Universidade do Minho
Escola de Engenharia
Departamento de Informática

Gustavo Linhares Galvão

Robi: A Visual Programming Language for Educational Robotics

April 2022

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Gustavo Linhares Galvão

Robi: A Visual Programming Language for Educational Robotics

Master dissertation
Master Degree in Informatics Engineering

Orientador / Supervisor
Pedro Rangel Henriques /
Cristiana Araújo

April 2022

STATEMENT OF INTEGRITY

Gustavo Linhares Galvão

____

ABSTRACT

This document presents a Master's thesis with researches focused on the teaching of computational thinking and present the development details of Robi, a block-based visual programming language that is able to program a robot built with an Arduino Uno. These researches had the purpose of evaluating if the development of Robi, a block-based programming language that communicates with Arduino, would really be needed. The researches have proved that from the popular programming environments that exist in the market, that were investigated, none have the requirements that Robi requires. The platform will be used to teach computational think through a block-based programming environment and educational robotics. Robi development is motivated by the intersection between the costs of educational robotics kits and the existing block-based programming language, in which simplicity and intuitiveness could be improved, so children with learning difficulties or even younger children, in the context of educational robotics, can leverage the learning benefits that the Robi environment can bring. The educational robotics kit used with the block-based programming environment developed, is the one based on Arduino Uno, a microcontroller board that, together with electronic components, can be considered cheaper than some of the famous educational robotics kits. The main goal of this project is to provide a simpler and more intuitive visual programming language platform to program a robot based on Arduino Uno.

Keywords: Computational Thinking; Visual programming language; Block-based programming language; Educational robotics

RESUMO

Este documento apresenta uma tese de Mestrado com investigações voltadas ao ensino do pensamento computacional e apresenta os detalhes do desenvolvimento de Robi, uma linguagem de programação visual baseada em blocos, que é possível programar um robô construído com um Arduino Uno. Essas investigações tiveram o objetivo de avaliar se o desenvolvimento de Robi, uma linguagem de programação baseada em blocos que se comunica com o Arduino, seria realmente necessário. As investigações comprovaram que dos ambientes de programação populares existentes no mercado, que foram investigados, nenhum possui os requisitos que Robi exige. A plataforma será utilizada para ensinar pensamento computacional por meio de um ambiente de programação baseado em blocos e robótica educacional. O desenvolvimento de Robi é motivado pela combinação entre os custos dos kits de robótica educacional existentes no mercado e linguagens de programação baseada em blocos existentes, em que simplicidade e intuitividade poderiam ser aprimoradas, para assim, crianças com dificuldades de aprendizagem ou até crianças mais novas, no contexto da robótica educacional, poderiam fazer proveito dos benefícios da aprendizagem que o ambiente Robi pode trazer. O kit de robótica educacional utilizado com o ambiente de programação baseado em blocos desenvolvido é um kit com o Arduino Uno, uma placa de microcontrolador que, junto com componentes eletrônicos, pode ser considerada mais barata que alguns dos famosos kits de robótica educacional. O objetivo principal deste projeto é fornecer uma plataforma de linguagem de programação visual mais simples e intuitiva para programar um robô baseado em Arduino Uno.

Palavras-Chave: Pensamento computacional; Linguagem de programação visual; Linguagem de programação baseada em blocos; Robótica educacional

CONTENTS

1 INTRODUCTION 1
- 1.1 Objectives 2
- 1.2 Research Approach 2
- 1.3 Research Hypothesis 3
- 1.4 Document Structure 3

2 STATE OF ART 4
- 2.1 Computational Thinking 5
- 2.2 Block-based programming language 6
  - 2.2.1 Scratch 6
  - 2.2.2 Code.org 7
  - 2.2.3 Alice 8
  - 2.2.4 Kodu Game Lab 8
- 2.3 Educational Robotics 8
  - 2.3.1 LEGO Mindstorms EV3 9
  - 2.3.2 mBot 9
  - 2.3.3 Robomind 10
- 2.4 Summary 11

3 PROPOSED APPROACH 12
- 3.1 System Architecture 12
- 3.2 Compiler 14
- 3.3 System Requirements 15
- 3.4 Technological decision 16
- 3.5 System architecture revised 17

4 LANGUAGE 20
- 4.1 Grammar 20
  - 4.1.1 Example 23
- 4.2 Blocks 25
  - 4.2.1 Movement Category 25
  - 4.2.2 Sensor Category 26
  - 4.2.3 Flux Category 26
  - 4.2.4 Operators Category 28
  - 4.2.5 Variable 30

5 ROBOT 31

5.1 Components 31
5.2 Arduino Programming code 34
5.3 Code Upload Process 35
6 ROBI PROGRAMMING SYSTEM 36
6.1 Visual Language Interface 36
6.1.1 Blocks Connections 37
6.1.2 Blocks Movements 40
6.1.3 Operator Blocks and Expressions 40
6.2 Code setup 41
6.3 Translator 45
7 COMPARISON BETWEEN ROBI AND SCRATCH 49
8 TESTING ROBI 52
8.1 Basic Exercise - Moving Robi robot 52
8.2 Intermediate Exercise - Looping through the robot movement 53
8.3 Rectangular Movement Exercise 55
8.4 Result 58
9 CONCLUSION 60
9.1 Future work 61
9.1.1 Bluetooth Integration 61
9.1.2 Improvements on Robi system 61
9.1.3 Class management 62
A CODE SENT TO THE ROBOT 63

LIST OF FIGURES

Figure 1	System architecture	13
Figure 2	Domain Model: Entity-Relationship Diagram	14
Figure 3	Compiler architecture	15
Figure 4	Updated system architecture	19
Figure 5	An example of a visual program created in the Web Interface	25
Figure 6	Robot components	32
Figure 7	Circuit diagram	33
Figure 8	Robi robot	34
Figure 9	LED blinking	35
Figure 10	Robi programming environment	37
Figure 11	Connection between blocks	39
Figure 12	Connection between aggregating blocks	39
Figure 13	Plus operation from a math operator block	40
Figure 14	Nested math operator blocks	41
Figure 15	Nested conditional operator blocks	41
Figure 16	Square exercise implemented in Robi language	49
Figure 17	Square exercise implemented in Scratch language	49
Figure 18	Categories existing in Robi platform	50
Figure 19	Categories existing in Scratch platform	50
Figure 20	Scratch programming environment	51
Figure 21	First exercise solution	52
Figure 22	Second exercise solution	54
Figure 23	Third exercise solution	55
Figure 24	Exercise 3 with variables	57

