change one of the values, your program will no longer work. So it is much easier to make the robot do this by itself. Therefore, we are going to develop a program that calibrates the sensors automatically.

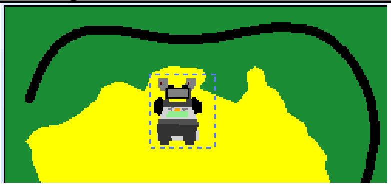
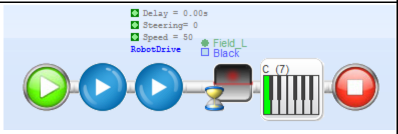
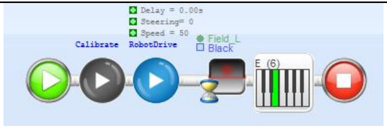
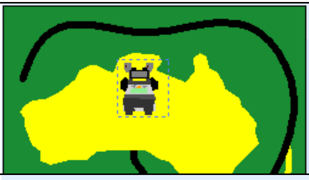
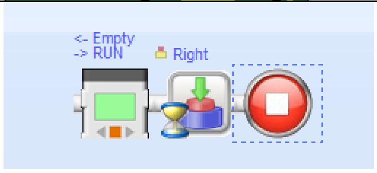
6A.1 Assignment: Automatic Calibration
Instead of writing down the sensor values and modifying your program, make the robot do this.

6A.1	Automatic Calibration	
1		Load program 6A - Calibration and save it as 6A.1 – Calibration. Make a new FlowCodeSheet and call it Calibrate. Remove the Driver icon in the middle. Also make three global variables for Black, Green and Yellow. Use the Global Green, GlobalYellow and GlobalBlack icons from Variables.
2	 Properties Name: Black DataType: level Default: 15 Type: GlobalBlack	We have used Variables before, but here we will use Global variables. This means that we can use them anywhere in a project. The variables that we used before as parameters in a subroutine are called Local variables. They are only valid within a subroutine and do not exist outside of the subroutine. Variables also have a DataType, like Time or Int. In this case the variable are of the type “Level”.
3	 Properties Sensor: Field_L Variable: <Select Variable to Rec Lamp Type --Global Variables-- Green Yellow Black --Constants--	Use the Record icon from Sensor Flow. This icon reads a sensor value and stores the measured value as a variable. In Properties, you can specify in which variable to store the value.

4	 Yellow (70) Green (50) Black (15) Field_L Yellow Field_L Green Field_L Black	As you need to do this for each color, you will need three icons. The robot has to be placed directly above the corresponding color, so we need something to indicate that you have to place the robot in a specific place.
5		Use the LCDMessage icon from Lights and Sounds to insert text under the left and right buttons. Select YELLOW from the Properties. This is a request to put the robot on the yellow area of the field. We also need to know when this action has been completed.
6		So, we have the program wait until the left button is pressed. This is the sign that the robot is correctly placed above the yellow area. Use the WaitForButtonPressed icon from Touch Button Flow and select Left in the Properties.
7		Read and store the other two colors in the same manner. At the end, have the robot sound a short beep and make sure that the last message is erased from the LCD screen. This completes the calibration program.
8		The last step is to test the program with the simulator. Make sure that the Calibrate subroutine is the startup program. In this way, you can test only this single subroutine, bypassing the Main program temporarily. Make sure it works.
9		You can now use the variables Yellow, Green and Black anywhere in your program, rather than having to measure and write down the values every time. The Margin is automatically set to 5% and if you want to use a different margin, you can specify it when you use a sensor. Save your program.

6A.2 Assignment: Using the Calibration

- You have now stored the sensor readings, but you must also start using them in your program. You need to do a number of things to this.
- Call the Calibration subroutine in your main program, but make sure this only happens the first time, when you start the robot.
- The values that you have stored must also be used in the program. So, you have to change all the icons that use color values. From now on, you will use variable names for colors rather than numbers.

6A.2	Using the Calibration Data	
1		Save the program as 6A2 – Calibration. Select your robot and choose Main as your startup program in Properties.
2	 Delay = 0.00s Steering= 0 Speed = 50 RobotDrive Field_L Black C (7)	Drag Main to the FlowCode sheet. Insert a new Subroutine called Calibrate. Give the icon the color Black (or another color) to show that it concerns a different subroutine. Also change the note that the program plays when it is finished, so that you will hear a different sound from the one in the calibration.
3	 Delay = 0.00s Steering= 0 Speed = 50 Calibrate RobotDrive Field_L Black E (6)	Test your program in the simulator and check that it reacts properly to the black line, but … OOPS! We have a problem. The robot is still on the black line. It will start moving and miss it.
4		We want the robot to wait until we have placed it at the starting point.
5		We need to change the calibration program to do this. Change the last part of the program so that a RUN message appears on the screen and the program waits until the right button is pressed. Now try again, but first put the robot back on the yellow area.
6		If this works, try to do the same with the NXT and make sure it works on the rescue field.

