

Robotics 1



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.

Modified versions of this module may only be distributed if this colophon states that it is a modified version, including the name of the author of the modifications.

© 2010/12. Version 4.3

The copyright of this module rests with RoboCup Junior Netherlands that is the owner under the terms of the creative commons license as mentioned below.

The authors of this module have used material from third parties during its development and have received permission to use this material. During research into the rights of text and illustrations, we have acted carefully.

Should, however, any person or organization deem to have rights to parts of the text or illustrations, they are advised to contact RoboCup Junior Netherlands (info@robocupjunior.nl).

This module has been compiled with care and has been tested extensively by the authors and several test schools. The authors accept no responsibility for incorrect or incomplete parts of this module, nor do they accept any claims for damages as a result of using this module or its associated software.



This module is distributed under the Creative Commons License 3.0, Netherlands.

► <http://creativecommons.org/licenses/by-nc-sa/3.0/nl>

COLOPHON	2
PREFACE	5
INTRODUCTION	6
ROBOTS IN SCI-FI FILMS.....	6
SERVICE ROBOTS	7
ENTERTAINMENT ROBOTS	8
INDUSTRIAL ROBOTS AND ASSEMBLY ROBOTS	8
EDUCATIONAL ROBOTS	9
1. GETTING TO KNOW YOUR ROBOT	14
THE ROBOT	14
TYPES OF SENSORS.....	17
FIELD OF VIEW OF THE DISTANCE SENSOR.....	17
THE PROCESSOR	18
WHAT IS PROGRAMMING?	19
LEGO MINDSTORMS NXT ROBOT COMPONENTS.....	20
ON/OFF SWITCH.....	20
THE BATTERY.....	20
THE PROCESSOR	21
SENSOR CONNECTORS	21
THE MOTORS	21
LCD DISPLAY	21
PUSHBUTTONS	21
DISTANCE SENSOR	21
LIGHT SENSORS	22
RESET BUTTON.....	22
USB CONNECTOR.....	22
THE ROBOPAL DONGLE.....	22
WHY USE A SIMULATION PROGRAM?.....	22
PROGRAMMING	23
2. GETTING TO KNOW THE SIMULATOR.....	26
USING VIEWS.....	28
ERROR MESSAGES	41
3. HOW DOES YOUR ROBOT WORK?	47
INSTALLING THE FIRMWARE	48
BASIC SETTINGS OF ROBOPAL	48
UPLOADING THE PROGRAM	50
CHECKING THE BATTERIES	50
READING SENSOR VALUES	52
4. DRIVING OVER THE RESCUE FIELD	55
INDICATORS.....	57
GOING BACK TO AN EARLIER VERSION	59

Preface

When the pioneers of Artificial Intelligence and Robotics developed the first programs and robotic devices, they foresaw a world inhabited by reasoning machines that would be able to make decisions. In this way, researchers created high expectations for the development of new theories in model-based reasoning, systems architecture, ethics and other fields. In the course of the development of intelligent devices, it became clear that more down-to-earth problems needed to be dealt with first: building mechanical structures, recognizing objects, avoiding obstacles, finding the optimal path to a goal, etc. These are the topics that will be addressed in this module and which are common to all intelligent devices such as robots.

The subjects treated in this module, which are all part of the robotics domain, mirror the basic assignments that need to be completed. A Lego NXT robot with sensors and actuators will be used to carry out a series of challenging, interesting and increasingly complex assignments.

This module is divided into three parts. The first part (in which the robot and the software are explained) is necessary to complete the other two parts. Its main goal is to introduce students to robot hardware (Lego NXT robot) and software (RoboPAL - Play And Learn). RoboPAL is a software development environment (IDE) with an integrated simulator. This means that programs can be developed and tested without a robot, thereby limiting the hardware requirements necessary at school. Moreover, it also means that students can work on their programs at home, on their own computers.

The second part focuses on the RoboCup Junior Rescue Mission and explains the operation of the Sense-Reason-Act loop. This loop is found in all robotic systems and lays the foundation for the control of the robot. This part also provides further information on the design criteria required to build more complex systems.

The third part provides a deeper explanation of sensors and actuators and allows students to design and develop more complex and autonomous systems.

Introduction

Background

This Robotics course is divided into three parts that can be studied independently.

This first part addresses the basics of handling a robot, its development environment and how to use the RoboPAL programming language.

The second part concentrates on programming the Rescue mission in which a dangerous container needs to be removed from a swamp.

The first and second parts are suitable for the lower classes of middle grade schools, while the third part is more demanding and concentrates on the development of a small Robotics project. It acquaints students with several aspects of the Artificial Intelligence techniques used in Robotics.

What are Robots?¹

It was not very long ago that computers became a part of our everyday lives and brought about so many changes. In a period of less than ten years, the world changed for a large number of people in companies and then in schools and at home. Nowadays, living without a computer is unthinkable.

More and more equipment around us contains small computers. These include TV set remote controls, washing machines and MP3 players. Often one device may contain several computers. These tiny computers are known as Embedded Systems. Modern cars, for instance, sometimes have more than ten of them. In most cases, we do not even know in what machines they are present and what kind of functions they perform. The same thing is about to happen, but this time with robots. More and more often, we see robots taking over difficult or dangerous jobs. So, it is important to understand what they can do and how they work.

Robots in Sci-Fi Films

Robots play an important role in many science fiction films. You may have seen *Wall-E*, the Disney film in which a lonely robot has to clean up waste on a deserted planet Earth.

¹ The word *robot* was used for the first time by the Czech writer [Karel Čapek](#) in his play [R.U.R.](#) (*Rossum's Universal Robots*) ([1920](#)) and is derived from the Czech word *robotá*, which means 'work'.

In films like *I Robot*, we see a robot jump from a window on the 5th floor and leap over cars. These are things that people will never be able to do. So, what can modern robots actually do? Will we ever have robots that can do what we see in these movies?



Fig 1: Wall-E

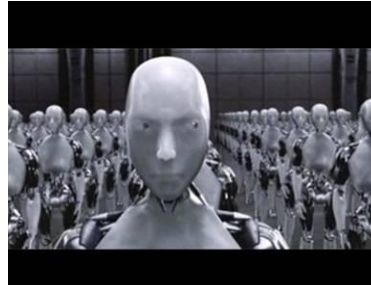


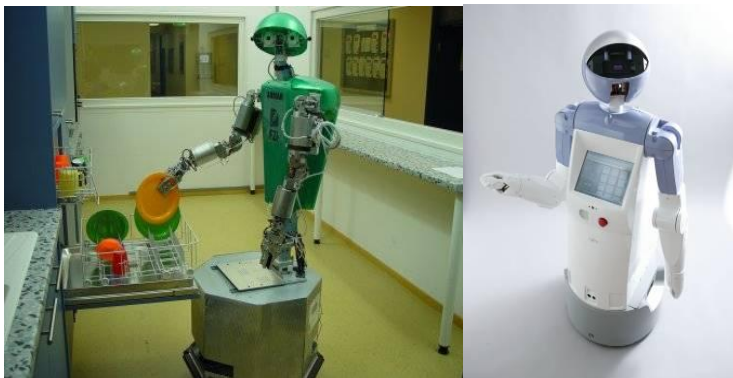
Fig 2: I Robot



Fig 3: Service Robot

Service Robots

The robots that we are speaking about are called *service robots*. More and more elderly people require care and assistance, but we are less and less capable of providing it. Many manufacturers are developing robots that can help with domestic tasks, such as making beds or reading out books, but we also need robots that can be used for security purposes in our homes (i.e., fire alarms or to check if somebody has fallen over). Honda and Toyota have been building robots for years with the goal of providing good assistants for our homes.



There is also a great deal of interest in robots that can help the elderly (i.e., make sure they take their medication on time), but these robots do not exist at present. Some hospitals are already using robots to deliver food and experiments are also being conducted with Rescue robots.

Figure 5 shows a robot that is used to save people in dangerous situations such as fires, gas leaks and collapsed buildings.

We also have robotic vacuum cleaners and lawn mowers (Fig 6). The behavior of these robots is very limited. Their movement pattern is random, which is not



Fig 5: Robot in a hospital (left) and Rescue Robot (right)



Fig 6: Robot vacuum cleaner (left) and lawn mower (right)

very economical for finding dust and grass to cut. Also, in order to let the lawn mower “know” where the grass is, a special wire must be placed around the field to be mowed.

Entertainment Robots



Fig 7: Robo Sapien

As Robotics has not advanced far enough to make robots that are sufficiently flexible and reliable, a good first step is the use of robots in the entertainment industry. The demands that we put on an entertainment robot are not as high as those we would have, for instance, in a robot meant to help people at home. If a robot does something wrong in an amusement park or in a movie, it is funny and has far less severe consequences than if it did something wrong in a hospital. In this case, a robotic mistake could cause a significant amount of problems. Sony has been selling entertainment robots, such as the *Aibo*, for many years. We also have robots like the *Robo Sapien*.

Industrial Robots and Assembly Robots

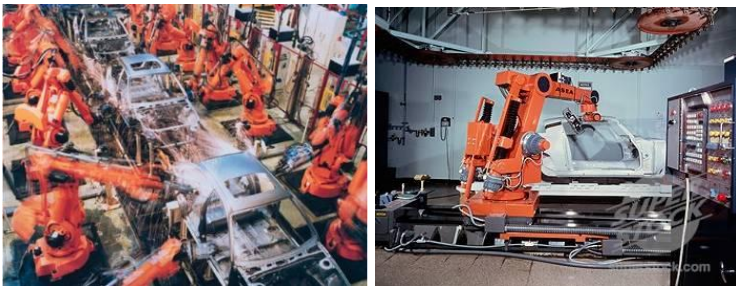


Fig 8: Assembly Robots

Robots are also used in industry to assemble cars and airplanes. They are used in almost all factories, but are not as flexible as the robots that we see in science fiction movies. Most of these robots are used to weld, insert screws or assemble parts such as printed circuit boards.



Fig 9: AGV



Fig 10: Salvage Robot

A special kind of robot, the so-called AGV (Automated Guided Vehicle) is used in Rotterdam harbor. These vehicles are used to automatically transport sea-containers. An increasing number of robots is also used in the army for a variety of tasks such as removing explosives.

Educational Robots

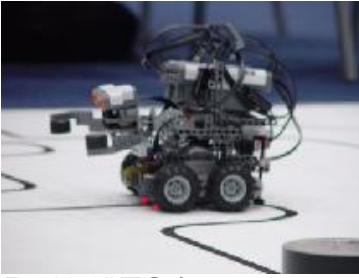


Fig 11: NXT Robot



Fig 12: the Tulip

The robots that we will be using in this module are educational robots. There are many brands on the market, but the Lego MindStorms is by far the best known. In addition, there also are other robots, such as the Board Of Education Robot (BOE-bot) and walking robots such as the RoboNova and the BioLoid.

Universities also develop robots. In the Netherlands, for example, the three Technical Universities, headed by TU Delft, are developing a new generation of walking robots: the Tulip of the Dutch Robotics team.

Building a service robot is a complex exercise, because a variety of elements must be taken into consideration. In this module, you will get a good idea of the things a robot needs to be able to do in order to act as a service robot. You will also see what knowledge and technology is necessary to design and build a service robot.

What Must a Service Robot Know and Do?

Two of the aspects that need to be considered during the development of a service robot are: what will it be used for? and what type of tasks must it be able to perform? A service robot used as a deep-sea diving assistant will require different elements than a kitchen robot that needs to handle large pans and pots. The idea of building the ultimate universal service robot is probably utopian, but whether or not this will ever be possible, building such a robot would not be very useful. A specialized robot will function better and will be cheaper to develop.

There are, however, a number of basic concepts that are valid for all service robots. A robot needs to be able to find its way around its environment and react to the changes that take place around it. This means that a robot must either collect information from its surroundings or be pre-programmed with the necessary data. Responses may consist in a direct reaction to a change, comparable to a reflex or the robot could plan its next step based on the information it collects. Moreover, its reaction also depends on the amount of time that is necessary for it to react appropriately.

A robot should be able to execute commands. It should be able to make decisions based on a variety of conditions. The conditions that a robot must use are determined by its observations, but also by data that is pre-programmed and/or collected over time (experience).

When a robot uses its sensors to collect information from the environment around it, it has to decide if it can execute a command. Clearly, the best situation would be for the robot to decide how to perform a given action. Three elements are involved in such a decision: *reaction time, conditions and execution*. Indeed, this is what many researchers are currently investigating.

What Should Robot Builders Know and Do?

The previous section should have made it clear that a designer, working alone, faces an impossible task; a team of designers is necessary to achieve a satisfactory result. Every team member must have an area of expertise and must explain to the other team members what is necessary to make a good design.

Such specialists range from programmers to mathematicians and from structural engineers to psychologists. They all have to address the question of what function the robot must perform. Questions such as: How big and how strong must it be? How much energy will it need and for how long? How will the robot receive its commands ?

The Lego NXT Robot



Fig 13: Lego NXT Robot

This Robotics module teaches you how to make a robot perform tasks. We will use a simple two-wheeled robot that is equipped with several sensors. Your job is to make your robot remove a dangerous container from a swamp (not really, it's a soft-drink can that is placed on a yellow surface, see Fig. 14). In order to accomplish this task, you will have to program your robot to behave like Wall-E. Your robot will need to follow the road to the swamp, then search for the container and push it outside of the swamp.

The world in which your robot lives has been made a bit simpler than reality, otherwise the task would be too difficult. Your robot will move over a field on which a black line leads to the swamp. There, a can or a small puppet has to be pushed ashore.

An even simpler field is the so-called *Grid field*, which is divided into small squares and has no further surface drawings. Both fields are used separately with the RoboPAL software that you will be using to create your program. This will be explained in greater detail in the following lessons.

The Leading Thread

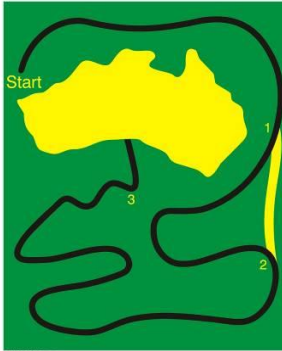


Fig 14: Rescue Field

We will use small programs to help you become familiar with programming and learn how to create your own programs. You will then start making simple programs, followed by more complex programs, to instruct your robot how to behave. Clearly, the small robot will only be capable of a limited repertoire of behaviors. In each lesson, we will explain the relevant principles underlying each type of behavior and how this knowledge is used in industry, at home or elsewhere.

The robot that we will use in the lessons (if programmed correctly) can detect colors, drive forward and backward, follow lines and detect and avoid obstacles. This simple repertoire is all you need to program the robot to perform some quite interesting actions.

The programs that you are about to work with are examples of the way in which problems are solved by many devices. Cars, washing machines, industrial robots and airplanes contain the same technology. This will help you learn how these machines work. The following subjects will be addressed:

H#	Subject	Learning Assignment	Working Assignment	Examples
1	Getting to know the robot	Understanding Sensors and Processors	Find out what a robot is	Where are robots used?
2	Getting to know the simulator	Using a simulator	Changing a program	Flight Simulator
3	How does your robot work?	What can a robot do?	Making your robot move	Bridge
4	Making the robot move over the field	Following patterns	Following a fixed track	Welding robots
5	Sensors for the 'Sense' step	Sense-Reason-Act loop. Robot- and animal behaviors.	Observe before acting	Automatic doors in elevators
6	Processing for the 'Reason' step	Process Information	Line-Follower	Automatic Guided Vehicles
7	Actuators for the 'Act' step	Moving and Steering	Adaptive behavior	Self-parking car
8	Adaptive behavior	Adapting to the environment	Not reacting the same way every time	'Pick and place' robots
9	Advanced sensors	Sensory information	Calibration and object avoidance	X-rays, infrared and ultrasound
10	Control systems	Feedback in control systems	React to disturbances	TV transmitters automatic programming

Setup of each Chapter

Every chapter in this module has the same setup:

1. What will you learn?
2. What do you need?
3. What are you going to do?
4. What can you do after studying each chapter?
5. Explanations and Assignments
6. In Practice
7. Test Questions








The Assignments

The idea behind this module is to learn through discovery, insofar as possible. Therefore, we purposely do not explain everything in great detail. As it's a new subject, you may discover that some steps are too difficult, but the main idea is to work out as many things as possible on your own. Each chapter has a number of optional assignments that you may skip. To make the lessons more challenging, we hope you will first try to work on the main assignments and only turn to the optional assignments when the main prove to be too difficult. Each chapter ends with an assignment that you will have to complete on your own. You must not skip these, as they will help you understand the subjects that are treated in each chapter. You will learn best by trying, making mistakes and discovering the right answer.

