NXTSim: A 2D simulator for a LEGO Mindstorms NXT robot controlled by MATLAB

BSc Thesis

Müttalip Küçük, 2524209

Department of Computer Science,
VU University Amsterdam, Netherlands

Supervisor: Natalia Silvis-Cividjian
Second reader: Anton Eliens

2015

Table of contents

Abstract

The role of simulators in robotics research is important because of its efficiency, safety and robustness of new algorithms. Research shows that a simulator for LEGO Mindstorms NXT robots, which accepts MATLAB-code as input, does not exist. We developed a simulator to solve this problem. In this thesis, we contribute a simulator for the LEGO Mindstorms NXT robotic kit that accepts MATLAB-code as input. The main focus is on simplicity and usefulness, because it will be used by students with no experience in programming.

Chapter 1. Introduction

In 1988, Mark Weiser introduced the term “ubiquitous computing” which means that computing should be accessible everywhere and anywhere. In 1991, Weiser described his idea as follows: “Specialized elements of hardware and software, connected by wires, radio waves and infrared, will be so ubiquitous that no one will notice their presence” (Weiser, 1991). This vision involves advantages in business, public and private fields. Friedewald (2011) describes applications of ubiquitous computing in retail, industrial production and material management, transport logistics, personal identification and authentication, health care and mobility. This shows the importance of computers in today’s world. Other experiments predict that their role will grow further (Anderson et al., 2002). In 2002, there were more than 150 million hosts connected to the Internet, and the number was growing exponentially. Staranowicz et al. (2011) says that the need of robots is motivated by modern applications in which robots are required to autonomously operate in close interaction with humans. Hence it can be concluded that a future world without computers is not imaginable.

As a result of this growth, several branches within the computer science became interesting. Examples of these branches are software engineering or artificial intelligence. Because of the growth of the Internet during the years, a lot of knowledge can be reached online. This makes it easy to learn programming. Nowadays, in the early years of school, basic programming courses are given to students due the importance of IT in business. Because there are a lot of ways to teach the basics of programming, the school has to choose an environment, which is suitable for the school as well as for the students. The school wants an environment, which has a lot of tools available. Students prefer an environment, which shows quick results of their programming efforts. To satisfy both, robots can be used to teach programming and became familiar with artificial intelligence (Kumar, 2004). A bad thing about using robots to teach programming is that it demands a lot of time and effort from the students and the instructor.

A widely used platform for teaching artificial intelligence and the basics of programming is the LEGO Mindstorms NXT robotic kit. This robot is used by a lot of universities and there are even competitions to compare the robots programmed by the students (Patterson-McNeill, 2001). As earlier said, using robots in program courses takes a lot of time. A reason for this is that the duration of a class is limited and students need time to test their program with a robot. A solution for this time problem is to use simulators more. This allows students to work at home in their own time.

In this thesis, we present a program called “NXTSim”, which simulates a LEGO Mindstorms NXT 2.0 robot in 2D. This simulation accepts a syntactically correct MATLAB code as input for the robot. We intend to use it for the course “Pervasive Computing” lab sessions at the VU University Amsterdam. The students of this course are beginners in MATLAB as well as in using the robot.

This thesis is built as follows. In chapter 2, we describe the problem statement. In chapter 3, we describe a systematic literature review about simulators and robots in general. Also, the used methods, some results and discussion are given in this chapter. In chapter 4, we describe the implementation of our 2D simulator in more technical detail, and test NXTSim with two examples. In chapter 5, we end up with a conclusion and give some ideas for future work. In the appendix, you can find a user manual for NXTSim.

Chapter 2. LEGO Mindstorms robot controlled by MATLAB - Problem statement

The purpose of this project is to develop a 2D simulator for a LEGO Mindstorms NXT 2.0 robotic kit controlled by MATLAB. It will be used during the course “Pervasive Computing” at the VU University Amsterdam. A requirement is that the performance speed has to be acceptable. This means that it may not work too slowly. This chapter explains the fundamentals of the hardware and the software of the project.

The LEGO Mindstorms robotic kit is dividable in three parts. The first part is the controller. The controller shown in Figure 2 is the brain of the robot and where the program runs. The controller has totally eight ports. Four of these port are the output ports located at the top of the controller, and four are the input port located at the bottom. The four output ports at the top are labelled as “A”, “B”, “C” and “USB”. The ports “A”, “B” and “C” should be connected with actuators which will be explained later. The port “USB” is to make an USB-connection with another device. The USB-port will not be used in this project. The four input ports at the bottom are numbered from 1 to 4. These four ports should be connected with sensors which will also be explained later. In short, the controller has four input ports (1, 2, 3 and 4) and three output ports (A, B, and C).

Sensors are connected to the controller. There are a lot of sensors available for the LEGO Mindstorms robotic, but we will use two of them. The first is the light sensor shown in Figure 3. This sensor measures the reflected light and returns a greyscale value between 0 (white) and 800 (black). It is often used for line following. The second sensor is the ultrasonic sensor shown in Figure 4. This sensor generates sound waves, and reads their echoes to detect and to measure the distance to objects. The maximal range of this sensor is 255 cm. So the returned value is between 0 and 255.

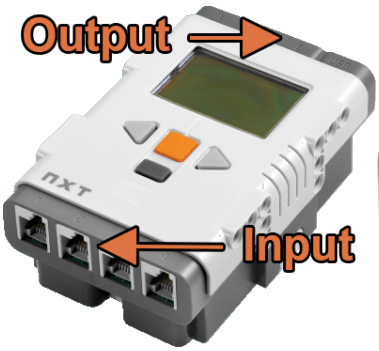


Figure 2. LEGO Mindstorms controller.

Figure 3. LEGO Mindstorms light sensor.

Figure 4. LEGO Mindstorms ultrasonic sensor.

Actuators are also connected to the controller. There are also numerous actuators for the LEGO Mindstorms robotic, but we would limit it to two actuators: motors and LEDs. The motor shown in Figure 5 makes rotations for a given speed between 0 and 100. It can be connected to a wheel. We used this actuator in this way. The second actuator is the LED shown in Figure 6 which can be switched on or off. The colour can be red, yellow or green.

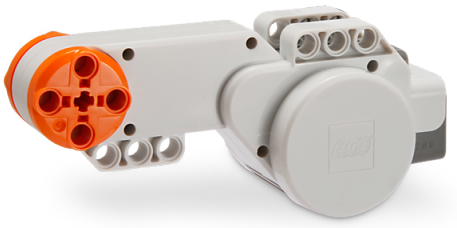


Figure 5. Motor.

Figure 6. LED

Figure 7(a) shows the robot which is used during the course “Pervasive Computing” and which we will implement in the simulator. The robot consists of a controller, two light sensors, one ultrasonic sensor and two motors. The other version of the robot is connected with LEDs instead of motors, and will also be implemented. The environment of the robot is a white surface on the floor where the robot would be placed on (see Figure 7(b)). This surface is 90 centimetres by 120 centimetres. There can be objects on this surface, so that we can analyse the behaviour of the robot. The available objects are black lines, horizontal and vertical walls and other 3D obstacles.

