SCHOOL OF TECHNOLOGY

UNIVERSITY OF EXTREMADURA

SCHOOL OF TECHNOLOGY

MASTER IN COMPUTER SCIENCE

MASTER DEGREE PROJECT

LearnBot 2.0:

A tool for programming teaching

and

emotional management through robotics

SCHOOL OF TECHNOLOGY

UNIVERSITY OF EXTREMADURA

SCHOOL OF TECHNOLOGY

MASTER IN COMPUTER SCIENCE

MASTER DEGREE PROJECT

LearnBot 2.0:

A tool for programming teaching

and

emotion management through robotics

Author: Iván Barbecho Delgado

Tutor: Pilar Bachiller Burgos

Abstract

The use of robots as tools to facilitate technological education is rapidly gaining interest. The educational robotics allows students to experience situations that contribute to acquire cognitive strategies for solving, planning and execution real problems. The robot LearnBot was designed in the area of educational robotics for promoting the development of computational thinking in different educational stages. LearnBot is a low cost robotic platform which has to be programmed using the Python language. This work aims at developing an improved version of LearnBot to extend this robotic tool to other usages related to emotional management. To this end, using the new robotic platform, called EBO, students can simulate emotional behaviors. In addition, we have developed a specific programming tool for EBO, called LearnBlock, designed for easy usage of the robot. LearnBlock provides a visual language through which children can program robot behaviors in an intuitive way by specifying what the robot has to do whenever a given situation occurs. The language can be easily extended by creating new blocks associated to Python functions. Moreover, LearnBlock programs can run in either the physical robot and a simulated robot. Both, EBO and LearnBlock, are open developments. In this document, the different aspects of the design, implementation and usage of both educational tools are described in detail. In addition, a review of the existing educational robots closely related to our approach is presented, comparing different features these educational tools.

Resumen

El uso de robots como herramienta para facilitar la educación tecnológica está ganando rápidamente interés. La robótica educativa permite a los estudiantes experimentar situaciones que contribuyen a adquirir estrategias cognitivas para resolver, planificar y ejecutar problemas reales. El robot LearnBot fue diseñado en el área de la robótica educativa para promover el desarrollo del pensamiento computacional en diferentes etapas educativas. LearnBot es una plataforma robótica de bajo coste que se programa utilizando el lenguaje de programación Python. Este trabajo tiene como objetivo desarrollar una versión mejorada de LearnBot para extender esta herramienta robótica a otros usos relacionados con la gestión emocional. Con este fin, utilizando la nueva plataforma robótica, llamada EBO, los estudiantes pueden simular comportamientos emocionales. Además, se ha desarrollado una herramienta de programación específica para EBO, llamada LearnBlock, diseñada para facilitar el uso del robot. LearnBlock proporciona un lenguaje visual a través del cual los niños pueden programar comportamientos en el robot de una manera intuitiva especificando qué tiene que hacer el robot cada vez que se produce una situación determinada. El lenguaje se puede extender fácilmente mediante la creación de nuevos bloques asociados a funciones de Python. Además, los programas pueden ejecutarse en el robot físico y en un robot simulado. Ambos, EBO y LearnBlock, son desarrollos abiertos. En este documento, se describen detalladamente los diferentes aspectos del diseño, la implementación y el uso de ambas herramientas educativas. Además, se presenta una revisión de los robots educativos existentes estrechamente relacionados con nuestro proyecto, comparando diferentes aspectos de estas herramientas educativas.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nowadays more and more schools are using different technological resources to prepare their students for the new digital world, where many jobs are related to computer science. For this reason, teachers have started introducing concepts of programming in early ages, using tools like **Scratch** [2], to teach programming to theirs students in a visual and simple way.

These new resources are not exclusively dedicated to learn concepts of programming. In addition, they can be used to complement different teaching units or projects related to logic, mathematics or language, facilitating the teacher work.

The main aim of this project is to create one of these technology resources, developing open-source and open-hardware tools, as flexible as possible, composed of a robot (EBO) capable of recognizing and showing emotions, a program development tool (LearnBlock) devoted to programming different robot behaviors, including emotional behaviors, using an easy-to-extend programming language and, finally, a robot simulator to test the programs when the physical robot is not available.

Next chapter exposes the different objectives of our work. In order to justify the need for a new educational robot, chapter 3 presents an analysis of existing robotic platforms and programming tools used in education. In chapter 4, the different decisions that have been taken for the design and implementation of both, the robot and the programming tool, are described. Chapter 5 details the different stages of

the development of the hardware and software tools composing our project as well as the final result. Results obtained after using the developed tools in different workshops with groups of students of different ages are exposed and analyzed in chapter 6. Finally, chapter 7 summarizes the main conclusions and potential extensions of our work.

Chapter 2

Project objectives

The main objective of this project is to develop an educational robot capable of exhibiting not only motor behaviors, but also emotional ones, acting as a tool to learn computer programming concepts and to work different aspects related to emotional management.

This tool will be composed of:

- A robot totally designed and built by us.
- A development tool to create programs for the robot in an easy way.
- A simulator of the robot and its environment, on which the same programs developed for the physical robot can be executed.

To achieve the maximum success of the project, the following specific goals are proposed:

- Search and analysis of existing educational robots and programming tools.
- Study of requisites and functionality.
- Design and implementation of the robot.
- Design and implementation of the programming tool.
- Selection of a simulator.

- Interconnection between the robot (simulated and physical) and the programming tool.
- Start-up in the classrooms.

Chapter 3

State of Art

This chapter presents a review of different robots and programming tools used in education. We mainly focus on those platforms that present features of interest for our project.

One of the most widely used tools is **WeDo 2.0** [3] developed by *Lego*. This project is composed of a Lego kit for building different robots, with a tilt sensor, a proximity sensor and a motor. In addition, this tool has a coding IDE (*Integrated Development Environment*) to develop different programs. Figure 3.1 shows a robot built using the kit along with an example of code to control the robot.

Figure 3.1: Lego WeDo 2.0 robot and code example

The next analyzed tool is also developed by *Lego* and is called **EV3** [4]. Like **WeDo 2.0**, this project is composed by a Lego building kit, with more sensors and actuators than WeDo, and a coding IDE. This project is designed to build more complex robots. In figure 3.2 some assembly examples using this Lego kit are shown.

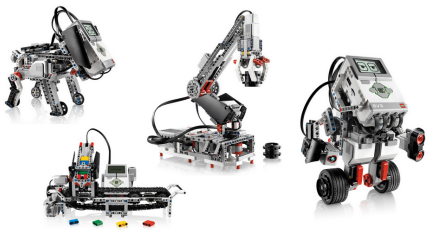


Figure 3.2: 4 assembly examples using the Lego EV3 kit

Another project, similar to the previous ones, is **Mbot** [5], developed by *Makeblock*, which provides a kit to build a robot and an IDE for programming. It is worth mentioning that this project is open-source and open-hardware. Figure 3.3 shows the Mbot Robot and a programming example.

Figure 3.3: Mbot robot and programming example

Besides the building kits, other interesting projects exist. One of them is the robot **Zowi** (figure 3.4) [6], a project developed by *bq* for *Clan TV*, a cartoon channel of *Radio Televisión Española*. Along with the robot, this project includes a coding IDE for programming its movements. The robot can not be modified to add more sensors or actuators.

Figure 3.4: The robot Zowi

Another project related to educational robotics is **Colby** (figure 3.5) [7], a mouse-shaped robot developed by *Learning Resources*. This project has a robot that is programmed by entering a sequence of movement by pressing different buttons. These buttons are on the top of the robot.

Figure 3.5: The robot Colby

Another project, developed by *Makeblock*, is **Codey Rocky** (figure 3.6) [8]. This educational tool includes a robot with the appearance of a panda that is composed of 2 parts which can work individually. One part has a display to show images, a speaker and three buttons among other sensors and actuators. The second part is a platform, where the display is connected, that allows for moving the robot in a horizontal plane. This robot can be programmed using a programming IDE or using a Python API.

Figure 3.6: The Robot Codey Rocky

The next project we have analyzed is **Joy robot** (figure 3.7) [9], develop by *Igor Fonseca Albuquerque*. It has a robot and a programming tool. Joy Robot is an *open-source* and *open-hardware* project and anyone can improve it or customize it.

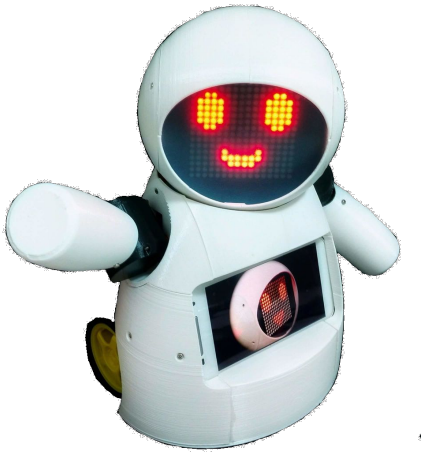


Figure 3.7: The robot Joy Robot

This analysis also includes the project **Thymio** (figure 3.8) [10], another project composed of a robot and a tool for programming. This robot, like **Joy robot**, is an *open-source* and *open-hardware* project. Thymio has numerous sensors and lights.

Figure 3.8: The robot Thymio

Anki is a robotic company that has 2 robots like the analyzed projects. The first one, is **Cozmo**. This robot is sold like a toy. It has a lot of features, but the most prominent one is that it has a ”life”, with a personality. Thus, it is endowed with an artificial intelligence software that allows the user for interacting with the robot. Cozmo can play some games with the user, but in addition, Cozmo can be programmed by the user through its development application, building programs with a visual programming language. This application is a version of the Scracth tool. Figure 3.9 is shows the Cozmo robot.

Figure 3.9: The robot Cozmo

The second one is **Vector** (figure 3.10), an evolution of Cozmo. This new robot extends the features of Cozmo, adding additional components, such as a microphone

matrix, a touch sensor and a laser scanner sensor. The main limitation of Vector is that it cannot be programmed with a specific tool designed for children.

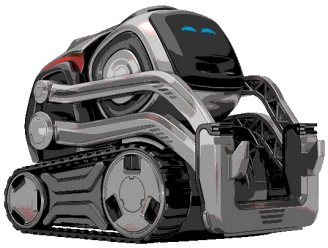


Figure 3.10: The robot Vector

To provide a complete comparison of the different analyzed projects, the following tables summarize the main features and functionality of each project according to certain issues of interest in our work. The first two tables (3.1 and 3.2) present the list of hardware components of each robot.

Table 3.3 compares more general features related to the use possibilities of each project from an educational point of view. Specifically the following features are considered:

- Building kit: as was mentioned above, some projects include robotic building kits that extend the use possibilities in education. Nevertheless, the functionality of these robots is more limited.

- Visual programming tool: visual programming languages have emerged to make programming more accessible to beginners. Among other features, visual languages reduce potential syntactic errors, which makes them suitable as introductory programming languages.

- Integration of general elements of programming languages: such as variables, control structures or functions.

- Simulator support: robot simulation is a very interesting feature of any

Table 3.1: Table with a list of hardware components of WeDo 2.0, EV3, Mbot, Zowi and Colby

	WeDo 2.0	EV3	Mbot	Zowi	Colby
Processor	?	ARM9 300MHz	ATmega 2560	ATmega 328	
Cores	?	1			
Prize	173€	438,99€	121.75€	79,90€	59,99$
Microphone			V	V	
Distance sensor		Ultrasound	Ultrasound	Ultrasound	
Touch sensor					
Cliff sensor		*Can use Ultrasound			
IMU	V	V			
Screen resolution		178×128	Matrix Led	LED Matrix (5 x 6 px)	
Screen Colors		Gray			
Screen Size					
Camera resolution					
Battery duration	?	3h	1-2h	8 h	?
Lights	1		1		V
Motors	Normal	Normal	Normal	Normal	Normal
Speaker					*Speaker or Buzzer
Buzzer			V	V	*Speaker or Buzzer
IR Reciver/ Trasmiting			V		
Line sensor			V		
Parts	280	541	+-50	+-30	?
Connections	Bluetooth/ Wifi	USB/ Wifi / Bluetooth	Bluetooth/ Wifi	Bluetooth	
Color sensor		V	V		
Termometer					

Table 3.2: Table with a list of hardware components of Joy robot, Thymio, Codey rocky, Cozmo and Vector

	Joy robot	Thymio	Codey rocky	Cozmo	Vector
Processor	ATmega2560	?	ESP32	ARM Cortex 4 100 MHz	Qualcomm Snapdragon 1.2GHz
Cores		?		1	4
Prize	?	109.90€	99$	180$	250$
Microphone		1	V		4
Distance sensor		9 InfraRed	1		1 laser (Max 1m)
Touch sensor	*Smartphone	5 capacitive buttons			On the top of the robot, (Capacitive)
Cliff sensor			* the same as the distance.	V	V
IMU		V	V	V	V
Screen resolution	LED Matrix (16x16 px)		LED Matrix	128x64	184x96
Screen Colors	Red		Blue	Blue	Full color
Screen Size	*Smartphone			1.8”	1.8”
Camera resolution	*Smartphone			320x240	720p
Battery duration	?	3-5 h	2h	45-60min	45-60min
Lights		39LEDS	V	4	5
Motors	Normal	Normal	Normal	Normal	Noiseless
Speaker		V	V	V	V
Buzzer					
IR Reciver/ Trasmiting		1	V		
Line sensor					
Parts	+-130	?	?	700	370
Connections	Bluetooth	Wifi	Wi-Fi/ Bluetooth/ USB	Wifi	Wifi
Color sensor			V		
Termometer		1			

Table 3.3: General features of the analyzed projects

	WeDo 2.0	**EV3**	**Mbot**	**Zowi**	**Colby**
Building kit	X	X	X		
Visual programming tool	X	X	X	X	
Integration of general elements of programming languages	X	X	X	X	
Simulator support					
Open-source project			X		
Open-hardware project			X		

	Joy robot	**Thymio**	**Codey rocky**	**Cozmo**	**Vector**
Building kit					
Visual programming tool		X	X	X	X
Integration of general elements of programming languages		X	X	X	X
Simulator support					
Open-source project	X	X	X		
Open-hardware project	X	X			

educational project related to programming and robotics, since this feature adds the possibility of testing the programs when the physical robot is not available.