If you cannot complete one of the assignments, turn to the detail assignments and try them first, until you understand how something works. Then, go back to the main assignment. By following the detail assignments, you should find enough information to continue with the main assignment.

Explanation of Pictograms

The symbols used in this module have the following meanings:

<i>Assignment Type</i>	Assignment
	Detail Assignment (optional)
	Find out about something
	Assignment on the Grid Field.
	Assignment on the Rescue Field
	Assignment with RoboPAL (programming)
	Assignment on the Simulator (testing)
	Assignment on the Robot

Assignments



1. Find Out

Use a search engine to find the meaning of the word ‘robot’. Use what you have found out to describe (½ A4) the similarities and differences between the old and modern meanings of “robot”.



2. Find Out

Use a search engine to answer the following questions. (½ A4)

- What is a robot?
- Are there different types of robots?
- What is the difference between a robot and a human?
- If you had to divide all robots into two main groups, what would they be?
- What is the main difference between these two groups?



Fig 15 Pictures of Robots

1. Getting to Know Your Robot

The most important parts of a robot are explained in this chapter. You may cover this chapter together in the classroom or read it on your own. If you want to get started right away and read this part later, go to Chapter 2 and start working with the Simulator.

1.1 You will learn

- what a robot is
- why we are using a simulation program
- about various programming languages



Fig 16: Lego NXT

1.2 You will need

- a computer
- a Lego MindStorms NXT Robot
- the RoboPAL development environment
- the simulation program (part of RoboPAL)

1.3 You will experiment with

- the components of the robot and learn to name them
- the how and what of a simulator
- the language used to program a robot

1.4 After following this chapter, you will be able to

- explain the meaning of the terms Robotics and AI (Artificial Intelligence)
- point out the most important parts of a robot, name them and describe their functions

1.5 The Lego MindStorms NXT Robot

The Robot

In order to follow these lessons, you will need a robot. Although you can follow all the lessons just with the simulator that is built into RoboPAL, using a robot makes the lessons much more interesting.

Only when you run your programs on a real robot will you discover the differences between a simulator and the real thing. The robot is influenced by external factors, such as friction, the slippage of wheels, low batteries,

inaccurate sensors, different lighting conditions and many more, that are absent in the simulator.

These factors will always make your robot react differently from a PC simulation. We will see more about this in the next chapter.

You will first have to assemble a robot using the Lego MindStorms kit. Most schools already have these robots and reuse them every year. In this case, there is no need to build your robot. Otherwise, use a Lego kit and put a standard robot together. The instructions for this can be found on the CD that comes with this module or downloaded from the RoboDidactics site (www.phyrtual.org).

Lego sells various different NXT kits, so the assembly instructions may refer to parts that are not available with your kit. In this case, you may need to find an alternative solution to construct your robot. What the robot looks like is not so important as long as it has two wheels and a castor wheel, or glider, and two light sensors, pointing to the ground, plus a distance sensor.

It is important that the motors and sensor are connected in the standard way to ensure that RoboPAL will function correctly.

The left and right motors must be connected to ports A and C. When we refer to the left side of the robot, we mean as seen from the position of an imaginary driver. The driver is always on the left side. So, if you look at the robot from the front, the left motor is on the right-hand side, just as with a car. Also, remember that if you mount the motors backward, the direction of movement will be reversed. The RoboPAL CD contains the building instructions. So ask your teacher for them or you may also make your own design. What the robot looks like is not important, as long as all the parts are connected correctly.

The two light sensors point downward and the underside of these sensors must be about 1 centimeter from the ground. The distance between the sensors should be 5-8 centimeters. In RoboPAL, you can specify the spacing between the light sensors, so the simulator can take this into account. We will explain how to do this later on. Now, keep a **standard** distance of **6.5 cm**. The sensors must be connected to ports 1 and 4, the outer two ports just as the motors.

The distance sensor must be connected to port 3. If you were also using a third light sensor, it would be connected to port 2, but we will not be using it in these lessons.

So, assemble your robot or make sure you have a previously built one. As we will be using a simulator, you will only use the robot for short periods and share it with other students. As a rule of thumb, 4-5 students can share a robot. If students are working in groups, you may even need fewer.

The Components of a Robot

All robots consist of three essential parts:

1. The brain, consisting of the processor(s) + program(s)
2. The sensory system, consisting of sensors
3. The muscles and skeleton, consisting of actuators and a mechanical frame.

The processor(s) + program(s) are necessary to process the information provided by the sensors and to make decisions. Based on these decisions, the actuators will be switched on or off. The generic name for sensors and actuators is transducers.

Definition

A *transducer* is a device that transforms one type of energy or physical property into another for a variety of purposes, including measurements and information transfer.

An example of a sensor is a microphone. It is used to transform sound (mechanical) into an electric signal. An example of an actuator is a motor. It transforms an electrical signal into movement (mechanical). So, an input-transducer is a sensor and an output-transducer is an actuator. The variation in the mechanical construction of robots is enormous. One specific type includes all wheeled robots that have the following mechanical components: a chassis, a steering mechanism, a number of wheels (powered or unpowered) and motors. The combination of the steering mechanism and separately powered wheels is the most distinctive property of wheeled robots. A frequently used feature is the so-called Ackermann² steering mechanism. Almost all cars and trucks use this principle.

The steering mechanism that we will use on our Lego NXT is very popular with small-wheeled robots. It features two independently controlled wheels and a so-called *castor wheel*, which often is simply a ball-shaped object that allows the robot to slide over the ground.

The simplicity of this construction and the easily programmable steering mechanism makes it very suitable for this kind of robot.

When a robot needs to drive at a high speed or over an uneven terrain, other steering mechanisms are required, especially when the load on the wheels or the accuracy of steering are important factors.

In our case, the robot has two independently powered wheels. This type of steering is often called a *differential drive* (not to be confused with the differential, which is a mechanism used in cars to control the back

² Designed by the German car designer Georg Lankensperger in 1816 and patented in England by Rudolph Ackermann.

wheels). The difference in speed between the wheels and the distance between them determine the direction and speed of the robot. This type of steering mechanism is also known as a *wheelchair drive*.

Types of Sensors

There are many different types of sensors, subdivided according to the physical properties that they transform. These include chemical, electrical and mechanical ones. Some examples are:

- a pressure sensor to 'feel' (mechanical)
- a reflection sensor to 'see' (electrical)
- an infrared (IR) sensor to see in the 'dark' (electrical)
- an ultrasonic sensor to 'hear' (electrical)
- a compass sensor to 'detect a direction' (magnetic)
- a gas sensor to detect gasses (chemical).

We can also distinguish between passive and active sensors.

Passive sensors do not need an extra source to detect something: for example, a camera without an external light source is a passive sensor. On the other hand, a reflection sensor and an ultrasonic sensor are *active sensors*, because they send out a signal and measure the returning signal. We will use some of these sensors to let our robot sense its environment so that it can determine where to go to and to check if there are any obstacles on its way. In particular, we will use the following sensors with the Lego NXT:

A *distance sensor* (DS) uses an ultrasonic signal to measure the distance between an object and itself (the robot in this case) and expresses the measured distance as a number. The robot uses this mechanism to detect obstacles.

The *reflection sensor* (FieldSensor, FS) is also called a 'ground sensor'. It emits visible light and measures the amount of reflected light. This type of sensor allows the robot to detect a black line on the floor. A dark surface reflects less light and thus returns a lower value.

Field of View of the Distance Sensor

In most cases, sensors that use light or sound have an opening that allows waves to enter the sensor. This opening, which is called the *field of view (FOV)*, is often a lens that allows the sensor to see a given area.

When using a distance sensor, the optimal situation would be to measure the distance in a straight line ahead, especially if the beam could keep the same width over a longer distance (field of view = 0°). A laser can do this, but it is expensive. An ultrasonic sensor, on the other hand, will always have a field of view that can be compared to the light beam of a flashlight.

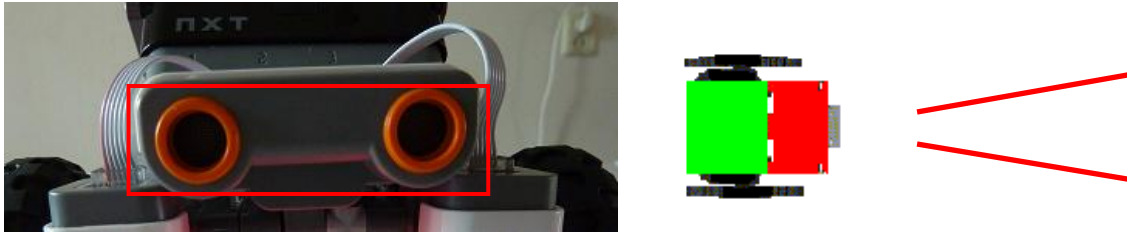


Fig 17: Ultrasonic Distance Sensor

The farther away the beam travels from the sensor, the wider it gets. This also means that the farther the objects are, the larger the detection area is and the sooner the sensor will react to obstacles. The advantage is that narrower objects are detected sooner, although it is unclear where exactly the object is located. Alternatively, a narrow beam provides greater certainty on the position of an object, but the chance of missing an object becomes greater. A solution to this situation is to use an array (row) of overlapping sensors, each with a known field of view. By correlating the information provided by an array of sensors, it is easier to detect small objects and determine their position.

We will use an ultrasonic sensor that has a relatively wide field of view with the Lego NXT. This, unfortunately, often makes it difficult to accurately measure nearby objects (see fig 17).

Types of Actuators

Actuators are mainly motors. There also are special actuators such as memory metal and artificial muscles, but these are mostly still research projects. In addition, there are actuators that let us see or hear things, like:

- lamps
- LCD screens
- loudspeakers

The Processor

In principle, any kind of PC can be used to control a robot. There are many robots that have a PC as their brain, but if space and energy consumption are limiting factors in the design, we need special **processors**. Moreover, the price-performance ratio often dictates the use of a less advanced processor.

The most important demands of a robot are of a different order than those of a PC. For calculations, accuracy is one of the most important aspects on a PC. Calculations must be fast, but a precise answer is even more essential.

A robot has completely different demands. The emphasis is not so much on the accuracy of a calculation, but the timely availability of a result, regardless of its accuracy. Which of the following two robots would be

more useful? One that can calculate its position with a precision of 100 decimal positions, but that discovers it is falling off a bridge because there was not enough time to read the information coming from its sensors, indicating that it has reached the edge of the bridge, or a robot that ‘only’ has 2 decimal places, but quickly reads the information from its sensors and correctly executes a stop command? The answer is clear.

You could opt to provide the robot with a processor that is both fast and accurate, but then its price and power consumption would certainly be much higher.

The mobile device market is not the only one that has a high demand for such processors. Processors are used in all kinds of electrical appliances, ranging from toys to kitchen machines to machines for heavy industry. A modern, average size car has about 15 processors, while more luxurious models may have as many as 65.

The number of processors used in this market are greater by a factor of 40 than that used in personal computers. In 2008, about 100 million PC processors were sold versus 4 billion so-called embedded processors.

What is Programming?

To make a robot do what you want, you have to give it instructions. We call these instructions a *program*. It would be nice if we could just say, “Go get that dangerous container and push it out of the swamp,” but the software used in robots is not that clever, yet. You need to give the robot more detailed instructions such as “Follow the black line to the yellow swamp. Then, search for the container and when you find it push it ashore.”

Although these instructions are far more detailed, the robot cannot understand these either. You will have to tell it what to do in even greater detail. A robot needs to know when to switch on its motors, how fast to move, in what direction to go and how to use its sensors to see where it is. You must also tell it how to distinguish black, green and yellow. The robot does not ‘know’ this and you will need to instruct it on each of these things. Telling a robot what to do is called ‘programming’ and we use a special language called a *programming language*.

There are many different programming languages, ranging from very simple to professional ones. This version of the RLT Robotics module uses a programming language that is based on little pictures called Icons. The more advanced version of this module is based on a more complicated programming language called Java and the Eclipse development environment. In fact, Eclipse is used by many professional programmers. In this version, we will use the RoboPAL (Play And Learn) development environment, which is based on a graphical programming language. A development environment is a system that lets you create programs, test them and then upload them onto a robot.

In RoboPAL, we have the following three components:

- The development environment - to create or modify a program
- A simulator - to check if a program works as intended (testing)
- A compiler/loader - to upload programs onto the robot

Lego MindStorms NXT Robot Components

In this module, we will use a robot with the following components (fig 18).

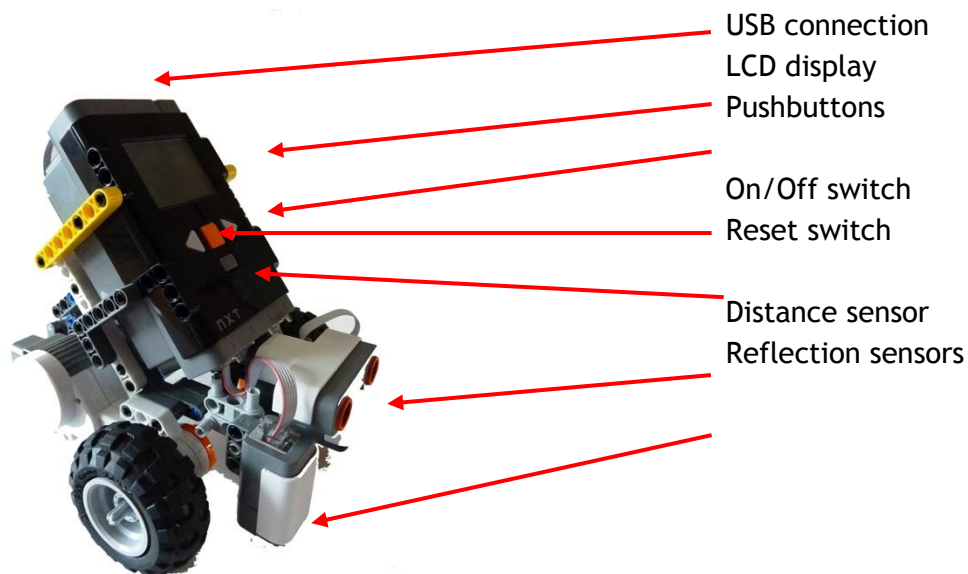


Fig 18: Components of the Lego MindStorms NXT

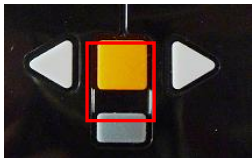


Fig 19: On/Off Switch

On/Off Switch

The NXT has various pushbuttons. The ON/OFF switch (the orange button) is used to turn the robot. Press the orange button together with the gray button underneath it to turn the robot off. Make sure you always switch off the robot when it is not in use, as the batteries run down quickly.

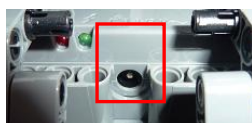


Fig 20: Charger Connector

The Battery

The NXT has a battery compartment on the back that holds six penlight batteries (or a rechargeable battery that is sold separately). The Lego Dacta school version comes with the rechargeable battery. This battery has a connector at the bottom to connect it to the charger. Just above this connector are two leds: a green and a red one. The red led is on when the battery is being charged, while the green led is on when the battery is fully charged. During charging, both leds usually are lit. The batteries run down rather quickly and take a while to fully recharge. If they are not charged, you can also use standard alkaline batteries, but in most cases you will have to disassemble your robot to reach the battery compartment. Batteries last approximately 30 minutes. So, to conserve your batteries, remember to switch the robot off when it is not in use.



Fig 21: The NXT Processor Brick

The Processor

The brain of the Lego NXT robot is the NXT brick which houses the processor. This brick contains a powerful 32 bit ARM7 processor with a 256 KB flash memory. This processor controls all the electronics and runs the RoboPAL programs. In order to work with RoboPAL, the standard Lego Firmware needs to be replaced with the RoboPAL firmware. This is done directly with the RoboPAL program. The Lego Firmware can be reloaded using the Lego NXT-G software, so both languages can be used with the same robot, but not at the same time.

Sensor Connectors



Fig 22: Sensor Connectors

The sensor connectors are located at the bottom of the NXT brick. Both light sensors are connected to the left and right sides (ports 1 & 4), while the two middle ports are used for the ultrasonic distance sensor (port 3) and a touch or third light sensor (port 2). You may also connect another sensor to port 2, like the Lego Sound Sensor.



Fig 23: Servomotor

The Motors

The actuators on the NXT are two motors that can be used as a regular motor or as a servomotor. In this module, we will only use the standard motor option. An explanation of the purpose of a servomotor is provided later on in the module..