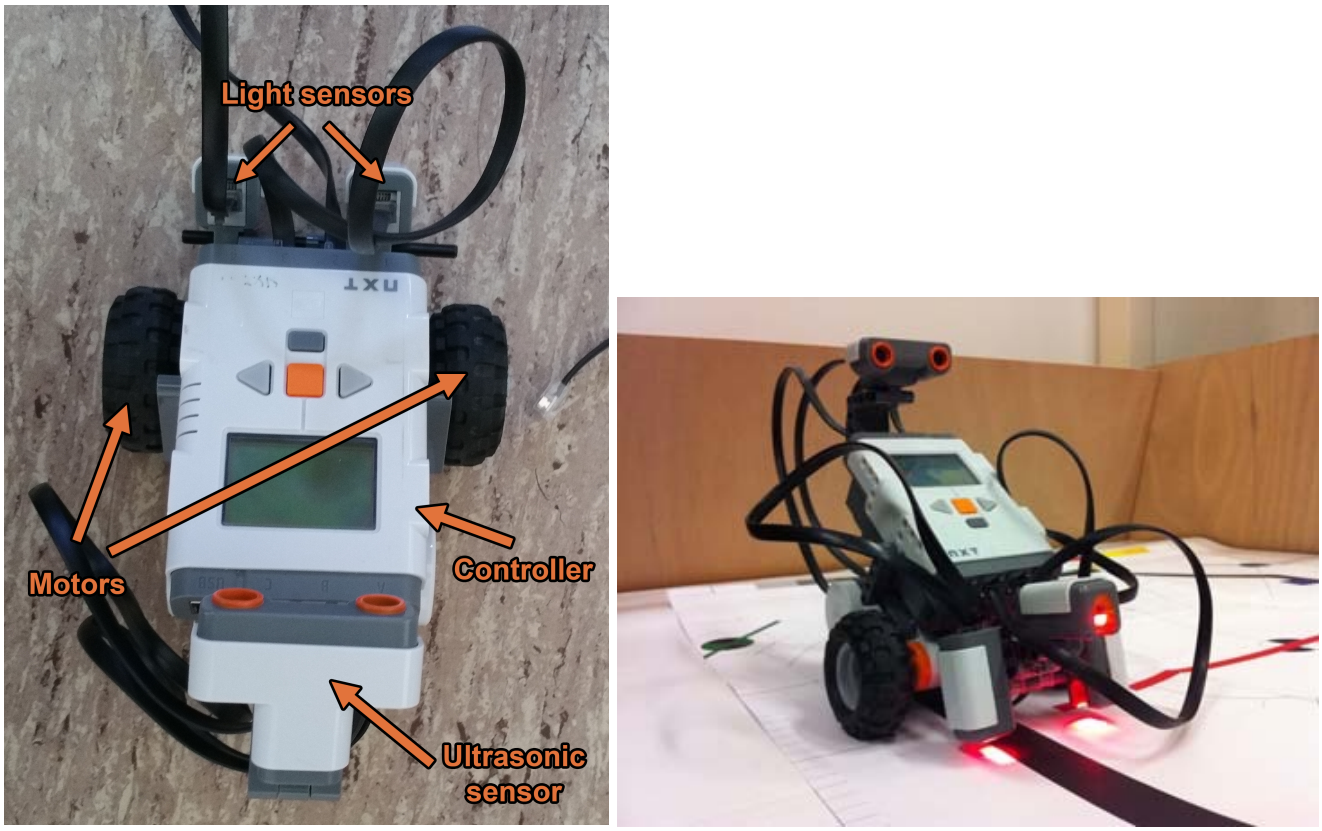


Figure 7. (a) LEGO Mindstorms robotic kit used during “Pervasive Computing”. **(b)** The environment with the robot.

The software which will be written by the students to control the robot is in MATLAB. This software uses the RWTH MATLAB toolboxes for LEGO (Aach et al., 2008). MATLAB is a numerical computing environment which has a lot of tools and functions. This program is widely used by schools and universities, because it is easy to learn to work with. We used MATLAB 8 version R2012b during this project. The written MATLAB program will also be one of the inputs of the simulator. Table 1 shows the commands that have to be recognized by the simulator. Combinations of code will also be possible and valid. The first column shows the code, the second column shows which part of the robot that code uses, and the third column shows a short description.

Command	**Category**	**Description**
COM_CloseNXT('all')	Initialization	Initialize the connections of the robot.
close all	Initialization	
h = COM_OpenNXT()	Initialization	
COM_SetDefaultNXT(h)	Initialization	
COM_CloseNXT(COM_GetDefaultNXT())	Initialization	Close the connection.
CloseSensor(SENSOR_1)	Sensor	Close sensor connected in port 1.
OpenLight(SENSOR_1, 'active')	Light sensor	Open light sensor connected in port 1.
GetLight(SENSOR_1))	Light sensor	Return value of light sensor connected in port 1.
OpenUltrasonic(SENSOR_1)	Ultrasonic sensor	Open ultrasonic sensor connected in port 1.
GetUltrasonic (SENSOR_1)	Ultrasonic sensor	Return value of ultrasonic sensor connected in port 1.
m = NXTMotor('AC', 'Power', 50)	Motor	Set the speed of motors connected in port A and C to 50, is assigned to variable m.
m.Power = 10	Motor	Assign the power of variable m to 10.
m.SendToNXT()	Motor	Send the instructions assigned to m to the controller.
m.Stop('off')	Motor	Turn the motors in m off.
m.Stop('brake')	Motor	Set speed of the motor in m to 0.
SwitchLamp(Motor_A, 'on')	LED	Switch LED connected in port A to on.
SwitchLamp(Motor_A, 'off')	LED	Switch LED connected in port A to off.
x = 10	General	Assign 10 to variable x.
x > 100	General	Return a Boolean expression.
vec(i) = 10	General	Assign 10 to place i of vector.
display(x)	General	Print variable x.
If (true) … else if (true) … end	General	If- and else if- statement.
Pause (0.5)	General	Pause for 0.5 milliseconds.
% this is a comment	General	Comment for the user.
While (true) … end	Loop	While-loop.
For i = 1:20 … end	Loop	For-loop.
Break	Loop	Break out (for or while) loop.

Table 1. Typical MATLAB commands used for our simulator.

Chapter 3. 2D simulators for robot programming - State of the art

3.1. Methods

A systematic literature review provides the relevant articles from all available literature to answer the research question or to learn more about other interests. To perform this review, we followed the steps defined by Kitchenham (2004) and Khan et al. (2001). The following stages and activities are performed:

Stage 1: Planning the review
- Activity 1.1: Identification of the need for a review
- Activity 1.2: Development of a review protocol

Stage 2: Conducting the review
- Activity 2.1: Identification of research
- Activity 2.2: Selection of primary studies
- Activity 2.3: Study quality assessment
- Activity 2.4: Data extraction and monitoring
- Activity 2.5: Data synthesis

Stage 3: Reporting the review
- Activity 3.1: Communicating the results

Firstly, we performed a search to check whether there are systematic reviews about programming simulators. No results were found. Further, we examined the state of research in programming simulators and we used the following research questions:
- RQ1: What is the platform of the robot?
- RQ2: Which languages are used for developing a simulator?
- RQ3: What are the components of the simulator from its maker point of view?
- RQ4: Which objects are defined in the simulator from its user point of view?

In April 2015, a systematic review was undertaken in the following online digital libraries: (a) ACM Digital Library, (b) ScienceDirect and (c) Google Scholar. Only results written in English were selected.
The search string used was: (simulation OR simulator OR simulate OR 2D or 3D) AND (robot OR robotic OR robotics OR LEGO OR Mindstorms) AND (programming OR program OR software OR JAVA) AND (teaching OR teach OR learning OR learn OR educate OR education OR educational OR school). Table 2 shows the protocol executed for each database.
Three criteria for exclusion (EC) articles were also identified:
EC1: The article is not about robots used in programming or artificial intelligence courses.
EC2: The article is about a robot instead of a simulator.
EC3: The simulator presented in the article is not achievable due to the duration of project and the small development team.

The initial search yielded 226 papers. Firstly, we selected on titles and abstracts using the inclusion criteria. Generally, it was easy to select on titles and abstracts, but there were articles of which we had the read the full text to decide whether it does not fit the exclusion criteria. Table 3 shows more detail about the selection we made.

Database	Protocol	Note
ACM Digital Library	(simulation OR simulator OR simulate OR 2D or 3D) AND (robot OR robotic OR robotics OR LEGO OR Mindstorms) AND (programming OR program OR software OR JAVA)	The search string is shortened, because it returned few results. The following text is deleted: (teaching OR teach OR learning OR learn OR educate OR education OR educational OR school)
ScienceDirect	(simulation OR simulator OR simulate OR 2D or 3D) AND (robot OR robotic OR robotics OR LEGO OR Mindstorms) AND (programming OR program OR software OR JAVA) AND (teaching OR teach OR learning OR learn OR educate OR education OR educational OR school)	Open access articles
Google Scholar	(simulation OR simulator OR simulate OR 2D or 3D) AND (robot OR robotic OR robotics OR LEGO OR Mindstorms) AND (programming OR program OR software OR JAVA) AND (teaching OR teach OR learning OR learn OR educate OR education OR educational OR school)	-

Table 2. The specific protocol executed in each database.

Database	Articles resulting from the search	EC1	EC2	EC3	Selected
ACM Digital Library	26	16	2	6	2
ScienceDirect	100	51	9	36	4
Google Scholar	100	55	13	24	8
Total	**226**	**122**	**24**	**66**	**14**

Table 3. Summary of article selection.

Table 3 shows that around 55% of the articles resulting from the search were excluded, because these are covered by *EC1*. This means that the resulting article is about another subject in the science and not about robots used in programming or artificial intelligence courses. Over 10% of the resulting articles were excluded due *EC2*, because these articles focus more on comparison of several robots and not or less on the simulation which uses a specific robot. Approximately 30% of the articles resulting from the search were not accepted due *EC3*, because the presented simulation is not achievable due to the duration of the project and the small development team.