- Open-source project: possibility to use or modify/adapt the code in a new project.
- Open-hardware project: possibility to make your own replication of the robotic platform.

Regarding the functionalities of the reviewed educational robots, table 3.4 summarizes the most significant ones. We have considered sensory and motor abilities that can be used to program robot behaviours.

Since the objective of this project is not only to create an educational robot, but also to develop a programming tool, a review of the main programming tools in education is also included. We have focused on Scratch and Blockly. Both tools can be modified to create a programming environment for a specific platform. The most complete

Table 3.4: General functionalities of the analyzed projects

	WeDo 2.0	EV3	Mbot	Zowi	Colby
Follow black line	X	X	X		
Follow color line					
Emotional expressions		X		X	
Emotion recognition					
Tag recognition					
Obstacle detection	X	X	X		X
Base motion	X	X	X	X	X
Face detection					
Touch detection				X	
Slant detection	X	X			
Sound detection		X			
Sound making		X	X	X	X

	Joy robot	Thymio	Codey rocky	Cozmo	Vector
Follow black line	X	X	X		
Follow color line					
Emotional expressions	X		X	X	X
Emotion recognition				X	
Tag recognition				X	X
Obstacle detection		X	X		X
Base motion	X	X	X	X	X
Face detection				X	X
Touch detection		X			X
Slant detection		X	X	X	X
Sound detection		X	X		X
Sound making		X	X	X	X

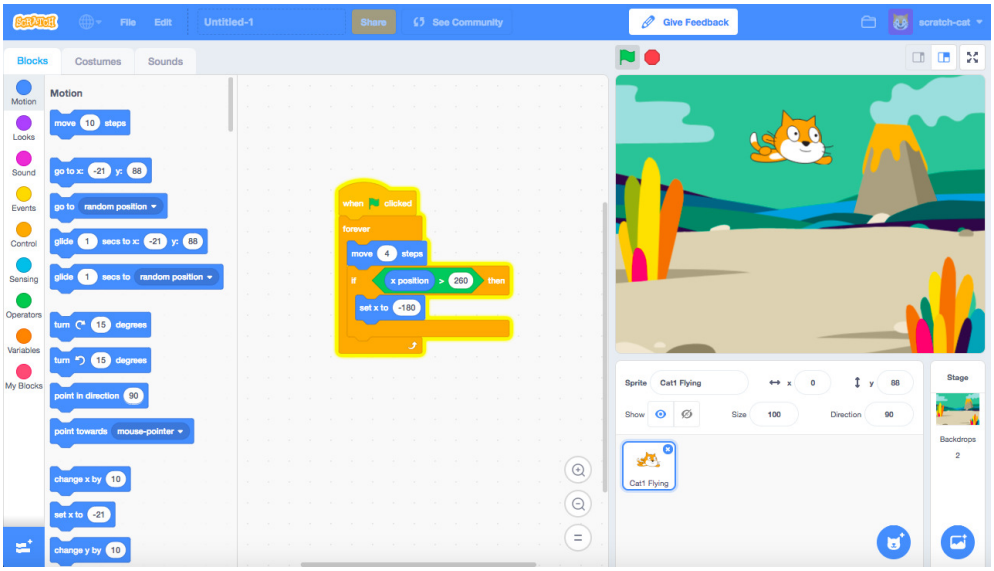


Figure 3.11: Scratch graphical interface

programming tools of the previously described projects are based on one of these two frameworks.

Scratch[11] has been developed by the *Team Lifelong Kindergarten*. Programs in Scratch are created in a visual way, through block connections. This tool offers two different programming models: sequential and event-driven[1]. Scratch is open-source and anyone can collaborate in the project. Figure 3.11 shows the graphical interface of this tool and a code example.

Blockly [12] has been developed by the *Google Team* with certain similarities to Scratch. Thus, it provides a graphical interface where programming is solved through interconnected blocks. The basic version of Blockly includes a very limited number of blocks, which are translated to JavaScript, PHP, Dart and Python. Nevertheless, since it is an open-source project, it can be adapted to an specific use. Many tools can be found created as extensions of Blockly. In fact, most of the programming tools of the aforementioned robots are extensions of Blockly. Blockly can be extended easily, using an online tool to create new blocks, adding libraries of functions and the translation to different programming languages for each block. Figure 3.12 shows an example of

[1]Event-driven programming is a type of programming where a program is composed of independent portions of code. Each portion is run when a given condition is activated.

Blockly that generates Arduino Code.

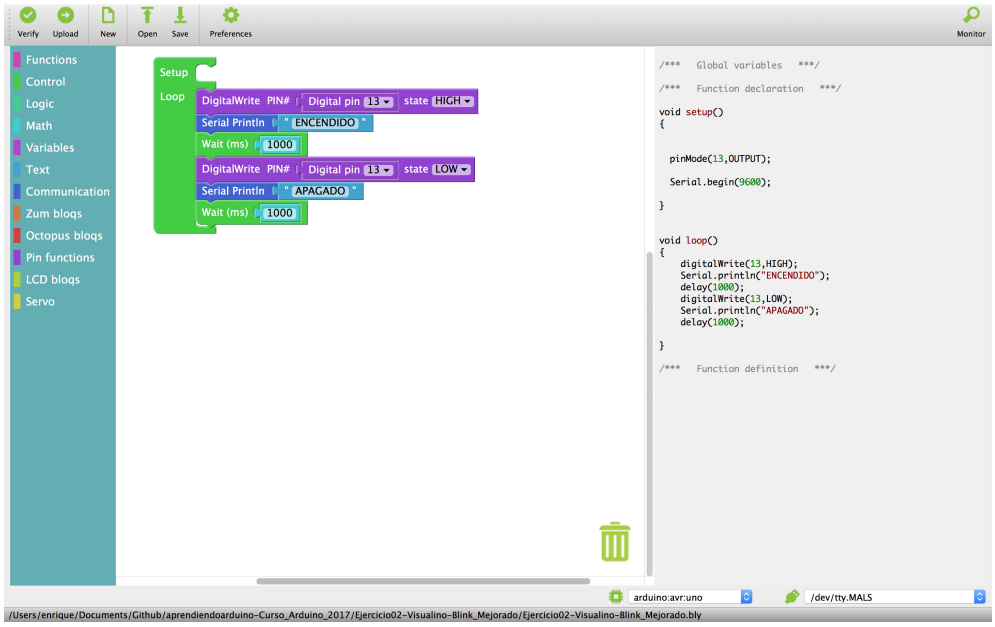


Figure 3.12: Extension of Blockly to generate code for Arduino

Table 3.5 compares the main features of both programming environments. In particular, the following characteristics are considered:

- Create functions with blocks: the user can create functions that are implemented through the use of other existing blocks.
- Functions created with blocks can return a value: defined functions can return values.
- Functions created with blocks can be exported to be used in other programs: defined functions can be exported to create a set of library-like functions. This library could be imported into new programs to use them in new projects.
- Create blocks from code: the tool allows defining blocks that are implemented using textual programming.
- Facilitates the creation of blocks from code: new blocks can be easily implemented using textual programming, without having to edit a lot of code.

This feature will be considered only when this process can be done from the tool itself.

- Multilingual options: The tool can be used in different languages, like Spanish, English, etc.
- Can be used from a web browser: he tool can be used from a web browser.
- Can be downloaded into the PC: The tool can be downloaded and installed in a PC to be used in local mode.
- Has a community of shared code: there is a community in which users publish their programs.
- Consistency of variables: when the user makes use of variables and eliminates some of them, these are deleted from the program code maintaining the consistency among all the variables.
- Can be adapted to other robots: The tool can be used to program other robots.
- High-level functions already implemented can be used in adapted projects: high-level functions, such as ”there is an obstacle” or ”there is a person”, can be used in projects adapted to other robots, without modifying the implementation of the function.
- Includes a hardware abstraction layer: the tool provides a hardware abstraction layer that facilitates the task of adapting the tool to other robots.
- The tool directly translates from block code to other programming languages: the tool directly translates a program written in visual programming to other programming languages, such as Python.

Analyzing both development tools using the features described above, it can be observed that Blockly includes all the features available in Scratch and provides additional ones. Thus, in Scratch, new function blocks can be created from JavaScript,

Table 3.5: General features of the analyzed programming tools

	Scratch	Blockly
Create functions with blocks	X	X
Functions created with blocks can return a value		X
Functions created with blocks can be exported to used in other programs		
Create blocks from code	X	X
Facilitates the creation of blocks from code		
Multilingual options	X	X
Can be used from web navigator	X	X
Can be downloaded in the PC	X	X
Has a community of shared code	X	
Consistency of variables	X	X
Can adapted to other robots	X	X
High-level functions already implemented can be used in adapted projects		
Includes a hardware abstraction layer		
The tool directly translates from block code to other programming languages		X

but this process is not an easy task, since it does nor provide any tool to create this code. In addition, when the user define a variable and delete it at some point, the variable is not deleted from the code, but is created again when the program is executed without any notification to the user. On the contrary, Blockly controls the delection of variables in a more proper way. In addition, Blockly can return values in functions defined with blocks and new blocks can be easily created from JavaScript using a specific tool, although this tool is not in integrated in the programming environment, but in another web service. Finally, Blockly can automatically translate the visual code to other programming languages, while Scratch does not provide a direct translation (although some external tools exist to translate a Scratch project to other programming languages).

From this analysis of the different robots and programming tools in education, different decisions have been made during the development of this project. These decisions are described in the next chapter of this document.

Chapter 4

Methodology

This chapter describes the main questions that were considered at the beginning of this project to make the most important decisions about the design of the robotic platform and the programming tool.

The first questions we should answer is:

At what ages will the project be destined?

According to Piaget's theory of cognitive development the child has four stages (table 4.1) . Taking into account the cognitive features of each stage, it can be said that the project is suitable for children between 10 and 12 years old, since the stages **"concrete operations"** and **"formal operations"** appropriately fit the features of this project.

The selected period involves that, in the Spanish educational system, this project will be deployed in the courses 4^o, 5^o and 6^o of primary education.

Once the first question has been answered, design decisions concerning the **robot** and the **programming tool** must be independently made, considering the different particularities of both components.

4.1. The Robot

Regarding the robot design, the following questions were initially considered:

Table 4.1: Stages table of Cognitive Growth from Piaget's Perspective[1]

Stage	Age	Features
Senrorimotor The active child	Birth to nearly 2 years	Children learn proactive behavior, thinking oriented to means and ends, the permanence of objects.
Preoperational The intuitive child	About 2 to 7 years	The child can use symbols and words to think. Intuitive solution of problems, but thought is limited by rigidity, centralization and self-centeredness.
Concrete Operations The practical child	About 7 to 11 years	The child learns the logical operations of seriation, classification on and conservation. The thought is linked to the phenomena and objects of the real world.
Formal Operations The thoughtful child	About 11 to 12 years and through adulthood	The child learns abstract systems of thought that allow him/her to use propositional logic, scientific reasoning and proportional reasoning.

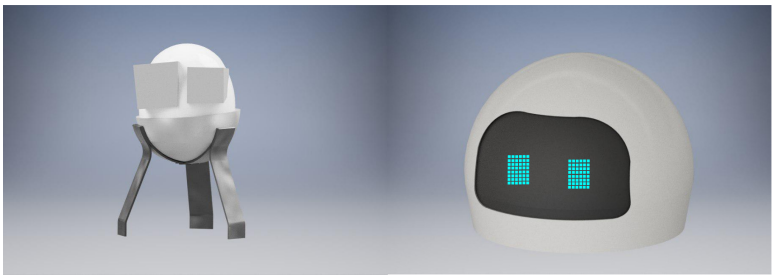
- What should be the aspect of the robot to receive greater acceptance from the children?
- What sensory-motor abilities should the robot have?
- Should it be a construction kit?
- Should the robot be designed from scratch or reusing parts of an existing project?

Each answer to these questions constitutes an important design decision. Next sections describe in detail the most important points of our proposal according to the previous questions.

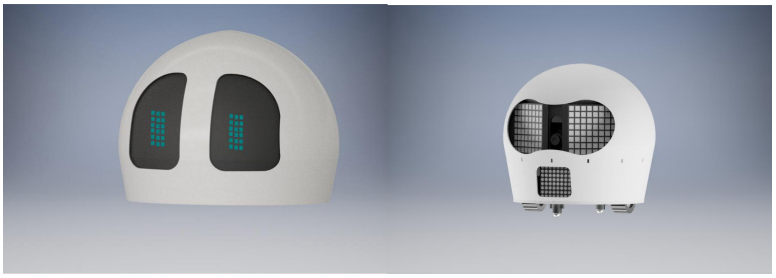
4.1.1. What should be the aspect of the robot to receive greater acceptance from the children?

To solve the first design issue, a survey was made to a group of children. Initially, children were asked to draw a robot with the aspect they thought it should have. The majority of the children drew complex robots, with arms and legs, that were difficult

to design and build. For this reason, several designs (figures 4.1a, 4.1b, 4.1c and 4.1d) were made and children were ask to select the favorite one.



(a) Design 1 (b) Design 2



(c) Design 3 (d) Design 4

Figure 4.1: Robot designs used in a survey.

The design that received more votes was 4.1b. Thus, this design was initially taken to build a prototype of our robot.

After detecting some design flaws, certain aspects of the robot were redesigned, but respecting the rounded shape of the robot, as far as possible. The final design is shown in the section 5.1.3

4.1.2. What sensory and motor abilities should the robot have?

Considering the main abilities of the analyzed educational robot, the following abilities were established as essential:

- Move through its surroundings.
- Perceive visual stimuli.
- Detect obstacles.

Extending these general abilities to more specific ones, the following skills were included:

- Turn to the left and to the right.
- Move forward.
- Move to the left and to the right.
- Detect different colors.
- Detect people/faces.
- Recognize emotions.
- Detect frontal, right and left obstacles.
- Express emotions.
- Recognize marks (e.g. *AprilTags*).

With this set of skills, there is a wide range of combinations to define different robot behaviors that can be of interest for different teaching units.