LCD display



Fig 24: LCD display

The LCD display is a small screen (110x64 pixels) that is used to display information by the program. It is especially useful not only for the program calibration process, but also to know what your robot is doing whilst testing a program. This is an example of an output device. As the NXT does not have any lamps, four simulated lamps are shown on the display. You can turn these simulated lamps on and off with your program.

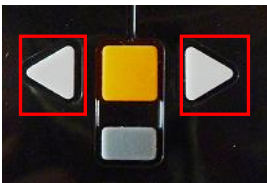


Fig 25: Passive Sensors

Pushbuttons

The NXT has various pushbuttons. The two triangular buttons are used to start the program or to provide information to the program. This is an example of a passive sensor.

Distance Sensor



Fig 26: Distance Sensor

This is an example of an active sensor. The sensor has two parts: an ultrasonic transmitter and a receiver. A short burst of ultrasonic sound (40 KHz) that humans cannot hear is transmitted. This burst is called a Ping. This ping is reflected by an object and then detected by the receiver as an echo. The receiver measures how long the ping took to echo back to calculate a given distance. This sensor can measure objects at a distance of 5 to over 100 cm.

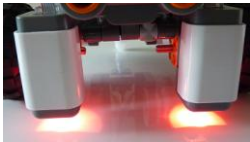


Fig 27: Reflection Sensors

Light Sensors

The light sensors are another example of an active sensor. The light sensors are mounted underneath the robot and use red light that is reflected by the ground. The sensor measures the intensity of the reflected light, allowing the robot to detect different shades of gray (values) on the ground. Colors are seen as shades of gray. Light sensors can also be used as passive sensors by switching off the leds. In this case, they only register the amount of ambient light. We will not use this feature in our lessons, but it is used by soccer robots to detect the ball.

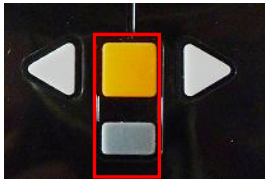


Fig 28: Reset Button

Reset Button

The orange button is used to stop the program, but if the program has crashed (it is blocked), you need to use the reset button. In order to reset you robot, press the orange button and the gray button underneath it at the same time.



Fig 29: USB Connector

USB Connector

The NXT uses two communication facilities. The USB connector on top of the NXT brick is only used to upload firmware onto the robot. We will use RoboPAL to load the firmware, but you must also install Lego Phantom drivers on your computer. If you are not going to change the firmware (true for most school computers) this driver does not need to be installed.



Fig 30: RoboPAL Dongle

The RoboPAL Dongle

To load a program onto the NXT you need to use RoboPAL with the special Bluetooth dongle, which takes care of all communication between a PC and the NXT, without requiring any extra software or drivers. The dongle also acts as a license key for the RoboPAL software. There are two versions of this dongle, a white one for personal use and a black one, which is the server version which is used by most schools.

RoboPAL can also be used without a license. You can also install it on your computer at home. You only need the license to load a program onto a NXT unit.

Why Use a Simulation Program?

It is dangerous to start training pilots in a real airplane! That's why they use a training device that simulates all the movements of an airplane, but without ever leaving the ground. An image is projected onto airplane windows for pilots in training. In this way, if the pilot makes a mistake, there won't be an accident. Moreover, many dangerous situations can be simulated and the pilot can gain important experience in a safe environment. These flight simulators are controlled by computers. The same holds true for developing robot software. Often, it is not practical and sometimes even dangerous to test a program directly on the robot. We first want to try it on a simulated robot. A simulator, however, has both advantages and disadvantages.

The Advantages

- You can develop and test a program without a robot, which saves money.
- It's faster: loading a program onto a robot takes about 2 minutes versus less than a second on the simulator.
- Everything in a simulator is stable and always works in the same way. You do not have to take external elements such as noise, lighting conditions, uncharged batteries, etc. into consideration.

The Disadvantages

- Not everything that happens with the real robot can be simulated: for instance, fluctuations in battery voltage or changing lighting conditions.
- A simulation program has a limited reality. Your program will almost always have to be modified to run on the real robot. So, be aware that if your program works fine with the simulator, this does not mean it will also work perfectly on the real robot.



Fig 31: Simulator

The simulator allows you to create and test programs without a robot. This is much quicker because you do not have to load the program onto the robot every time you complete a new version. In a classroom, this also saves money as not everyone needs their own robot. Moreover, you can also develop your program at home. Last but not least, robots are also expensive and intensive use will wear them out or damage them.

If your program works on the simulator, load it onto the robot to see what it does in reality. In chapter 4, we will explain how to upload the program to the robot.

Programming

FlowCode 1 shows an example of RoboPAL code, while FlowCode 2 provides an example of the associated pseudo code. Pseudo means 'not real' or 'fake'. It describes the program in an easier to understand 'natural' language. To keep code readable, a commonly accepted lay-out is used. In principle, the code could be written as one long sentence, but this does not improve the readability, although the computer does not care about such things.

In the RoboPAL version, we will not use pseudo code very much. (We will see some examples for the more complicated tasks in part 3.) The RoboPAL graphical programming language makes code easy to understand itself.



FlowCode 1

The pseudo code looks like this:

```
Do four times {
    Set motors to half speed forward // left speed 60, right speed 60
    Wait for 2 seconds
    Make a turn to the right
    During half a second
}
```

FlowCode 2: pseudo code.

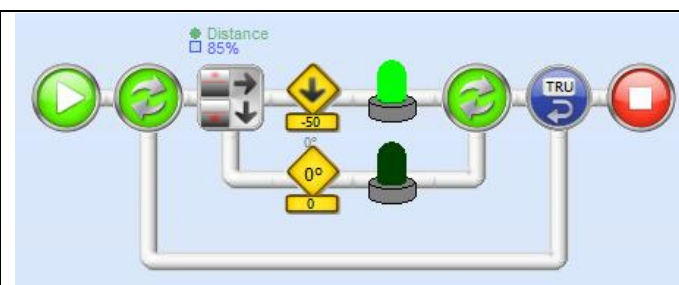
Pseudo code is written first to get an idea of what a program should do. It makes it easier to write the real code. In our examples, we also use pseudo code to explain what the RoboPAL flow code does.

The robot's control system consists of a RoboPAL program. Every program consists of a number of RoboPAL worksheets that we call *routines*. Every block you see in the program is called an *'icon'* and a series of these icons form a *routine*.

The total of all routines is called the *program*. Every program has a name and is stored in a workspace, which in our case is called *'My NLT Projects'*.

You can also store your programs on a USB stick and then decide what you want to name your workspace. We have separate definitions in the form of icons that we specify in the so-called *world* for each different playing field (Grid, Rescue and Soccer/Football), but also for each type of robot.

In each chapter, you will find these programs in the form of icons (see the example in FlowCode 3). In the lessons, we will explain what the meaning is for each icon.



FlowCode 3

FlowCode 3 consists of a number of icons that we will discuss in greater detail in the following lessons. It does not matter right now if you do not understand it.

1.6 The First Assignment

It is a good idea to have a look at the Dutch RoboCup Junior website ► www.robocupjunior.nl to get a better understanding of the challenge that you will meet in these lessons. The site also has a limited English section.



1.1 Rescue Assignment Rules

- Open **Downloads** via ► www.robocupjunior.nl
- Open Rules
- Open rules Rescue 200X
- Study these rules



1.2 Find Other Programming Languages (search engine)

- Find out if there are other programming languages. Why are there so many of them?
- Mention at least three and what they are used for
- What are currently the most used programming languages? Why are they so popular?

1.7 Test

1. What sensors do you know about and what is their function? Explain to what human senses they correspond.
2. Explain what the following robot components are used for: push buttons (three of them), batteries, processor, motors, connectors, distance sensor, reflection sensors and USB connection.
3. What is the purpose of the LCD screen on the NXT brick?
4. Why are we using a simulation program ?
5. Which programming language will we use to control our robot?

2. Getting to Know the Simulator

We will introduce you to reading and changing a program by showing you a small example program that will also serve as an introduction on how to use the simulator.

The ‘Flee Behavior’ program has already been written. You need to read and try to understand where, in the instructions, the actual ‘flee behavior’ takes place. You do not need to fully understand all parts of the program. Having a general idea is sufficient to find out where you need to make a change, so that the robot will become curious, rather than scared. Change the program into ‘Curious Behavior’. This will give you an idea of how the program controls the robot’s behavior.

2.1 You will learn

- to interpret a RoboPAL program
- to understand *FleeBehavior* and *CuriousBehavior*, both as behavior and as a program
- how to work and test programs on the simulator
- how to recognize errors in a RoboPAL program

2.2 You will need

















- a computer with RoboPAL
- the Grid field (a white surface)
- two programs: *FleeBehavior* and *CuriousBehavior* (You actually get *FleeBehavior* twice and need to change one of them into *CuriousBehavior*.)

2.3 You will experiment with

- some simulator features
- changing an existing program and testing it
- the behavior of the robot on the simulator

2.4 After following this chapter, you will be able to

- explain how the simulator works and why it is useful
- work with the development environment and the simulator
- use loops and draw lines between program icons
- make small changes to an existing program

Type	#	Assignment	Description
	2A	Simulator	Working with the Simulator
	2A.1	Simulator	Starting the Simulator
	2A.2	View	Looking at parts of the Simulator
	2A.3	Insert	Adding parts
	2A.4	Making a copy	Making a backup copy
	2A.5	Drawing lines	Using pipes
	2B	FleeBehavior	Testing Flee Behavior
	2B.1	FleeBehavior	Making the robot flee
	2B.2	FleeBehavior	Looking at FleeBehavior in RoboPAL
	2C	CuriousBehavior	Making CuriousBehavior
	2C.1	CuriousBehavior	Making the robot curious
	2D	CuriousBehavior	Reading Sensor values
	2D.1	Experiment	Experimenting with sensor values
	2D.2	Experiment	Experimenting with turns
	2E	CuriousBehavior	Making the robot stop in front of the ball
	2E.1	Personal assignment	Stopping the Robot in front of the ball
	2F	CuriousBehavior	Error messages
	2F.1	CuriousBehavior	Errors in your program
	2F.2	CuriousBehavior	Debugging a program

2.5 The Assignments

Detail Assignments




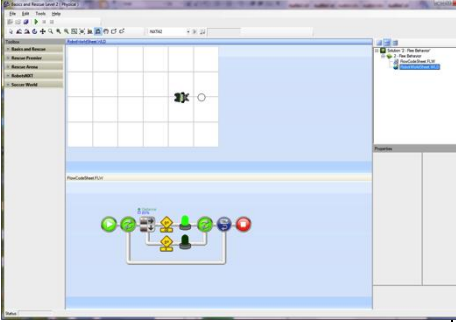
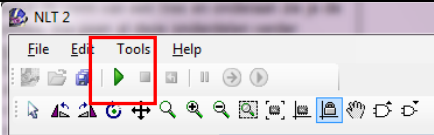
In each chapter, you will find both main and detail assignments. The detail assignments are meant to help you along only if the main assignments are too difficult or if you do not understand how something works. These assignments lead you step-by-step through everything you need to complete the assignment. If you think that you understand the main assignment, go directly to the next main assignment in each chapter. Only work on the detail assignments if you think you need them. Sometimes, you may find clues in the detail assignments that will help you without having to actually complete them.

Detailed assignments are shown in light gray and indicate that the assignment is optional, as the next assignment.



2A.1 Assignment: Working with the Simulator

The first thing we will show you is how to start RoboPAL, select a program and start the Simulator. Then, we will explain what you can do with the simulator. Follow the steps given below. If you think you can already do this, you may skip this assignment.

2A.1 Starting the Simulator	
1	 <p>Start RoboPAL by double clicking on the RoboPAL icon on your desktop.</p>
2	 <p>Select the leftmost tab "Perspective" from the window that appears. Now, select the Physical Level (Level 2) in which you will find the NLT modules.</p>
3	 <p>Now, select the NLT 1 module. All examples, mentioned in this module are provided as external modules and can be found on the CD that comes with the course. In most cases the school will place these lessons on the server, so ask your teacher where you can find these lessons in your case. Now load module 2A from the student examples.</p>
4	 <p>In the window that will appear, there is a Toolbox on the left (more about this later) and RobotWorldSheet in the middle at the top. This is called the World. It displays the playing field, robots and any other object in use (a ball, for example). Underneath this is the FlowCodeSheet with your program. This is where you will be working. The project structure is displayed on the right in the form of an expandable list (tree) and, at the bottom, there is the properties box. We will explain all of these parts further on.</p>
5	 <p>We will begin by starting the simulator. Click on the green 'Run' button in the left upper toolbar. Take note of these icons as you will be seeing them regularly. Always first make the top window active by clicking in it anywhere, otherwise you cannot start the Simulator.</p>

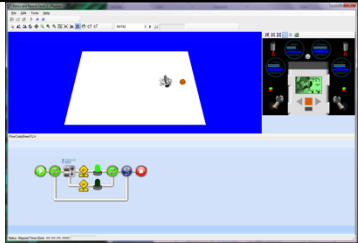
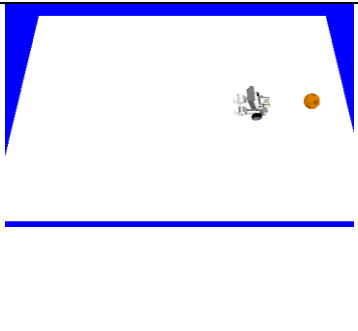
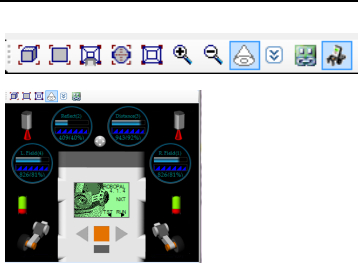
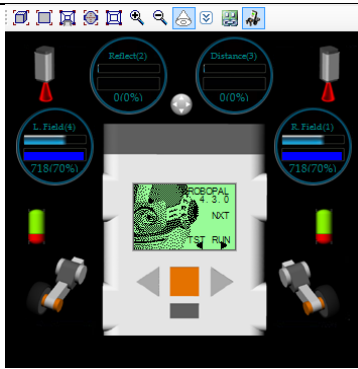
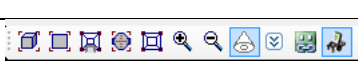



2A.2 Assignment: Looking at Parts of the Simulator

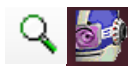
We are going to find out how to use a simulator and what it can be used for.

Using Views

The various views available on the simulator allow you to choose how you observe the playing field.

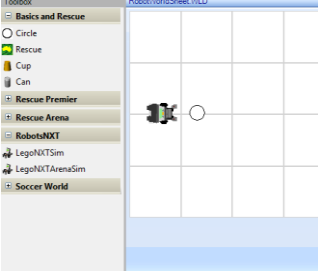
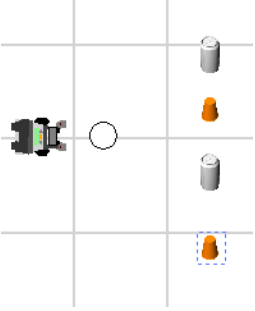
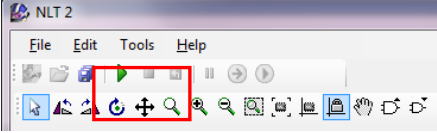
2A.2 Looking at Parts of the Simulator		
1		As soon as the simulator starts, you will see three screens: the Grid field on top with the robot and a ball, the Program underneath that the Control Panel on the right. We will first look at some of the details of the playing field (the world).
2		This is how the robot and ball appear in the simulator. You can move the robot and the ball by pressing the left mouse button and dragging the robot with the mouse. You can make the robot face a different direction by using the right mouse button. The blue area around the field is a kind of nowhere land where the robot should not go. If the robot drives too far into nowhere land, the simulator will put it back in the middle of the field.
3		The Control Panel is at the top on the right with a row of icons. The first five icons determine how you look at the field: 3D is the standard view, 2D allows you to look at the field from above and Point Of View (POV) shows the field from the point of view of the robot. Try all of them.
4		The Control Panel also has four dials corresponding to the four sensors that can be attached to your robot. We will not use the second sensor (the reflection sensor). When your robot gets close to a ball or other object, you will see the value of the Distance sensor change. You will need to use this later on. The NXT brick in the simulator also displays the same buttons as the real NXT that we discussed in the previous lesson.
5		The icon number eight is used to switch the lights on and off in the simulator. It can be used to slightly alter the lighting conditions in the simulation. We will not be using it, so make sure the lights are always on. The icon next to it is used to change the speed of the simulator itself. Always keep this at a 100% setting, but you can make the robots move faster or slower in the simulation. Its effect is similar to making a video play faster or slower. Your program stays the same, but everything happens faster or slower.