A Line-Follower with a Single Sensor

6B.1 Assignment: a Line-Follower with a Single Sensor

- You are going to make a program that reads the left sensor every time a *behavior* is called. If this sensor has a higher value than that of black, the robot moves to the right, but if it detects black it moves to the left. This will make the robot zigzag on the border of the black line. As we are using the left sensor, the robot will follow the line on the left side (as shown in figure 7).
- We will be using the calibration data measured in assignment 6A.

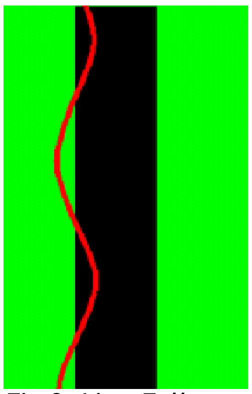
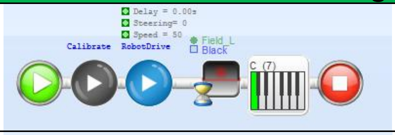
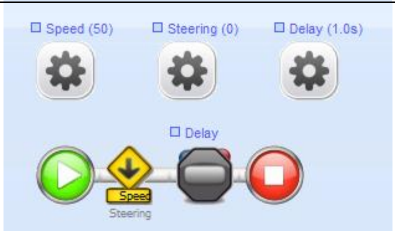
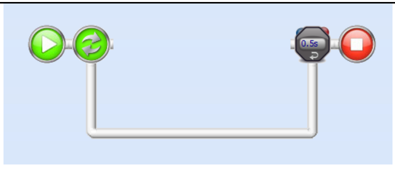
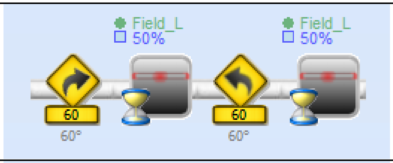
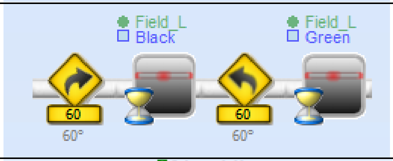
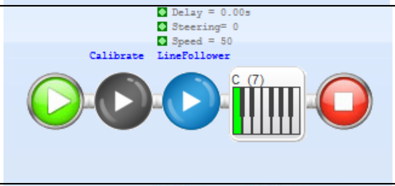
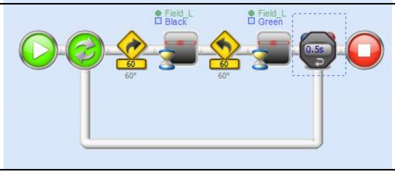


Fig 8: Line-Follower with One Sensor

6b.1	A Line-Follower with a Single Sensor	
1	 Delay = 0.00s Steering= 0 Speed = 50 Calibrate RobotDrive Field_L Black C (7)	Load program 6B – Line Follower and save it as 6B.1 – Line Follower. If you completed assignment 6A, start with 6A.2.
2	 Speed (50) Steering (0) Delay (1.0s) Delay Speed Steering	We are going to transform RobotDrive into a line-follower. Select the RobotDrive in the flow code from the Project Browser in the upper right-hand corner and press the right mouse button. Rename it as LineFollower. Make sure you save the program first since renaming is only allowed on programs that have been saved to disk. Now, drag LineFollower to the FlowCodeSheet.
3	 0.5s	As described in the assignment, the robot needs to move away from the line if the sensor detects Black and towards it when it detects Green. So, we need to modify and expand the program. First, we have to insert a loop that will allow the robot to follow the line for a set time.
4	 Field_L 50% Field_L 50% 60 60° 60 60°	We must make the robot turn right until the color of the field changes. Then, the robot must turn left until the field color changes again. All we need to do is specify which colors these are with the WaitForMatch icon.
5	 Field_L Black Field_L Green 60 60° 60 60°	To perform the comparison, we use the calibrated colors and therefore change the percentages values to the Black and Green variables.
6	 Delay = 0.00s Steering= 0 Speed = 50 Calibrate LineFollower C (7)	Be sure that the robot's starting program is the Main program so that it calls Calibrate first and then starts the line-follower program. Save your program and start it in the simulator.
9	 Field_L Black Field_L Green 60 60° 60 60° 0.5s	It seems to work, but the program stops almost immediately. This is because the LineFollower is using the default time of 0.5 sec in the LoopTimer. We will change this in the next assignment.

Tuning the Line-Follower

If the robot does not have enough time to detect the black line, it will miss it, ending up on the wrong side of it. So, make sure that the line is wide enough and that the robot is not moving too fast. This will also happen if the frequency with which *behavior* is called is too low. If the Sense-Reason-Act loop has a lot of work to do, the call frequency will get lower and the program will become slower. This will not happen in our examples, but as a program grows larger this risk increases.
If the robot is driving too fast, the black line will only be visible for a short time before the robot passes over it. So, experiment a little with your program to see if the robot works reliably.