4.1.3. Should it be a construction kit?

Taking into account that the robot that is intended to design requires complex hardware, it is considered that the design of a mounting kit for the construction of the robot is not a priority. However, since our project is open-hardware, anyone has access to the designs and the construction process, being able to replicate the robot or even modify its appearance or incorporate new sensors or actuators.

4.1.4. Should the robot be designed from scratch or reusing parts of an existing project?

In 2005, RoboLab, the Robotic Laboratory of the University of Extremadura, developed an educational robot [13], called LearnBot, which meets some of the

expectations of the present project. LearnBot (figure 4.2) uses the hardware platform Odroid-C1 and is composed of the following sensors and actuators.

- 4 Ultrasonic sensors.
- 1 Fixed camera.
- 1 Differential base.

Figure 4.2: Robot of LearnBot's Project.

LearnBot is programmed in Python using the open-source framework RoboComp [14], which has been designed to be used in the field of robotics and is heavily based on the component-oriented programming (COP) paradigm. It offers the possibility to create components in an easy and simple way. The communication between these components is carried out with public interfaces using the communication middleware ICE [15].

4.2. Integrated Development Environment

Design decisions regarding a programming tool for our robot have been based on the following questions

- What types of programming languages should the tool allow for?

- What programming models should the tool provide?
- Can the development start from an existing project?

As in the previous section, each answer supposes an important decision for the development of the coding tool. Next subsections analyze these questions in detail.

4.2.1. What types of programming languages should the tool allow for?

The purpose of this question is to determine whether the programming language should be textual, visual or both. Each type of programming language has certain benefits from an educational point of view. For textual programming languages, the following points can be highlighted:

- Similar to those used in industry.
- Wide variety of languages.

For **visual** programming languages:

- Easy to learn and use.
- Suitable for beginners.
- Reduce or even eliminate syntactic errors.
- Soft learning curve.

Considering that our project is destined to children, the IDE should have a visual programming language, since the properties of this type of programming language make it ideal for non-programmers. Nevertheless, in the long term, we believe that the tool has to offer other options that provide the possibility of extending and improving the programming skills of the children. For this reason, we decided that the programming tool should include 3 programming languages:

- A visual programming language.
- A textual representation of the visual programming language.
- An industry programming language.

Thus, providing different types of languages facilitates a complete learning of an industrial programming language, starting with visual programming, then with a middle language, and, finally, using an industrial language.

4.2.2. What programming models should the tool provide?

Regarding programming models, the two basic types available in other educational tools are considered: **sequential programming** and **event-driven programming**. **Sequential programming** is the basic model of programming. Many programming languages use this model, so providing this feature in the selected programming languages is essential. On the other side, in the **event-driven programming** paradigm, a program is designed to detect events as they occur and react to those events by executing the associated portions of code, which can be seen as *reactions* to different potential situations. This conception is very interesting in robotics for programming robot behaviours. After analyzing the advantages of these two programming models, we decided to include both of them. Thus, the user can select which model is more suitable to solve an specific problem.

4.2.3. Can the development start from an existing project?

Since the robot design is initially based on the LearnBot robot, the design of the programming tool also starts from the same project. As previously mentioned, LearnBot is programmed in Python, using an intermediary class to communicate with the physical robot. Thus, this intermediary class acts as a *Hardware Abstraction Layer* (HAL).

Since the LearnBot project does not include an IDE, the programming tool has been entirely developed from the beginning. Nevertheless, we have taken advantage of

the idea of using a HAL class due to the benefits of separating access to hardware from high-level functionality. In addition, Python has been selected as the industrial textual language of our tool (and also as the language for the tool development) because of its simplicity and its small learning curve.

Beside these two inherited features from the LearnBot project, we have also included the possibility of executing the created code in the RoboComp simulator RCIS [16]. Using the HAL class, this additional feature can be included with little effort. Moreover, the same principles used to add this extension can be applied for adapting the tool to any other robotic platform. This constitutes a significant feature of our tool that marks the different with regards to other educational programming tools.

Chapter 5

Implementation and development

This chapter is organized in three sections that deals with the robot development, the communication between the robot and the user PC and the programming tool development.

5.1. The robot

Regarding the robot, many changes have been made to the original LearnBot robot due to the different requirements of the new robotic platform. These changes are described in the following subsections according to 3 issues: hardware, software and design The final can be seen in figure 5.1.

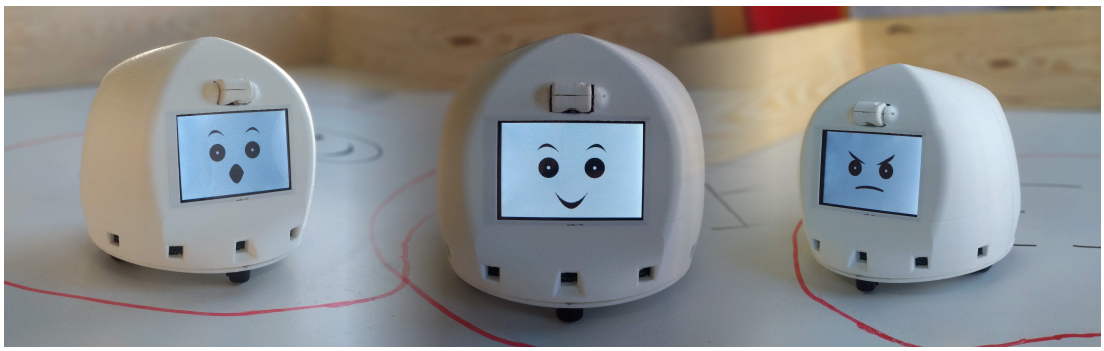


Figure 5.1: The EBO robot.

5.1.1. Hardware

The new features of the robot make necessary to include the following changes in the hardware components:

- Replace the USB camera with a raspberry camera.
- Replace the ultrasonic sensors with laser sensors.
- Add a screen to show images.
- Add 1 degree of freedom to the camera to vertically cover a wide field of view.

Beside these changes of sensors and actuators, the Odroid-C1 hardware platform used in LearnBot has been replaced with a Raspberry pi 3B +. This decision was made because Raspberry includes a CSI port (a video output through pins) to connect a camera, while in Odroid video output can only be obtained through HDMI. Additionaly, Raspberry has an integrated Wi-Fi module that is not available in Odroid.

With these changes, our robot will have the following hardware components:

- Raspberry Pi 3 Model B+: where the host system runs to control the other hardware components
- Camera with CSI connector: to capture visual information.
- Servomotor model SG90: it provides movements to place the camera at a certain vertical angle.
- 3.5 inch Resistive Screen (PiTFT 3.5”): to show images (e.g. emotions)
- 5 laser sensors (VL53L0X): to obtain distance information from objects around the robot.
- PWM pin extender (Adafruit 16-Channel PWM): to provide a stable output to the servomotor and also to configure the 5 lasers.

- 2 298:1 DC motors (73 RPM) in differential configuration to move the robot base.
- Motor controller (DRV8835)
- Battery (7.4V) responsible for supplying power to the robot.
- DC-DC voltage regulator (D24V50F5) to reduce the voltage from 7.4V to 5V.

In addition to these hardware components, it has been decided to design a *shield* (figure 5.2) to connect all these components, reducing significantly the number of wires.

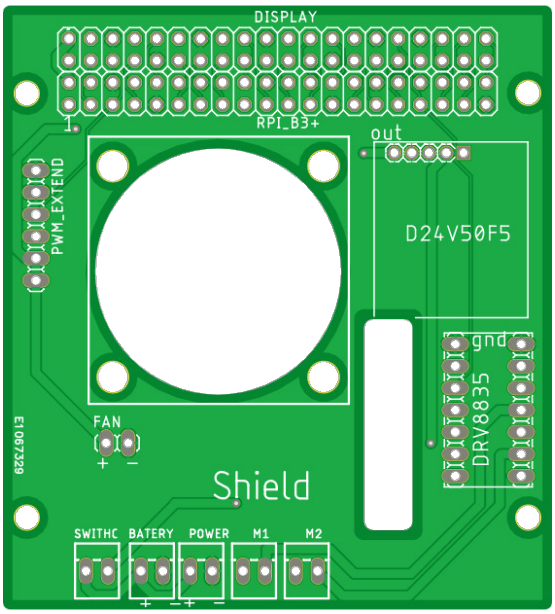


Figure 5.2: Shield that connects the components to the Raspberry.

Figure 5.3 shows a scheme of the connections between the different components of the robot. In this figure, it can be observed that the Raspberry is connected to the camera though the CSI port and to the shield through a 40-pin GPIO header. The shield provides connection to the screen, the DC regulator, the motors of the base and their driver, the power connector, the fan, the battery, the switch and a PWM extender. Finally the PWM extender connects the lasers and the camera servomotor.

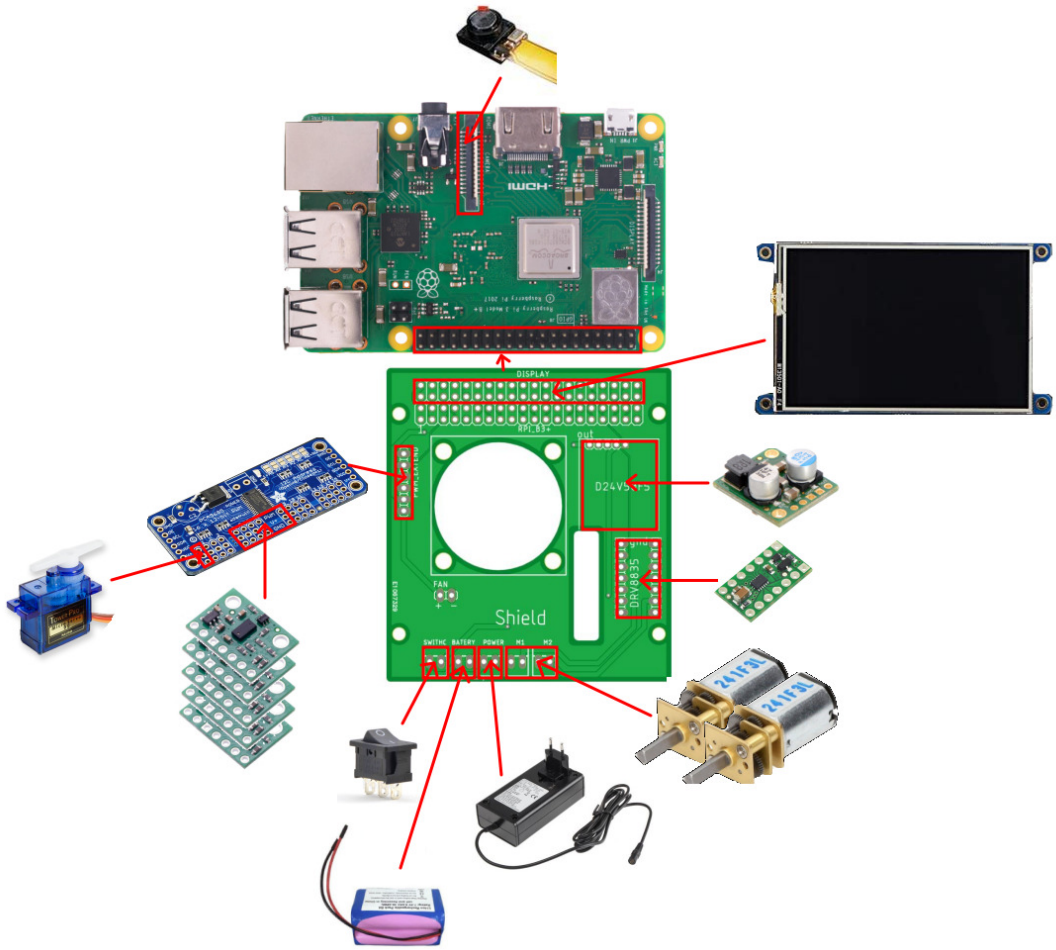


Figure 5.3: Connection diagram of the EBO hardware.

5.1.2. Software

The different changes performed in the control software are motivated by the applied hardware changes. Next, these changes are detailed:

- Creation of a *camera streamer*. This change consists of adapting the existing streamer to the new camera. To improve the *fps* rate, we decided to create a *mosquitto broker* [17] and publish the data under a *topic*. Thus, clients have to subscribe to this topic to obtain the images. Using this protocol, a frame rate of 20 *fps* is obtained.

- Creation of a component that offers the lasers information. Since ultrasonic sensors have been replaced with laser sensors, a new component was created

to provide information of the distance to obstacles around the robot. This component firstly assigns an I2C address to each laser by switching off and then on each laser through a PWM of the PWM extender. After assigning each I2C address, the information of the lasers is read and transmitted under request through an ICE interface.

- Creation of a component to show images on the screen. This change is due to the inclusion of a screen in the new robot. The component has an ICE interface through which it receives the path of a file or the information of an image to be displayed on the screen. This image is sent to the *framebuffer* of the screen so that it is displayed in it.

- Creation of a component to move the servomotor of the camera. This new component is used to control the servomotor that moves the camera. The servomotor is connected to a PWM pin of the PWM extender. When this component receives an angle through an ICE interface, it is responsible for sending the signal to the PWM. This signal is finally received by the servomotor to move the camera.

Beside these changes, a new component was created to generate emotional expressions and send the generated images to the screen component. This component has an interface with 7 methods, one for each emotion that the robot can express:

- void expressJoy()
- void expressSadness()
- void expressSurprise()
- void expressFear()
- void expressAnger()
- void expressDisgust()

- void expressNeutral()

When one of these functions is called, this component is responsible for making a smooth transition from the current emotion to the new emotion. To make these transitions, each emotion has been parameterized with different points and radii.

- 1 center and 2 radii to define the bounding box of each eye.
- 1 center and 1 radius to define the bounding box of the pupil of each eye.
- 4 points for each eyebrow, 2 for the ends, 1 for the upper midpoint and another one for the lower midpoint.
- 6 points for the mouth, 3 for the upper lip and 3 for the lower one.
- 4 points for the tongue, 2 for the ends and 2 for the tip of the tongue.

Summarizing, the control software of the robot is composed of the following components and scripts:

- DifferentialBase: the same component employed in the robot LearnBot.
- Laser.
- JointMotor: control of the camera servomotor.
- Display: screen.
- Emotionalmotor: generation of emotional expressions
- Camera.