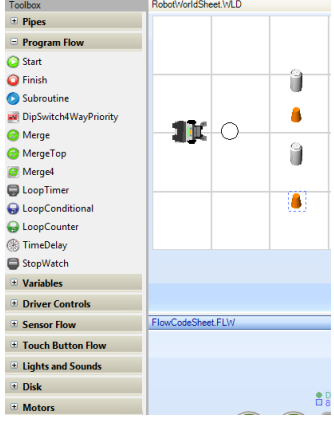
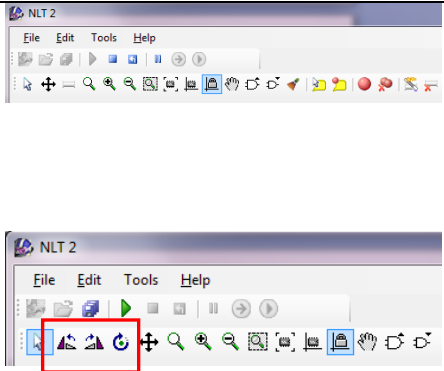
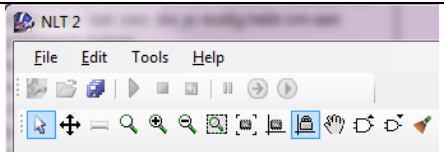
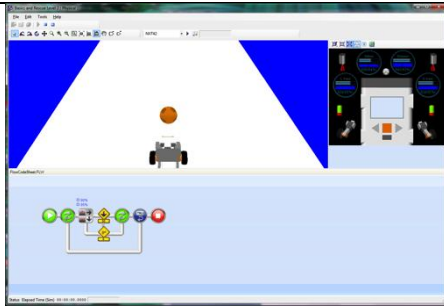
6		<p>To end this explanation, we return to the icons in the upper left side of the toolbar. We started the simulator using the green arrow. We also need to be able to go back to the program and we use the small blue square to do this. The next blue square with the arrow in it makes the simulator start the program again from the beginning. This is practical because you do not have to go back to the program every time and restart the simulator. So, if you want to run a program more than once, you can use the button with the blue square and the arrow. When you are done with the simulator, click on the blue square without the arrow.</p>
---	---	--



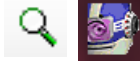
2A.3 Assignment: Adding Parts

In the RobotWorldSheet, also called the World View, you can create new robots and objects and insert them onto the playing field. We have a Ball, Cup, Wall and a Can (the container) as objects and a number of robots. You will mostly be working with the **Basic or Rescue Robot**, but other robots may be added as well.

2A.3 Adding Parts		
1		<p>Load Program 2A FleeBehavior, if you have not already opened it. Select the RobotWorldSheet by clicking somewhere on that panel. Please note that the ToolBox (on the left) only shows the parts that belong to the World View.</p> <p>Open “Basics and Rescue” and “RobotsNXT”. Now, you will see various elements that you can insert onto the playing field, as well as the two versions of the NXT robot that you can use. We will always use the Rescue Robot, because it has a distance sensor. The other robot does not have this sensor.</p>
2		<p>Select a Cup or a Can from “Basics and Rescue” and insert it onto the field. You can put all kinds of objects onto the playing field in this way. Almost all programs in the RLT lessons are created as a QuickStart project, in which all the necessary parts have already been selected for you, but you can change them at any moment.</p> <p>Also look at the other parts, like the Soccer (Football) World and the Rescue Arena.</p>
3		<p>You can move any element with the four-arrow cursor, both in the World and in the FlowCode View.</p>


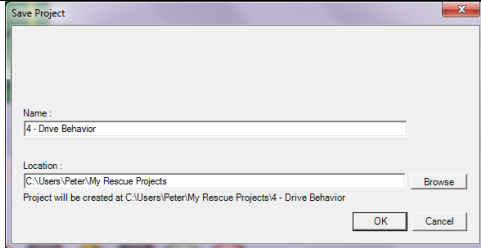
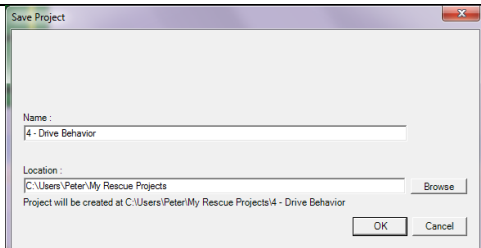
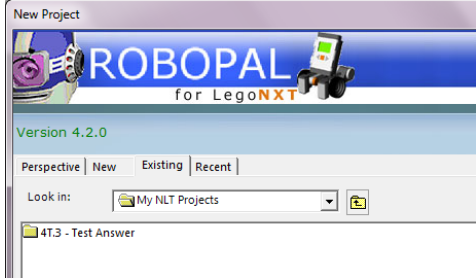
4		<p>If you place objects on the playing field, they will always appear on top of any other object that is already there. So, if you place a tile on top of a robot, the robot will disappear underneath the tile. Use the right mouse button and choose the 'Bring to Front' or 'Send to Back' option from the pop-up menu to move an object to the foreground or the background.</p>
5		<p>Click with your mouse on the bottom panel, the FlowCodeSheet. You will see that the ToolBox now changes and shows all the parts that are needed to make a program. Look at the contents of Program Flow, Driver Controls and Sensor Flow. You will find the icons that have been used so far. Later on, you will be using other icons as well. So take note of which of the two WorkSheets you are working on, as each has a different ToolBox.</p>
6		<p>Another important point is that when you are working in the FlowCode Sheet, the green Run button on top is not active, so you cannot start the simulator from there. You first have to activate the ToolBar of the RobotWorldSheet by clicking on the upper panel. This will activate the Run button and display a series of different icons in the ToolBar. In the WorldView, you can rotate objects with the rotation icons (Program View does not have these icons).</p>
7		<p>A little more about these icons. In the Program View (FlowCodeSheet), the last icon is a little brush that you use when your program becomes a little too messy. The Brush icon removes all the icons that are unconnected. The Hand icon is used to move the entire panel, while the magnifying glasses are used to change the size of the icons in the panel. There are some other icons, but we will discuss these later on.</p>
8		<p>You now have enough information to start experimenting with the Simulator. Experiment with how all its parts work, but make sure that you do not change an existing program by accident. Also, make sure that you always keep a version of a working program, so that you can go back to it in case something goes wrong. We will explain how to do this in the next assignment.</p>

Making a Backup Copy



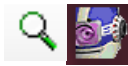
2A.4 assignment: Making and Using a Safety Copy

It is a good idea to make a copy of your program before you work on it. So, from here on, we will first make a copy of every program that we are going to modify.

2A.4 Making a Backup Copy	
1	 <p>We are going to make a copy of NLT 2A – Flee Behavior. Load this program and select ‘Save As’ from the menu. You can save your program in your own workspace or on a USB stick. Do not save your program in the student example directory or you will overwrite the original and you may want to go back to that in case of problems.</p>
2	 <p>Use the Browse button on the right of the window to find the directory (folder) where your programs are stored. If you are using a USB stick, select it; otherwise, select the folder where the NLT projects are normally stored.</p>
3	 <p>Change the name of the project. For instance: 2A.4 – Drive behavior. You will be making several new versions so that you can easily go back to a working version if something goes wrong.</p>
4	 <p>If you have saved a program, you may load it again later on. The programs that are built into RoboPAL are loaded from the startup screen. To reload a program that has been saved, select the File menu and then the Existing tab (instead of New) from the startup screen. Now, select the directory where you saved your program.</p>

You now have a new version of the program that you can modify.

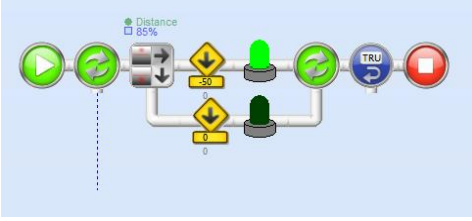
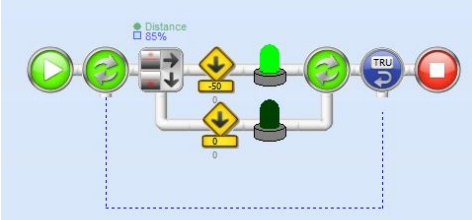
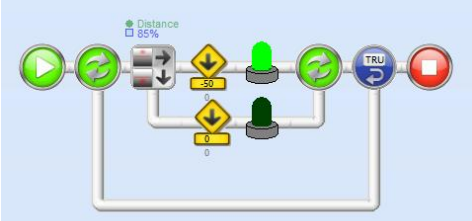
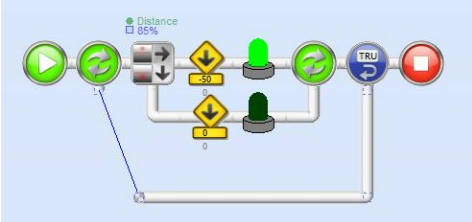
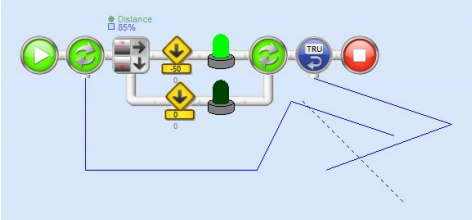
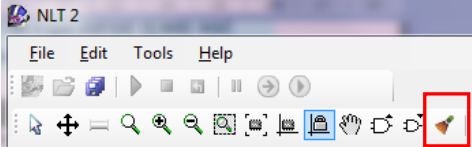
Connecting Icons Using Pipes



2A.5 assignment: Using Pipes

- In order to connect icons, you normally just place them side by side, but sometimes it is necessary to make longer connections. We use the Wiring Wizard to do this. The Wizard connects all open connections, but sometimes it does not know which icons to connect and you will have to connect them manually.
- We use the Pipe cursor to manually draw a line between any two icons.
- When icons are connected, the connecting line will move with them when you change their position. Drawing a line with the Pipe cursor is more difficult than using the Wizard, so we will try this out first.

2A.5	Using Pipes	
1		<p>Load program 2A.4 – FleeBehavior from the previous assignment. The program contains icons that are connected by lines called Pipes. We will experiment a bit with this program, but first save it as 2A.5 – FleeBehavior.</p>
2		<p>The two Wiring Wizard icons are on the top right side of the menu bar. The first is the Wizard, which makes connections, while the second one removes all connections. Click on the second icon to remove all pipes from the program.</p>
3		<p>Wiring is automatically inserted from left to right and from top to bottom. The Wizard finds unconnected icons that are open on the same level and connects them. So, click on the first icon and watch how the Wizard restores the connections.</p>
4		<p>If the icons are not on the same level, the Wizard does not work well. You can solve this by moving the icons so that the Wizard understands what belongs together. (Use the cursor with the four arrows to move icons.) Sometimes, however, the wiring is too complicated for the Wizard and you will have to connect the icons manually. We will now see how to do this. First, remove the pipes again and put the icons in the correct position.</p>
5		<p>You are now going to use the Pipe Cursor, which is in the top menu bar, next to the other cursor icons.</p>

6		<p>Select the Pipe Cursor and move with the mouse to the second icon on the left side. Click with the center of the cross on the connection point at the bottom of the icon. Release the mouse button and move the mouse downward. Then, click the left mouse button again. This allows the line to make a turn to the right.</p>
7		<p>Drag the line to under the blue icon. Click the left mouse button again to bend the line and move the cursor up to the connection point of the blue Loop icon. Now, click the left mouse button again. The line is now on the Loop icon, but it is still connected to your cursor.</p>
8		<p>Click the right mouse button to indicate that you have finished and release the line from the cursor. The line will remain connected to your mouse until you press the right mouse button. After you have pressed the right mouse button, the line will change into a pipe, but only if it has been connected correctly.</p>
9		<p>If the connection is not correct, a pipe will not appear, indicating that something went wrong. If you move the icons, the pipe will move with them. Use the cursor with the four arrows to move icons. Note that pipes can only be drawn horizontally or vertically.</p>
10		<p>If connections made with the pipe cursor do not work as you expected, you may end up with a mess of unconnected lines in your program. If two lines are drawn on top of each other, they erase each other and it looks as if they are not there.</p>
11		<p>Use the icon with the brush on it to clean up all these unused lines. You can also use the 'delete all links' option from the Wizard. If the program works fine again, save it.</p>

The Main Assignment

You will now make a small change to an existing program and modify the robot's behavior, changing its *FleeBehavior* into *CuriousBehavior*. Open *FleeBehavior* and run it in the simulator. If you have difficulty starting and using the simulator, look at the examples on how to use the demo programs. The detail assignments - indicated by the magnifying glass icon as in assignment 2A - can also help you with this. You may skip the detail assignments if you think you can work directly on the main assignment, but if you have problems go back to the detail assignments.



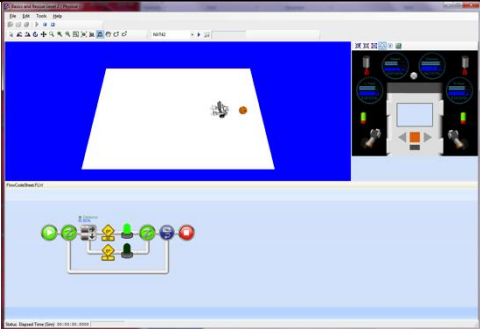
Now, look at the program *CuriousBehavior*. It is exactly the same as *FleeBehavior* and is ready to be modified. Your job is to modify the program so that the robot moves toward a ball or can in the simulator. Once this works, you have to make sure that the robot does not bump into the object and stops about 10 cm before it.

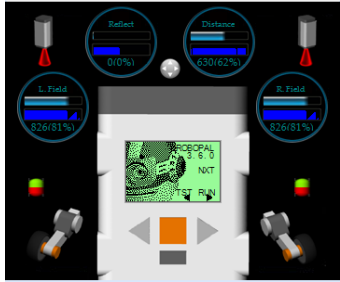
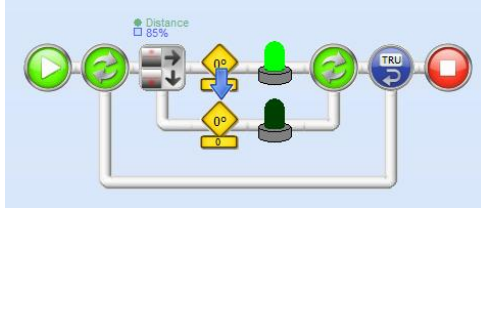
Important: You can change the direction in which the robot is facing in the simulator by pressing the right mouse button and dragging the mouse with the right button pressed.



2B.1 Assignment: Your first programming assignment - FleeBehavior

You are going to watch the robot display flee behavior. If this first step is too hard, first try out assignment 2A from the detail assignments.

2B.1 Making the Robot Flee		
1		Start RoboPAL by clicking on the startup icon on your desktop.
2		Select the first lesson 2A – FleeBehavior from the NLT student examples. Your teacher will have provided you with a directory where these examples are stored.
3		Press the green start icon to start the simulator. Use the mouse to place the robot on the left side of the field. Then, move the ball closer and further away and watch the value of the distance sensor change in the top right of the Control Panel. Find out from how far the robot can see the ball.

4		<p>Put the robot right in front of the ball, as shown in the picture in step 3.</p> <p>Start the program by pressing the triangular button on the right under RUN. This will start the FleeBehavior program.</p>
5		<p>Watch the blue arrow (the HighLight) in the simulator carefully. It shows the part of the program that is currently being executed.</p> <p>If you manually move the robot you will not only see the sensor value change, but also that another part of the program is activated.</p> <p>Watch carefully when the robot stops moving backward and try to understand what condition makes the robot stop.</p>



2B.2 Assignment: Watching the Program

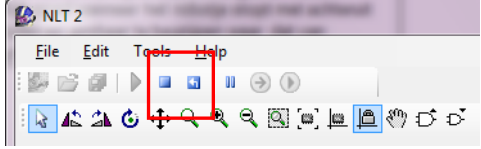
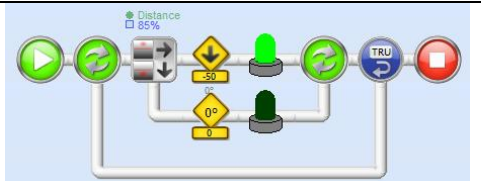
Stop the program and go back to the programming environment using the small blue square button next to the green start button. Let's have a closer look at the *FleeBehavior* program. In the pseudo code 4 below, you can see what the program code does.