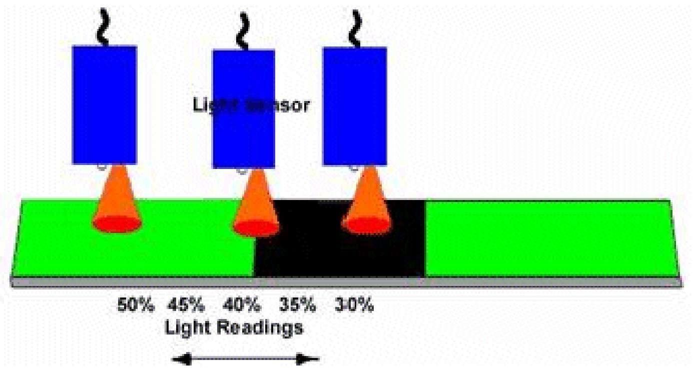
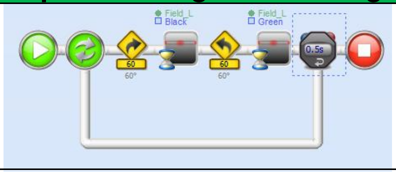
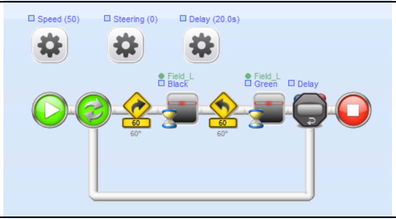
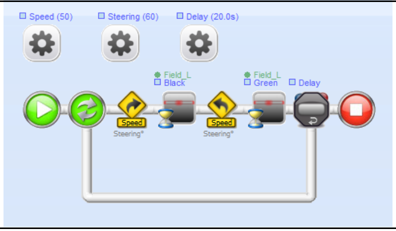


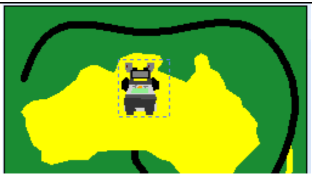
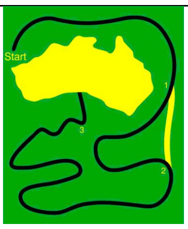
Fig 9: Measurements of the Light Sensor

Moreover, there also is another phenomenon that we discussed earlier. If the sensor reaches the border of the black line, it will detect both black and green and return a value somewhere between these two colors. This value can actually be used to make the robot react more quickly.
By making the value in your program slightly higher, you are actually making the line a little wider (by including the transition between black and green as part of the line). However, if you make this value too high, it will start reacting to green, too. So it is important to find the best value. In most cases, increasing the value by 3-5% is sufficient. You define this correction factor in the Margin property.

6B.2 Assignment: Experimenting with Settings

You are going to make the robot follow the line by using one sensor. The sharper the turns in the road are, the harder it will become for the robot to follow its path. In addition, following a black line on a green background is clearly more difficult than following a black line on a white background.

6B.2	Experimenting with Settings	
1		Save your program as 6B.2 – Line Follower. The first thing we will do is change the timing. You could just change the time in the LoopTimer, but it is better to use a parameter.
2	 Speed (50) Steering (0) Delay (20.0s) Field_L Black Field_L Green Delay 60 60°	Change the time of the Delay variable in the LoopTimer. Set its default value to 20 sec. You may use the time that the line-follower continues to follow the line as a parameter in the call from the Mainline. If you do not change the value in the call, it will assume a standard value of 20 seconds.
3	 Speed (50) Steering (60) Delay (20.0s) Field_L Black Field_L Green Delay Speed Steering°	We can also change the speed and angle parameters. The higher the speed, the larger the chance that the robot will not follow sharp turns, so a larger value for the angle will improve tracking on the turns. Change the settings of all Driver icons into variables.

4		If you have modified the Speed and Steering and given the variables a default value, save your program and try it out. Please note that the parameter values in Main are not automatically updated with the new defaults. So check that their values are correct.
5		After 20 seconds, the robot will stop following the black line, as indicated.

6B.3 Assignment: Testing on the Robot
Experiment with the various settings. Check how you can make the robot move faster. Watch what happens if you set the Steering to a higher or a lower value. Try to make the robot follow the entire track. Also try this out on the Rescue field.

Driving on the Other Side of the Line

If you put the robot on the other side of the line, you will find that the line-follower no longer works.

6C.1 Assignment: What Needs to be Changed?

- Start the simulator with program 6B.2
- Put your robot on the right-hand side of the line
- Try your line-follower
- On that side of the line, the robot does not work
- Find out why
- Change your program to follow the line on the right-hand side
- Make another change to your program so that the green led is lit when the robot detects green and the blue led is lit when the it detects the black line. This is not very easy to do with the current program, because a WaitForMatch icon is being used. So, change the program to use a BranchOn. This is a rather significant change to the logic of your program.