Together with this control software, two additional components are executed in the robot. These two components provide perceptual high-level information related to recognition of facial expressions and detection of artificial marks:

- EmotionRecognition: recognizes the 5 basic emotions. This component receives an image, detects faces in it and identifies an emotion for each detected face.

- AprilTag: recognizes AprilTag markers. This component receives an image, detects AprilTag markers and returns information about the position of each detected marker and its identifier.

5.1.3. Design

To deal with the new requirements, many parts of the robot were redesigned.

The final robot is composed of the following parts:

- Base: upon it, the other parts are placed.

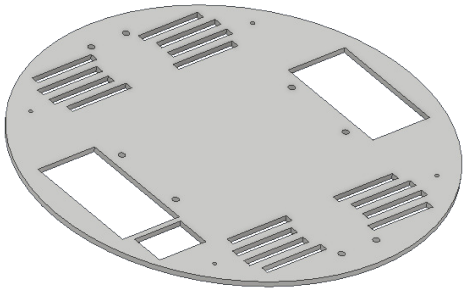


Figure 5.4: Base of the robot

- Camera housing (2 pieces): the camera is inside it.

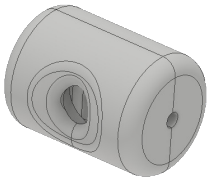


Figure 5.5: Housing of the camera

- Support for the Raspberry: the Raspberry and the shield with all its components are placed upon it. It also ensures that the motors of the wheels are attached to the base.

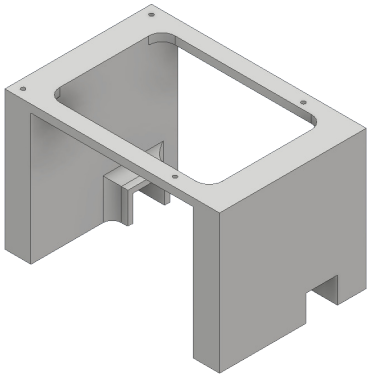


Figure 5.6: Raspberry support

- Frontal shell: it contains the screen, the 5 lasers, the servomotor, the camera and the PWM pin extender.

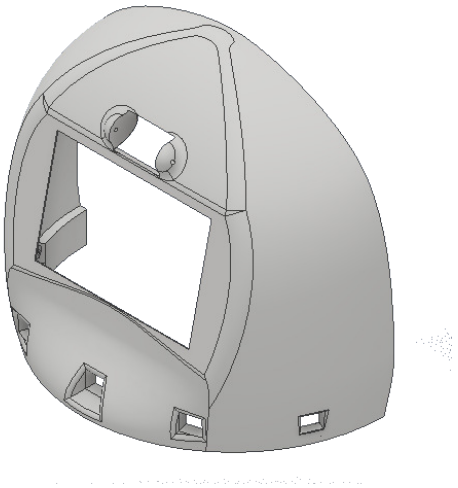


Figure 5.7: Frontal shell

- Rear shell: it contains the charging connector. It also includes 5 ventilation slots to prevent the Raspberry from overheating.

Figure 5.8: Rear shell

5.2. Communication between EBO and the PC

This section details the different steps followed, as well as the modifications made to the client class of the LearnBot project, for the implementation of communications between the EBO robot and the end user's computer.

In the LearnBot project, the user PC is connected to the robot through a Wi-Fi access point provided by the robot. We decided to maintain this way of connection since a program only models the behavior of one robot.

Once the connection between the robot and the PC has been established, the communication with sensors and actuators is carried out using an intermediary class written in Python, as was discussed in section 4.2.3. This class is connected to the different components deployed in the robot through different proxies associated to each component.

The intermediary class had to be modified to contemplate the different new components that were added to the robot (*display*, *jointmotor*, *emotionalMotor*). These modifications entail connecting to the proxies of the new components and implementing new methods to send or receive data from these components. After including all the changes, the new intermediary class presents the structure shown in

listing 5.1.

By means of this intermediary class, the user can write a Python program that controls the EBO robot. Nevertheless, writing this program would be too complex for the educational levels to which the project is destined. For this reason, we decided to implement high-level functions that make data processing easier to the users.

These functions are defined according to the following rules:

- Each function is implemented in a separated file.
- The name of the file must be the same as the name of the function.
- The function must receive at least one parameter corresponding to an instance of the intermediary class.

Code 5.2 shows an example of function implementation. This function checks if the distance measured by 3 laser sensors, located in front of the robot, exceeds or not a threshold.

All these functions are available for use in a dictionary. A call to the implemented function of the example in listing 5.2 is done as follows:

functions.get("front_obstacle")(lbot, 200)

Once, this dictionary of functions is available, more complex programs can be created in a more abstract way. To ease programming even more, the programmer does not need to program the robot in Python, but using a visual language through the programming IDE developed as part of our project. Next section describes the implementation of this tool.

5.3. The Programming Tool

The new programming IDE, called LearnBlock, has been designed to integrate all the necessary tools to program and control the robot

As previously mentioned in section 4.2, using this IDE, the robot can be programmed from three types of programming languages:

Listing 5.1: Client class that interconnects the robot to a PC

```
class Client(Ice.Application, threading.Thread):
    def run(self):
        ...
    def lookingLabel(self, id):
        ...
    def stop(self):
        ...
    def stopped(self):
        ...
    def getImageStream(self,image):
        ...
    def readSonars(self):
        ...
    def getSonars(self):
        ...
    def getImage(self):
        ...
    def getPose(self):
        ...
    def setAngleJointMotor(self, angle):
        ...
    def setRobotSpeed(self, vAdvance=0, vRotation=0):
        ...
    def expressJoy(self):
        ...
    def expressSadness(self):
        ...
    def expressSurprise(self):
        ...
    def expressFear(self):
        ...
    def expressAnger(self):
        ...
    def expressDisgust(self):
        ...
    def expressNeutral(self):
        ...
    def setJointAngle(self, angle):
        ...
    def getCurrentEmotion(self):
        ...
    def getEmotions(self):
        ...
```

Listing 5.2: Example implementation of a function. (Front_obstacle.py)

```
def front_obstacle(lbot, threshold= 200, verbose=False):
        sonarsValue = lbot.getSonars()[1:4]
        if min(sonarsValue) < threshold:
                if verbose:
                        print('Obstacle in front of LearnBot')
                return True
        if verbose:
                print('No obstacle in front of LearnBot')
        return False
```

- Industrial language (Python).
- Domain specific language (EBOCode).
- Visual language.

5.3.1. Industrial programming language (Python)

LearnBlock includes a text editor (figure 5.9), where Python programs can be created. The user can create the code to control the robot using the intermediary class that gives access to its low-level components. In addition, all the available high-level functions can be included in the code to program complex robot behaviors with few lines of code.

5.3.2. Domain specific language (EBOCode)

Together with the text editor for Python programming, the tool includes a text editor window for programming the robot by means of s domain specific language[1](DSL).

The DSL was defined as a middle language between Python and visual programming languages. This new language can be understood as a textual representation of the visual language. To create our DSL, called **EBOCode**, different

[1]A domain specific language is a programming language or specification defined for solving a particular problem, representing a specific problem domain and providing a technique to solve a particular situation.

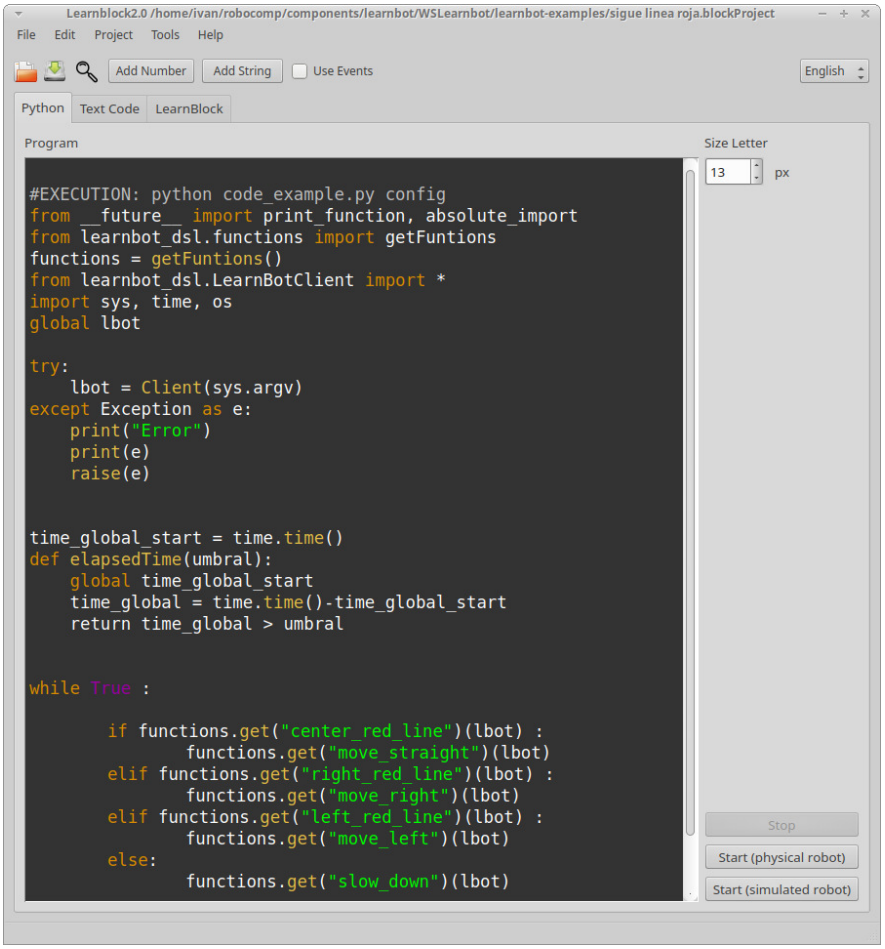


Figure 5.9: Integrated text editor for programming in Python. Example of a Python code that makes the robot to follow a red line.

symbols and rules were defined. The following three types of sentence constitute an example of our language syntax:

- If-elif-else:

```
if condition:
    ...
[elif codition:
    ...
]*
[else:
    ...
]
end
```

- While:

```
while condition:
    ...
end
```

- Robot functions:

```
function.<function name>(lbot,[arguments]*)
```

Beside these basic grammar rules, two programming models are considered in our language:

- **Sequential programming**: the code is include inside a *main* section. There is only one running thread, the one associated to this *main* section. Figure 5.10 shows an example of this programming type.

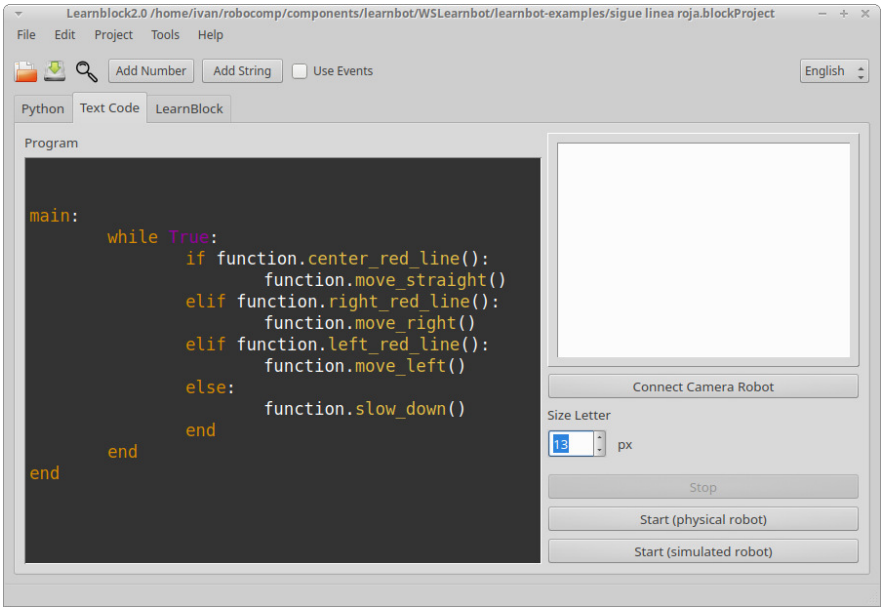


Figure 5.10: Implementation of the "follow red line" behavior using EBOCode with sequential programming.

- **Event-driven programming**: using this programming model a program is composed of different blocks of code associated to certain events. Each event

is defined using a sentence type called “when”. The code inside this sentence is executed whenever the defined condition is satisfied. A condition can be achieved by an external event or by an internal activation (explicit activation during the execution of a block of code). Every block of code associated to a “when” sentence whose condition is satisfied runs in a separated thread. Beside external events, an event called *start* exists. This event is used to execute a block of code when the execution starts. Figure 5.11 shows an example of this programming type.