1

INTRODUCTION

This thesis has the purpose of developing a new programming language, aiming to be the most intuitive and simple as possible in a way that makes it is possible to improve computational robotics skills, so that younger children or even children with a learning difficulties can leverage. Children with special educational needs are an example of students with greater learning difficulties. Although children with learning disabilities, if taught in conventional ways, may have difficulty reading, writing, spelling, reasoning, recalling and/or organize information, they are as smart as than their peers. Having the right support and intervention, these children can improve their academic grades, succeed in school and even achieve successful, often distinguished careers later in life (Ismail et al., 2009). In order to support and improve the learning of students by making use of the benefits that Computational Thinking brings, it is intended in the context of the project under which this Master's works appears, resort to the development of a block-based programming language. According to Weintrop and Wilensky (2017), Block-based programming environments leverage a programming primitive-as-puzzle-piece metaphor that provides visual cues to the user about how and where commands can be used as their means of constraining program composition. This type of visual programming language consists of dragging blocks into a scripting area, and fitting them together, to form scripts (Weintrop and Wilensky, 2017). Although the aim of this thesis is not directly associated to children with special educational needs, but since they have learning difficulties, and the purpose of this thesis is the develop a simpler and more intuitive visual programming language to support the teaching of Computational Thinking through educational robotics.

As Virnes et al. (2008) stated, educational robotics has the potential for improving special needs education and for eliminating barriers to learning if it can be focused on the special needs of the children. Motivated by the benefits that robotics tools brings, in which made it possible for the students to practice and learn many necessary skills, like collaboration, cognitive skills, self-confidence, perception, and spatial understanding (Karna-Lin et al., 2006), we considered the integration of the block-based programming language with an Arduino Uno, an open source microcontroller that can be easily programmed and reprogrammed at any instant of time. As reported by Louis (2016), the Arduino platform was introduced in 2005, and was designed to provide a low cost and easy way for hobbyists, students and professionals to create devices that interact with their environment using sensors and actuators. One of the reasons to the Arduino be a low cost product, is due to the independence of suppliers of parts and components. This contrasts with robotics kits such as Lego, Fischer and other manufacturers, which have their own standard and proprietary components, making them more expensive (Junior et al., 2013).

Some examples of block-based programming environments that are very used to introduce programming are the following: Scratch, Alice, Snap!, App Inventor and Blockly. They provide a fun and engaging introduction to programming concepts without the need to deal with syntax that has been a historically obstacle to students that recently started learning text-based programming (Grover and Basu, 2017). However, despite the fun and engaging block-based programming environments, they either didn't have integration with robot kits, or when they had, either the robot kit was expensive, or they lacked a level of simplicity and intuitiveness that could benefit younger children. Not having integration with a robot makes them less flexible as to the use of fundamental features that could support the learning process of those students. For that reason, it was decided to develop Robi, a visual programming language for educational robotics dedicated on having a higher level simplicity and intuitiveness, using a cheaper solution when it comes to the robot.

1.1 OBJECTIVES

The development of this project have the following objectives:

- Design a programming language;
- Develop a block-based programming environment as a platform that could be used to teach programming to children;
- Generate code compatible with Arduino programming language;
- Use Arduino compiler to make the code developed in the environment to control the Robi robot.

1.2 RESEARCH APPROACH

To accomplish this Master's work, a methodology based on literature revision, solution proposal and implementation will be followed, going through an iterating over the next steps:

- Bibliographic study to understand the state of the art in the areas of educational robotics, educational programming in a block-based environment;

- Definition of the features and design the block-based programming language.
- Definition and design of the block-based programming language.
- Integration of the visual programming environment developed with the Arduino-based robot.
- Testing, results evaluation and discussion.

1.3 RESEARCH HYPOTHESIS

Using a very easy and iconic Block Programming Language to control Robots, it is possible help children on the training of Computational Thinking, improving their skills to solve problems.

1.4 DOCUMENT STRUCTURE

This document is organized into nine chapters. In Chapter 2 is presented the State of the Art of this Master thesis, in which contains researches about computational thinking and its interventions and what are in the market, as well as researches about the relation between computational thinking and students with learning difficulties. In Chapter 3 is presented the overall architecture, system requirements and the technological decisions of the system that will be implemented. In Chapter 4 is presented the grammar and the blocks that is part of the visual programming language developed. In Chapter 8 is presented the components used to built the robot and how it works. In Chapter 6 is presented Robi interface and the translation process developed. In Chapter 7 is presented a comparison between Robi and Sratch platforms. In Chapter 8 is presented three exercises that were given to children, and the correspondent solutions. Finally, in Chapter 9 is presented the conclusion of this Master Thesis.

2

STATE OF ART

In this chapter will be described relevant topics to the development of this Master Thesis, that involves researches regarding special educational needs. The main purpose of that research is to evaluate that computer-based interventions can be a factor that benefits these students with special educational needs. Other relevant topics that will be discussed in this chapter, are computational thinking and its impact; block-based programming languages; and educational robotics. A study on the popular block-based programming environments and educational robotics kits that exist on the market will also be carried out, in order to assess whether the development of the visual programming language Robi, is really necessary.

There are children with special educational needs as Hyperactivity and Autism that can eventually leverage the learning of this approach.

Children who exhibit developmentally inappropriate levels of inattention and/or hyperactivity-impulsivity are recognized as children with Attention-Deficit Hyperactivity Disorder (ADHD). These children are known to experience not only behavioral and social problems frequently, but also academic difficulties (Jitendra et al., 2008).

As stated by Jitendra et al. (2008), in order to improve the academic achievement of students with ADHD, it's important to have interventions that directly focus on academic skill deficits in content areas such as reading and mathematics. Jitendra et al. (2008) also mentioned that several models of peer tutoring have been investigated for use with these students, and most of these models include characteristics known to be effective with this population. Some of these models consists of:

- working one-to-one with another individual;
- the learner determining instructional pace;
- continuous prompting of academic responses;
- providing frequent, immediate feedback about quality of performance.