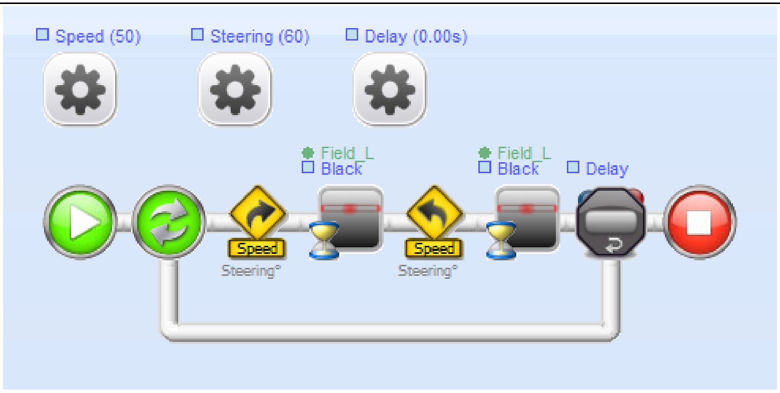
6.6 In Practice

In large warehouses and container terminals, goods and/or containers are transported by autonomous robots. One of the ways in which this is done is to install a metal guide wire in the ground. The robot sensors detect this wire and follow it. If there is an some obstacle on the way, the robot needs to know how to avoid it.

Fig 10: Automatic Guided Robots. Source: http://www.wampfler.com/index.asp?id=75&e1=11&e2=31&kgref=127&lang=E

6.7 Test

1. Suppose you read the following sensor values on your robot: `Black 200, Green 280, Yellow 420`. Explain what values you will use in the line-follower flowcode for the color black.
2. What is wrong in FlowCode 4?



FlowCode 4

3. What will happen if you do not change this?
4. Under what conditions can something go wrong with a line-follower?
5. What can you do to resolve this?

7. Actuators: the ‘Act’ Phase

In this chapter, you will be developing most of the code on your own. The robot can follow a line with its sensor, but the zigzag behavior slows it down. The motors need to be controlled in a better way to increase the robots speed and improve its ability to follow the track.
So, we have to improve the “cooperation” between the Reason and the Act phases. We are going to make the line-follower a bit more complex and use two sensors. In addition, we are going to make the line-follower behavior more flexible.

7.1 You will learn

- what type of actuators exist
- how actuators are controlled by information from the reason phase

7.2 You will need

- a NXT robot
- a computer with RoboPAL
- the rescue field
- the *Line-Follower* program

7.3 You will experiment with

- controlling motors based on data from the reflection sensors
- using a line-follower with two reflection sensors

Type	#	Assignment	Description
	7A	**LineFollower**	**A faster line follower**
	7A.1	LineFollower	A line follower with two sensors
	7A.2	Experiment	Testing in the simulator and on the NXT
	7B	**LineFollower**	**Following the black line**
	7B.1	LineFollower	Following the line using a subroutine
	7B.2	Testing	Following the black road
	7C	**LineFollower**	**Following the yellow road**
	7C.1	LineFollower	Following the yellow road
	7C.2	Testing	Testing YellowLineFollower
	7C.3	Testing	Following the rest of the trajectory
	7D	**Exam**	**Exam on chapters 6 & 7**

7.4 After Completing This Chapter, you will be able

- to explain how a line-follower with two sensors works
- to use information from the sensors to control the motors
- to explain why a line-follower with two sensors makes the robot move faster

7.5 Explanation

The Sense-Reason-*Act* Loop

Living organisms continually collect information. First, the *Sense* step takes place, then information is processed (the *Reason* step) and this results in a meaningful action (the *Act* step).

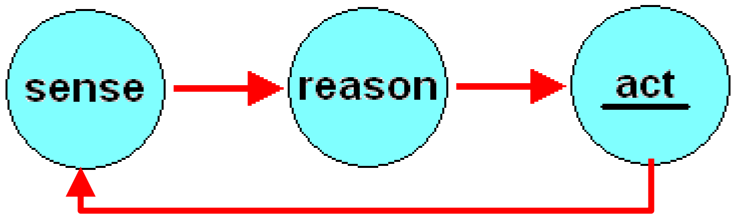


Fig 11 Sense, Reason, Act Loop

Robots process information from their environment and then control one or more actuators. In this chapter, we will see how actuators are controlled during the Act step.
Actuators are the parts of a robot that take care of movement. They usually are electric motors; other types include pneumatic or hydraulic actuators, linear motors, memory shape alloy artificial muscles, etc. The Lego NXT only uses electric motors.

Servomotors

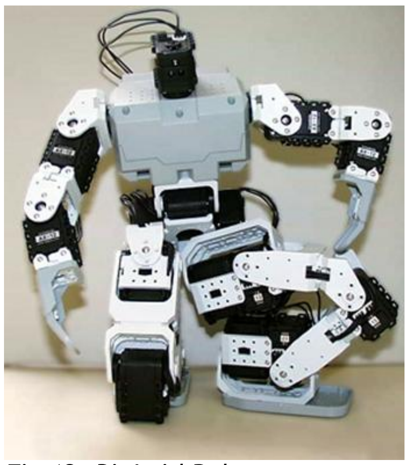


Fig 12: BioLoid Robot Source:http://www.robotis.com

Servomotors are often used in simple robots, because they are inexpensive and all the mechanic and electronic parts are nicely integrated into a single standard package. They are often used to control the steering of model boats, airplanes and cars.
In a walking robot, they are used as a joint to move the arms and legs.
On the NXT, small motors are used to power the wheels. The motors of the NXT can both be used as standard motors or as a servomotor. The difference between a servomotor and a standard motor is that if you use a servomotor you will need to specify the position of the motor shaft to control steering. Without a servo mechanism, the motor just turns and you will have to count how long it is turning to make it reach a certain position. This becomes very inaccurate as a result of slippage and braking.