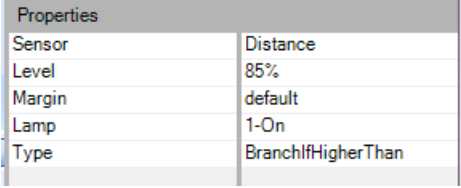
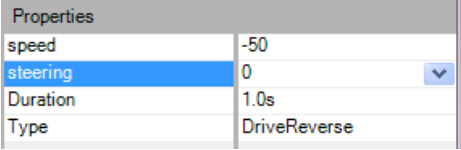
```

Start of FleeBehavior                                     // FleeBehavior
Repeat always:                                          // True means always
{
  if distance sensor value is larger than 85% then do
  {
    motor left and right -50                             // half power backward
    turn on green LED                                    // green = see bal
  }
Else
{
  motor left and right 0
  Turn green LED off
}
}

```

FlowCode 4: Pseudo Code FleeBehavior

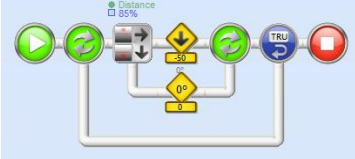

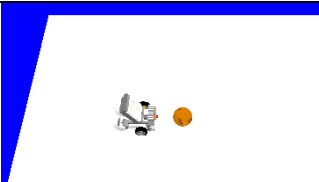
2B.2 Looking at Fleebehavior in RoboPAL		
1		<p>Close the simulator by pressing the blue square button. Look at the program in the FlowCodeSheet.</p>
2		<p>FlowCode 4 shows the pseudo code in which the processor is told what to do. See if you can find each step from the pseudo code in this program.</p>

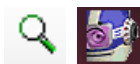
3		<p>You can make a number of changes to this program, but do not do this yet. First, let's take a look at what we can change.</p> <p>Select the icon that reads the sensor value (the gray icon with the two arrows on it, a BranchIfHigherThan icon). If you click on it, the properties shown on the right appear and you can see that the Distance sensor is in use. Underneath that you can see the Level and Margin that are important values for the Distance sensor. This percentage corresponds to the value that you see in the simulator when the ball is moved, while the Level field allows you to specify at what distance the robot must react to the ball or another object.</p>
4		<p>Select the icon to drive backward. Here, you will see that you can change its speed. It is set at half speed backward. You can also change the Steering, which is the size of the turn that the robot makes or, in other words, the difference in speed between the left and right wheels.</p>



2C.1 Assignment: Curious Behavior

We will now change the robot's behavior so that it becomes curious and moves toward the ball. The program that you will find in *CuriousBehavior*, is not ready to be curious. In fact, *CuriousBehavior* is identical to *FleeBehavior*. Look carefully at the explanation provided by pseudo code 4.

2C.1 Making the Robot Curious		
1		<p>Close the program by selecting Close Solution from the File menu. Select File New and then program 2C – Curious Behavior from the NLT student examples folder. This program is almost the same as FleeBehavior, but the LEDs are missing.</p>
2		<p>Change this program so that the robot moves forward instead of backward. It will now push the ball away. After you have made your changes, save the program as 2C.1 – Curious Behavior to make sure you do not lose anything if your program crashes, but first check to make sure that your program works.</p>
3		<p>As the robot pushes against the ball, it will roll away and the robot will continue to follow it. Experiment a little bit with the distance at which the robot reacts and with the speed, too.</p>



2D.1 Assignment: Effects of the Sensor Values

Now that you know enough about the simulator, you can start to experiment with the sensors. Watch how the sensors react to movement in the simulator.

2D.1	Experimenting with Sensor Values													
1		<p>You have changed CuriousBehavior to make the robot move toward the ball. Save the program again as 2D.1 – Curious Behavior to prepare for the next experiment.</p>												
2	<table border="1" data-bbox="293 710 727 891"> <thead> <tr> <th colspan="2">Properties</th> </tr> </thead> <tbody> <tr> <td>Sensor</td> <td>Distance</td> </tr> <tr> <td>Level</td> <td>85%</td> </tr> <tr> <td>Margin</td> <td>default</td> </tr> <tr> <td>Lamp</td> <td>1-On</td> </tr> <tr> <td>Type</td> <td>BranchIfHigherThan</td> </tr> </tbody> </table>	Properties		Sensor	Distance	Level	85%	Margin	default	Lamp	1-On	Type	BranchIfHigherThan	<p>You are going to experiment with the sensor values and see what happens. Select the gray 'BranchIfHigherThan' sensor and look at its Properties. Change the value of Level and make it higher. Test the program in the simulator and see what happens. What has changed?</p>
Properties														
Sensor	Distance													
Level	85%													
Margin	default													
Lamp	1-On													
Type	BranchIfHigherThan													
3		<p>Move the ball toward the robot or away from it. Watch the dial of the Distance Sensor and see how the value changes. You will see what value belongs to which distance.</p>												
4		<p>Now, lower the value and see what happens.</p>												
5		<p>Carefully watch the value of the Distance sensor in the Control Panel during your experiments. You will see that the value of the sensor varies at different distances from the ball. Make sure you position the ball in a straight line and look at how far the sensor can see the ball on different positions on the field.</p>												
6		<p>Try to determine what the sensor values are at different distances, as in the picture on the left. This will give you an idea of how a sensor works, what its field-of-view is, and at what distance the sensor detects the ball. This works not only with the ball, but also with the cup or can. Find out the exact value straight ahead of the robot at a distance of about 10 and 40 cm. You can estimate this based on the dimensions of the robot, which is approximately 15 cm in diameter.</p>												



2D.2 Assignment: Experimenting with Turns

So far, the robot has moved in a straight line, either toward a ball or away from it, but the robot can also make turns. In this case, different things may happen. Experiment with this.

2D.2	Experimenting with Turns	
1		<p>So far, the robot has moved in a straight line: backward with FleeBehavior and forward with CuriousBehavior. However, the robot can also make turns. Save your program as 2D.2 – Curious Behavior. Do not forget to put the sensor level back to 85%.</p>
2		<p>Replace the Drive Forward icon with one to drive in a Curve, either forward or backward. Start with a backward curve by using the DriveReverseRight icon from Driver Controls.</p>
3		<p>Put the ball and robot forward to give you space to drive backward. When you run this program, you will see that the robot drives backward, avoiding the ball. If you then make the robot drive forward, it will have successfully avoided the ball. We will take a better look at this behavior in later lessons.</p>
4		<p>Place a number of balls on the field, as shown on the left. You can find the balls in the Soccer World. Run the program and see what happens.</p>

Do Not Bump into the Ball

Everything that you have learned so far will now be used to make a final change to the program. If you think this is too hard, then study the detail assignments 2D.1 and 2D.2 first.

You have to keep the robot from bumping into the ball. Start by finding out at what value of the distance sensor the robot must stop. Once you have found this value, make the robot drive toward the ball and stop just in front of it.

You will need something new to create this program: a command to make the sensor look at two different values. First, the robot must check if it sees the ball (`sensor > 85`, so it will start moving, but it must also check if it is close enough to the ball and then stop. So you will need two conditions to decide if the robot needs to move. To do this you define a

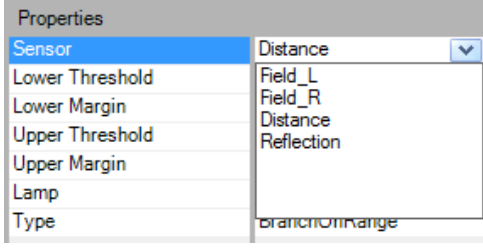
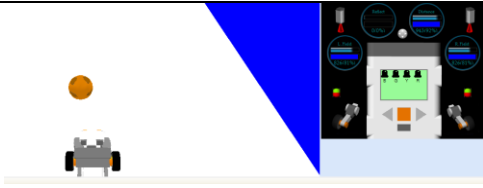
range in which you specify a lower and an upper margin. Within that range the robot needs to move, outside the range it must stop.



2E.1 Assignment: Making the Robot Stop

Change *CuriousBehavior* so that the robot no longer bumps into the ball, but stops right in front of it. You will first need to find out what the value of the distance sensor is at a distance of about 10 cm.

2E.1 Making the Robot Stop in Front of the Ball		
1		<p>First save your program as 2E.1 – Curious Behavior. Try to change it so the robot does not touch the ball, but stops in front of it. The ball will no longer roll away. It is not that easy. In the current program, the robot is instructed to start moving when the sensor value is higher than 85%. Find out what the value of sensor is at a distance of about 10 cm.</p>
2		<p>We need to tell the robot that it should move forward only if the value is higher than 85%, but that it must stop if the value is larger than the one you just measured. The closer you get to the ball, the higher the value becomes, so the range must be between 85 and the other, higher value, which represents a distance of 10 cm.</p>
3		<p>Select the Sensor Flow from the ToolBox and select the BranchOnRange icon. Remember? It is attached to your mouse. If you move the mouse to the FlowCodeSheet at the bottom, you will see that it moves with your mouse cursor.</p>
4		<p>Remove the BranchIfHigherThan icon with the delete button and insert the BranchOnRange icon in its place. Make sure that the pipe at the bottom is connected to it. You can do this by clicking on the pipe when the icon is in the right position. Move icons with the four-arrow cursor.</p>

5		<p>Now, you have to change the properties of the BranchOnRange icon. First, you need to change the Sensor. Select the Distance sensor and change the Lower and the Upper Threshold values. The Threshold tells the robot within which boundaries to react to a given value. There also is an additional parameter, the Margin, which is set to Default. (We will explain more about this in the next chapter.) Set this value to zero for both Margins or it will not work. We will explain why you have to do this later. Remember that the values of a sensor are always a Level and are expressed as a percentage. This will also be explained later on.</p>
6		<p>Once you have done this, save your program. Then, start the simulator and check if your program works. Change it until it does what you intended it to do. Experiment a little bit with the speed and maybe also with different distances. If it works fine, then try to test it on the robot.</p>

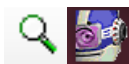
While you are changing and testing your program all kinds of things can go wrong. If you want to know what kind of things can happen and what you need to do to solve these problems, look at detail assignment 2F.

Error Messages

When a computer runs a program, the program must not contain any errors. If there are errors, three different situations can occur:


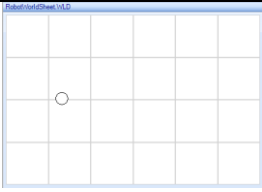
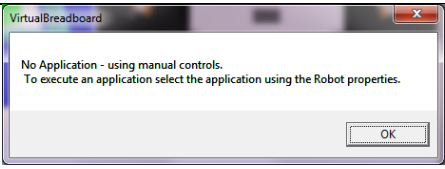
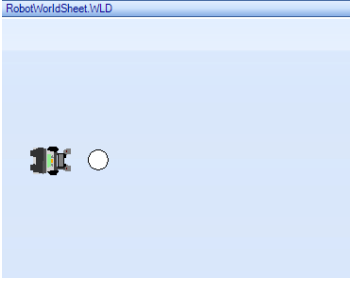
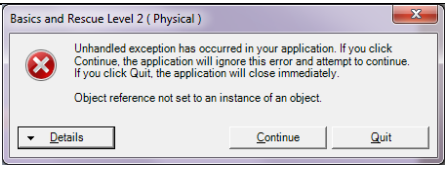
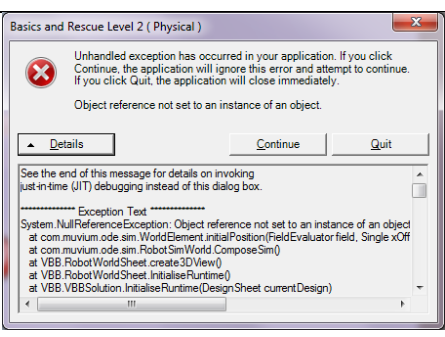
- The program does not know how to handle the situation and will not work. You will get an error message telling you what went wrong.
- The error is so severe that RoboPAL cannot handle it and crashes without producing an error message.
- The program does not stop, but does something completely different from what you expected. This kind of error is not detected by the computer, because it only does what it is told and not what is “expected”.

In the first situation, there is something wrong in your program. There may be many causes for this. In the next assignment, we will look at some of them.



2F.1 Assignment: Error Messages

You are going to look at what can go wrong with your program. As unexpected things can always happen, we always advise you to frequently save your program. If something goes wrong and your program crashes, you can always go back to the last working version you saved.

2F.1 Errors in your Program	
1	 <p>The Driver icon is missing from the Flowcode on the left. If you try to run this program, the simulator will just stop at that place in the program, because it does not know what to do next.</p>
2	 <p>If you remove the robot from the RobotWorldSheet or forget to insert one, the simulator gets confused and will show the playing field, but the Control Panel disappears and you cannot start the robot program.</p>
3	 <p>If you accidentally remove the program you made from the FlowCodeSheet, RoboPAL will not start and you will get an error message. There are a number of these messages that help you to identify what went wrong.</p>
4	 <p>The situation becomes more serious when there is no playing field in the WorldView. This will not happen so easily in our lessons, because QuickStart always inserts a playing field, but you could accidentally remove it. With the Grid field this is harder, as it consists of a number of tiles, but in the later lessons, where you only use the Rescue field, it can happen.</p>
5	 <p>When you use a program that contains errors, RoboPAL may crash and produce an error message. To find out what went wrong, you can click on Details.</p>
6	 <p>This kind of error is a so-called “Unhandled Exception Error”. RoboPAL is in a situation that was not anticipated and shows you what it was doing when the problem occurred. You will see a list of the steps that the program was trying to perform and that may help you determine what caused the error. In this case, you see a NullreferenceException (1st line) in RobotSimWorld (3rd line). This means that RoboPAL was trying to start the RobotWorld but could not find it.</p>
7	<p>It is even more serious when RoboPAL crashes without producing any error message. Sometimes, you will get an error message from Windows, telling you that the program crashed. In this case, you do not know what caused crash and you will have to restart RoboPAL. This is why it is important to always save your program before you start testing, so you can always go back to the last saved version. You can then look at the last changes you made, which probably caused the problem.</p>

Debugging

When your program is ready to be tested on the simulator, you will often discover that it does not do exactly what you had in mind. You will then have to find out where the program goes wrong. RoboPAL checks many parts of your program to see if everything it needs has been specified, but the computer cannot determine what you intended your program to do. So, if a program does not do what you expected, there must be an error in the logic of your program. You either forgot to insert something or there may be something wrong with the logical order of your icons.

This kind of mistake is very hard to find, because most of the time you only see what you think your program is supposed to do. The true art of programming is to put yourself in the position of the computer. You have to look exactly at what is specified in your program, not at what you meant it to do. You will gradually get better at looking at a program in this way.

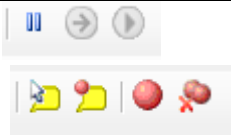
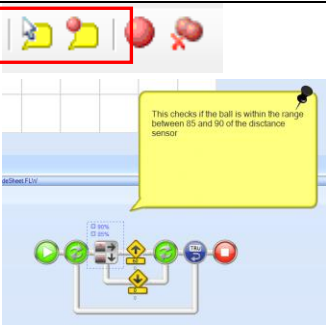
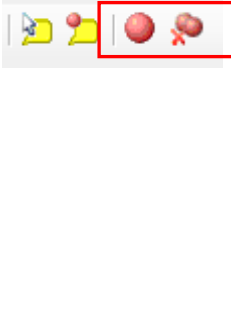



To help you determine where a program goes wrong, there are a number of tools available that we will describe briefly. The process of finding errors in a program is called ‘Debugging’. In early computers that used very large components, like switches and relays, a fly or other insect (also called bugs in English) would often get trapped between the contacts of a switch and prevent it from working correctly. The computer then contained a ‘bug’ and the programmers had to find where the bug was and remove it. We still call an error in a program a ‘bug’ and the process of solving errors is called ‘debugging’.



2F.2 Assignment: Debugging

You are going to look at a number of debugging tools and use them in the program *CuriousBehavior*. Load the program 2F - CuriousBehavior. If you are working on a computer with a small screen, some parts may not be visible. You can use the mouse to move the borders of the panels and make them bigger or smaller.