According to Román-González et al. (2018), teachers reported unexpected brilliant performance and behavior of students usually disruptive and inattentive, when faced with computer programming experiences. To put in another way, it's possible to say that students with hyperactivity, disruptive conduct or inattentiveness, for example, seem to respond especially well to programming tasks.

Autism Spectrum Disorder (ASD) is a neurodevelopmental condition mainly characterized by a deficit in social interaction as well as in creative behavior (Munoz et al., 2018).

According to Elshahawy et al. (2020), computer-based interventions for children with ASD have proven to be useful and effective due to the following reasons:

- Most technologies use visual interfaces, which goes along with the characteristics and preferences of individuals with ASD;
- Considering that social interactions might be overwhelming for children with ASD, they are more drawn to computing which is free from social demands;
- One of the key benefits of software development is that it provides a controlled environment with immediate feedback and no surprises, which is suitable for children with ASD;
- One of the reasons why they might be attracted more to computer programming specifically, is that it is consistent and logical.

2.1 COMPUTATIONAL THINKING

With the advances in technology in the past years, computing has become an area of great interest and present in almost every job. With this popularity and being programming an area in which students present several difficulties, the need arose to explore the education of Computational Thinking (Teixeira et al., 2020). Computational Thinking is currently an important subject of national education in many countries, some of them have even classified Computational Thinking as a national program or have prepared new teaching content and textbooks (Hsu et al., 2018). As mentioned by Grover et al. (2017), Computational Thinking is recognized as a fundamental competency for success in the fields of science, technology, engineering and mathematics (STEM). In addition, Computational Thinking also improves creative problem solving in other disciplines.

Tang et al. (2020) stated that many researchers defined Computational Thinking in terms of programming or computing concepts, so that they could not only design interventions based on programming activities but also assess students' programming skills as evidence of students' Computational Thinking skills.

Teaching computational thinking from an early age will benefit students not only to overcome difficulties in programming courses, but also to improve students skills that can be leveraged in all areas. This happens because the computational thinking ability promotes the improvement of skills such as the development of critical thinking, abstract thinking, logical reasoning, problem solving strategies and persistence (Araújo et al., 2019).

2.2 BLOCK-BASED PROGRAMMING LANGUAGE

The body of introductory programming should not correspond to the understanding of the grammar or syntax of that particular programming language, but it should be the development of the problem-solving skills with computing, that is called "Computational Thinking" (Matsuzawa et al., 2016). As mentioned before, the block-based programming languages are a way to develop interventions of Computational thinking. Computing is an essential skill for all students to develop in order to fully participate in an increasingly digital world (Weintrop and Wilensky, 2017).

Block-based environments uses a programming primitive-as-puzzle-piece metaphor to provide visual cues to the user as to how and where commands may be used. Programming is done by dragging command blocks from a palette into the scripting pane and assembling them, like puzzle pieces, creating "stacks" of blocks (Weintrop, 2019). Some of the popular environments of block-based programming are Scratch, Alice, and Code.org's Hour of Code activities. (Brown et al., 2016).

Block-based environments are today commonly used to introduce programming. An example is the Hour of Code campaign, which provides online introductory programming activities that reach millions of students. (Effenberger and Pelánek, 2018).

2.2.1 *Scratch*

Scratch is a block-based programming environment that emphasizes media manipulation and supports programming activities that resonate with the interests of children, such as creating animated stories, games, and interactive presentations (Maloney et al., 2008).

According to Weintrop (2019), since Scratch was launch, over 35 million users have created accounts on the Scratch website and almost 40 million projects have been shared with a majority of users being under the age of 14.

This visual programming language, despite interesting the youth, doesn't have a clear objective embedded in the platform, depending mainly on the teacher to use his creativity to develop programming activities. In other words, Scratch does not contain a system of levels with different activities and difficulties, with different subjects or themes, in which could improve students motivation, or track the students progress. The platform has more than 100 possible blocks to use. They are organized into sections, such as:

- Motion;

- Looks;
- Sound;
- Events;
- Control;
- Sensing;
- Operators;
- Variables;
- My Blocks.

2.2.2 *Code.org*

Code.org is a nonprofit organization that was founded in 2012, whose vision is that every student in every school should have the opportunity to learn. Code.org created an enormous interest in its Hour of Code initiative, an one-hour introduction to computer science. The goal of the Hour of Code is to demystify code and show that everyone can learn the programming basics (Liu et al., 2016).

The Code.org website contains a block-based environment with lots of different programming activities, making use of different and well known games and characters, to motivate, attract and interest the young public. Code.org contains a catalog of courses that can be used since the kindergarten until the high school. There are courses the are planned for students that can't read, having blocks that contains only images. The platform makes possible for teachers to have an account, to create classes and start assigning courses and check students' progress. The courses consist of a variety of lessons and each lesson corresponds to either a set of levels or unplugged activities. As the student progresses through the levels, the difficulty increases. Each lesson corresponds to a different topic or subject, for example: computational thinking, the artist, functions, the farmer and conditionals. A programming activity level doesn't show all the blocks available in the whole platform, only the ones relevant to the level, what is good for the student to not divert the focus on the programming question.

The visual programming language development will leverage some of the features found on Code.org, such as the management area that the teachers have, that can track the students progress and assigning courses with lessons to their students. It was decided to create Robi, a visual programming language, and not use the Code.org because its creation didn't have educational robotics as its main focus. Having in consideration that the programming language will be used to program a particular robot, the blocks could be better identified by containing pictures of parts of the robot. A section of blocks focused on the robot sensors could also be created. Robi platform will also enable teachers to create their own courses, lessons and levels.

2.2.3 *Alice*

The Alice environment provides to the students the possibility to create animation and/or games, without the need to focus on the syntax of the language. Alice allows the programmer to create code without worrying about semicolons or curly braces, for example, and allows the students to focus on the concepts (Ali and Smith, 2014). The code is constructed in the form of puzzle pieces, causing the characters s in the virtual environment to execute tasks (Liu et al., 2016).