A servo mechanism actually measures the position of the shaft and a feedback mechanism allows the motor to stop automatically when the required position is reached. This feedback mechanism makes the motor turn faster at the beginning and slow down as the desired position is about to be reached. This prevents the motor from overshooting the desired position due to high speed.

This is a very different way of using a motor than that we do with our robot. A servo mechanism is more useful, for instance, to steer a model car as you want the front wheels to be put precisely in a preset position. The motor in such a servo has a high rotational speed to obtain a large torque and enough power to move the shaft. And this is exactly what we want to achieve with the driving motors of the robot: sufficient power to move the robot, but with the possibility of changing the speed and direction of rotation of the wheels. So, we sometimes use modified servo motors in which the feedback is “misused” to control the rotational direction and speed of a motor. You can instruct the program to use the motors of the NXT in both ways.

Rotational Direction of the Motors

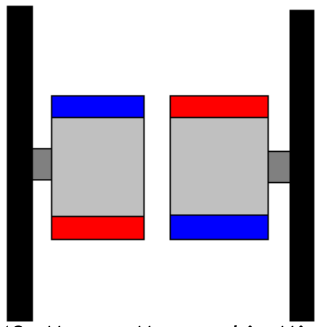


Fig 13: Motors Mounted in Mirror Image

Our robot has two motors. As they are mounted on either side of the robot, we obtain a “wheelchair” effect that is usually referred to as a ***‘differential drive’.***
Unfortunately, this name causes some confusion with the ‘differential’ used in automobiles.
The advantage of a wheelchair drive is that the robot can rotate on its own axis, making it rather agile. Also, sharp turns can be negotiated without a complicated steering mechanism such as that used in cars.

As the robot is powered by two motors that are mounted in a “mirror image,” one motor must spin forward and the other in reverse to make the robot move forward. Clearly, this may cause some confusion. RoboPAL is preset to take care of this and avoid confusion with the Driver icons. You have already encountered the driver functions in previous chapters.

Adaptive Behavior

To make the robot move forward, we use the Driver icons that have a predefined motor speed: 100 means full speed ahead, -100 (minus) means full speed backward and 0 means stand still.
However, these easy-to-use motors have a disadvantage. They are never exactly equal, which means they do not rotate at exactly the same speed. Thus, when both motors are given the same speed command, the robot will not actually drive forward in a straight line.

If a robot uses its sensors to move forward in a straight line, this does not pose a problem. It will use its sensors to determine where to go and automatically compensate for small differences in its motors. We will frequently encounter this kind of feedback mechanism. Feedback is one of the most important properties of autonomous (independent) robots. We call this ***adaptive*** (self-changing) behavior. The third part of the Robotics module is entirely dedicated to this behavior and contains a number of assignments to become acquainted with it.
What is interesting, in this case, is the difference in behaviors resulting from the intrinsic and extrinsic environment. The intrinsic environment, for instance, includes differences between motors, motor temperature and battery level, as well as sensor readings of the internal state of the robot. In humans, we call this ***proprioception*** or the ability to experience the position of one's own body parts (i.e., knowing the position of your arm or leg).
The extrinsic environment includes obstacles, color differences on the surface and the temperature of the environment. So, anything that can be detected with sensors outside of the robot. In humans, we call this ability to detect external stimuli through our senses ***exteroception***.
The LCD screen of your robot can be used to tell something about the intrinsic state of the robot. A led for instance can be used to indicate the battery power level. Another example is the so-called heartbeat-led that indicates that the robot is 'alive'.

Leds as Indicators

Leds are really not actuators, but indicators. They kind of fit in with the Act-part, because they deliver a form of *output*.
It is useful to know that your robot sensors detect something as dark. You can use one of the leds for this or display a short message on the screen.