3.2. Results & discussion

Table 4 shows an overview of all relevant articles resulting from the literature research. For each article, Table 4 shows the following attributes: (a) column 1: the name of the article or the presented simulator, (b) column 2: *Robot type* shows the type of the used robot, (c) *Language simulator* shows the language which is used by the simulator, (d) *Language input file* shows the language which is used as input by the simulator, (e) *Components* presents the component of the simulator, (f) *Defined objects* present which objects are defined in the simulator.

Article or simulator	Robot platform	Language simulator	Language input file	Components	Defined objects
Tian (2007)	LEGO Mindstorms RCX	ODE, C++ (OpenGL), ROBOLAB	ROBOLAB	GUI, Object Environment, Properties	Predefined robots and environments
Gonçalves et al. (2009)	LEGO Mindstorms NXT	SimTwo, ODE, C++ (OpenGL), XML	leJOSl	Object Environment	Robot with two motors and light sensor, line
Meyer et al. (2009)	LEGO Mindstorms RCX	Robotran	leJOS	IDE for editing code, simulator for visualizing robot	Robot, line, circle, square, move direction
Webots	Many, also LEGO Mindstorms NXT	C, C++, Java, Matlab, Python	leJOS	GUI, 3D, Documentation, Tutorial	Customizable robots, obstacles, walls, line, more
Easy-Rob	Robotic platforms in manufacturing plants	C++ (OpenGL)	None	GUI, 3D, Documentation	Multiple robots
Rob. Tbx.	Not specified	C-like	MATLAB	GUI, 2D/3D, Documentation, Tutorial	Robots with many sensors
Player/Stage	Not specified	C, C++, Python, Java	Any language	2D, Documentation, Tutorial	Robot with many sensors, obstacles and accessories
Gazebo	Not specified	C, C++, Python, Java	Any language	3D, Documentation, Tutorial	Robot with many sensors, obstacles and accessories
ROS	Not specified	C++, Python, Octave, LISP, Java, Lau	Not specified	2D/3D, Documentation, Tutorial, Community of researchers	Robots, nodes
Simbad	Not specified	Java, Python	Not specified	3D, Documentation,	Single and multiples robots, obstacles
CARMEN	Not specified	C, Java	Self-made software	2D, Documentation,	Real mobile-robot platforms and sensors
Sarsi	Not specified	C, C++, Java	GameBot interface	GUI, 3D, Documentation,	Several default maps, allows creating own maps, a variety of sensors
MRDS	Not specified	VPL, C#, Visual Basic, Jscript, IronPython	Visual programming language	GUI, 3D, Documentation, Tutorial	A variety of robotic platforms and sensors
MissionLab	Not specified	VPL	Self-made software	GUI, 3D, Documentation,	Multi-agent robotic mission specification, several tools to plan the missions

Table 4. Context of the articles.

There is a relation between the attributes *Language simulator* and *Language input file* shown in Table 4. Figure 8 shows the way the two compilers translate the language of the input code to the language of the simulator. Firstly, the language of the input file is compiled to an intermediate code which is the bridge between the languages of the input file and the simulator. Then, the intermediate code is compiled to the language of the simulator which will show a simulation that interprets the input file.

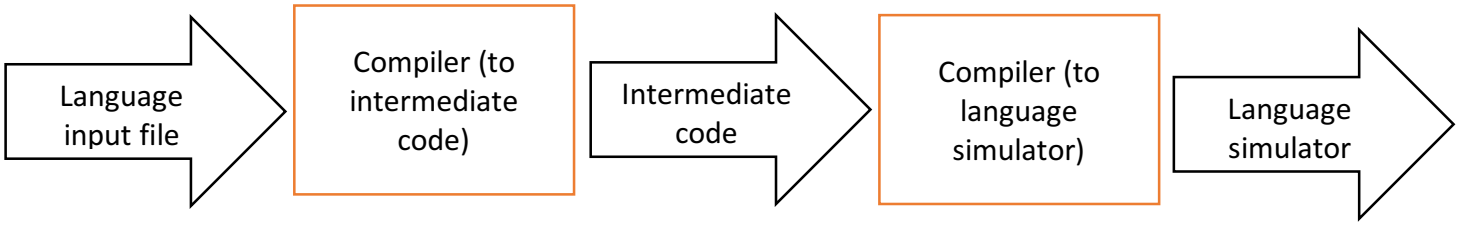


Figure 8. Transformation sequence.

We analyse the results of the systematic review and try to answer the research questions defined in paragraph 3.1.

RQ1: What is the platform of the robot?
In most of the retrieved articles, it is not specified for which robot type it is be used (Webots, Easy-Rob, Rob. TX., Player/Stage, Gazebo, ROS, Simbad, CARMEN, USARSim, MRDS, MissionLab). So this allows it to use it for different robot types. The other articles presents a simulator for the RCX (Tian, 2007; Meyer et al., 2009) and NXT (Gonçalves et al, 2009) models of Lego Mindstorms.

RQ2: Which languages are used for developing a simulator?
The results show that two of the most used programming languages are C++ and Java (Webots, Easy-Rob, Player/Stage, Gazebo, ROS, Simbad, USARSim). Other often used programming languages are C and Python (CARMEN). The following programming languages are used by a few articles: VPL (MissionLab), LISP, Lua, Visual Basic, Jscript, IronPython (MRDS), Octave, ODE, Robotran (Meyer et al., 2009), ROBOLAB (Tian, 2007), SimTwo (Gonçalves et al., 2009), XML and a C-like language (Rob. Tbx.).

RQ3: What are the components of the simulator from its maker point of view?
Almost all articles contain a graphical user interface (GUI) and documentation. Most of the GUI's are in 3D (Webots, Easy-Rob, Gazebo, Simbad, USARSim, MRDS, MissionLab), a few are in 2D (Player/Stage, CARMEN), and a few in both 2D and 3D (Rob. Tbx., ROS). Another well common component in the articles is a tutorial. Other mentioned components are object environments (Gonçalves et al., 2009), properties (Tian, 2007), and an IDE for editing the code (Meyer et al., 2009). One article uses MATLAB as the language of the input file (Rob. Tbx., ROS).

RQ4: Which objects are defined in the simulator from its user point of view?
In most of the articles, the user is able to customize the robot with several sensors (Gonçalves et al., 2009; Webots, Easy-Rob, Rob. Tbx., Player/Stage, Gazebo, ROS, Simbad, CARMEN, MRDS, MissionLab). All simulators focus on the programming language and the graphical user interface (Tian, 2007; Meyer et al., 2009; USARSim). So the defined objects are clearly the same: robots, obstacles, lines and nodes.

Chapter 4. A 2D simulator for LEGO Mindstorms controlled by MATLAB - NXTSim

This chapter is about the implementation of the hardware which is mentioned in the previous chapter. We will also tell about the parsing of the MATLAB code to intermediate code. At the end of this chapter, we present the user interface and the design of NXTSim.

4.1. Approach

The main requirement is as follows: Given a MATLAB code as input file, develop a 2D program which simulates the physical LEGO Mindstorms NXT robot. Before developing the simulator we had to choose which programming language we would use. Our team[1] is most familiar with JAVA, so we decided that the language of the simulator is JAVA. Then, we thought about ideas to develop a digital version of the hardware and the environment mentioned in the previous chapter, and about parsing the software.

4.2. Sensors, actuators and environment

Before explaining the implementations, we have to notice that 10 cm in the physical world is 64 pixels in the simulation. Firstly, we design a 68 pixel by 119 pixels image shown in Figure 9 which represents the robots which we will use. The left one is connected with motors, and the right one with LEDs. The implementations of motors and LEDs will be explained later. Then, we thought about the implementations of the three parts of the hardware which are controller, sensors and actuators. We do not have to implement the controller, because that is the program itself. The first sensor to implement is the light sensor. The light sensor shown in Figure 10 has a width of 12 pixels and a height of 18 pixels. We select nine coordinates of each light sensor shown as yellow in Figure 10. These nine coordinates will measure the colour under these coordinates. After knowing the nine colours from each sensor, the majority of the nine colours will be the colour of that sensor. The nine coordinates are combinations of the x-coordinates 1, 6 and 12, and the y-coordinates 1, 9 and 18. The second sensor is the ultrasonic sensor which is implemented as follows. We select five coordinates with x-coordinates 1, 17, 34, 51 and 68 and with y-coordinate 92 shown in Figure 11. From these five points, we go one pixel further till one of these point measures an obstacle. If we measure an obstacle, we save the number of the pixel we went through. Otherwise, if we do not measure an obstacle and the maximum range of the ultrasonic sensor is reached, then we saved the maximum range as observed value to an obstacle.