According to Liu et al. (2016), Alice was shown to be an effective alternative to standard programming languages in teaching programming to undergraduate computer science students, since Alice's approach addresses many of the issues afflicting programming education.

Like Scratch, Alice shares the same downsides related to the problem raised on this thesis. This software doesn't contain a system of levels with different activities and difficulties, with different subjects or themes, in which could support the teacher in the classes and improve students motivation. Alice doesn't have a teacher management area as the one found on Code.org, which is a great feature that could support the teachers analysing their students progress, or assigning different courses that could fit the students profile.

2.2.4 *Kodu Game Lab*

Similar to the programming languages described above, Microsoft Research's Kodu Game Lab are used as well to introduce students to computer science concepts and programming (Stolee and Fristoe, 2011). Kodu is integrated in a real-time 3D gaming environment, with a intuitive user interface (MacLaurin, 2011).

2.3 EDUCATIONAL ROBOTICS

Educational robotics in early childhood education facilitates the students learning in a playful way, based on principles of interactivity, social interrelations, collaborative work, creativity, constructivist and constructionist learning and didactic approach centered on the student, allowing them to improve their digital skills and develop logical and computational thinking (González-González, 2019).

Robotics platforms, together with an easy-to-use visual programming tools, can be an effective way to introduce computational thinking as it involves students being able to sequence the coding commands needed to program a robot (Chalmers, 2018).

Educational robotics is used worldwide in education as a learning tool, but rarely focused on students with with individual needs (Karna-Lin et al., 2006). If it can be focused mainly on the special needs of children, educational robotics has the potential for improving special needs education and for eliminating barriers to learning (Virnes et al., 2008).

2.3.1 *LEGO Mindstorms EV3*

Lego robotic systems comprise Lego bricks, a programming language, a microprocessor, wheels and friction gears, such as gears and cogwheels. The visual programming language based on the Lego system turns out to be a coherent and amusing subject since it uses flow diagrams instead of written text. Lego Mindstorms is a robotic set created by Lego for children aged 10 and above. It is aimed at getting students to acquire programming, basic design and robotic principles. (Korkmaz, 2016).

According to Korkmaz (2018), Lego sets have many advantages, such as:

- Providing real-life experiences to students;
- Enabling both individual and team work;
- Allowing students to actively take part in learning processes;
- Using interdisciplinary knowledge;
- Presenting alternative ways for solving problems.

The visual programming language that will be developed, will make use of the way Lego chose to provide its programming blocks, making more use of pictures instead of written text. Although Lego provides a great approach at building the robot, and a platform to implement code that is intuitive and easy to learn, Lego is more expensive than an Arduino robot kit. The purpose of this thesis is also to provide a non expensive solution, so it was decided to work with an Arduino, that makes possible to create economical robots (Pisarov and Mester, 2019).

2.3.2 *mBot*

The robot mBot is built using an Arduino, and makes use of the mBlock software, a visual programming language based on Scratch 2.0 to transfer the programming commands to the mBot. The mBlock software does not only inherit the characteristics of Scratch, but also increases many script modules to interact with hardware (Pisarov and Mester, 2019).

The Arduino platform has become very popular with people who are involved with electronics. As reported by Badamasi (2014), unlike most previous programmable circuit boards, the Arduino does not have a separate piece of hardware in order to load new code onto the board, it just needs a USB cable to upload the code. Arduino consists of both a physical programmable circuit board and and an Integrated Development Environment (IDE) that runs on the computer, and is used to write and upload computer code to the physical board (Pisarov and Mester, 2019).

The mBot is an economical mobile robot equipped with Bluetooth and various sensors. With the drag and drop technique to implement code that can run on the robot, children can learn to program quickly, so they control the robot and try out different features that mBot has. This robot is an educational tool that gives beginners basic knowledge in programming, electronics and robotics (Pisarov and Mester, 2019).

The solution provided by this thesis will also use an Arduino to create the robot. Since mBot makes use of the mBlock software, that is based on Scratch, mBot doesn't fit as a solution to the problem raised on this thesis. The visual programming language that will be developed, will have personalized programming blocks with pictures related to the robot, making it more intuitive, with less text to read. In other words, the visual programming language development will be dedicated to the Arduino robot.

2.3.3 *Robomind*

Different from the programming platforms described above, Robomind is not block–based concept with a drag-and-drop interface, but a syntax based concept. (Faisal et al., 2017). This platform can be used for students ranging from primary schools, secondary schools and further education. Basically, the main objects of Robomind are how to control the robot movement through a sequence of instructions written in a particular programming language (Yuana and Maryono, 2016). The interface allows not only the users to inspect which robot behaviors are running, but also the data collected from the robot sensors in run-time (Svendsen, 2014).

Although Robomind contains a simple and attractive graphical display (Faisal et al., 2017), this platform is textual based, so it restricts the public that have difficulties at reading and writing, in which could demotivating. This solution doesn't attend to the problem raised by this thesis, since the proposal is to have a platform with a programming language more visual, so it could be a better fit for students with special needs that the learning process. For example, the learning process for the individuals with ASD is more effective when working with technologies that use visual interfaces, in which goes along with their characteristics and preferences. (Elshahawy et al., 2020).

2.4 SUMMARY

Based on the research done and discussed along this chapter, it's possible to state that computer-based interventions to students with special educational needs[1] have proven to be useful and effective for their learning. It's also possible to state that the popular block-based programming languages and educational robotics kits, that were presented in this chapter, even with a lot of motivational and engagement potential still lack some simplicity to be used with full success when working with children with special educational needs.

1 Students with hyperactivity and autism

3 PROPOSED APPROACH

This chapter will present the proposal describing how the desired programming environment will be built. As this project aims at the creation of a new programming language, it will be necessary to create a compiler, therefore, this chapter contains not only the system architecture, but also what the compiler will consist of. This chapter will also present the requirements elicited during the preliminary analysis phase, and the technological decision.

3.1 SYSTEM ARCHITECTURE

The system will contain two web applications. One web application will be running in the cloud, and the other one will be a web application that will run locally in the user's computer. The web application that will be running in the cloud will have the responsibility of registering the users, providing the download of the web application that will run locally and storing in its database all the data that comes from the local web application. The local web application will contain the block-based programming environment, as well as the management area for the teachers to assign and create courses for their respective students. The local web application will contain also a compiler service, that as the name suggests, will compile the code developed by the user to a code that the Arduino understands, so the code upload can be possible. This web application will also contain a database.