The Assignment

A Line-Follower with Two Sensors

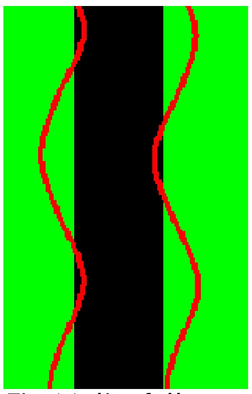
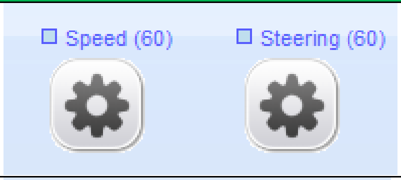
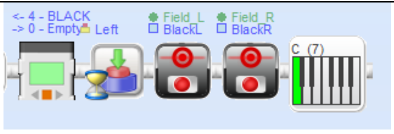
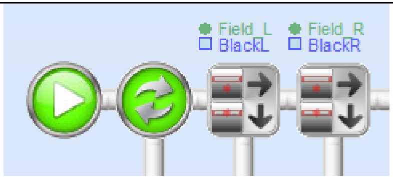
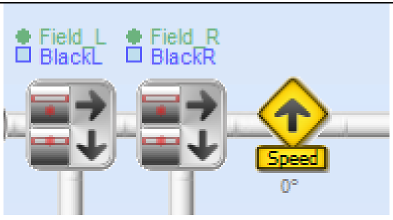
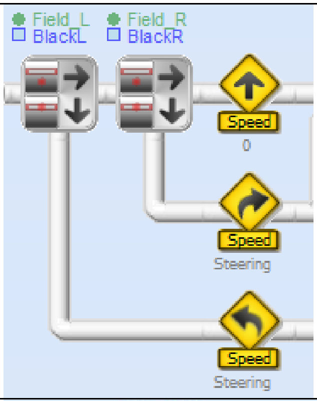
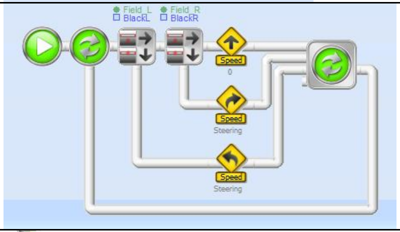


Fig 14: linefollower with two sensors

A line-follower with a single sensor is easy to program, but it is not very fast. The robot is slow because it has to make small zigzag movements to follow the edge of the line and must therefore limit its speed to not overshoot the line. Moreover, the robot cannot distinguish between leaving the line on the left or the right side and therefore has to stay on one side of the line.
By using two sensors, we can limit this zigzag behavior. This is clearly visible when following a straight line. The robot moves in a straight line until one of its sensors detects the line. The correction of the motor speed and decision of which motor to slow down will depend on which sensor detects the black line. This makes the robot move much faster than if it only uses one sensor.

7A.1	A Line-Follower with Two Sensors	
1		Open 7A – Fast Line Follower and save it as 7A.1, then open the LineFollower FlowCodeSheet. We will modify the program to use two sensors. Start by creating two local variables - Speed and Steering - both with a default value of 60 (int).
2	 <- 4 - BLACK -> 0 - Empty Left Field_L BlackL Field_R BlackR C (7)	We have to read the values of the right and left sensors. As we saw earlier, we should have stored the comparison value in the calibration process. So far, we have only used the left sensor. So, first change the calibration so that both sensors are read and their values stored in the variables BlackL and BlackR. We will change the other colors later on.
2		In this case, we read the sensor values in LineFollower with the BranchNotOn icon. This allows us to put the action underneath the icon and keep the program readable (so we use the inverted version of BranchOn). Start by reading both sensors and using the BlackL and BlackR global variables from calibration.
3		You will see that if the left sensor does NOT detect the black line, the program checks for the black line with the right sensor. If the right sensor also does not detect it, the robot continues to drive straight ahead. Add this to the program. We are using Speed as a variable, while Steering should be 0 to drive straight ahead.
4	 Field_L BlackL Field_R BlackR Speed 0 Speed Steering Speed Steering	If the robot detects the line with its right sensor, it should turn right; when it detects the line with its left sensor, it should turn left. This can be added easily. We also use local variables for Speed and Steering to specify how sharp the turn away from the line must be and the Speed at which it acts. Now, we must connect these three wires into the loop.
5		Use the Merge4 to do this. Note that one wire will remain unconnected. Connect the exit of Merge4 to the merge at the beginning to complete the loop. As the loop never ends, there is no Finish icon. It also means that the robot will not stop and you will have to switch it off by using the reset button.
6		Try the program out in the simulator and then on the Rescue field with the NXT. You will find that the robot loses the line at the first sharp turn. We will deal with this later on.

7A.2 Assignment: Testing in the Simulator and on the NXT

- Notice that the robot misses a turn once in a while. You could lower the speed, of course, but the real goal is to make the robot react as quickly as possible. So, you will need to make more changes to your program.
- These improvements will not be treated separately. You will have to discover what to do on your own. A hint: by changing the turning angle of the robot the turns can be negotiated much better.