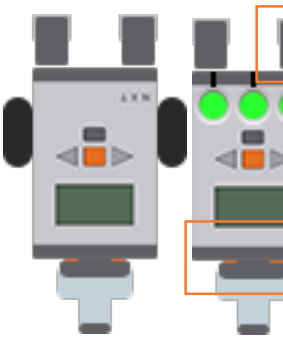


Figure 9. The two robots in the simulation.

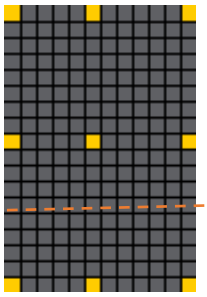


Figure 10. Nine coordinates of the light sensor.

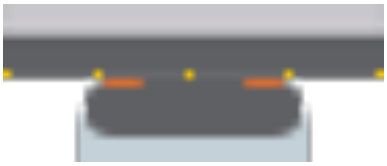


Figure 11. Five coordinates of the ultrasonic sensor.

[1] The team consists of one student which is Müttalip Küçük.

The first actuator implemented is the motor. We do not have to implement a graphical item for the motor. The only important thing about the motor is that the simulation has to know whether it is on or off, and the speed of the motor. Given these two data for both motors, we move the robot in the simulation in two steps. The first step is to move the robot with the lowest speed of the two motors in the direction which is saved in the program. The second step of moving the robot is to calculate the difference between the speed of the two motors, and rotate the robot. This looks like the robot drives a curve. When the two motors have equal speed, then the degree of rotation is 0 which will not cause a curve. We will explain this with the example (see Figure 12) to make the idea clear. For example, if the speed of the left motor is 30 and the right motor is 10, and both motors are on, then the robot has to drive to the right with a certain angle. The first step of our implementation tells that the right motor has the lowest speed which is 10. The robot drives straight forward with speed 10 which is shown as two red arrows in Figure 12(b). The second step calculates the difference of the two speeds which is 20. The following formula calculates the degree of a rotation:

$$\text{rotation} = \tan^{-1}\left(\left(\text{speed}_{\text{left motor}} - \text{speed}_{\text{right motor}}\right) / \text{width}_{\text{robot}}\right), \quad \text{where } \text{width}_{\text{robot}} = 68.$$

Using this formula, we calculate the angle A shown in Figure 12(c). Then, we rotate the robot with this angle shown in Figure 12(d). Finally, the robot has moved. Figure 12(a) and 12(e) show respectively the robot before and after it was moved.

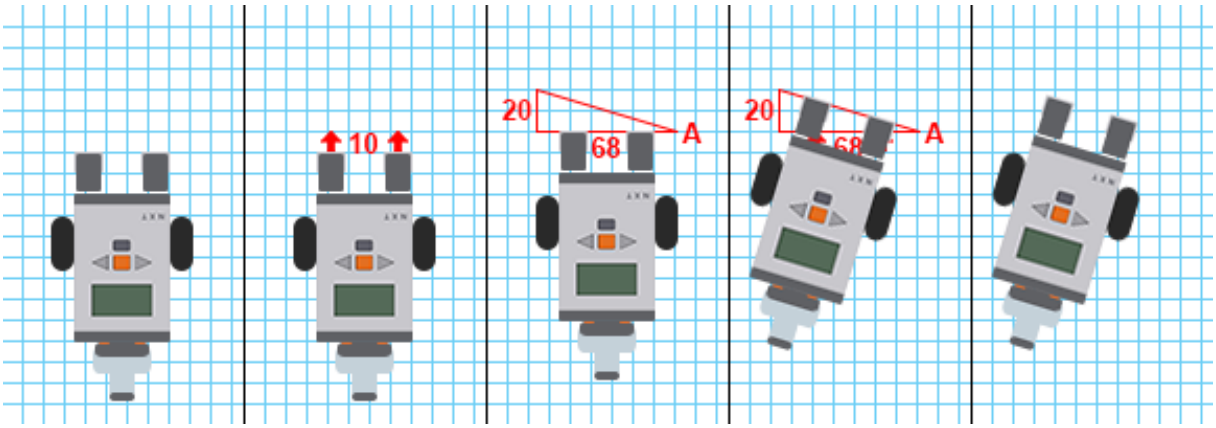


Figure 12. (a) Before the robot moved, **(b)** before step 1 **(c)** after step 1 and before step 2, **(d)** after step 2, **(e)** after the robot moved.

The implementation of the LED is the simplest, because the program just had to select one of the robots in Figure 13. The robot can be connected with at most three LEDs. If the LED is connected and is switched to on, the LED becomes green. Otherwise the LED will be red.

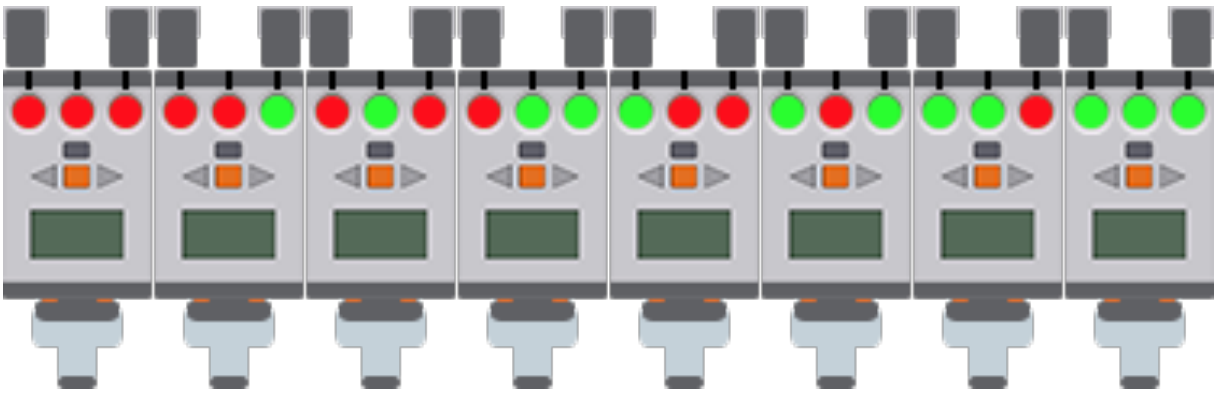


Figure 13. All possible states of a robot connected with at least one LED.

The environment in the physical world is a 90 cm by 120 cm white surface. We designed a grid with nine columns and twelve rows. Each cell represents a part which is 10 cm by 10 cm, but in the program it is 64 pixels by 64 pixels.

4.3. Parsing the code

Our assumption is that the MATLAB file is syntactically correct. The first step is to parse the MATLAB file to an intermediate code, see the left rectangle in Figure 14. In this step, we determine which category a MATLAB command belongs to. Table 5 shows all the possible categories with the corresponding MATLAB commands. The second step is to parse the intermediate code to the language of the simulator, which are both in JAVA. The code in the language of the simulator performs our implementations of the simulator.

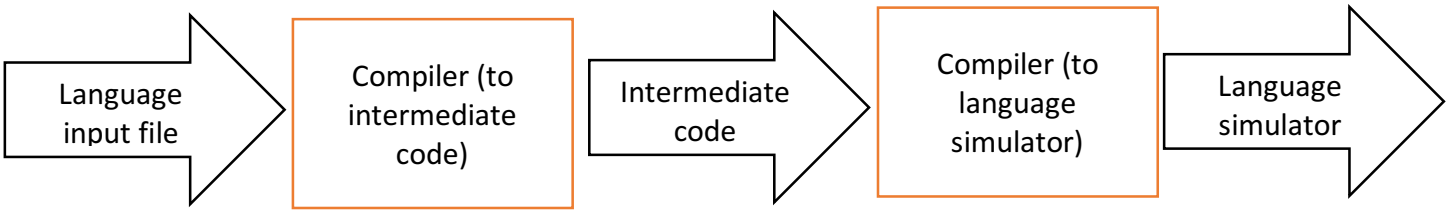


Figure 14. Transformation sequence.