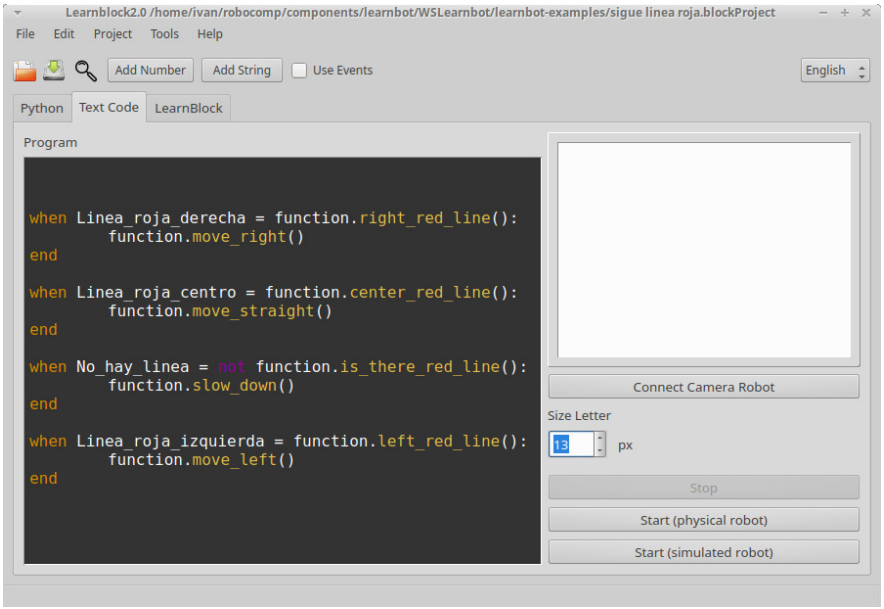
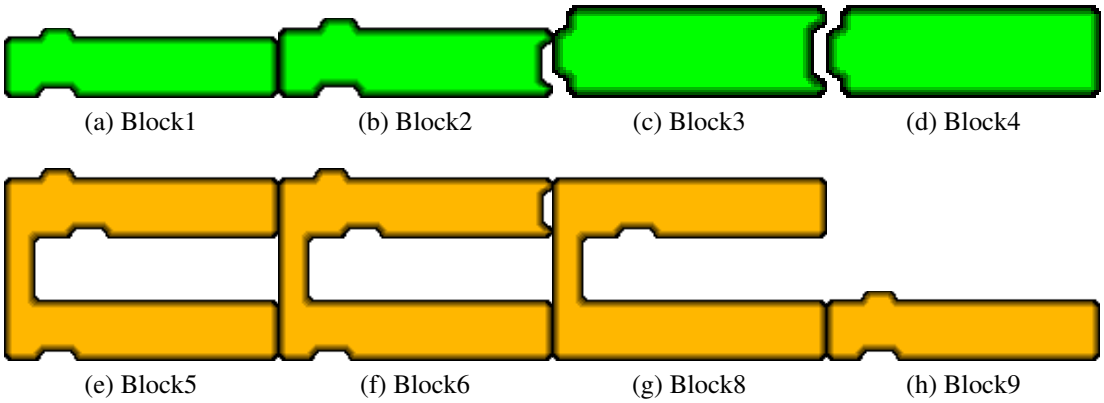


Figure 5.11: Implementation of the “follow red line” behavior using EBOCode with event-driven programming.

Once a program using EBOCode is created, the tool Learnbock translates it to Python language and generates a final program ready to be executed.

5.3.3. Visual programming language

The most complex part of the development of our programming tool is related to the visual programming language, since it requires a higher level of abstraction for the final user.



(a) Block1 (b) Block2 (c) Block3 (d) Block4

(e) Block5 (f) Block6 (g) Block8 (h) Block9

Figure 5.12: Shapes of the different available blocks.

The following steps were carried out to include the visual programming option in our tool:

- Definition of the different blocks (shapes).
- Definition of the different types of blocks (functions, operations, control, etc.)
- Connection between blocks.
- Translation from visual language to EBOCode.

To define the block shapes, we decided to create images with the shapes of the different blocks. These are shown in figure 5.12.

To define the different blocks that will be available to the user, a configuration file is used. This file contains the attributes of each block in JSON format. The listing 5.3 shows a configuration example of a block.

From this configuration, 2 blocks are created with the shapes of 5.12d and 5.12c that correspond to the final blocks of figures 5.13b and 5.13a.

Once all the blocks are configured, they appear on the left side of the tool (figure 5.14), organized according to the the field *type* in different tabs.

To create a program, users have to place and connect the blocks in the central panel. Once the user clicks over a block of the left tabs, it appears in the central panel. Then, the user can move the block to any position of this panel.

Listing 5.3: Example configuration of the front_obstacle block

```
[
{
    "type" : "perceptual",
    "name" : "front_obstacle",
    "variables" : [
           {
               "type": "float",
               "name": "threshold",
               "default": "200",
               "translate": {"ES": "umbral", "EN": "threshold"}
           }
       ],
    "img" : ["block4", "block3"],
    "languages" : {"ES": "hay_obstaculo_delante",
                   "EN": "front_obstacle"},
    "tooltip" : {"ES": "Devuelve verdadero si la distancia de los
                    láseres frontales es menor que el umbral",
                 "EN" : "Returns true if the distance of the front
                 lasers is less than the threshold" }
}
]
```

(a) Block generated from block3.

(b) Block generated from block5.

Figure 5.13: Two different types of blocks for the same function.

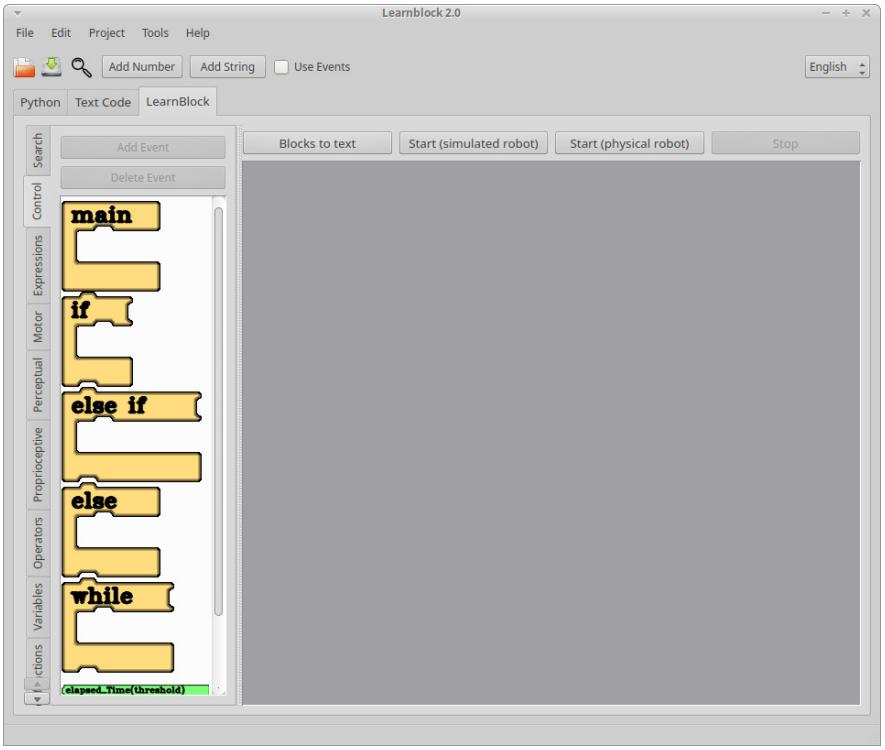


Figure 5.14: Interface of the programming tool for visual programming. Blocks are shown on the left, organized according to the *type* property of the configuration file.

Types of connectors of each block limits the connections among blocks with the main objective of reducing potential syntactic errors. According to their position in the block, connectors are classified as follows:

- TOP: located at the upper side of the block.
- BOTTOM: located at the lower side of the block.
- BOTTOMIN: located inside the block.
- RIGHT: located at the right side of the block.

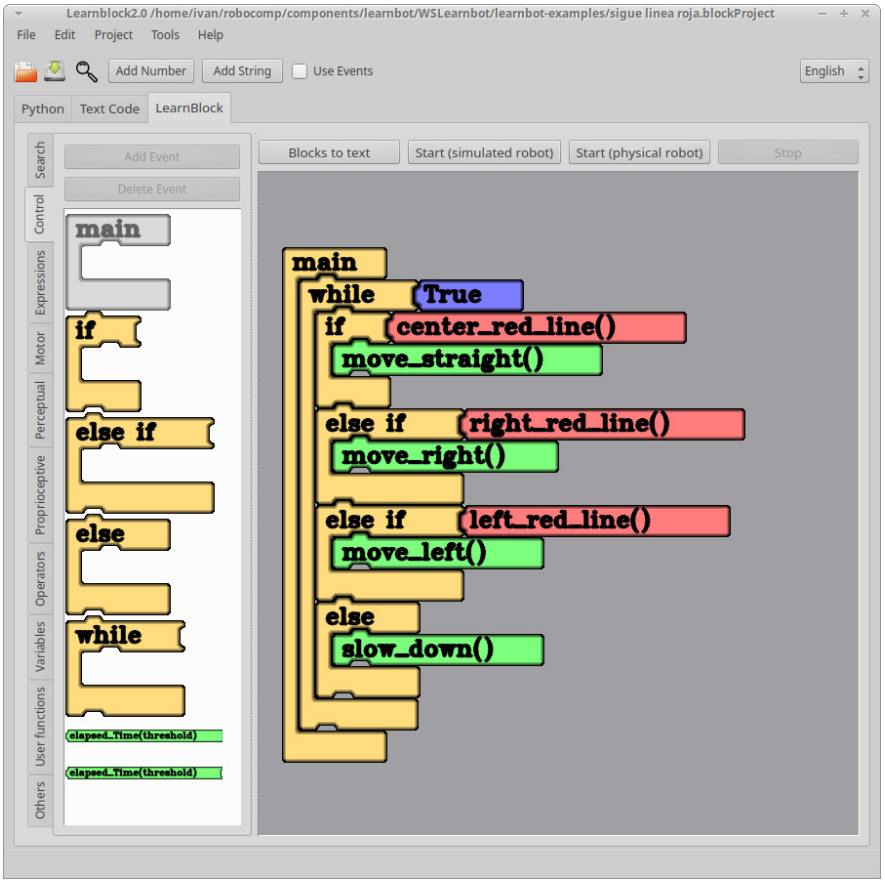


Figure 5.15: Implementation of the “follow red line” behavior using sequential visual programming.

- LEFT: located at the left side of the block.

Using these types of connectors, connections between blocks are restricted to:

- $TOP \Longleftrightarrow$BOTTOM
- $TOP \Longleftrightarrow$BOTTOMIN
- $RIGHT \Longleftrightarrow$LEFT

As in the EBOCode language, the two programming models, sequential and event-driven, are available in the visual programming mode of the tool. Figures 5.15 and 5.16 show examples of the two programming models using blocks.

When the user build a program with the visual language, this code is translated to EBOCode and then, translated again from EBOCode to Python. The generated Python

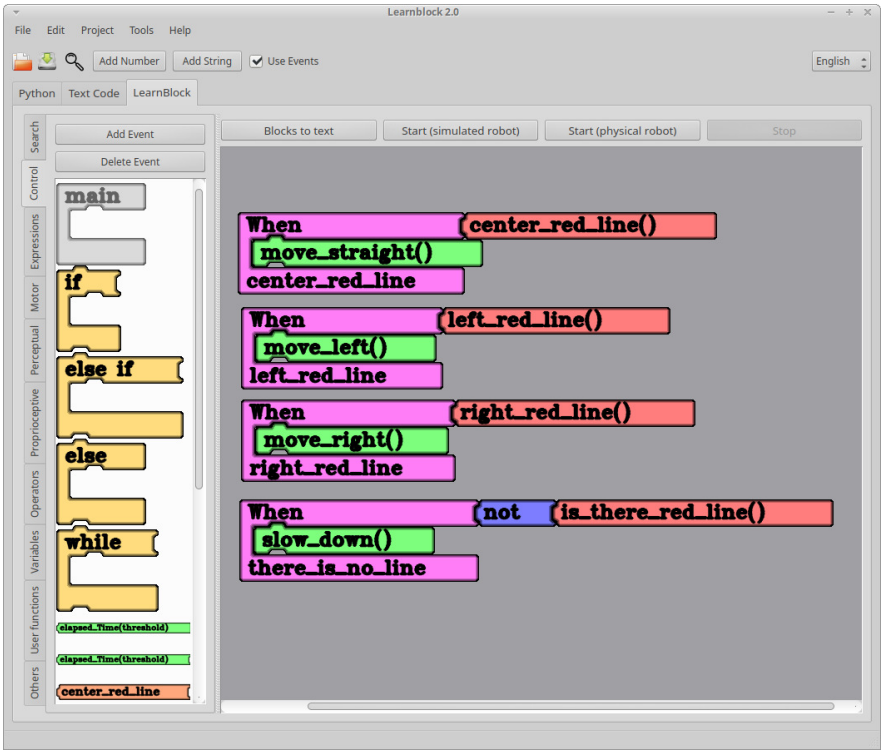


Figure 5.16: Implementation of the "follow red line" behavior using event-driven visual programming.

program is finally executed to make the robot behaves as specified by the initial visual code.

5.4. The simulator

At this point, the programming tool includes all the necessary elements to program the physical robot using one of the three available programming languages. Nevertheless, some adaptations are necessary to run the generated code in a simulated robot.

As previously mentioned in chapter 4, simulation support is obtained form RCIS, the RoboComp robotic simulator. This is the same simulator that was used in the LearnBot project.

To use RCIS in our project, a 3D model of EBO was created, as well as different simulated environments for the robot. Figures 5.17 and 5.18 show two examples of

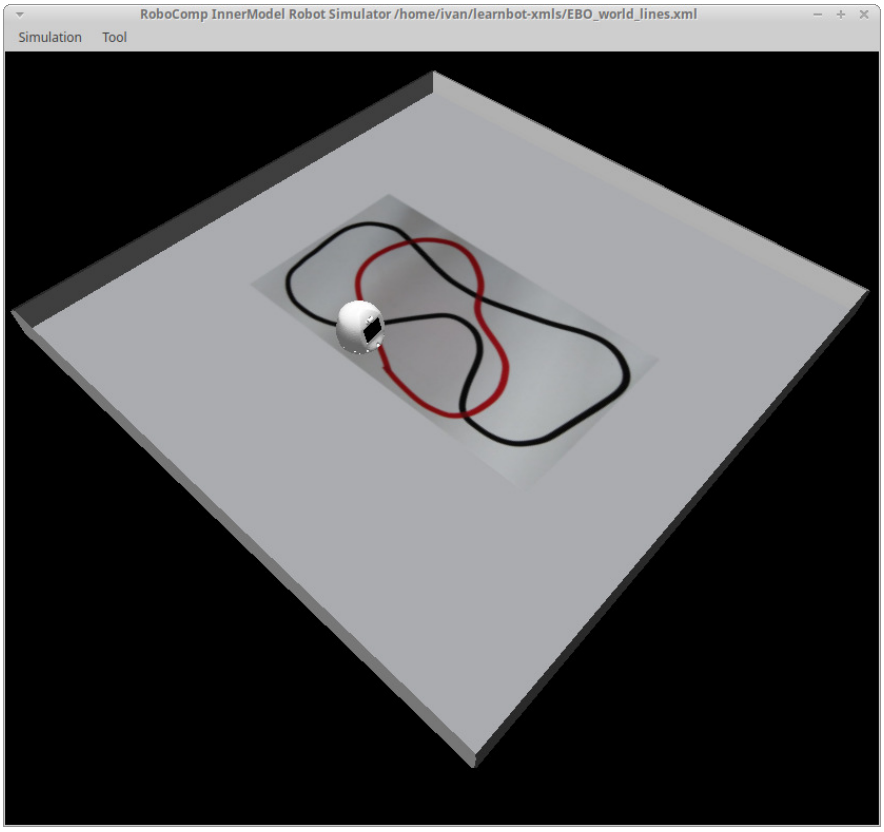


Figure 5.17: Simulated environment where the robot is in a limited surface with lines.

these simulated environments.