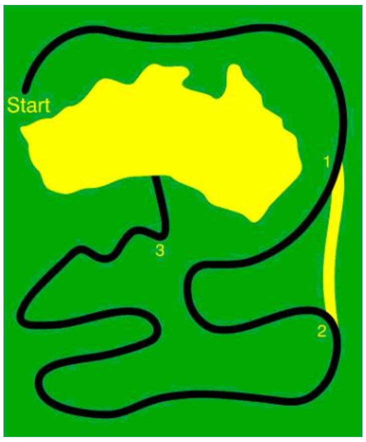
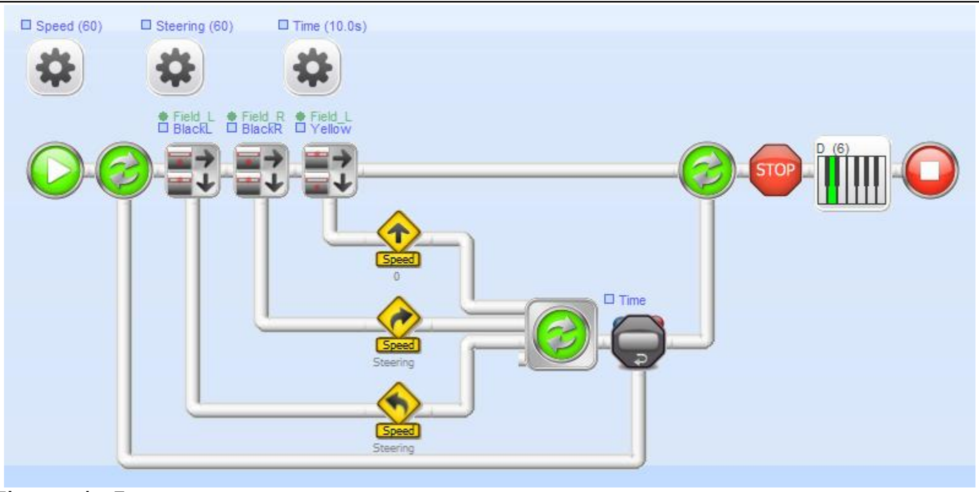


Fig 15: Rescue Field

Following the Road by Using a Stop Condition

After following the first part of the road, there is a yellow road that can be used as a shortcut. First, the robot needs to find the yellow road and then react to this different color.
We saw earlier how using a subroutine can make a program easier to understand. A subroutine can take care of following the line as well as adapting to the color change.
While following the black road we look for values that are lower than the surrounding green field, but with the yellow road we need to check for values that are higher than green.
Although these two steps could be combined into a single subroutine, it is much simpler to make two separate ones. Moreover, we also want to include a stop condition to indicate when the line-follower has reached the end of its trajectory (i.e., with the black road it is the yellow area and with the yellow road it is the black area). Either may be on the left or on the right, so we must use separate values. The code for this is shown in FlowCode 5.

Note that Speed and Steering are passed as parameters, but a Time parameter is also used to define how long the line-follower must move forward. This means that if the robot misses the stop condition, the timer will make sure it stops after a set period. The color of the stop condition is defined in the subroutine itself.



Flowcode 5

This part of the assignment requires further insight, because you have to develop the code on your own. The idea is to create a new subroutine called YellowLineFollower. You can create it by using the example above. You also have to modify the LineFollower for the black line so that it has a stop condition, but also stops if it detects the yellow line. Finally, you also have to change the calibration routine to register the color yellow.
To help you along, we will explain what needs to be changed in your program.

- First, you must make sure that the line-follower has a stop condition. You do this by including a third Branch command, but this time a BranchOn that tests if the left sensor detects yellow. For the yellow line-follower, the right sensor must test for black.
- If this is NOT the case (lower exit), the robot must continue to drive straight ahead as in the previous version. If it DOES see yellow, the robot should stop and sound a beep.
- We also want the robot to stop after a while as it might miss the yellow line. You do this by including a LoopTimer in the loop that gets its value from the Time parameter. As soon as that amount of time expires, the program will automatically stop. This gives the line-follower two stop conditions, one of which it will always react to.

The following assignments look at how to fit all of this into your program.

7B.1 Assignment: Following the Line by Using a Subroutine

- Load program 7B – Yellow Line Follower and save it as 7B.1 – Yellow Line Follower.
- Complete the black line-follower by including two stop conditions.
- Make sure that the black line-follower is called from the mainline and the calibration is updated.
- Make sure your program follows the black line and stops when it detects the yellow line.
- The framework of the code is shown in FlowCode 5

Following the Black Road

Finish the program so that the robot can follow the black line. Make sure that the robot stops when it detects the yellow line. Test this part well, because you will be using the black line-follower throughout the rest of the mission. It is important that you make sure the line-follower reacts properly to both stop conditions. During testing, temporarily set the timer to a lower value, so that you can check if it stops before detecting the yellow road.

7B.2 Assignment: Following the Black Road

- Test *LineFollower* with the simulator and check that the robot stops at the yellow line. If this works fine, test it on the NXT.
- Check that your robot stops before reaching the yellow line, if you decrease the time value. You will need this later.
- Use different tones to find out where and when the line-follower stops.

Following the Yellow Road

The goal is for the robot to start following the yellow line as soon as it is detected. To follow the yellow line, the sensors need to detect yellow. So this means modifying the calibration so that both sensors can detect the value for yellow and including the YellowL and YellowR variables in your program. The stop conditions also have to be changed, because this line-follower must stop on black.
After having followed the yellow line, the robot must continue following the black line again in the proper direction. This can be improved by lowering the speed after the straight part of the road, just before the sharp turns. You will have to count how long the robot can move at high speed and then modify the behavior of the line-follower.
Problem: after the robot detects the yellow line and starts following it, it must stop following it as soon as it detects the black line again. The field is designed so that the robot still detects the black line at the beginning of the yellow line. This makes the robot think, erroneously, that it has already reached the end of the yellow line.
Make sure that the robot does not interpret this first part of the black line as a stop condition.