Category	**Command(s)**
Space	All commands consisting of whitespaces or tabs.
Initialization	COM_CloseNXT('all') close all h = COM_OpenNXT() COM_SetDefaultNXT(h) COM_CLoseNXT(COM_GetDefaultNXT())
For	For i = 1:20 … end
While	While (true) … end
If	If (true) … else if (true) … end
Assignment	m = NXTMotor('AC', 'Power', 50) m.Power = 10 x = 10 vec(i) = 10
SendToNXT	m.SendToNXT()
Stop	m.Stop('off') m.Stop('brake')
Break	Break
Pause	Pause (0.5)
SwitchLamp	SwitchLamp(Motor_A, 'on') SwitchLamp(Motor_A, 'off')
OpenLight	OpenLight(SENSOR_1, 'active')
GetLight	GetLight(SENSOR_1))
OpenUltrasonic	OpenUltrasonic(SENSOR_1)
GetUltrasonic	GetUltrasonic (SENSOR_1)
CloseSensor	CloseSensor(SENSOR_1)
Display	display(x)
Expression	x > 100
Unknown	All command which do not belong to one of the categories above.

Table 5. Categories in parsing a MATLAB command.

4.4. User interface

We decided to make two screens: one screen for initialization of the ports, and another screen to input the file and to run the simulation.

Figure 15 shows the initialization screen. The main goal of the initialization screen is to connect sensors with ports 1, 2, 3 and 4 and actuators with ports A, B and C. This screen is divided in two parts. The left part shows the controller with the connected sensors and actuators. The right part is to select a sensor or actuator, or to end the communication process. All the seven ports are clickable. After a click on one of the ports, the right part of the screen will show one of the following two scenarios: a menu with the available sensors, or a menu with the available actuators. When the initialization is completed, the user clicks on the OK button.

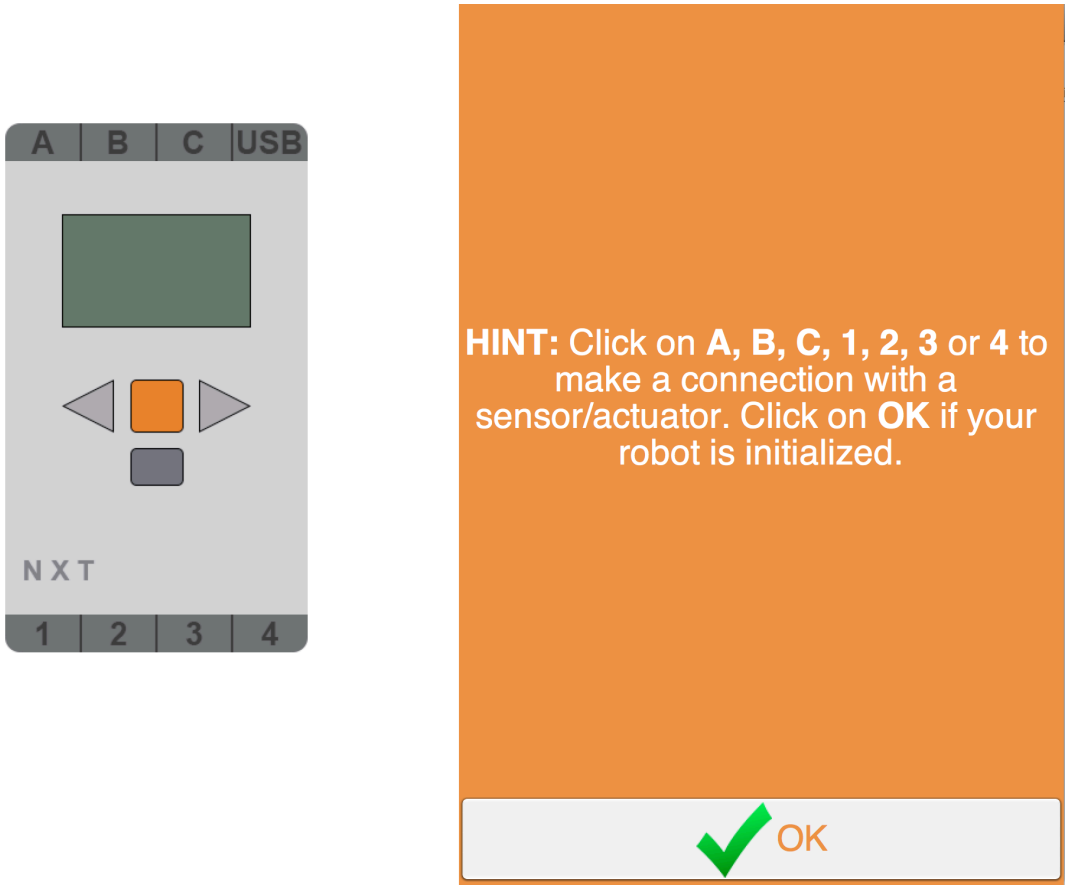


Figure 15. The initialization screen.

The second screen of our program is the simulation screen shown in Figure 16. The goal of this screen is to build the environment of the robot, to read in the MATLAB file, and to run the simulation. The simulation screen contains a menu bar at the top, a tools panel at the left, an environment panel at the centre, a code panel at the top right, and an output panel at the bottom right.

The menu bar at the top contains the options “File” and “Run”. The “File”-option can be used to clear the environment, to import the file with the MATLAB code, or to quit the program. The user can delete all objects in the built environment by File>Clear environment. After clicking on File>Import MATLAB-file, the user can browse through his or her folders to select the MATLAB-file. To quit the program, the user selects File>Quit. The option Run>Run is to

run the simulation and becomes clickable after the robot is placed and the MATLAB-file is imported. To abort the simulation which is running, the user can select Run>Abort.

The leftmost part of the screen is the tools panel. The tools panel contains sixteen objects which can be used to build an environment, four initial robots which are the same robot but have different rotations, and a delete button to correct a wrongly placed object. There are sixteen objects available to build an environment. Ten of them are black lines which simulate a black tape. The user can use these lines for example for testing a line following robot. The other six objects are 3D obstacles. These objects can be used for testing the ultrasonic sensor which measures the distance to the nearest object. These six obstacles are the following: a big grey cube, a small grey cube, a big grey ball, a small grey ball and a wall which can be placed horizontally and vertically.

The gridded panel in the centre is the surface in which the objects and the robot are placed. Each cell in the grid represents a 10 cm by 10 cm surface. The whole grid has nine columns and twelve rows from which it follows that this grid represents a 90 cm by 120 cm surface. The simulation runs in this environment panel.

The black panel at the top right of the screen is the code panel. This panel shows the code of the imported MATLAB-file. The user can click on this panel to import the MATLAB-file instead of using the menu bar. The white panel at the bottom right is the output panel. This panel reports the connections with the ports, the measured values of the sensors, the line of the MATLAB code which is running, and the print statements in the MATLAB-file.