The simulated robot offers the same interfaces to access most of the low-level components than the physical robot. However, certain changes are required for communicating with some components. These changes only concern the implementation of certain functions of the intermediary class. This way no modification is required neither in the structure of this class nor in any other part of the programming tool. This implies that a code created by a user using our tool can be executed in both, the physical and the simulated robot, by exclusively including the intermediary class implemented for the specific robot. Next section details this important feature of our project.

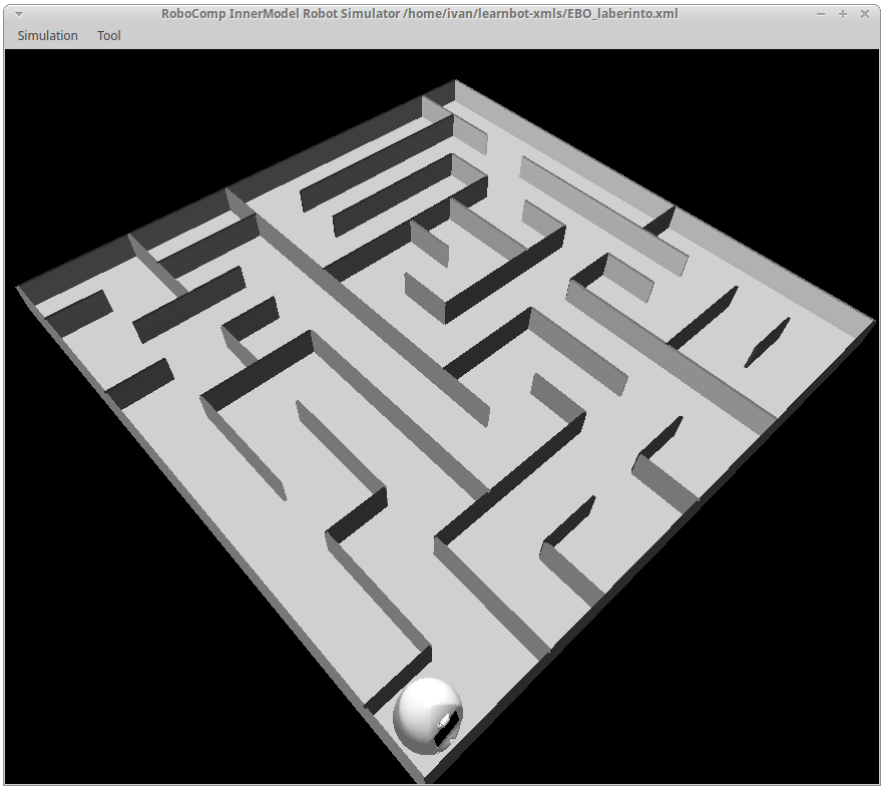


Figure 5.18: Simulated environment of a labyrinth.

5.5. Execution of the user code

When the user creates a code using the visual programming mode, this code is translated to EBOCode and, finally, to Python code. This code can be executed in the physical robot or in the simulated one using the corresponding intermediary class. Each intermediary class is in charge of communicating with the specific robot accessing to its low-level components.

Figure 5.19 shows a diagram of connections between the tool, the physical robot and the simulated robot. In this figure, the execution process of the code generated by the user can be observed. In addition, this scheme reflects how the tool can be easily extended to be used for other robots. Thus, the extension of our tool to other robotic platform only entails the adaptation of the intermediary class exposed in listing 5.1, maintaining the names of the different functions. With this simple step, the whole tool will work with the robot to which it has been adapted.

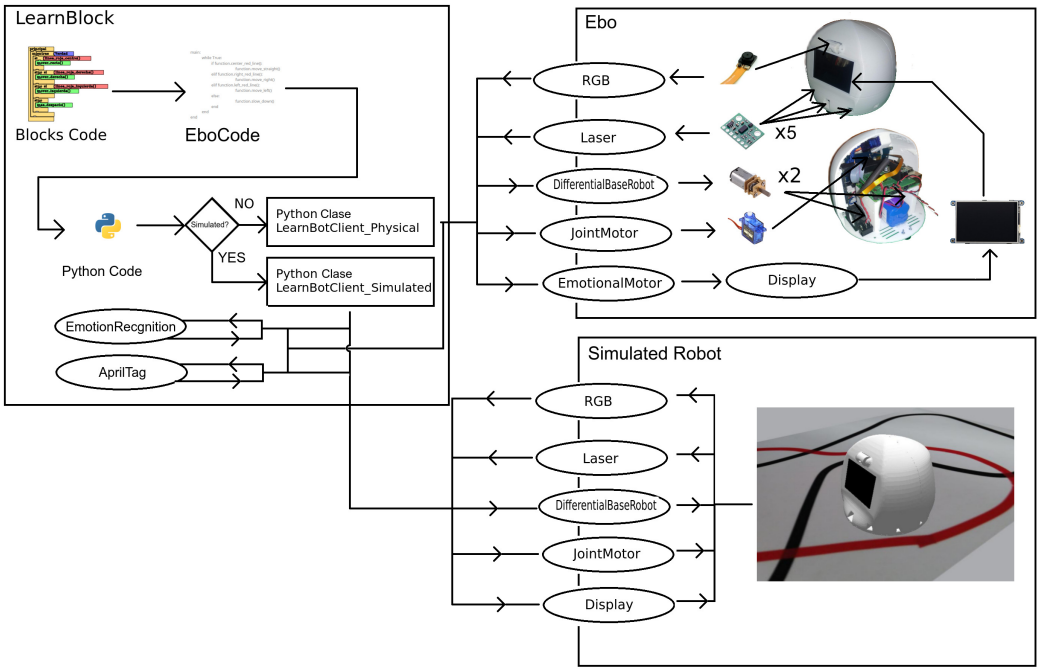


Figure 5.19: Diagram of connections between the tool, the physical robot and the simulated robot.

Chapter 6

Results

To obtain validation results of our project and verify that it meets all the proposed expectations, different workshops were organized. These activities provide us with feedback about the different features of both, the robot and the programming tool.

Two types of workshops were performed:

- Short workshops: workshops with a duration of about 15/20 minutes where some programming examples were exhibited. In this kind of activity the interaction between the child and the tools was limited.
- Programming introduction workshops: workshops with a duration of about 2/3 hours, where a group of children had to use the programming tool, autonomously solving different exercises and testing them in the simulator or in the robot.

In both types of workshops, child opinions were collected with a survey of 7 questions. The next two sections present the result of this survey for each type of activity.

6.1. Short workshops

Several short workshops were organized as part of the *"European Researchers' Night"* during the 2018 edition. Due to the short duration of each workshop,

the activity consisted in explaining different features of the project through several programming examples.

66 children participated in these workshops. The ages of these children ranged between 5 and 13 (figure 6.1), although 75% of them were from 7 to 9 years old, which are appropriate ages for our project.

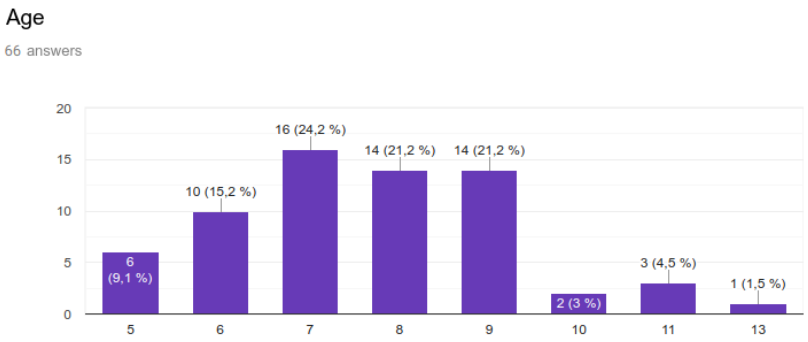


Figure 6.1: Ages of the children who participated in the short workshops.

Regarding the gender of the participants (figure 6.2), 63.6% were boys and 36.4% were girls. Figure 6.3 shows the gender distribution by ages.

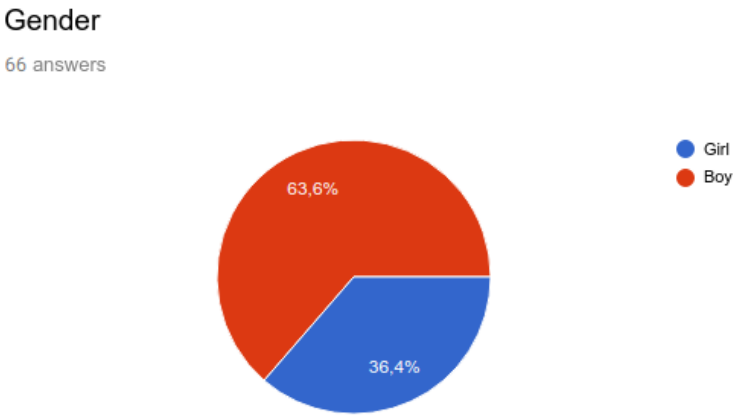


Figure 6.2: Gender of the participants of the short workshops.

The remaining questions of the survey were designed to obtain an evaluation of different features of our project. To quantify the responses to these questions, a

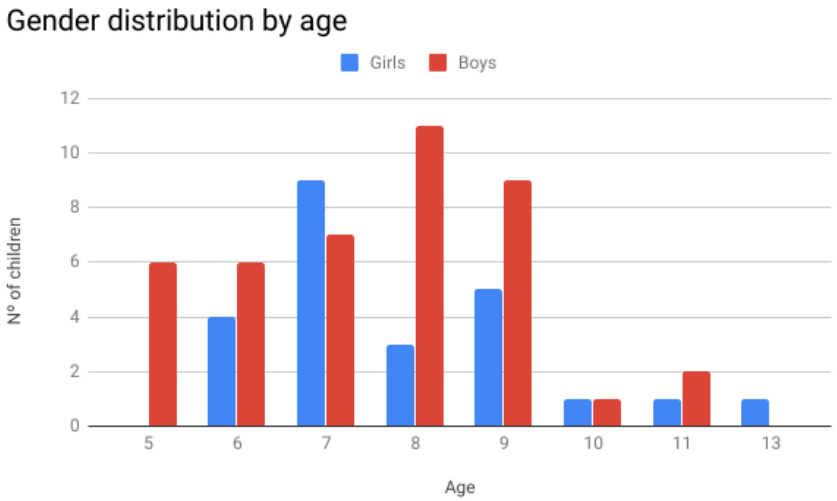


Figure 6.3: Gender distribution by age of the children who participated in the short workshops.

scale from 1 to 10 was used, where 1 represents the most negative opinion of the corresponding feature and 10 the most positive one.

Next, the different questions are presented and analyzed.

What do you think about the appearance of the robot?

Results of this question are shown in figure 6.4. As can be observed in this figure, all the children quantified the appearance of the robot with a value of 5 or above. 89.3% of the children graded the robot appearance with a value higher than 7. According to these results, we can conclude that the appearance of the robot is appropriated to achieve acceptance by children, although new improvements can be made.

The next question of the survey was:

Do you think the robot correctly expresses the different emotions? (joy, sadness, surprise, ...)

Figure 6.5 shows the evaluation of this feature of our robot. From these results, it can be stated that the representation of some emotional expressions of the robot should be redesigned to get a greater approval. It should be noted that the current version of

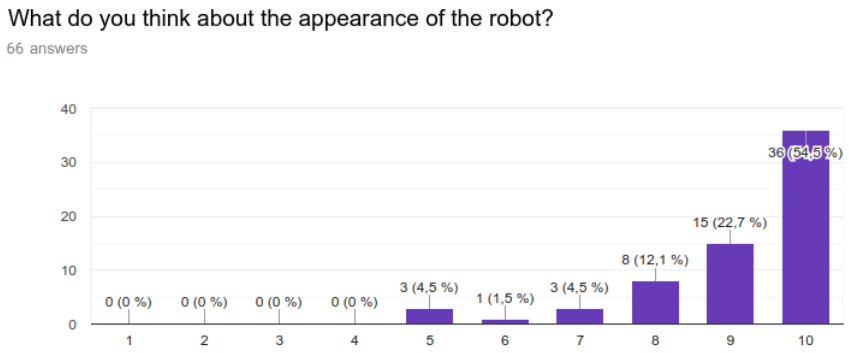


Figure 6.4: Evaluation of the appearance of the robot by the participants of the short workshops.

the robot includes new emotional expressions that where not available at the time of these workshops.

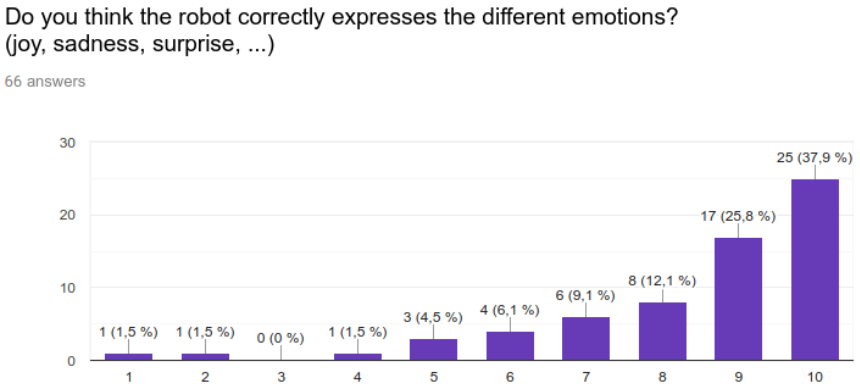


Figure 6.5: Evaluation of the representation of emotions in our robot by the participants of the short workshops.