7C.1 Assignment: Following the Yellow Road

- Save your program as 7C.1 – Yellow LineFollower
- Make a copy of the black line-follower and call it *YellowLineFollower*.
- Make the necessary color and stop condition changes to *YellowLineFollower*.
- Make the necessary changes to the calibration subroutine, so that it can detect yellow with both sensors.
- Call the subroutine *YellowLineFollower* from the Main program.

- Make sure that the stop condition for the end of the yellow line is detected properly. (The problem described above occurs when you are following the yellow line.)
- Use beeps and maybe leds or a message to help you follow what happens.
- Complete the code and check that it all looks OK.

7C.2 Assignment: Test YellowLineFollower
Test *YellowLineFollower* in the simulator and check that the robot stops at the black line after following the yellow line. If this works fine, test it on the NXT.

The Rest of the Trajectory

Finish the program, so that your robot is able to follow the rest of the trajectory. The first part after the yellow line can be followed at high speed. Then, count how long the robot takes to reach the next turn, just before the sharp turns begin. At that point, make sure that the black line-follower slows down, so that it can follow the turns.
Especially in the final part, it is hard to keep the robot on the track. In fact, it may be easier to just use a single sensor for this final part.

This is the final assignment for the second part. It serves to find out how much you have learned from these lessons on robotics.
Show how well your robot can follow the trajectory. When grading your work, your teacher will check both how your program was built and what it does. The faster the robot follows the road, the better you have succeeded in completing the mission.

7C.3 Assignment: Following the Rest of the Trajectory
- Save your program as 7C.3 – Rescue Line Follower.
- Complete the program so that it follows the rest of the trajectory.
- Test Rescue Line Follower in the simulator and check that the robot detects the end of the yellow line properly.
- Make sure the robot can follow the first part of the black line at high speed.
- Slow down the robot so that it can negotiate the sharp turns.
- Increase the Steering here; if this does not work, try a line-follower with a single sensor.
- Make sure the robot stops when it reaches the yellow swamp.
- Test the program on the rescue field, too.

7.6 In Practice: the RoboCup Competition

If you have finished the program and it works correctly, you can move on to finding the can in the swamp and having the robot push it out. If you can do this, you might just be ready to participate in the RoboCup Junior Competition, which is held annually in Rome (March/April). More details are available at www.mondodigitale.it.
You can participate in these competitions with your Lego robot, but you will have to design your own robot. You cannot use the example robot that was built for these lessons. You can find additional educational material at the same website. Check if the competitions in Rome also include the green Rescue field we used in this course. In international competitions usually a more complicated field is used on which a black line on a white floor is used, but with corridors where no lines are present. There you may have to follow the wall instead of the line.

Exam 7D: Exam on chapters 6 & 7
These three chapters end with a written exam.

7.7 Test

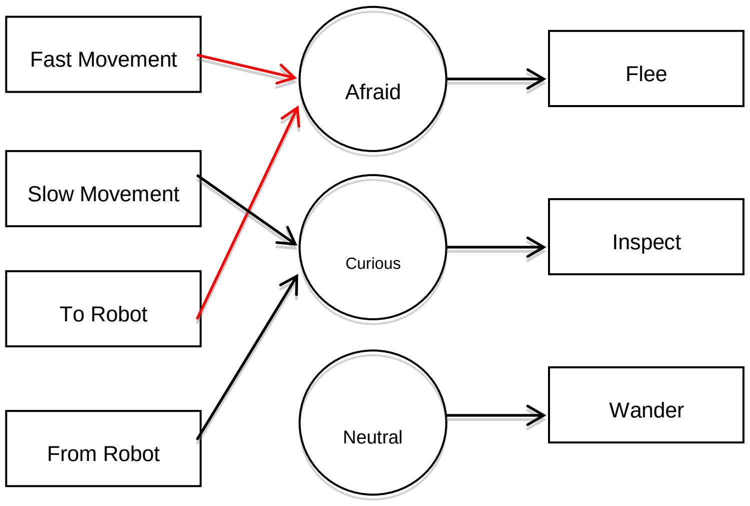
1. What is the difference between a servomotor and a standard electrical motor?
2. Why is a line-follower with two sensors faster ?
3. Which three variables determine the behavior of a line-follower?
4. What happens when both sensors see the black line at the same time?
5. Could this really happen and, if it does, what can be done about it?
6. Why do we have a stop condition in the line-follower?

Intro Part 3

In the third part of the RLT Robotics Module, we will explain Adaptive Behavior. This part is much more difficult than the previous two and is more suitable for older students or in information technology lessons. Part 3 addresses the Sense-Reason-Act loop in much greater detail and also explains the use of State Machines.

We will look at the behavior of a (very) simple organism and try to copy flee behavior and curious behavior or just let it drive around at random.

We will be using a State Diagram, that looks like this:



The diagram indicates under what conditions the robot becomes scared or curious and what behavior it should exhibit for each State. The three boxes on the right are the Behaviors that the robot will execute and that autonomously determine when their task has been completed.

These lessons also draw a parallel to the biological aspects of behavior and illustrate the initial principles of the Artificial Intelligence used in the field of Robotics.