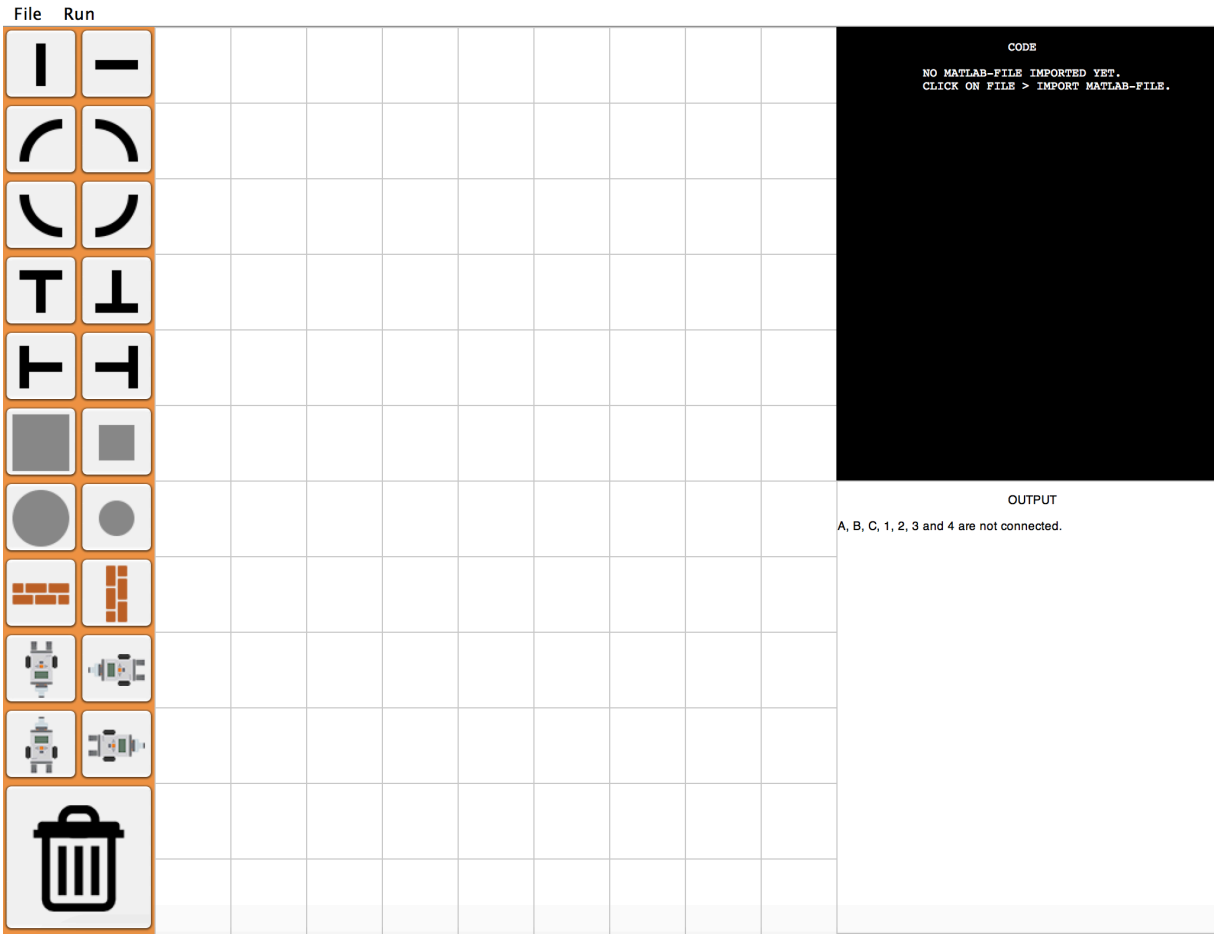


Figure 16. The simulation screen.

4.5. Testing

We tested our simulator with a lot of MATLAB programs and scenarios, but in this chapter we will explain two scenarios which are used during the lab session by the students. The ports are connected as follows: A with the left motor, B with the right motor, 1 with the left light sensor, 2 with the right light sensor, and 3 with the ultrasonic sensor. Table 6 shows the MATLAB codes of the two scenarios.

```
OpenLight(SENSOR_1, 'ACTIVE');
OpenLight(SENSOR_2, 'ACTIVE');
mA = NXTMotor('A', 'Power', 20);
mB = NXTMotor('B', 'Power', 20);

while(true)
      val1 = GetLight(SENSOR_1);
      val2 = GetLight(SENSOR_2);

      if (val1 > 450 && val2 > 400)
            mA.Power = 20;
            mB.Power = 20;
      elseif (val1 < 410 && val2 > 380)
            mA.Power = 0;
            mB.Power = 15;
      elseif (val2 < 410 && val1 > 400)
            mB.Power = 0;
            mA.Power = 15;
      elseif (val2 < 380 && val1 < 400)
            mAB.Stop('brake');
            break;
      end
      mA.SendToNXT();
      mB.SendToNXT();
end
```

```
OpenUltrasonic(SENSOR_3);

m = NXTMotor('AB', 'Power', 10);

m.SendToNXT();

minDistance = 20;

while(GetUltrasonic(SENSOR_3) > minDistance)

end

m.Stop('off');
```

Table 6. (a) MATLAB code for line follower, **(b)** MATLAB code for obstacle avoidance

4.5.1. Line follower

The idea of the line follower program is as follows. There are two light sensors, a left and a right one, and two motors, also a left and a right one. Continuously, the two light sensors measure the reflected light. This results that there are four combinations possible. If both sensors measure white, both motors drive straight forward with the same speed (see Figure 17(a)). If the left light sensor measures black and the right one white, then the left motor stops driving while the right motor drives with a certain speed (see Figure 17(b)). This causes a curve to the left. The same idea holds for the case in Figure 17(c), but this causes a curve to the other direction. If both sensors measure black, then both motor stop driving and the program will stop running (see Figure 17(d)). The pseudo code of the line follower is as follows.

Figure 17. (a) Left and right measure white, **(b)** left measures black and right measures white, **(c)** left measures white and right measures black, **(d)** left and right measure black.

```
WHILE motor A and B are on
      measure values of left and right light sensor
      IF values of left and right light sensor are white THEN
            drive straight forward
      ELSEIF value of left light sensor is black and right is white THEN
            drive to the left
      ELSEIF value of left light sensor is white and right is black THEN
            drive to the right
      ELSEIF values of left and right light sensor are black THEN
            turn off the both motors
END
```

The user wrote the line follower code in MATLAB shown in Table 6a. The ports of the controller are connected as described above. After setting the connections right in the initialization screen, the initialization is completed. The simulation screen opens, and the connected ports are shown in Figure 18 as light blue. The user clicks on the objects and the robot marked as blue, and adds these to the environment and positions the robot. Then, the user adds the MATLAB code shown in Table 6a by clicking on File>Import MATLAB-file in the menu bar (marked as yellow). After clicking on Run>Run in the menu bar which is marked as green, the program will start running the simulation. The robot will follow the line and will stop when it arrives at the T-junction. The user clicks on File>Quit to quit the program.

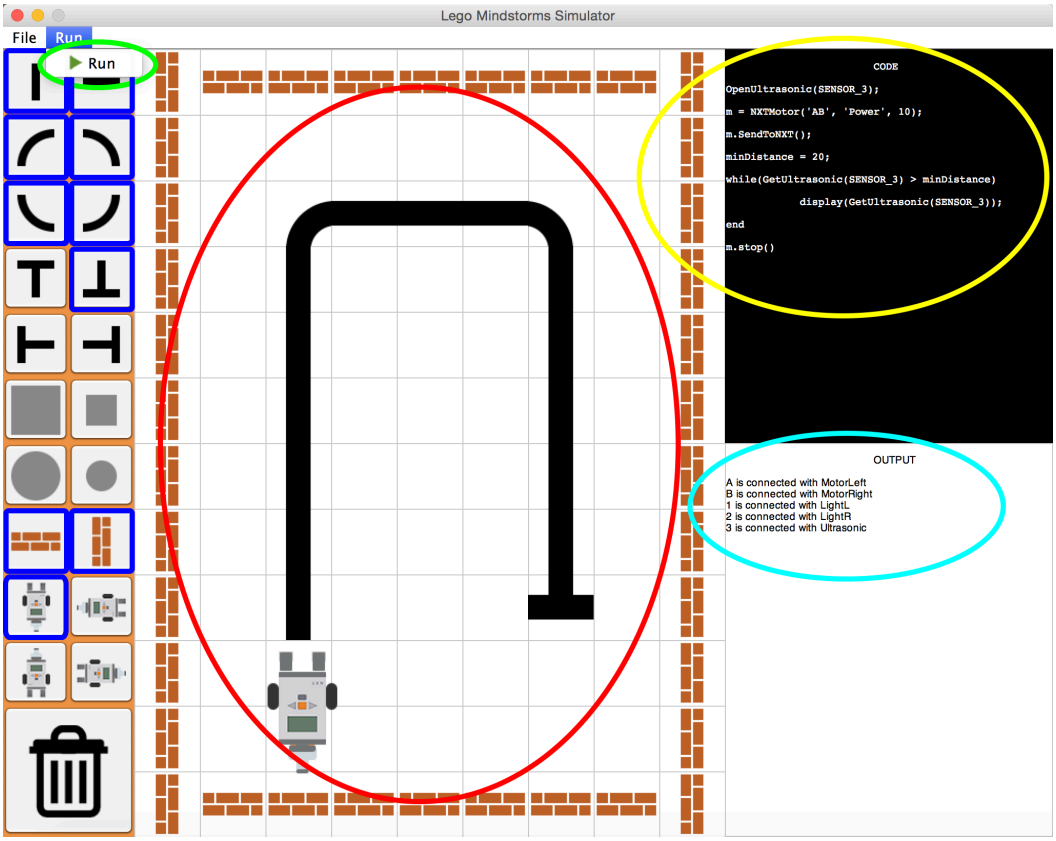


Figure 18. The line follower simulation.

4.5.2. Obstacle avoidance

The idea of the obstacle avoidance program is as follows. The robot is connected to an ultrasonic sensor. This sensor measures continuously the distance from the robot to an obstacle. The robot drives straight forward, until it measures a value which is smaller than a given minimum distance, and then it stops. The corresponding pseudo code of the obstacle avoidance is as follows.

```
WHILE measured distance is greater than the minimum distance
      drive straight forward
END
stop driving
```

The user has written the code obstacle avoidance in MATLAB shown in Table 6b. The initialization is the same as in the line follower example. The simulation screen is opened and some info about the initialization marked as light blue is shown (see Figure 19). The user adds objects and the robot, which are marked as blue to the environment, which is marked as red. After importing the code of Table 6b by clicking File>Import MATLAB-file, the code is shown. This is marked as yellow. After clicking on Run>Run which is marked as green, the robot will drive straight and stop when the ultrasonic sensor measures a distance less than 20 cm.