The next question was related to the programming tool:

What is your opinion about the tool to program the robot (suitable interface, programming using blocks and connections among blocks, etc.)?)

Children opinion regarding this question is shown in figure 6.6. Only 64 children answered to this question. The main reason is that some children did not pay attention to the explanations about how to program the robot and they decided to not introduce false responses. As can be observed in figure 6.6, the majority of the participants

expressed a very positive opinion about the programming tool. This means that our programming tool meets all the requirements to be an appealing programming tool for children.

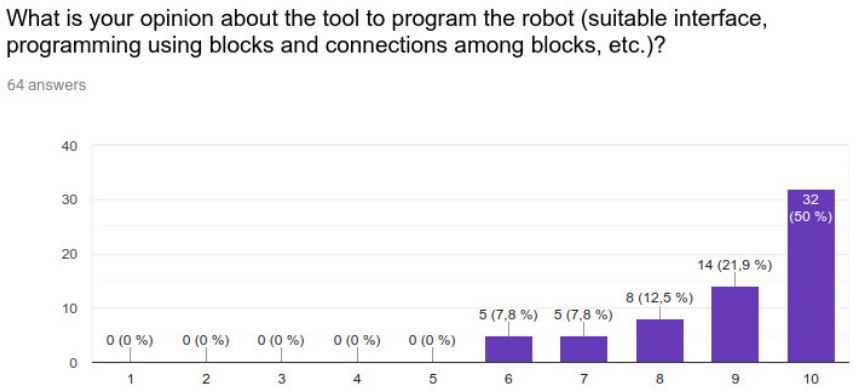


Figure 6.6: Evaluation of general features of the programming tool by the participants of the short workshops.

The next question is also related to the programming tool, but in this case the question is whether the tool is easy to use or not:

What is your opinion about the easiness of use of the programming tool?)

64 answers were obtained. As shown in figure 6.7, 65.7% of the participants evaluated the tool for programming with a value higher than 7, although 15.7% of the children believed that the programming utilities of the tool were rather complex. This result is in accordance with our expectations, since the duration of this kind of workshop is insufficient to appropriately introduce the tool for programming.

The last question and the most generic is:

Would you like to have a robot like EBO and learn to program it?

The results of this question are shown in figure 6.8. From these results, it can be stated that the project had a great acceptance by the children. Thus, 90.9% of the children evaluated the project with a value of 10, which shows that a great majority of the children were very interested in using our robot and programming tool beyond this activity.

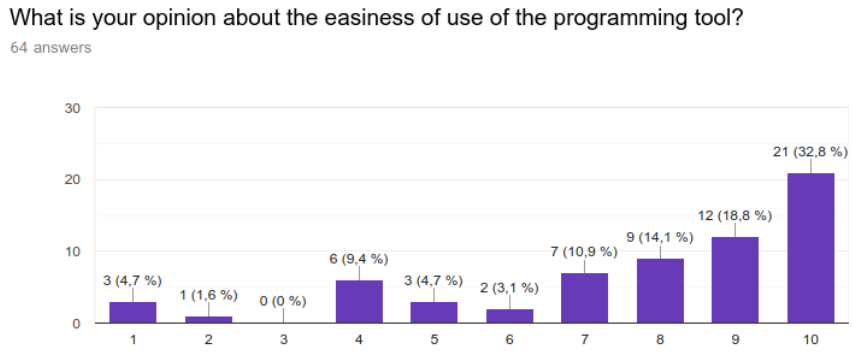


Figure 6.7: Opinion about the complexity of the tool for programming (short workshops).

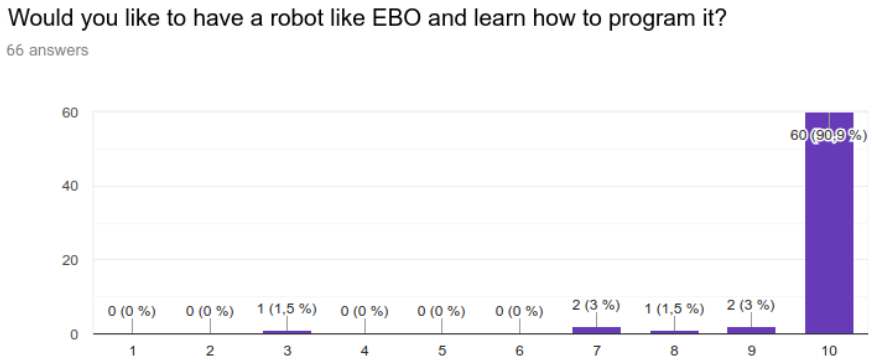


Figure 6.8: Evaluation about the interest of the whole project by the participants of the short workshops.

6.2. Programming introduction workshops

In this kind of workshop, children received a brief introduction on how to use the tool for programming the robot. Then, they implemented some examples of simple robot behaviors. The duration of this activity was between 2 and 3 hours. The total number of participants was 18.

We used the same survey as in the short workshops to collect the opinion of the children about our project. Although there were fewer participants than in the previous type of workshop, the results of these surveys have a greater relevance, since the children had a greater interaction with the robot, the programming tool and the simulator.

The profile (age and gender) of the children who participated in this kind of activity

is shown in figures 6.9, 6.10 and 6.11. As shown in these figures, the mean age of the participants is in line with the age for which the project is designed. Regarding the gender of the participants, the proportion of girls over the total of children was similar to the one observed in the short workshops, with the distribution by age of figure 6.11.

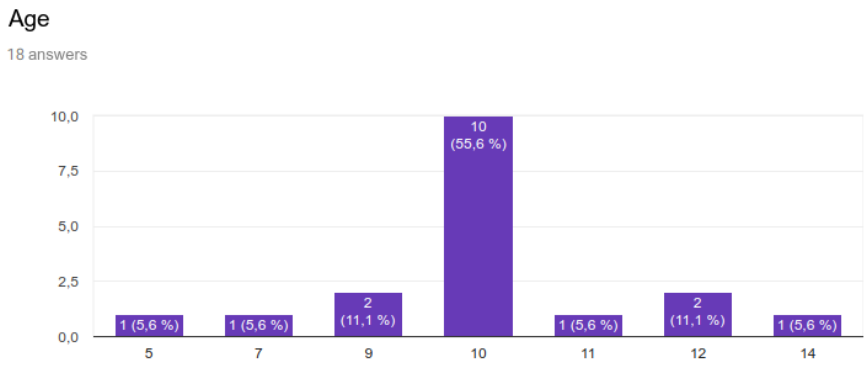


Figure 6.9: Ages of the children who participated in the programming introduction workshops.

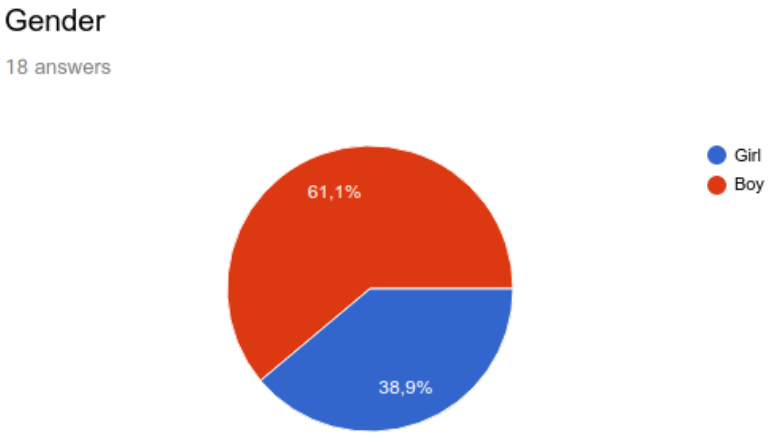


Figure 6.10: Gender of the participants of the programming introduction workshops.

The evaluation of the different features of our project by the participants of programming introduction workshops is depicted in figures from 6.12 to 6.16.

In relation to the robot appearance (figure 6.12), it is evaluated with a very positive opinion by a significant percentage of the participants. In addition, the great majority of the children consider that the emotions of the robot are correctly expressed (figure

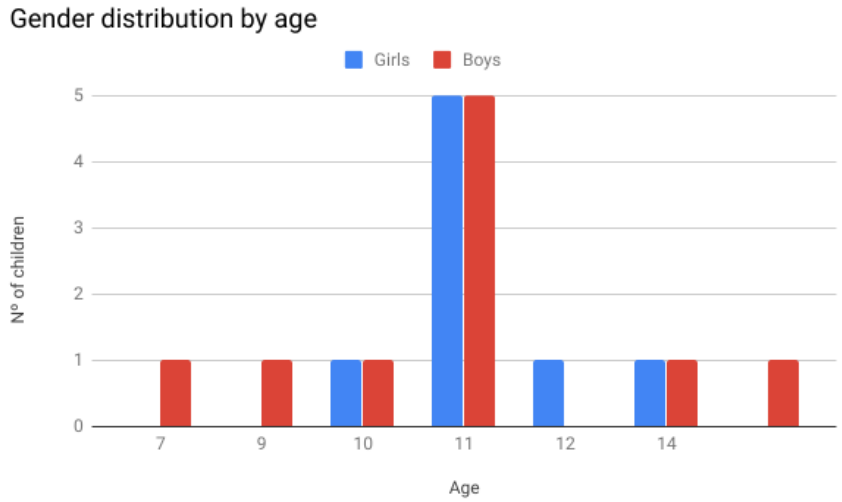


Figure 6.11: Gender distribution by age of the children who participated in the programming introduction workshops.

6.13). According to this result and the evaluation obtained to the same feature by the participants of the short workshops, we decided to keep the representation of every emotion, but we included a smooth transition between expressions when the robot changes its emotional state, to improve its expressiveness.

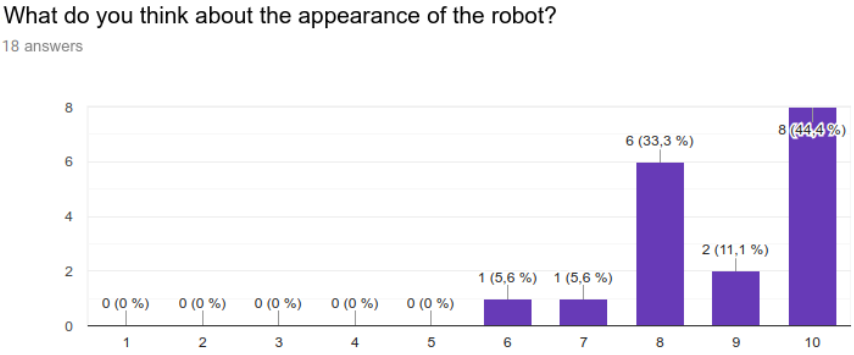


Figure 6.12: Evaluation of the appearance of the robot by the participants of the programming introduction workshops.

With regard to the programming tool, the rating of its general features (figure 6.14) is comparable to the evaluation of the participants of the short workshops. However, the children opinion about the complexity of the tool for programming is much more positive (figure 6.15). This indicates that, once the child has interacted with the tool,

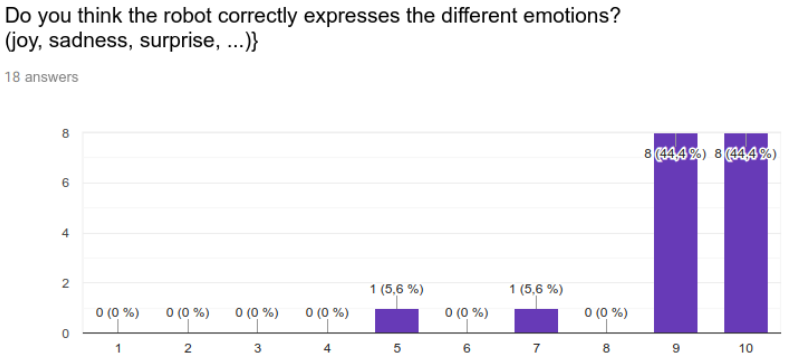


Figure 6.13: Evaluation of the representation of emotions in our robot by the participants of programming introduction workshops.

the complexity of programming the robot using the visual mode is moderate.

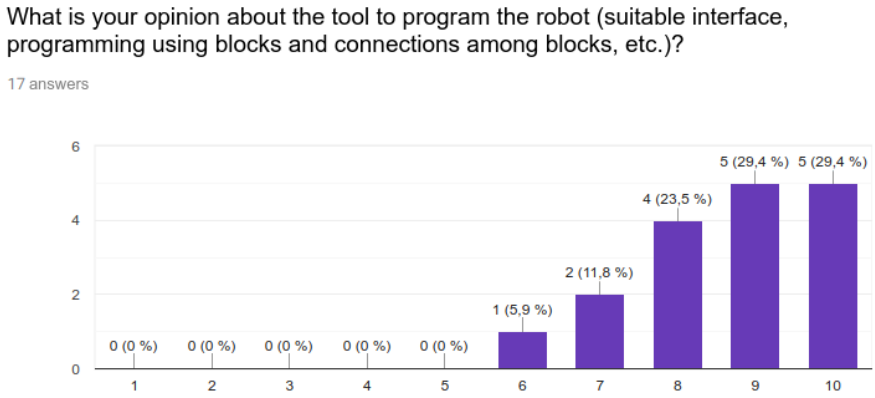


Figure 6.14: Evaluation of general features of the programming tool by the participants of the programming introduction workshops.

Finally, the overall evaluation of the project is shown in figure 6.16. As can be seen, 100% of the children have a great interest in continuing using the project to learn programming.

6.3. Comparison with other robotic educational projects

To finish this chapter, a comparison of our project with the educational projects presented in chapter 3 is included. With this purpose, the comparative tables of chapter

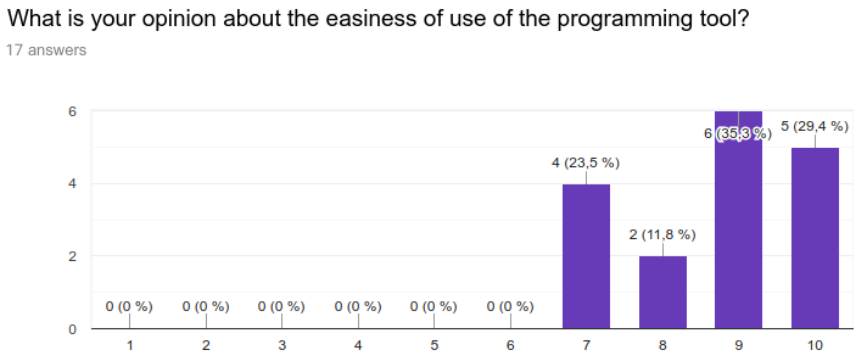


Figure 6.15: Opinion about the complexity of the tool for programming (programming introduction workshops).