The database associated with the local web application will also persist data that comes from the user. Although the data will be persisted twice (in the remote database and in the local database), it will increase the performance.

The local web application will contain a text converter, that will convert the visual code implemented by the user to a text, so that it can be compiled and turned into a file with a format that the Arduino can understand. Figure 1 shows the system architecture as explained above.

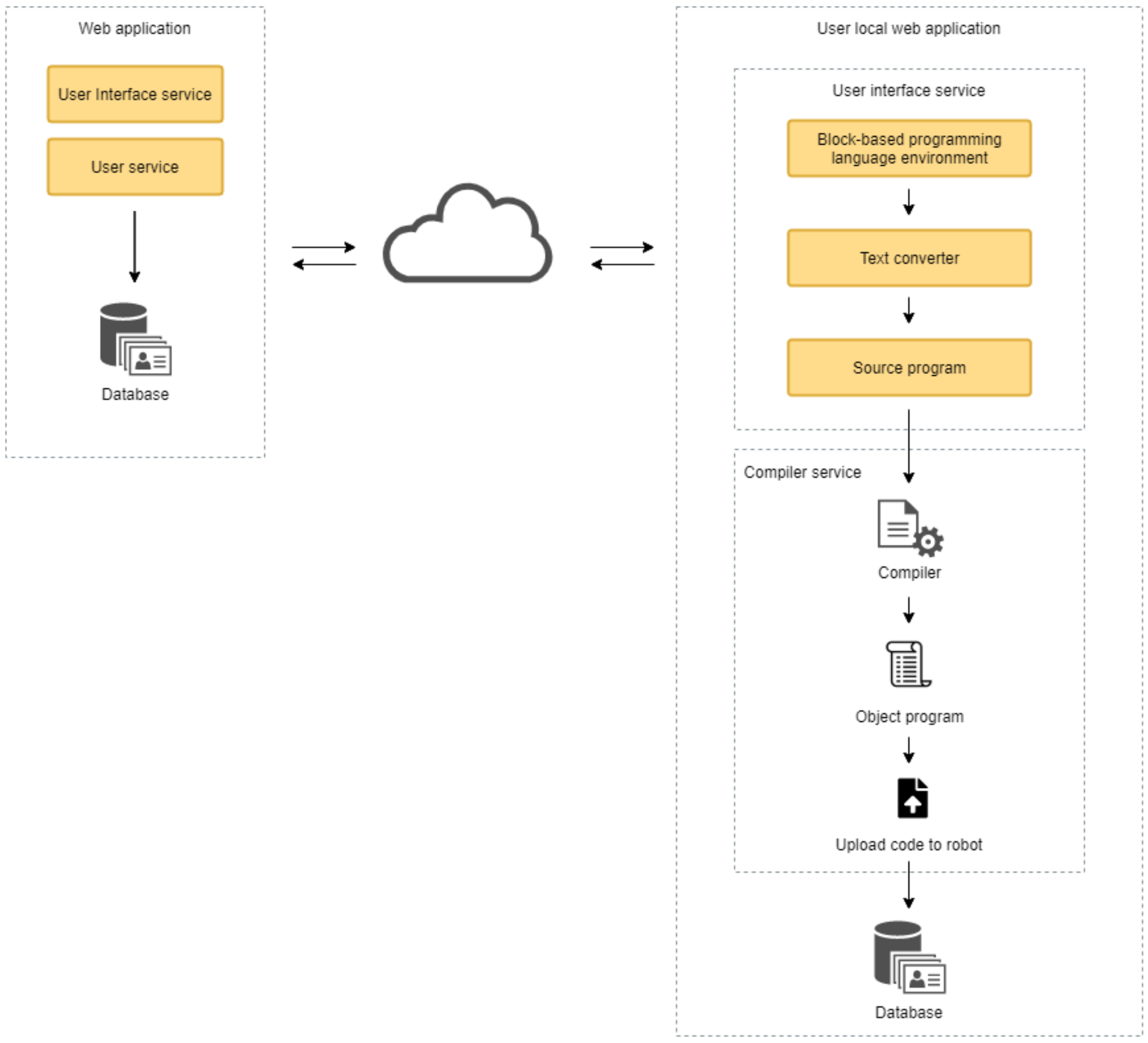


Figure 1: System architecture

After being clear the system behavior and architecture, it was decided to create a class diagram to have a better view of how the entities relate to each other. It can be seen in the Figure 2.