2F.1	Debugging Your Program	
1		<p>The simplest way to check what your program is doing is to turn a led on or off in your program, or use a beep. You will do this frequently and it is a quick and easy method, but sometimes it is not enough. You may need more detailed information about what is going on.</p>

2		<p>RoboPAL contains a Debugger that allows you to check all kinds of things while your program is running. A number of icons is available and we will now look at them in greater detail.</p>
3		<p>Comment. You can use the Comment property and insert a description for every icon. When you activate the first icon with the arrow in it, a comment will appear in a pop-up window as soon as you select the icon. Turn on the first icon and click on each of the icons in the program and look at the comment. You can drag the pop-up window with the mouse to a position where it does not cover a part of the program you want to see.</p>
4		<p>Breakpoints. By clicking on an icon and selecting the red ball, you insert a Breakpoint on this icon in your program. When you then start the program, it will run until it reaches the breakpoint and then stop and show you what is happening. The last icon serves to remove all breakpoints from the current part of the program you are working on. You can also remove a single breakpoint by selecting the breakpoint icon and then clicking on the ball again.</p>
5		<p>Stepping. As soon as a program stops at a breakpoint, these two icons become active. Now, you can follow the program step-by-step, so that you can see exactly what is happening. When you push the Run button (the triangular arrow), the program continues normally until it reaches the next breakpoint. The pointed arrow is the Step button.</p>
6		<p>Pause Mode. If you do not use breakpoints in your program, you can use the first icon to pause your program. The blue arrow shows at what point the the program is halted and you can progress from that point step-by-step and use the Run button to resume your program.</p>
7		<p>Inspectors. As soon as your program has stopped, either via a pause or a breakpoint, a pop-up screen appears displaying the content of the variables in your program. In the example, a counter is increased and as you step through your program you can look at the contents of the counter.</p> <p>In this program, we are using a few icons that you have not seen before, so do not worry if you do not understand them, we will explain them in later lessons.</p>

Comments

Putting comments in a program is a good habit. They clarify what a program is supposed to do. This is not only practical for your own understanding (after a while you may look at your program and not remember why you did something in a particular way), but it also especially important for other people, who might look at your program and need to understand how it works.

In RoboPAL, there are two ways to insert comments in a program:

1. Include comments with every icon in your program. In the previous assignments, you have already seen examples of this. If you turn on the Comment icon in the Debugger Toolbar, your program will automatically show the comments in a pop-up window when you select the icon. You have to activate the comment button to see the pop-up comments. During debugging, these comment may also be viewed by using the icon for displaying a breakpoint comment. In this case, the comment is displayed when it reaches the icon during debugging.
2. The second way to include comments in your program is by using a comment block in the Documentation part of the ToolBox. The Comment icon allows you to include text in your program to explain what is happening in the program. We will be using this feature in later programs, too.

2.6 In Practice

RoboPAL is a development environment with a built-in simulator and debugger. The programming language is a graphical language based on icons. In real life, professional programmers use an Interactive Development Environment (IDE) such as Eclipse, which is used in the Java version of this module. Other well-known IDEs include MicroSoft Visual Studio and Oracle jDeveloper. Most major manufacturers have their own IDE. Most are very similar and all of them have a debugger.

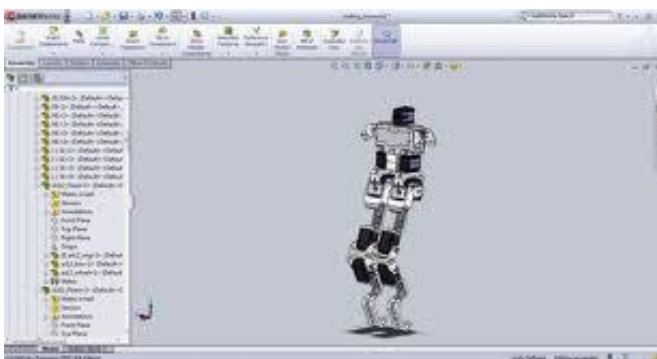


Fig 32: Bioloid simulator.

Simulators are usually available for applications in which the use of real equipment is too expensive or dangerous, as we have seen in the case of flight simulators. Simulators are also entering the business community. There are business simulators that help people to plan and calculate the consequences of their projects.

Nonetheless, simulators are most frequently used in robotics. The development of a robot is simplified by using a simulator. Having to load the program onto the robot and then testing it takes a lot of time. Almost every modern robot has its own simulation environment that is used by programmers and developers. You can look at examples such as the BioLoid, the RoboNova or the Aldebaran NAO robots.

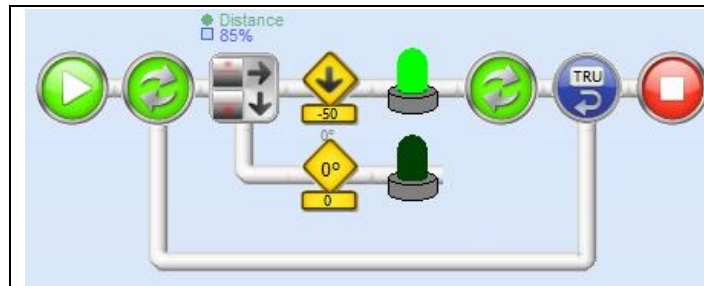
2.7 Test

For some of the questions below, you may need to look at the detail assignments.

1. Explain what the following parts of RoboPAL are used for:
FlowCodeSheet, WorldViewSheet, ToolBox, Simulation, Control Panel



2. What is the meaning of this icon in RoboPAL?
3. What is wrong with the program in FlowCode 5?



FlowCode 5

4. What will happen if you run this program?
5. How are errors indicated in RoboPAL?
6. What is the function of a debugger? Of a breakpoint?

3. How Does Your Robot Work?

In chapter 2, you worked with the simulator. In this chapter, we will move on to the real world and show you how to work with the robot itself. To transfer your program from the simulator to the robot, your program code has to be translated into a machine language that the processor in the robot will understand. This process is called *compiling* or *packaging*. The “translated” program is uploaded to the robot memory via the RoboPAL simulator. We will now look at how the program is executed by the robot.

3.1 You will learn

- how to make a connection with the Lego NXT robot
- how to upload a program and have the robot execute it







3.2 You will need

- a computer with RoboPAL
- a RoboPAL dongle
- charged batteries / adapter
- a table to make your robot move on
- the Grid field (a white surface)
- the *CuriousBehavior* program



3.3 You will experiment with

- making contact with your robot
- finding the NXT robot in RoboPAL
- uploading a program to the robot from the computer
- trying out the program on the robot.

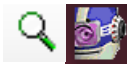
Type	#	Assignment	Description
	3A	RoboPAL	Loading the Firmware
	3A.1	Basic Settings	Basic settings in RoboPAL
	3B	CuriousBehavior	Loading the Program
	3B.1	Uploading	Uploading the Program
	3B.2	Reading Sensors	Reading the Sensors
	3B.3	Testing	Making the Robot Move
	3C	Test	Written test on the first three chapters

3.4 After following this chapter, you will be able to

- explain how to write a program, how to test it, how to upload it to the robot and run it on the robot
- use the RoboPAL dongle.

3.5 The Assignments

In order to use the robot, we must first connect it to a PC. We must also make sure that we can upload our program to the robot.



3A.1 Assignment: Setting Up RoboPAL

Before you upload your program onto the NXT storage, you have to make some preparations. You must make sure that the RoboPAL firmware has been installed on the NXT and that it is set to Level 2.

When this is done, you can start with assignment 3B. Check first, however, that everything has been installed correctly.

Installing the Firmware

To run the Lego NXT robot with RoboPAL you need to install the RoboPAL firmware on the robot. In most cases, this will have already been done, so you can safely skip this step. If, however, it has not already been done, ask your teacher to load the firmware onto the Lego NXT brick. When you start up the NXT by pressing the orange button, you will see the following picture in the NXT display:



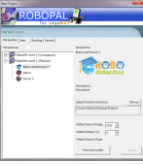
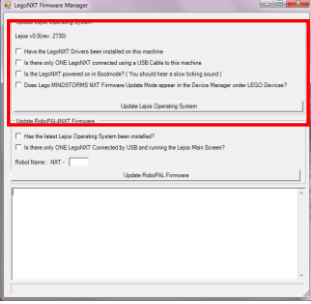
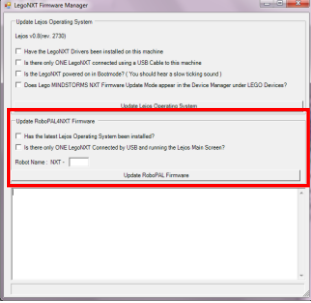
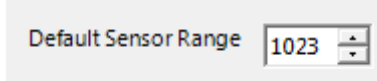
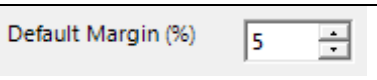
Fig 33 RoboPAL Startup Screen

If you see this picture something entirely different, the firmware needs to be loaded. The version number in the display has to be at least 4.1.4.

Basic Settings of RoboPAL

In these lessons, you will work with RoboPAL Level 2, also known as the Physical Level. First, however, you have to setup RoboPAL to make your robot work correctly. Then, you need to set the so-called Default values. Default means standard or, in other words, the values that the program refers to when you use the sensors. Using default values means that, in most cases, you will not have to make any additional changes to your program. The two values that we need to set are the Sensor Range, so that RoboPAL knows the maximum value of the sensors. The standard is 1023. As all icons in RoboPAL use percentages, the program needs to know the maximum value. So, before you continue, make sure that RoboPAL has been set to the correct level.

To make sure your computer is ready to make contact with the robot, follow these steps:

3A.1 Basic Settings of RoboPAL		
1		After starting RoboPAL, make sure that the startup screen is visible, otherwise select File New. Then, select the Perspective tab and RoboPAL Level 2 for NLT Part 1. Now, press the Apply button.
2		If the firmware for the NXT has not been installed, do it now. Press the Firmware button to see the screen that allows you to upload the firmware. You must also specify the number of your robot, here. Your robot needs to be connected to the PC via a USB cable to complete these steps. The screen is divided into two parts. The first part concerns the installation of the LeJos operating system. If it is already installed on your NXT, there is no need to do this again. Otherwise, look at the detailed instructions for RoboPAL4NXT that are included in the teacher's guide, as extra are steps required for this.
3		The second part is used to upload the RoboPAL firmware. Make sure that the robot is on and displaying the LeJos main menu. This step will NOT work if the standard RoboPAL screen is visible. If that is the case, ask your teacher to set the NXT to the LeJos menu for you. Now, insert your robot's number and press the button to upload the firmware.
4		Set the two default values above the Apply button. The first one is the default sensor range that must be set to 1023, the maximum value a sensor can return.
5		Then, select the default sensor margin and set it to 5. This is the safety margin that we use when reading a sensor value (i.e., to check if the sensor sees Black). The value that you specify here is used to check the range of values that the sensor uses to recognize a certain color. If, for instance, you measure a value of 19% for Black and specify a margin of 5%, everything between 14% and 24% will be considered to be Black. The margin that you specify here is used automatically in any program in which an Icon uses a Margin. If you choose Default, it will automatically use this value. You can always overwrite the Margin with your own temporary value. This is then only used when that particular Icon becomes active.

Uploading the Program



Fig 34: NXT Control Panel

We will now upload a program to the robot. The robot needs to be switched on and connected to the PC. Please remember these steps, as you will have to repeat this procedure in all following lessons. You have already seen the “brain” of the robot in the Control Panel when you used the Simulator.

We are going to load the *CuriousBehavior* program onto the NXT robot memory and then work with the robot.

Once you have loaded the program, you can use the RUN button to start the program on the NXT just as you did in the simulator. The screen also displays “TST.” This is used to read the sensor values and will be explained shortly.

Once it has been loaded, you can restart a program many times by selecting RUN from the main screen of the NXT. This always starts the most recently loaded program. If you want to stop the robot, press the orange button. If this does not work, switch the robot off by pressing the orange button together with the dark gray button underneath it.

You will notice that the real NXT robot reacts slightly differently from the simulator. It has more problems with lighting conditions and shadows. So, there will be a lot of trial and error to make it work correctly.

Checking the Batteries

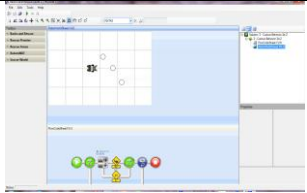
In assignment 3B, you are going to connect your robot to the computer and upload a program to the robot. Make sure the robot batteries are fully charged before you start the lessons.

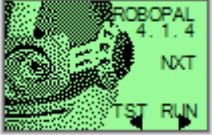
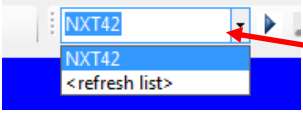

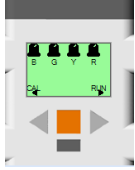
You can use the Lego NXT with regular alkaline batteries or with the Lego rechargeable battery pack. Always switch off the robot when it is not in use and between tests.



3B.1 assignment: Uploading the Program

You are going to load the *CuriousBehavior* program onto the NXT flash memory. Insert a RoboPAL dongle into a free USB port on your computer and make sure the batteries are fully charged. If you are using the Server version, you need to connect to the server first instead of using a dongle.

3B.1 Uploading the Program	
1	 A screenshot of the software simulator interface. It shows a top menu bar, a central workspace with a grid and a blue robot icon, and a bottom toolbar with various icons for simulation control. <p>Load 3B <i>CuriousBehavior</i> onto the simulator to upload it to the flash memory of the NXT. Make sure the program is ready in the simulator and that it works properly.</p>

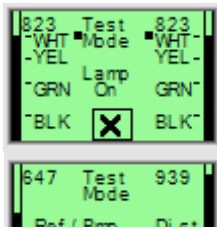
2		<p>Start the Lego NXT robot by pressing the orange button. When the NXT is ready you will see the word NXT followed by a number on the display. (You do not see this number in the simulator). Remember this number. It identifies your robot and you will need it to load programs.</p>
3		<p>In the simulator, just above the playing field, there are a number of boxes that you can use to select information. When you click on the small arrow on the right, you will see a list of all the NXT robots your computer knows about. If your computer has not identified any robots yet or if your robot is not listed, you need to select the last line <refresh list>. Make sure that the special RoboPAL dongle is inserted in one of the USB ports before continuing or when using the Server, make sure you know how to reach the server. When you select <refresh list>, the computer will check which NXT robots are on: you will see their names in the list and you can select your robot. So make sure your robot is switched on.</p>
4		<p>Once your robot has been identified, select it from the list and then click on the blue triangle in the next box on the right to upload the program. You will briefly see an orange 'progress bar'. When the upload has been completed you will see a message. The NXT will beep when it has received the program and you can then start using your NXT just like you did on the simulator. If the upload does not work, try it once again or ask your teacher to help you. You can try to switch the NXT off and on again and try once more. You switch the NXT off by pressing the orange button together with the dark gray button underneath it.</p>
5		<p>Start your program by pressing the RUN button. You will then see a menu in which you need to press RUN a second time, just as you did in the simulator. Turn the robot off when it is not in use as the batteries run down quickly and it takes more than half an hour to recharge them. Remember the upload procedure, because from now on you will need to do use it every time you want to test a program on the NXT.</p>




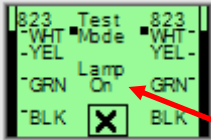
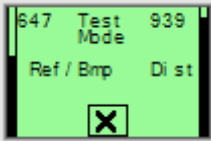
3B.2 Assignment: Reading the Sensors

You will need to write down the sensor values, just like you did whilst using the simulator. As the sensor values of the NXT robot differ from those of the simulator, you will need to determine the real values first. Then, you may have to modify your program in order to use the robot. Please note that the NXT displays less information than the simulator. Indeed, automatic calibration is not used with the NXT robot in these lessons, but is only a part of the program on the simulator.

Reading Sensor Values



Once the program has been uploaded you can test it, but first, as we have seen in previous lessons, you need to know what the sensor values are. In the simulator, you could use the dials in the Control Panel, but now you need to know the real values the robot sees. The NXT has a small program that displays sensor values.

3B.2		Reading the Sensors
1		Start the Lego NXT robot again so you see the startup menu. Now, press the TST button, the triangular button on the left. The screen view that appear here also works in the simulator. It's a bit simpler on the NXT robot than on the simulator; moreover, color indicators like WHT, YEL, GRN and BLK are not available on the NXT.
2		You will then see the screen on the left. It displays the light sensors, also known as ground or field sensors. We will explain more about them in the next chapter. Press the triangular button on the right and you will see a second screen. This one is almost the same, but instead of Lamp On, you will now see Lamp Off in the center. This is used to read the value of the ambient light with the red Leds switched off. We will not be using this feature in these lessons.
3		Press the right button again until you can see the screen view on the left. This is where you can measure the value of the Distance and the Reflection Sensors. We are not using the Reflection Sensor in these lessons, but it is basically a forward-pointing light sensor.
4		Hold an object in front of the Distance Sensor and watch the number on top. This number indicates how far the object is from the sensor. You will have to perform a calculation. The maximum value of the sensor has been set to 1023. Here, we see a value of 939. This means $939/1023 \times 100$ or ca. 92 %. This is the number that you will see and use in the simulator any time a sensor value is used.
5	Measure the value at a distance of about 10 and 25 cm. as these are the values that you need in your program. Calculate the percentage and then change your program so that it works with the NXT. Remember that the sensor values differ between sensors and, if your robot indicates 90%, the same distance may be interpreted as a higher or lower value on another robot or with another sensor.	