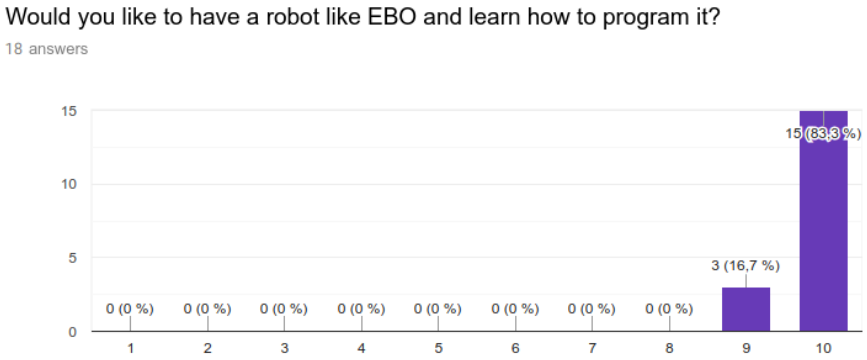


Figure 6.16: Evaluation about the interest of the whole project by the participants of the programming introduction workshops.

3 have been replicated, adding a new column to represent the features of EBO and LearnBlock.

Tables 6.1 and 6.2 present a comparative list of hardware components of the different educational robots, including EBO.

Table 6.3 shows a comparison of functionalities between EBO and the other studied robots. As previously mentioned, EBO is not designed to be a building kit, thus this is the unique feature that is not satisfied.

Table 6.4 shows that EBO meets many of the selected features. Discarding the *Vector* robot, since it can not be programmed with a visual language, EBO is the robot that satisfies the majority of the features. Moreover, 3 of the 4 characteristics that are not meet, can be satisfied in the near future, because the required hardware components

Table 6.1: Table with a list of hardware components of WeDo 2.0, EV3, Mbot, Zowi, Colby and EBO

	WeDo 2.0	EV3	Mbot	Zowi	Colby	EBO
Processor	?	ARM9 300MHz	ATmega 2560	ATmega 328		ARM Cortex-A53 1.4GHz
Cores	?	1				4
Prize	173€	438,99€	121.75€	79,90€	59,99$	500€ (per unit)
Microphone			V	V		
Distance sensor		Ultrasound	Ultrasound	Ultrasound		5 laser (Max 2m)
Touch sensor						All Screen (Capacitive)
Cliff sensor		*Can use Ultrasound				
IMU	V	V				
Screen resolution		178×128	Matrix Led	LED Matrix (5 x 6 px)		480x320
Screen Colors		Gray				Full color
Screen Size						3.5”
Camera resolution						1080p30
Battery duration	?	3h	2-3h	8 h	?	3-5h
Lights	1		1		V	
Motors	Normal	Normal	Normal	Normal	Normal	Normal
Speaker					*Speaker or Buzzer	
Buzzer			V	V	*Speaker or Buzzer	
IR Reciver/ Trasmiting			V			
Line sensor			V			
Parts	280	541	+-50	+-30	?	63
Connections	Bluetooth/ Wifi	USB/ Wifi/ Bluetooth	Bluetooth/ Wifi	Bluetooth		Wifi
Color sensor		V	V			
Termometer						

Table 6.2: Table with a list of hardware components of Joy robot, Thymio, Codey rocky, Cozmo, Vector and EBO

	Joy robot	Thymio	Codey rocky	Cozmo	Vector	EBO
Processor	ATmega 2560	?	ESP32	ARM Cortex 4 100 MHz	Qualcomm Snapdragon 1.2GHz	ARM Cortex-A53 1.4GHz
Cores		?		1	4	4
Prize	?	109.90€	99$	180$	250$	500€ (per unit)
Microphone		1	V		4	
Distance sensor		9 InfraRed	1		1 laser (Max 1m)	5 laser (Max 2m)
Touch sensor	*Smartphone	5 capacitive buttons			On the top of the robot, (Capacitive)	All Screen (Capacitive)
Cliff sensor			* the same as the distance.	V	V	
IMU		V	V	V	V	
Screen resolution	LED Matrix (16 x 16 px)		LED matrix	128x64	184x96	480x320
Screen Colors	Red		Blue	Blue	Full color	Full color
Screen Size	*Smartphone			1.8”	1.8”	3.5”
Camera resolution	*Smartphone			320x240	720p	1080p30
Battery duration	?	3-5 h	2h	45-60min	45-60min	3-5h
Lights		39LEDS	V	4	5	
Motors	Normal	Normal	Normal	Normal	Noiseless	Normal
Speaker		V	V	V	V	
Buzzer						
IR Reciver/ Trasmiting		1	V			
Line sensor						
Parts	+-130	?	?	700	370	63
Connections	Bluetooth	Wifi	Wi-Fi/ Bluetooth/ USB	Wifi	Wifi	Wifi
Color sensor			V			
Termometer		1				

are already available.

Finally, table 6.5 shows that LearnBlock meets all the selected features, excepting three of them. The first feature that is not provided by LearnBlock is *"Functions created with blocks can return"*. The reason is that we initially considered that it would not be necessary because returning functions can be implemented from code using the tool itself. Nevertheless, this feature will be included in the next version of LearnBlock. The second feature that is not satisfied in our programming tool is *"Can be used from a web browser"*, since LearnBlock requires a connection with the physical or the simulated robot for program execution. Scratch and Blockly are more general tools, not specifically developed for programming robots. Thus, the use of these tools from a web browser makes them more accessible for general educational projects related to programming. Finally, the last unsatisfied feature, *"Has a community of shared code"*, is not applicable to our project since its availability is too recent.

Regarding the other features, LearnBlock can extend its blocks creating new blocks from code using the tool itself. In addition, functions created with blocks can be grouped forming a library-like set of new blocks that can be imported into any project. Last and most importantly, LearnBlock communicates with the robot through a special software class that acts as a hardware abstraction layer. This implies that any robotic platform can be programmed using our tool by simply replacing the EBO HAL with a specific software layer that gives access to the hardware components of the new robot. No additional change is required. Thus, any high-level function already implemented can be used in new projects independently of the robot being used.

Table 6.3: General features of the analyzed projects and EBO

	WeDo 2.0	**EV3**	**Mbot**	**Zowi**	**Colby**
Building kit	X	X	X		
Visual programming tool	X	X	X	X	
Integration of general elements of programming languages	X	X	X	X	
Simulator support					
Open-source project			X		
Open-hardware project			X		

	Joy robot	**Thymio**	**Codey rocky**	**Cozmo**	**Vector**	**EBO**
Building kit						
Visual programming tool		X	X	X	X	X
Integration of general elements of programming languages		X	X	X	X	X
Simulator support						X
Open-source project	X	X	X			X
Open-hardware project	X	X				X

Table 6.4: General functionalities of the analyzed projects and EBO

	WeDo 2.0	EV3	Mbot	Zowi	Colby
Follow black line	X	X	X		
Follow color line					
Emotional expressions		X		X	
Emotion recognition					
Tag recognition					
Obstacle detection	X	X	X		X
Base motion	X	X	X	X	X
Face detection					
Touch detection				X	
Slant detection	X	X			
Sound detection		X			
Sound making		X	X	X	X

	Joy robot	Thymio	Codey rocky	Cozmo	Vector	EBO
Follow black line	X	X	X			X
Follow color line						X
Emotional expressions	X		X	X	X	X
Emotion recognition				X		X
Tag recognition				X	X	X
Obstacle detection		X	X		X	X
Base motion	X	X	X	X	X	X
Face detection				X	X	X
Touch detection		X			X	
Slant detection		X	X	X	X	
Sound detection		X	X		X	
Sound making		X	X	X	X	

Table 6.5: General features of the analyzed programming tools and LearnBlock

	Scratch	**Blockly**	**LearnBlock**
Create functions with blocks.	X	X	X
Functions created with blocks can return a value		X	
Functions created with blocks can be exported to used in other programs			X
Create blocks from code	X	X	X
Facilitates the creation of blocks from code			X
Multilingual options	X	X	X
Can be used from web navigator	X	X	
Can be downloaded in the PC	X	X	X
Has a community of shared code	X		
Consistency of variables	X	X	X
Can adapted to other robots	X	X	X
High-level functions already implemented can be used in adapted projects			X
Includes a hardware abstraction layer			X
The tool directly translates from block code to other programming languages		X	X

Chapter 7

Conclusions and future works

This chapter exposes the main conclusions of our work. Beside these conclusions, improvement proposals are also described for both, the EBO robot and the programming tool LearnBlock.

Starting with EBO, it meets all the objectives that were considered at the beginning of the project and, in addition, it has had a great acceptance by the children. In general, according to its features and components, EBO is comparable to many other educational robots, although it can be improved in several ways:

- Add audio hardware so that the robot can play sounds or even talk with the help of a text-to-speech converter.
- Add accelerometer and oscilloscope to control odometry and detect sudden movements.
- Take advantage of the touch screen to extend the human-robot interaction possibilities.
- Include a laser sensor under the base of the robot to detect edges of tables or steps.
- Reduce production costs
- Reduce the computational load

- Add interaction with other robots, to extend the project to collaborative robotics.

Regarding the programming tool, LearnBlock also meets all the objectives outlined during the design phase of the project. Additionally, LearnBlock includes interesting features that are not provided by any other educational programming tool. Specifically, LearnBlock can be used with any robot with little programming effort. This makes it a very useful tool. Nevertheless, new extensions can be added to improve even more our tool:

- Include new functions that provide additional functionality related to the new hardware components of the robot.
- Add more restrictions to the connections of the blocks to eliminate potential syntactic errors.
- Add collaborative robotics extensions, so that different clients that control the robots can communicate with each other and know what the others robots are doing.
- Add options for automatic updates, when a new version of the programming tool is available.

Considering the project as a whole, it can be stated that the aim of creating a new educational tool to work up different skills in children has been largely achieved. There are few educational tools as complete as ours. Moreover, new improvements will lead to a very competitive tool. “Imagination is the only limit” (Charles Kettering).

Annexes

Appendix A

Installation

The requisites to install our tool in a computer are the following:

- Linux Operative System.
- Version 18.04 or higher.

Once these requirements are met, the following steps should be performed:

1. Install the **Robocomp** framework. The steps to install this framework are described in the following link: `https://github.com/robocomp/robocomp`
2. Install **Python**. To install Python, the following versions can be chosen: 2.7.* or higher, 3.5.* or higher, lower than 4. As an example, to install Python 3.7 with the *apt* command, the following line has to be executed:

   sudo apt install python3.7

3. Install **pip**. The version of pip has to be the same than the one chosen for Python:

   sudo apt install python3-pip

4. Install LearnBlock. It can be installed from the repository `https://pypi.org/project/learnbot-dsl/`. Depending on the installed Python version, pip or pip3 must be used: **Python-2.***

sudo pip2 install learnbot-dsl

Python-3.*

sudo pip3 install learnbot-dsl

Once the installation process is finished, the LearnBlock tool is already available for execution.

All the additional information of the project can be found in the LearnBot github repository:

`https://github.com/robocomp/learnbot/`

Bibliography

[1] Piaget - desarrollo cognitivo del niño. https://desarrollocognitivoinfantil.blogspot.com/2016/04/piaget-desarrollo-cognitivo-del-nino.html Accedido a 06-01-2019.

[2] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[3] Lego Education. Wedo 2.0, 2017.

[4] Liberios Vokorokos, Juraj Mihalov, and Eva Chovancová. Potential of lego© ev3 mobile robots. *Acta Electrotechnica et Informatica*, 15(2):31–34, 2015.

[5] Makeblock-mbot. https://makeblock.es/productos/robot_educativo_mbot/ Accedido a 06-01-2019.

[6] Bq-zowi. urlhttps://www.bq.com/es/zowi Accedido a 06-01-2019.

[7] Learnigresources-colby. https://www.learningresources.com/product/learning+essentials--8482-+stem+robot+mouse+coding+activity+set.do?sortby=ourPicks&refType=&from=Search&ecList=6&ecCategory= Accedido a 06-01-2019.

[8] Makeblock-codey roky. https://www.makeblock.com/steam-kits/codey-rocky Accedido a 06-01-2019.

[9] Joy robot (robô da alegria). https://www.instructables.com/id/Joy-Robot-Rob\%C3\%B4-Da-Alegria-Open-Source-3D-Printed-A/ Accedido a 06-01-2019.

[10] Fanny Riedo, Morgane Chevalier, Stéphane Magnenat, and Francesco Mondada. Thymio ii, a robot that grows wiser with children. In *2013 IEEE workshop on advanced robotics and its social impacts (ARSO)*, pages 187–193. Eidgenössische Technische Hochschule Zürich, Autonomous System Lab, 2013.

[11] Scracth. https://scratch.mit.edu/ Accedido a 06-01-2019.

[12] Neil Fraser. Google blockly-a visual programming editor. https://developers.google.com/blockly/ Accedido a 06-01-2019.

[13] JM Haut, ME Paoletti1 P Bustos, and N Garcıa. Code2bot, a social robot for the classroom. In *Conference of the Spanish Association for Artificial Intelligence, CAEPIA*, volume 15, 2015.

[14] Luis Manso, Pilar Bachiller, Pablo Bustos, Pedro Núnez, Ramón Cintas, and Luis Calderita. Robocomp: a tool-based robotics framework. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010.

[15] Michi Henning and Mark Spruiell. Distributed programming with ice. *ZeroC Inc. Revision*, 3:97, 2003.

[16] Marco Antonio Gutierrez Giraldo, Adrián Romero Garcés, Pablo Bustos García de Castro, and Jesús Martínez Cruz. Progress in robocomp. 2013.

[17] Roger A Light. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13), 2017.