There are more tests done, for example with robots connected to LEDs. All the tests are completed successfully. The simulator works as expected.

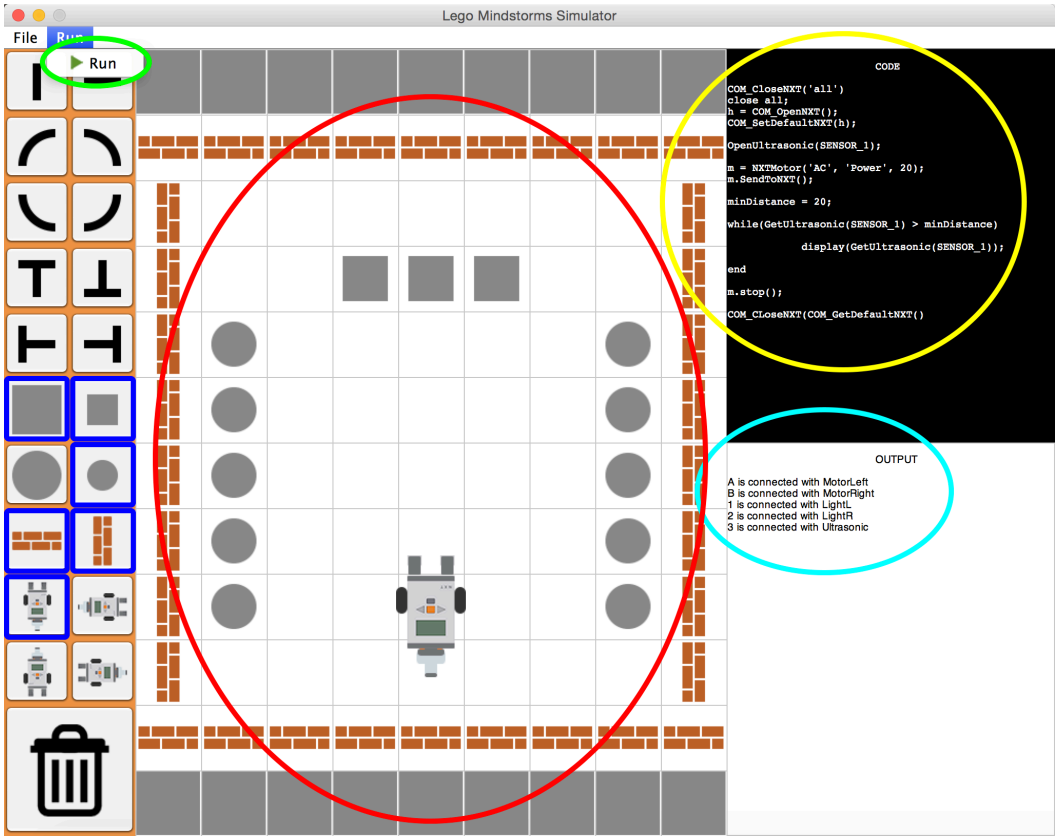


Figure 19. The obstacle avoidance simulation.

Chapter 5. Conclusion

This thesis described a 2D simulator for the LEGO Mindstorms NXT robotic kit controlled by MATLAB. We named the developed simulator “NXTSim”. For developing NXTSim, the programming language JAVA was used. The user is able to initialize the settings of the controller, to build an environment as desired, and run a simulation controlled by a correctly written MATLAB-code. The robot is equipped with two sensors (light and ultrasonic sensors), and two actuators (motors and LEDs). The user is able to build the desired robot and test it in an environment. The defined objects for building an environment are black lines, walls and other 3D obstacles.

The developed simulator meets all the requirements. One of the requirements was that the speed of the simulator had to be acceptable, and we are satisfied about the speed. If we compare NXTSim to other simulators, we see that there are simulators where the user can use more sensors (Webots, Rob. Tbx.) or even use multiple robots (Simbad). But the focus of NXTSim is on simplicity. NXTSim will be used by students without experience in programming. For these students, it is easier to use a simple simulator and focus on the MATLAB code.

For more future work, we suggest to define more objects which can be used for building the environment. Examples of these objects are objects in different colours or other geometrical figures. Our program recognizes twenty-six different MATLAB commands. This number can be increased which makes the simulator accept more different codes. The functions in the menu bar are limited. This can be extended by adding more functions. The duration of this project was two months and the team which developed this program consists of one Computer Science student. So using another programming language and starting a whole new project is possible.

In order to use or to extend the simulator, permission should be asked to Müttalip Küçük. The VU University Amsterdam has the permission to use this program during the lab sessions of the course “Pervasive Computing”. You can find more information about NXTSim on its official page: *www.mkucuk.nl/nxtsim.*

Personally, my main goal of choosing this project was to improve my programming skills in JAVA. Before this project, I had some experience which I gained during my study. But after this project, I learned a lot about the graphical user interface components in JAVA. So my personal goal is also completed successfully.

I thank Natalia Silvis-Cividjian for helping us with the literature study and for supervising us during this project. I also thank Anton Eliens for being the second reader of this paper.

References

Aach, T., Behrens, A. (2008). Freshman Engineers Build MATLAB Powered LEGO Robots. MATLAB Digest. Academic Edition. The MathWorks. 08/08.

Anderson, D. P., Kubiatowicz, J. (2002). The Worldwide Computer. Scientific American. March 2002.

Carnegie Mellon Robot Navigation Toolkit [Online]. Available: http://carmen.sourceforge.net/.

Corke, P. I. (1995). A computer tool for simulation and analysis: the robotics toolbox for matlab. Proceedings of the 1995 National Conference of the Australian Robot Association, Melbourne, Australia, pp 319-330, July 1995.

Easy Rob [Online]. Available: http://www.easy-rob.com/en/easy rob.html.

Friedewald, M., Raabe, O. (2011). Ubiquitous computing: An overview of technology impacts. Telematics and Informatics 28 (2011) 66-65.

Gazebo [Online]. Available: http://gazebosim.org

Gonçalves, J., Lima, J., Malheiros, P., Costa, P. (2009). Realistic simulation of a Lego Mindstorms NXT based robot. 18th IEEE International Conference on Control Applications Part of 2009 IEEE Multi-conference on Systems and Control Saint Petersburg, Russia, July 8-10, 2009.

Khan, K. S., ter Riet, G., Glanville, J., Sowden, A. J., & Kleijnen, J. (2001). Undertaking systematic review of research on effectiveness. April, 2001.

Kitchenham, B. (2004). Procedures for Performing Systematic Reviews. Keele University Technical Report TR/SE-0401. ISSN:1353-7776. July, 2004. Research Report. CRD Report (4 (2nd Edition)). NHS Centre for Reviews and Dissemination, York, UK.

Kumar, A. M. (2004). Three Years of Using Robots in the Artificial Intelligence Course – Lessons Learned.

Meyer, R. M., Puehn, D. C. (2009). Simulating a LEGO Mindstorms RCX Robot in the Robotran Environment. Proceedings of the Twenty-Second International FLAIRS Conference (2009)

Microsoft Robotic Developer Studio [Online]. Available: http://www.microsoft.com/robotics/.
MissionLab [Online]. Available: http://www.cc.portch.edu/ai/robot lab/research/MissionLab/.

Patterson-McNeill, H., Binkerd, C. L. (2001). Resources for using Lego Mindstorms. Consortium for Computing in Small Colleges. March, 2001.

Player/Stage [Online]. Available: http://playerstage.sourceforge.net/.

Robot Operating System [Online]. Available: http://www.ros.org/wiki/.

Simbad [Online]. Available: http://simbad.sourceforge.net/.

Staranowicz, A., Mariottini, G. L. (2011). A Survey and Comparison of Commercial and Open-Source Robotic Simulator Software. PETRA&11, May 25-27, 2011, Crete, Greece.

Tian, Y. (2007). Simulation for LEGO Mindstorms Robotics. A thesis at Lincoln University.

USARSim Matlab Toolbox [Online]. Available: http://robotics.mem.drexel.edu/USAR/index.html.

Webots [Online]. Available: http://www.cyberbotics.com/overview.

Weiser, M. (1991). The Computer for the 21st Century. Scientific American, 1991.