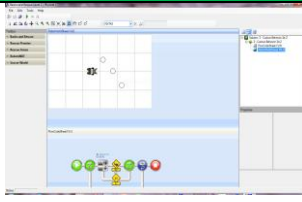


Making the Robot Move Forward

You are going to make the robot drive forward, but before you do so, you have to insert the measured values into your program.



3B.3 Assignment: Making the Robot Move Forward

You are going to change the sensor values for your program and then upload it to the robot. You are now ready to make the robot move.

3B.3 Making the Robot Move	
1	 <p>Continue using the program from 3B, but change the sensor values in the icon where the sensor value is indicated as a number and replace it with the percentage from your measurement.</p>
2	 <p>Make sure the robot startup menu is visible. If the TST button was pressed, you can use the orange button to return to the main menu. The robot is now ready for a new command.</p>
3	 <p>Press the RUN button, just as you did on the simulator. You will see four simulated lamps on the screen that represent four leds for Blue, Green, Yellow and Red. You will see the red led blinking. This means that the program is running. This led is called the heartbeat. If the red led is not blinking, something is wrong with your program.</p>
4	<p>If your program is correct and you have entered the right values in the program, when you hold your hand in front of the sensor, the robot will move toward you. If it does not do this, check if your program works on the simulator. If it still does not work, check that it has been loaded correctly and that all cables on the robot are connected correctly. The cables for the motors should be on ports A and C and should not be crossed. The sensors should be connected to ports 1 and 4 and the distance sensor to port 3.</p>



Test 3C: Written Test on Chapters 1-3

There is a written test once you have completed these three chapters. (It is not the same test that you find at the end of this chapter.)

3.6 In Practice

In this module, we only use the Lego MindStorms robot or the JoBot if you are following the Java version. In practice, however, robots are being used more and more. The first were the so-called industrial robots, like the ones used in factories. Just like computers, they are a part of modern society and we could not work without them.

The military also uses robots in a variety of situations: not only to detect and remove landmines, but also as unmanned and autonomous airplanes and boats.

Slowly, robots are entering our daily lives. The first domestic robots are now appearing on the market; manufacturers such as Honda and Toyota have been working on their development for years.



Fig 36 TU Eindhoven Soccer Robots: runner up world champion for the fourth time in 2012.

Nowadays, however, robots are mostly used by scientists for research purposes. This is done to gain a good understanding of the problems involved in the development of robots. Robot Soccer (football) is one of the most important research topics. In this discipline, a number of robots have to cooperate, learn about their environment and communicate with each other.

The scientific community has set itself a goal: allow a team of soccer-playing robots to defeat the human world champions by 2050. The RoboCup World

Championships are held yearly and progress is made every year. We are still far from reaching our goal, but soccer matches between robots already are very spectacular.

3.7 Test

1. What needs to be done to the NXT before you use it with RoboPAL?
2. Where can you find the 'number' of your robot?
3. What mode should RoboPAL be in to upload a program onto the NXT?
4. If you are not able to upload a program to your robot, what could be wrong?
5. How can you read the values of the sensors?

4. Driving Over the Rescue Field



Fig 37: rescue field.

Now that you know how to work with the simulator and the robot, you are ready to make the robot do even more.

We are going to make it follow a road on a field and move along an imaginary square. You will find that this is not easy and really demands a different approach, but first we will first try the naïve approach. The way this is programmed seems to be a rather easy solution. In the second and third parts of this module, we will look at other solutions and point out further possible approaches.

4.1 You will learn

- to set the speed of the robot
- to make the robot stop, after a certain time, by counting
- to let the robot make turns
- to show a message on the display

4.2 You will need

- a computer with RoboPAL and the simulator
- a robot
- the rescue field
- the program *DriveBehavior*













4.3 You will experiment with

- making the robot move forward for 2 seconds and then stop
- letting the robot make turns
- programming it to follow part of a road

4.4 After following this chapter, you will be able to

- describe the properties of a robot environment from the simulator and the actual situation on the rescue field
- explain why following a fixed pattern is difficult
- explain what a subroutine is and be able to use one
- make a simple program in which a robot moves straight ahead and makes turns

Type	#	Assignment	Description
	4A	DriveBehavior	Changing the robot's speed
	4A.1	DriveBehavior	Adapting the 'Drive' speed
	4B	DriveBehavior	Starting the program
	4B.1	Code change	Showing the current status
	4B.2	Code change	Cleaning up your program
	4C	DriveBehavior	Driving forward and stopping
	4C.1	DriveBehavior	Making the robot drive forward
	4C.2	DriveBehavior	Making a turn
	4D	SubRoutine	Using a subroutine
	4D.1	SubRoutine	Making a subroutine
	4D.2	SubRoutine	Using the subroutine
	4D.3	SubRoutine	Making the Main program
	4E	SubRoutine	Driving in a square
	4E.1	SubRoutine	Making the robot move in an imaginary square

4.5 Assignments

From here on, we will make quite a few changes to several programs. So, it will be wise to save your programs frequently.






4A.1 assignment: Let the Robot Stop by Counting

When you start the program you will see that the robot moves forward to the border of the field or just beyond it. Driving to the border of the field is not exactly what we want. The robot should move forward and then stop.

So, we need to tell the robot that it needs to stop moving after a while. We use the stopwatch, which counts how long the robot has been moving forward, to do this.

To make the robot faces a given direction, you have to give it the right speed and modify the timing.

4A.1		Modifying the Driving Speed
1		Load the program 4A Drive Behavior and look at the FlowCodeSheet. It is called RobotDrive here.
2		On the left, you can see the program that makes the robot move forward at a speed of 60 for 1 second.

3	<table border="1"> <thead> <tr> <th colspan="2">Properties</th> </tr> </thead> <tbody> <tr> <td>speed</td> <td>60</td> </tr> <tr> <td>steering</td> <td>0</td> </tr> <tr> <td>Duration</td> <td>1.0s</td> </tr> <tr> <td>Type</td> <td>DriveStraightAhead</td> </tr> </tbody> </table>	Properties		speed	60	steering	0	Duration	1.0s	Type	DriveStraightAhead	<p>Use the properties to change the speed to 80 and then set the stopwatch to 5 seconds. Please note that DriveStraightAhead also has a Duration. We are NOT going to use it. We will set the timing through the stopwatch because it makes it easier to change the program again.</p> <p>Select the numbers from the stopwatch drop-down list or type in a number. In RoboPAL, we always use a decimal point (not a comma) as is customary in all programming languages.</p>
Properties												
speed	60											
steering	0											
Duration	1.0s											
Type	DriveStraightAhead											
4		<p>Start your program in the simulator and move the robot to the starting point as shown on the left. Use the right mouse button, if you need to make the robot face a different position. Use the second icon above the Control Panel to set the field view to 2D. This makes it easier to place the robot in the right position.</p> <p>Start the program by pressing the triangular RUN button. The robot will move forward and beyond the field. This is not what we want. Change the program so that it stops right in front of the first curve to the right. Watch the counter in the bottom left and determine after how many seconds you must stop it. Change the program and save it when it works as 4A.1 – Drive Behavior.</p>										


Tip: If you have been using RoboPAL for a while, you will have made several changes to your program. Save it after every change so you can always restart with the most recent version in case something goes wrong.

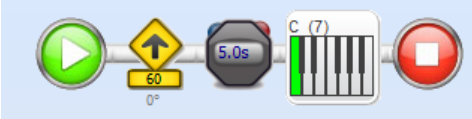
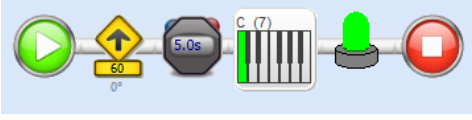
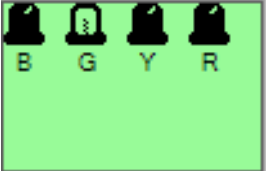
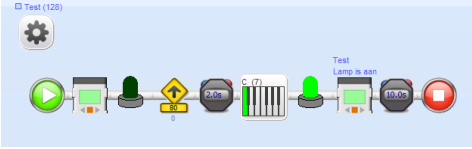

Indicators



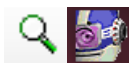
4B.1 assignment: Showing the Current Status

While your program is running, it can be useful to know how far it has progressed. There are a number of ways to do this. We can turn the leds on the display on and off or we can have it sound a number of beeps to understand what part of the program it is running.

4B.1 Showing the Current Status	
1	 <p>Save your program as 4B.1 – Drive Behavior. When the robot reaches the first curve, we want to it to signal that it has arrived. We have several options. One is to turn on one of the Leds on the NXT display or to show a text. The other one is to make the robot sound a beep.</p>

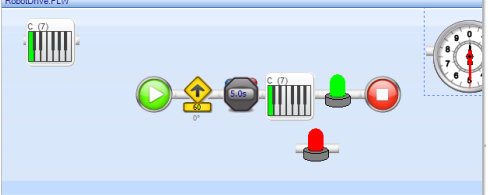
2		<p>First, we will let the robot sound a beep when it reaches the first curve. Select the MusicalNote from Lights and Sounds and insert it right after the stopwatch. If you do not change anything it will play a C, but you can modify the properties to change the pitch.</p>
3		<p>We can also turn on a lamp on the LCD screen. You do this with the led icons, which also are in Lights and Sounds.</p>
4		<p>Test this program in the simulator and then on the NXT to find out which indicator is best for you. The lamp is practical, but in this case the program stops and returns directly to the main menu, so you will not see the lamps. The lamps are only useful as long as the program continues to run.</p>
5		<p>A better possibility is to display a short text on the LCD screen. You do that with the LCDDebugMessage icon from Lights and Sounds. You can choose to display three short messages or show the content of a variable. Use the properties of the LCDDebugMessage for this. You can insert either a variable or a text message in each of the three fields.</p>
6		<p>While you are running your program you will see the text and the LEDs on the screen. Do not forget to turn the leds and/or the messages off when the situation they are representing no longer exists. Old indicators that are left on are frequently the cause of misunderstandings.</p>

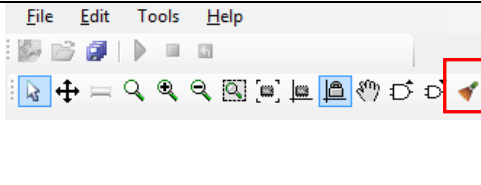
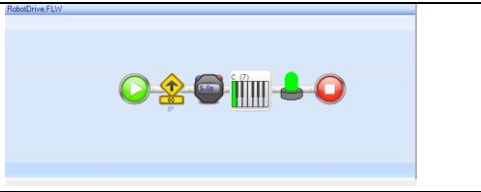

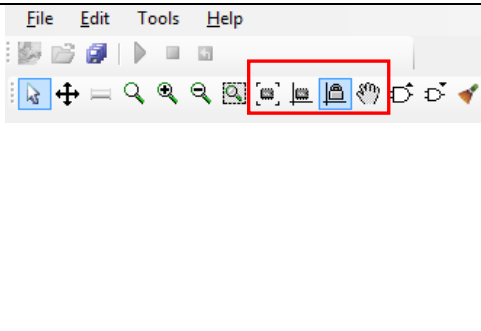
Cleaning Up Your Program



4B.2 assignment: Cleaning Up Your Program

If you make frequent changes to your program, it may end up not looking very tidy. The program may not be nicely lined up or maybe you think it should look different. There may also be some icons in it that you are not even using any more. RoboPAL provides tools to help you clean up your program.

4B.2 Cleaning Up Your Program	
1	 <p>If you have been busily working and have tried out various possibilities, your screen might look like the one on the left. Your program is in the middle and there are a number of unused icons scattered around it.</p> <p>Insert a number of unused icons into your program and then try the following steps, to clean it up.</p>

2		<p>Naturally, you can remove each icon manually, by clicking on them and then pressing delete, but if you use the icon with the brush, RoboPAL will automatically remove all the icons that are not connected to anything.</p>
3		<p>Your program should now look as you meant it to. You can move each icon individually with the move cursor, but you can also move all of them together.</p>
4		<p>First, select all icons that you want to move with the four-arrow cursor. You can select multiple icons by dragging a rectangle around them with the cursor, but you can also select individual icons by holding the Shift key down while clicking them. Then, press the right mouse button and keep it pressed while dragging the icons to the place where you want them and then releasing the mouse button.</p>
5		<p>You can also use the hand cursor to move all the icons in your program around. Be sure to reset the "origin" (that is the position of the left upper corner) so the icons are neatly placed on the left side. Finally, you can also use the magnifying glasses to make the icons larger or smaller (or zoom in or out with the mouse). Try out the various possibilities, so that you know what is available and how these cursors work.</p>

Going Back to an Earlier Version

If your program does not work and you want to revert to an earlier version, go back to the original of DriveBehavior or reload the last saved version of your program. You can then insert the most recent changes and resume where you left off.


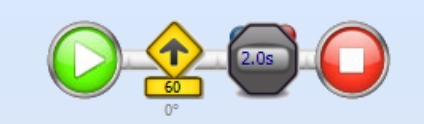
Making a Turn



4C.1 assignment: Moving Forward and Stopping

We will now work on the rescue field with the Rescue examples. You will be adding new parts. Follow the detailed assignments if you find these steps too difficult.


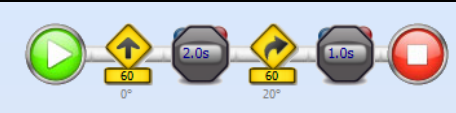

Please note that the program also contains instructions on how to show information on the LCD display. More details on this can be found in Assignment 4B in this chapter.

4C.1 Making the Robot Move Forward		
1		Load program 4A Drive Behavior again, to start working with a fresh copy of the program and save it as 4C.1 – Drive Behavior
2		Set the value of the Stopwatch to 2.0 seconds. Save your program every time you make changes.



4C.2 assignment: Making a Turn by Counting

A two-wheeled robot makes turns just like a wheelchair. If you make the right wheel turn slower than the left wheel, the robot will turn to the right. You will now try to create a new part for an existing program to make the robot turn. At the end of the turn, the robot must stop.

4C.2 Making a Turn		
1		Save your program as 4C.2 – Drive Behavior. You are going to have your robot make its first turn.
2		Add a DriveSwurveRight icon to the program followed by another Stopwatch. You must be sure the turn is not too sharp, so that the robot can follow the curving black line. The standard setting for Steering is 60, but this is too sharp. So, you have to change the Steering property.
3		Start the simulator and find out step-by-step what kind of turn you need to make and how long it should last so that the robot stops at the yellow line. Save your program and write down the values that you are using, as you will need them again later.

Using a Subroutine

Your robot is now able to follow the black line until it reaches the yellow line. You did this by using a combination of a Driver icon and a Stopwatch icon. The Driver icons also contain a Duration that you can change in their Properties. Although this is a good solution, we are going to show you something else that will help you further develop your programs: the use of a Subroutine.



4D.1 assignment: A More Flexible Program - Using a Subroutine

Whenever we need to program several movements, we end up using the same combination of code with different values for the Driver and Stopwatch icons. This is not very efficient as it takes up extra space in the

program. In addition, the code tends to become lengthy and unreadable. Too much code makes the purpose of a program hard to understand. The solution is to rewrite the code to do the same thing but through a series of *subroutines*. In a subroutine, you specify what needs to be done and the different values are handed over to the subroutine in the form of parameters. This makes the code variable. These values were previously fixed in the program, now they can easily be changed.

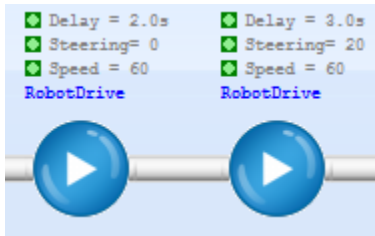
FlowCode 6 shows a part of code that is repeated twice in our program.



FlowCode 6: Same pattern with different values

We can put this sequence in a subroutine in which the fixed values are replaced by variables. We call these variables the parameters of the subroutine. Moreover, the four icons can be reduced into just one. First, we start by combining the Driver and the Stopwatch into a single icon.

These commands are executed
When the subroutine is called.
The parameters in the subroutine get
The same values as in the first example.