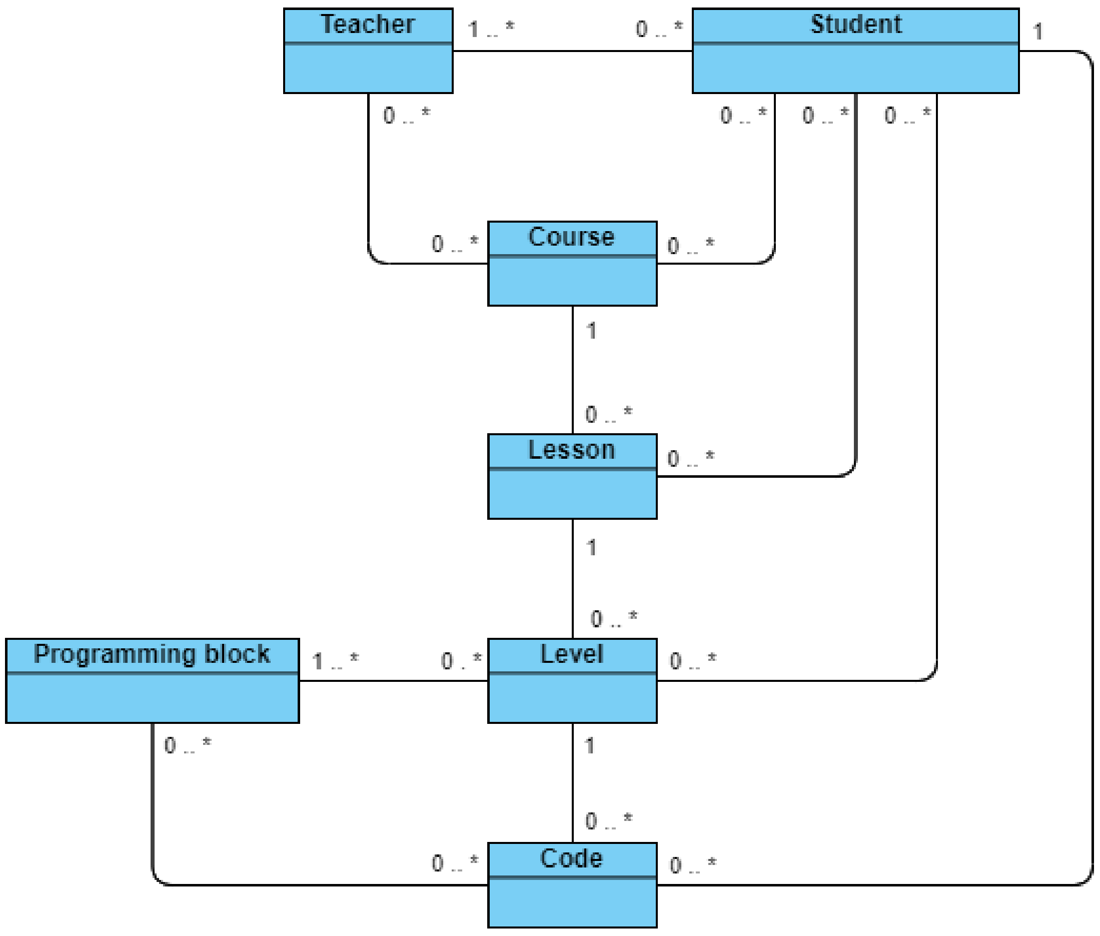


Figure 2: Domain Model: Entity-Relationship Diagram

3.2 COMPILER

While programming visually, using Robi block programming language, Robi editor generates a textual image of the visual program developed. When the student presses the button to upload the code to the Arduino robot, the textual file, so far created, is sent to a compiler that transform that source program into the Arduino target language. In other words, the text converter will generate a source program, that is sent to the compiler. As the Figure 3 shows, the compiler will consist of two phases:

ANALYSIS PHASE: This phase will responsible for reading the source program, dividing it into core parts and then checking for lexical and grammar. This phase is also responsible to generate an intermediate representation of the source program and identifier table, in which becomes an input to the Synthesis phase.

SYNTHESIS PHASE : This phase will be responsible for generating the object program with help of the Analysis phase output and the identifier table. The object program is the output of the compiler, and will be in a format that the Arduino can understand, so it can be uploaded without any problems.

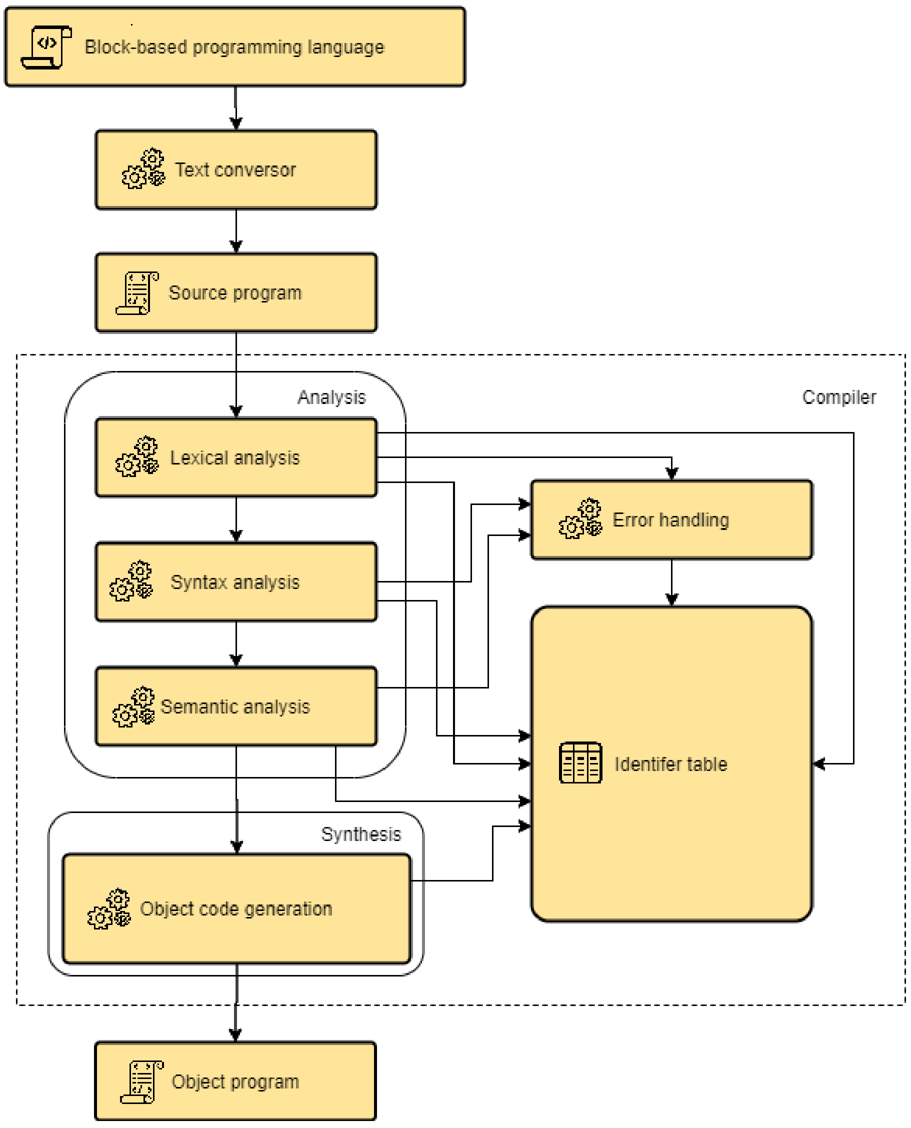


Figure 3: Compiler architecture

3.3 SYSTEM REQUIREMENTS

During the analysis phase some requirements were identified in order to list the features that will be necessary for the proper design of the visual programming language, as well as to develop the software system with the functionalities necessary to accomplish the objectives above stated. Below is the list of requirements so far collected:

1. A section to register users;
2. A section to provide the download of the platform;
3. An IDE with all the possible command blocks to program the robot;
4. Robot command blocks that should be as less textual as possible;
5. A programming script that should be built by connecting blocks together;
6. The system script area with code implemented by the user to be able to upload to an Arduino Uno;
7. All users registered to the system to have teacher roles;
8. The user to be able to register student accounts;
9. The user to able to assign courses to the students they registered;
10. Courses that will contain correspondent lessons;
11. Lessons that can contain one or more levels;
12. The levels might contain only the relevant command blocks to that particular level.
13. The user to be able to create courses and correspondent lessons and levels;
14. The user to be able to track students progress;
15. The student to be able to log in to the account registered by his teacher;
16. The student to be able to access the courses assigned by his teacher;
17. The student to be able to track his own progress through the courses assigned to him;

3.4 TECHNOLOGICAL DECISION

There are many languages and packages that could be used in the development of this project, however, the technologies were chosen based on the author's experience. The languages and frameworks adopted are well known and proved to be efficient in similar projects. Below are described the technologies that will be used to the development of Robi platform:

- The frontend will be developed using the framework Angular.

- The backend will be developed using Java, Gradle and Spring boot.
- The API will be tested using the following technologies:
  - JUnit
  - WireMock
  - RestAssured
- The API will be documented using the following technologies:
  - Spring Rest Docs
  - Asciidoctor
- The relational database will be implemented with PostgreSQL.
- The programming blocks and the script area will be implemented by the use of an SVG library.
- The textual script equivalent to the visual will be compiled to an hex file, so that the Arduino can receive, upload, understand and execute the code (Fezari and Al Dahoud, 2018).

3.5 SYSTEM ARCHITECTURE REVISED

During Robi development, it was decided not to develop the class Management[1] wouldn't be needed, since the scope of the project was getting too large for a one year Master's work. But the main reason for that decision is that the referred class would provide features that are secondary considering the project main focus. Not implementing the class Management has impact in the following points:

- Technologies to support the system API documentation are no more needed, since the system now contains only one endpoint;
- Database associated with the first block (the cloud component) is no more necessary, since there is not more information related to the students to store in the cloud;
- Courses, lessons, levels and teacher-student relation can be removed from the Entity-Relationship Diagram of Figure 2, since they make no more sense because they were related to the class Management.

1 That class should offer to the Teacher functionalities to create classes, populate them with students and track students performance.

Another decision that was made during Robi development was about the creation of a compiler. In order to compile and upload the code to the robot, Robi system is using the same compiler as the Arduino IDE uses. The translation from the script implemented in Robi platform to the program that will be sent to the compiler was necessary to develop.

To sum up, the new version of the system contains two services. One service will be in the cloud, which corresponds to Robi platform, where the user creates the visual program in Robi visual language, generating the script that will be sent to the robot. The other service will run locally, in the user's computer. That service is called Compiler service, and is responsible to receive the code implemented by the user in the cloud Robo editor and to compile and upload the code to the robot the user has. Figure 4 shows the updated system architecture.