Appendix

User Manual: NXTSim

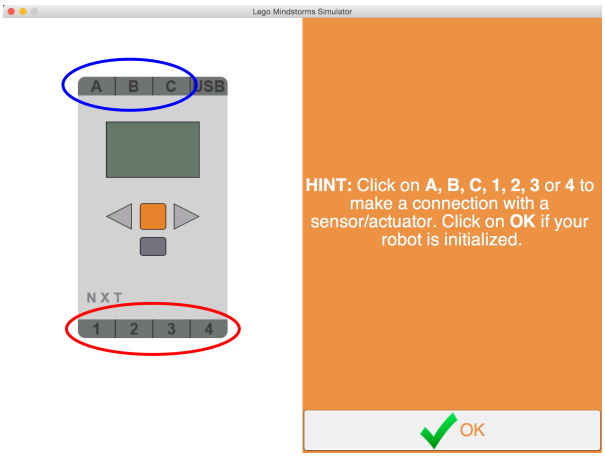
Before starting the NXTSim, you should prepare the following two things.

- At least one MATLAB file (.m) which will be the program of the simulated robot.
- The settings of the ports (1, 2, 3, 4, A, B, C) which tells the connections between ports and sensors/actuators.

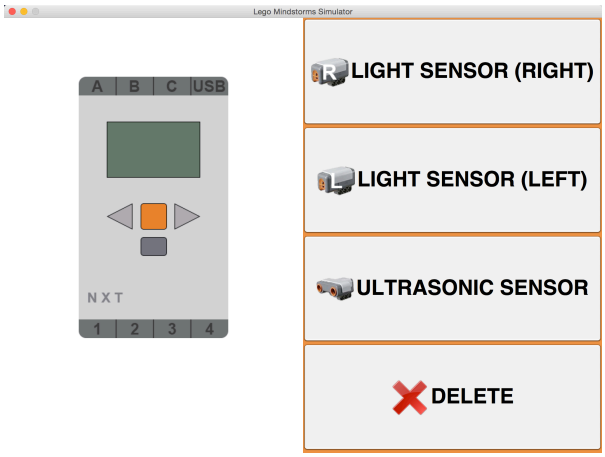
You are now ready to start NXTSim. Open NXTSim.jar. Screen 1 will appear.

Firstly, click on one of the sensor ports (1, 2, 3, or 4) which are encircled with red to connect it with a sensor. After clicking on one of the sensor ports, screen 2 will appear which allows you to choose one of the available sensors. After choosing a sensor, you will see a black line between the sensor port and the chosen sensor which represents a connection.

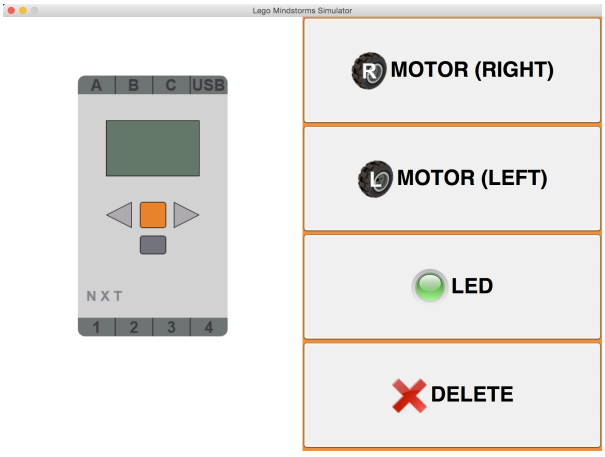
Secondly, click on one of the actuator ports (A, B, or C) which is encircled with blue to connect it with an actuator. After clicking on one of the actuator ports, screen 3 will appear. After clicking on one of the available actuators, it will be connected with the selected actuator port. Screen 4 shows an example where the controller is connected with a number of sensors and actuators.



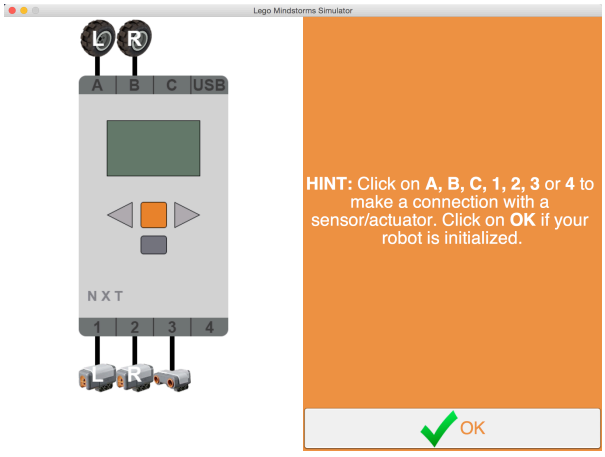
Screen 1



Screen 2



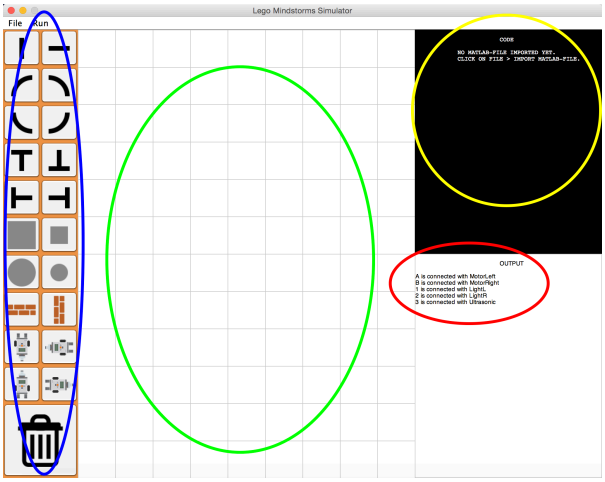
Screen 3



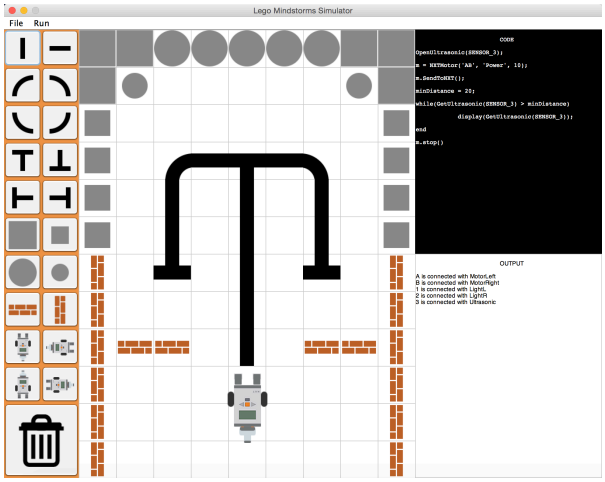
Screen 4

If you are completed with initialization, you can click on the OK button at the bottom right. Screen 5 will appear. This screen consists of the following panels:

- The tools panel at the left, which is encircled as blue. This panel contains objects which can be positioned in the environment panel.
- The environment panel at the middle, which is encircled as green. This panel represents the environment of the simulation.
- The code panel at the top right, which is encircled as yellow. This panel shows the MATLAB file which is imported.
- The output panel at the bottom right, which is encircled as red. This panel shows the following information:
    - the connections initialized in the previous screen.
    - the values of the connected sensors, if there are any.
    - the line of the MATLAB-code which is running by the simulator.



Screen 5



Screen 6

Click on the one of the objects in the tools panel, except the robot. Click on the place in the environment panel where you want to position this selected object. This object will be positioned. Do to the same for all the objects you want to use to build the environment.

You can delete an object or the robot by first clicking on the delete icon at he bottom left, and then on the object or robot which you want to delete from the environment panel.

If you have placed all the objects, you have to position the robot. Click on one of the four rotations of the robot, place it in the environment like you did with the objects in the previous step. Now, the environment is built, and you will import the MATLAB-file.

There are two ways of doing this. The first way is to click on File>Import MATLAB-file in the menu bar at the top of the screen. The second way is to click on the code panel. After doing one of the two ways, you are able to browse through your files, and select the MATLAB file.

Select the MATLAB file. The code panel will show the code of the inputted MATLAB as shown in screen 6.

Now, everything is ready and you can run the program by clicking on Run>Run in the menu bar at the top. The simulation will start.

In the output panel, you can see the line of the MATLAB code which is running. If the simulator is completed with running the MATLAB code, the output panel will show: “Running the MATLAB-file is completed.”

If you want to stop the simulation, click on Run>Abort. The simulation will stop.

You are able to perform a new simulation with the same initialized connections. To perform a new simulation, click on File>Clear environment. All objects and the robot will be deleted from the environment panel. After building a new environment, you can import another MATLAB file, like explained earlier.

To quit NXTSim, click on File>Quit in the menu bar. The program stops and will disappear.

More information on *www.mkucuk.nl/nxtsim.*