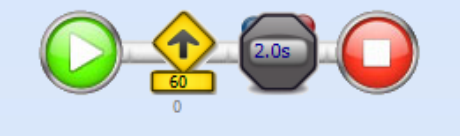

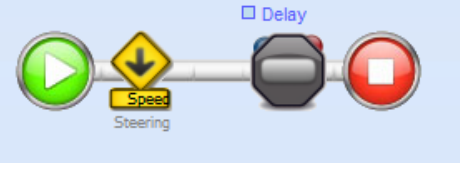


FlowCode 7: These icons perform the same operation as FlowCode 6

You can see here that the different values used in FlowCode 6 have been replaced by variables. To achieve the same results as FlowCode 7, we just need to call the subroutine and insert the values that we had in the code. You can see how this works in FlowCode 8. The explanation is given below.



FlowCode 8: The RobotDrive Subroutine

4D.1 Making a Subroutine																				
1		Save your program as 4D.1 – Subroutine. You will now make a subroutine.																		
2		We will make the FlowCode of RobotDrive so generic that it can handle any type of movement. First, remove the two icons in front of the Finish icon and insert the Finish next to the first two icons. This is our basic program.																		
3		Remove the DriveStraightAhead icon and replace it with a DriveAny icon. We have to make sure that the DriveAny icon knows how fast to drive and what the Steering value is.																		
4		This information will be provided to the icon from outside this program in the form of Parameters. It is just like the Properties that we saw before, but here we make our own properties using a Parameter. In this FlowCode, we will create three variables to input the required values.																		
5	<table border="1"> <thead> <tr> <th colspan="2">Properties</th> </tr> </thead> <tbody> <tr> <td>Name</td> <td>Speed</td> </tr> <tr> <td>DataType</td> <td>int</td> </tr> <tr> <td>Default</td> <td>50</td> </tr> <tr> <td>Parameter</td> <td>true</td> </tr> <tr> <td>Type</td> <td>Variable</td> </tr> <tr> <td>Comment</td> <td>Speed is passed by caller</td> </tr> </tbody> </table>	Properties		Name	Speed	DataType	int	Default	50	Parameter	true	Type	Variable	Comment	Speed is passed by caller	Go to the Toolbox and select a Variable icon from Variables. Go to the properties of this new Variable and give it the name Speed. Then select the INT value (Integer means a whole number, so without any decimal positions) from Data Type. Set the Default value to 50. This means that the variable Speed will get a value of 50 if no other value is transmitted to it.				
Properties																				
Name	Speed																			
DataType	int																			
Default	50																			
Parameter	true																			
Type	Variable																			
Comment	Speed is passed by caller																			
6	<table border="1"> <thead> <tr> <th>Steering (0)</th> <th>Speed (50)</th> <th>Delay (2.0s)</th> </tr> </thead> <tbody> <tr> <td></td> <td></td> <td></td> </tr> </tbody> </table>	Steering (0)	Speed (50)	Delay (2.0s)				Then, make two new variables and call them Steering and Delay. Give the Steering variable a default value of 0 (int) and the Delay variable 2 seconds using DataType Time.												
Steering (0)	Speed (50)	Delay (2.0s)																		
7	<table border="1"> <thead> <tr> <th colspan="2">Properties</th> </tr> </thead> <tbody> <tr> <td>speed</td> <td>local_Speed</td> </tr> <tr> <td>steering</td> <td>--Local Variables--</td> </tr> <tr> <td>Duration</td> <td>Steering</td> </tr> <tr> <td>Type</td> <td>Speed</td> </tr> <tr> <td></td> <td>Delay</td> </tr> <tr> <td></td> <td>--Constants--</td> </tr> <tr> <td></td> <td>0 - Stop</td> </tr> <tr> <td></td> <td>10</td> </tr> </tbody> </table>	Properties		speed	local_Speed	steering	--Local Variables--	Duration	Steering	Type	Speed		Delay		--Constants--		0 - Stop		10	Now, we have to tell the icons in the subroutine where they can find their property values. Go to the Properties of DriveAny and select the name of the Speed variable from its properties instead of a fixed number. This will make the speed of this icon variable and modifiable by the caller of the subroutine. If you do change a value, we call it a 'constant' but we now have turned it into a 'variable'.
Properties																				
speed	local_Speed																			
steering	--Local Variables--																			
Duration	Steering																			
Type	Speed																			
	Delay																			
	--Constants--																			
	0 - Stop																			
	10																			
8		In the same manner, change the Steering with the Steering variable. In StopWatch, change the Delay property and connect it to the Delay variable. Your program should look like the one on the left. Save your program and then continue with the next assignment.																		




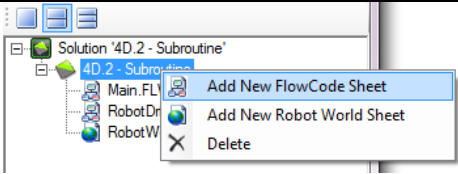
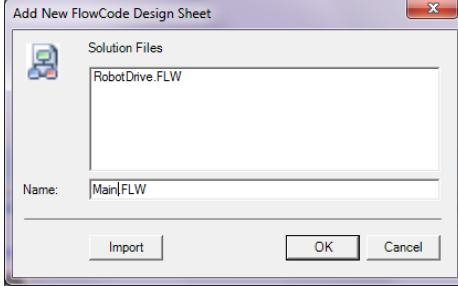
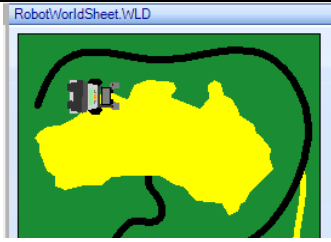
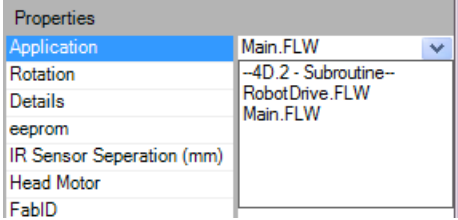
4D.2 assignment: Using the Subroutine

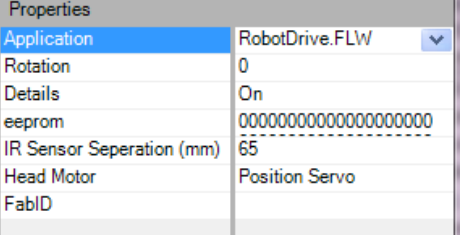
The subroutine in FlowCode 8 makes sure that the robot will start moving in the desired direction at the desired speed. Then, it waits for some time and then returns. Where does it return to?

A subroutine must always be activated by another part of a program. We refer to this as a CALL to a subroutine.

So we must have another program that calls this subroutine. We are going to make that program now. You will see that the subroutine is basically nothing but a new icon that we make ourselves.

You can use this feature to expand the programming environment with a variety of new icons, or functions, as they are also called.

4D.2 Using the Subroutine		
1		Save your program as 4D.2 – Subroutine. You are now going to use the subroutine.
2		So far, every program we have used consisted of a single FlowCode sheet, but in order to use subroutines we need a second FlowCode sheet. Go to the list on the top right (it is called the Project Browser) and use the right mouse button to select the Subroutine menu. Select 'Add New FlowCode Sheet' from the menu.
3		RoboPAL will ask you to name the new FlowCodeSheet. Call it Main and click on OK. Main means that this is the main program. Every program that uses subroutines has a main program, which is where the program begins. But how does the robot know where to begin?
4		Go to the RobotWorldSheet and select the robot. A robot has a number of Properties just as the icons in the FlowCodeSheet.
5		Look at the properties of the robot. Select Main.FLW from the Application list. This will allow you to tell the robot to start executing the Main program. Sometimes, it is useful to test a single subroutine at a time. You can do this by selecting RobotDrive, for example.

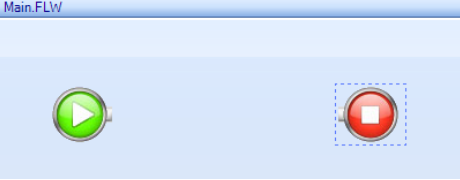

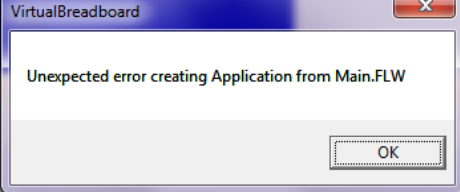
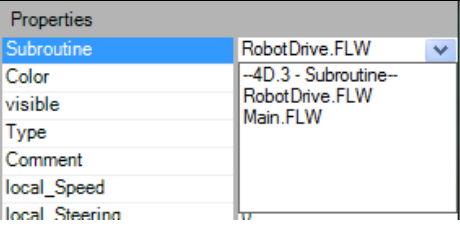
6		<p>The robot has more properties. The Rotation is set automatically when you rotate the robot in the RobotWorldSheet. The only other parameter that we will be using here is the IR Sensor Separation, that we already described briefly in Chapter 1. You only need to change it if the distance between the light sensors on your robot is different from the standard 6.5 cm. Save your program.</p>
---	---	---

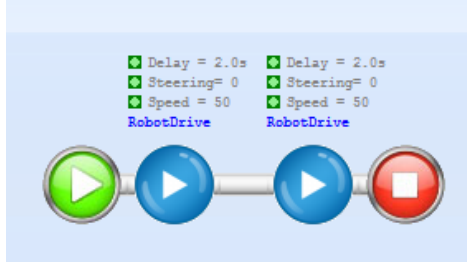


Making the Main Program



4D.3 assignment: Making the Main Program

Now we need to make the *Main* program and make sure that the robot moves forward until it finds the yellow line. This time, however, we are going to use subroutines. We have to change the program and use the two icons from program 4D.1.

4D.3 Making the Main Program		
1		<p>First save your program as 4D.3 – Subroutine. Go to the Project Browser on the right at the top and select Main.FLW and drag it to the FlowCodeSheet. You now have an empty screen at the bottom with the name Main. Now, insert a Start and a Finish icon from Program Flow. You will be creating the program between these two icons.</p>
2		<p>Also choose Subroutine twice from Program Flow and place the two icons between the Start and Finish icons and connect them. The program now knows that it has to call a subroutine twice, but it does not know which subroutine, yet. That is what we will tell it next.</p>
3		<p>If you try to execute or even save this program, you will get a RoboPAL error message, because it does not know what to do with these two icons. So we must now tell it what to do.</p>
4		<p>Select the first subroutine icon and go to its properties. Select Subroutine and choose RobotDrive from the list. Do the same with the second subroutine.</p> <p>You can also select a color for the subroutine, so that you can easily tell which subroutine the program is calling. It is a good idea to always use the same color for the same subroutine.</p>

5		<p>Once you have done this, the program will change the icons. It will put the name of the subroutine above the icons and display their parameters. You can now insert new values in the parameters of each subroutine. These values will be used when the subroutines are executed.</p>
6		<p>Use the same values you found earlier in program 4C.2 and insert them in the Properties of the Subroutine call. Save your program.</p>
7		<p>Now, try to run your program in the simulator and if it works well upload it to the robot. If everything works as expected, your robot should drive up to the yellow line. This version of the program does exactly the same as the previous, but it is more suited for inserting new actions in the following lessons.</p>






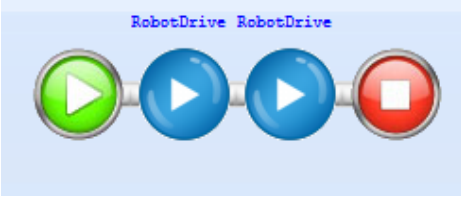




4E.1 assignment: **Driving in a Square**

You have made significant progress with your program. You have used the flexible idea of a subroutine and have found out how to make the robot follow the first part of the road and stop at the yellow line.

The idea now is that you are going to use what you have learned so far to make a new program, all by yourself, that will make the robot drive along an imaginary square. You will find some help for this task below. You have to change this program to use a subroutine.

In assignment 4E, the robot will move along an imaginary square. This means that the robot must move forward for a while, then make a 90° turn, then drive forward again and so on.

4E.1 Making the Robot Drive along an Imaginary Square												
1		<p>Select Drive Behavior 4A from NLT again. Save the program as 4E.1 – Drive Square.</p>										
2		<p>We want the robot to drive forward for a short while and then make a 90 degree turn.</p>										
3	<table border="1" data-bbox="293 1792 762 1977"> <thead> <tr> <th colspan="2">Properties</th> </tr> </thead> <tbody> <tr> <td>speed</td> <td>60</td> </tr> <tr> <td>steering</td> <td>90</td> </tr> <tr> <td>Duration</td> <td>1.0s</td> </tr> <tr> <td>Type</td> <td>DriveSwurveRight</td> </tr> </tbody> </table>	Properties		speed	60	steering	90	Duration	1.0s	Type	DriveSwurveRight	<p>You have a number of possibilities to find the best way to turn a corner. You can change the Speed, the Steering or the time on the Stopwatch (making it shorter or longer). Together these three variables determine how sharp the turn will be.</p>
Properties												
speed	60											
steering	90											
Duration	1.0s											
Type	DriveSwurveRight											

4		<p>First, test if this program can make a 90 degree turn on the simulator. Then, modify the program until the robot makes the turn correctly. Write down the value that works best.</p>
5		<p>To make the robot drive along an imaginary square, it has to execute this piece of code four times. You can do this with a subroutine, so make a Main program and make sure that the subroutine is first called once and then added a second time.</p>
6		<p>Test your program in the simulator to see if it makes the turn correctly the second time, too. If it does not, change the turn properties a little. You will see that the error in the turn angle will get larger as you execute the subroutine various times. If you cannot get it to work properly, reread the parts of this chapter on subroutines.</p>
7		<p>Now, change your program to call the subroutine four times, so that the robot ends at the same place where it started.</p>
8		<p>The last step is to be able to change the size of the square by introducing a single parameter. Add a local variable to the Main program and to the subroutine. Use this parameter to tell the subroutine for how long to move along a straight side of the imaginary square. If the corners stay the same, you can resize the square very easily in this way. By inserting the value into a single variable in Main and passing it to the subroutine, you can tell the program how big the imaginary square must be.</p>
9		<p>If your program works fine, test it on the NXT robot on the field. You will find that now it is more difficult to make the robot drive in an exact square. Try to find out what causes this.</p>

4.6 In Practice

In these lessons, we have only used wheeled robots; in our case, a robot with two powered wheels and a castor wheel. In practice, most robots are either have wheels or cannot move at all, like an industrial robot. Military robots are mostly wheeled robots, robots with caterpillar tracks or flying robots.



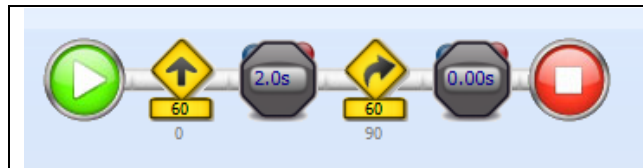
Fig 38: the Tulip

However, the most interesting robots are walking robots. There are many different robots on the market for research purposes that vary in price (1.000-15.000 Euro). If a robot is to be useful in and around the house, it will have to walk around and be able to use the tools that we use ourselves. A walking robot that can use a vacuum cleaner or can make the beds is far more useful than having individual, specialized robots for each task. So being able to walk is very important.

In the Netherlands, the three technical universities are especially active in this area. The newest robot, the Tulip of Dutch Robotics, has been developed by the Bio Robotics Lab at the Technical University of Delft and is a very advanced, state-of-the-art and internationally renowned walking robot.

4.7 Test

1. What does flowcode 9 do, describe this in pseudo code.



FlowCode 9

2. How long does it take to execute the code in FlowCode 9?
3. How big is the turn that this code makes?
4. Insert FlowCode 9 into a subroutine and change the size of the turn by using a parameter.
5. How can you make a robot do the same thing for a couple of seconds?
6. What kind of indicators can you use in RoboPAL?

Intro Part 2

Part two focuses on the Rescue mission in which the robot uses a line follower to find and retrieve a dangerous container in the yellow swamp area. The Sense-Reason-Act loop, a basic programming tool that is used in all robotics programs, is introduced in Part 2.

Intro Part 3

The third part of the RLT module Robotics mainly addresses what we call Adaptive Behavior. This is a slightly more complicated section aimed at higher classes or to be used as follow-up lessons for information technology classes. The Sense-Reason-Act loop is treated in greater detail and we introduce the use of so-called State Machines.

We will be copying the behavior of a (very) simple organism that displays flee behavior, which can become curious or move around randomly.

So, a choice must be made on how to continue after part 1. Part 3 is more difficult and delves deeper into the subject, while Part 2 is simpler and focuses on the Rescue Mission.

Clearly, if you have enough time, both parts may be completed.