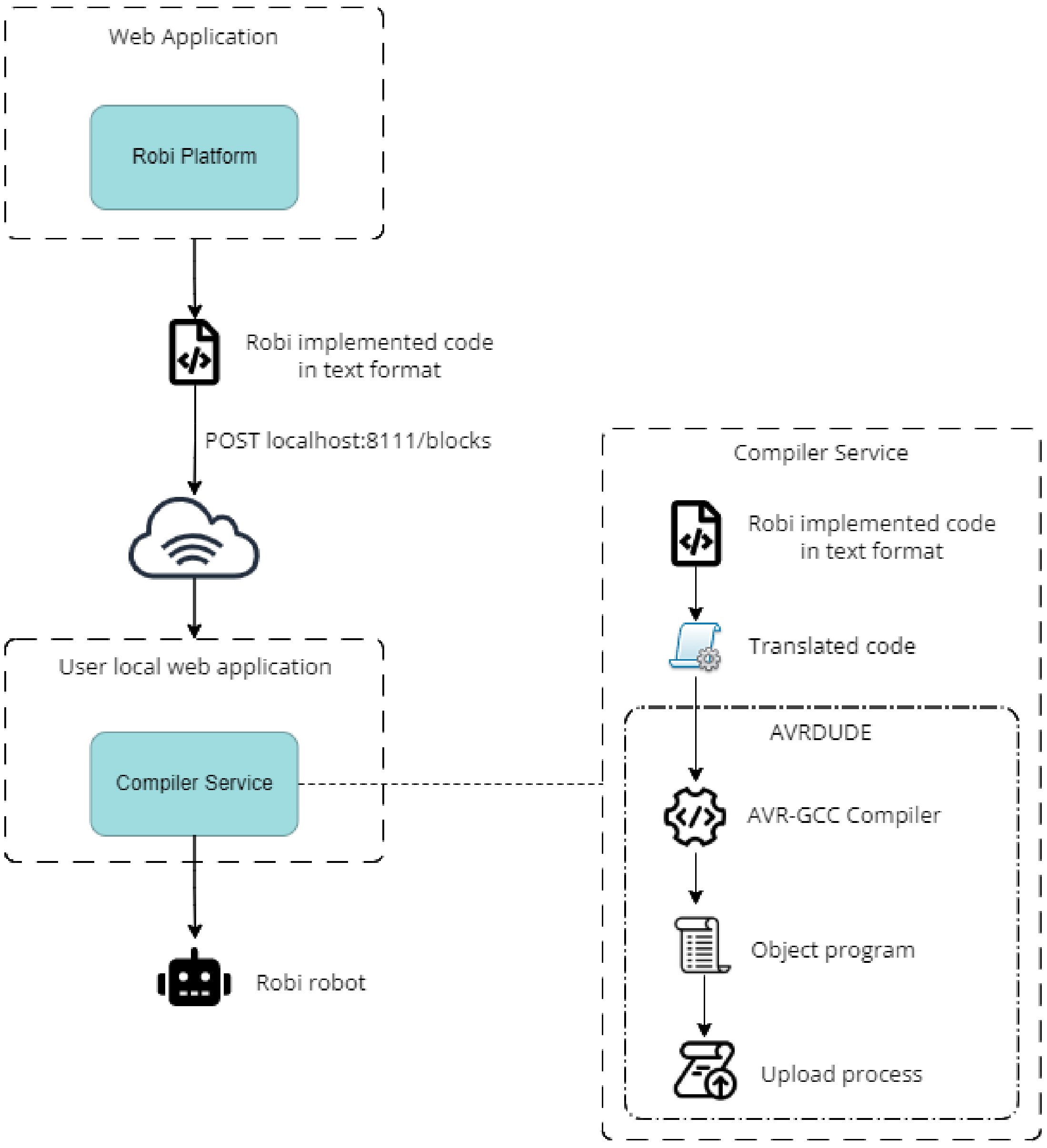


Figure 4: Updated system architecture

In Robi platform, when the user clicks on the button to send the code to the robot, the system will do a POST request with the script object to an endpoint located at localhost, on port 8111. This endpoint is served by the Compiler service, and will be accessible to receive requests from Robi platform system.

4 LANGUAGE

Robi visual programming language, a block-based environment, was developed using Angular 11, a free and open-source web application framework based on TypeScript, implemented by Google. The main purpose of this development is to offer an intuitive and easy to learn programming language and development environment, aiming at making more effective the inclusion of children with special educational needs.

Robi programming language was designed with the purpose of programming an Arduino-based robot. For this reason, this language is chiefly composed of robot movement and sensor blocks. Besides these robot-oriented blocks, Robi contains also programming-oriented blocks, such as repeat block, if block, if-else block, delay block, math operator blocks, conditional blocks, variable block and set variable block.

4.1 GRAMMAR

Bellow is defined the complete set of terminal symbols:

- INIT
- VARIABLE
- IF-THAN
- IF-THAN-ELSE
- FORWARD
- BACKWARDS
- TURN_LEFT
- TURN_RIGHT
- DO
- TIMES

- WHILE
- UPDATE
- TO
- AND
- OR
- GREATER_THAN
- LESS_THAN
- EQUALS
- NUMBER
- SENSOR
- PLUS
- MINUS
- TIMES
- DIVISION

A program is formally defined by the following rule:

- Variables INIT Block.

The INIT symbol corresponds to the Init block. The grammar rules are described below.

Variables	: VARIABLE
.	| Variables VARIABLE
Block	: Instruction
.	| Block Instruction
Instruction	: Condition
.	| Assignment
.	| Loop
.	| Move Expression
Condition	: IF ConditionalInput THAN Block
.	| IF ConditionalInput THAN Block ELSE Block
Move	: FORWARD
.	| BACKWARDS

```
.                         | TURN_LEFT
.                         | TURN_RIGHT
Loop                      : DO Expression TIMES Block
.                         | DO WHILE ConditionalInput Block
Assignment                : UPDATE VARIABLE TO Expression
ConditionalInput          : Expression ConditionalOperator Expression
.                         | ConditionalInput NestedConditionOperator ConditionalInput
NestedConditionOperator   : AND
.                         | OR
ConditionalOperator       : GREATER_THAN
.                         | LESS_THAN
.                         | EQUALS
Expression                : NUMBER
.                         | VARIABLE
.                         | SENSOR
.                         | Expression MathOperator Expression
MathOperator              : PLUS
.                         | MINUS
.                         | TIMES
.                         | DIVISION
```

The "Variables" rule generates one or multiple VARIABLE symbols. The VARIABLE symbol corresponds to the Variable block.

The "Block" rule can produce one or more "Instruction". That "Block" rule corresponds to the blocks present in Robi environment.

The "Instruction" rule produces "Condition", "Assignment", "Loop" and "Move Expression". It corresponds to the programming instructions, such as condition, assignment and loop. Another instruction that this rule generates is robot related, it generates an instruction correspondent to the robot movement.

The "Condition" rule is analog to a programming condition. That rule generates the IF symbol, that is analog to the conditional "if" and the THAN symbol. Every code after the THAN symbol will be part of the correspondent instruction.

The "Move" rule corresponds to the robot movement. It generates the following symbols: FORWARD, BACKWARDS, TURN_LEFT and TURN_RIGHT. The FORWARD symbol corresponds the Move forward block, the BACKWARDS symbol corresponds to the Move backwards block, the TURN_LEFT symbol corresponds to the Turn left block and finally, the TURN_RIGHT symbol corresponds to the Turn right block.

The "Loop" rule can produce "DO Expression TIMES Block" and "DO WHILE Conditional-Input" Block. The code after the DO symbol corresponds to the amount of times the code that is inside the correspondent programming instruction will be repeated. The code after the TIMES symbol corresponds to the code that will be executed in the loop instruction. The first generation done by the "Loop" rule is analog to the for loop, and corresponds to the Repeat block. The second generation is analog to the while loop, and corresponds to the While block.

The "Assignment" rule corresponds to the Set variable block. The UPDATE symbol that is part of what this rule produce, corresponds to the set action of a variable. The VARIABLE symbol corresponds to the variable that will have its value set. The value that comes after the TO symbol corresponds to the value that will be set in the correspondent variable.

The "ConditionalInput" rule corresponds to the input found in the conditional operator blocks. This rule can produce multiple conditions joined by AND and/or OR operators, or this rule can generate a condition with the conditional operators greater than, less than or equals.

The "NestedConditionOperator" rule generates the AND and OR symbols. They correspond to the "and" operator and "or" conditional operators, respectively.

The "ConditionalOperator" rule generates the symbols GREATER_THAN, correspondent to the "greater than" conditional operator, LESS_THAN, correspondent to the "less than" conditional operator and EQUALS, correspondent to the "equals" conditional operator.

The "Expression" rule can generate a "NUMBER" symbol, correspondent to a number, VARIABLE symbol, SENSOR symbol, correspondent to the Sensor block, and can generate multiple expresssions joined by a math operator, such as plus, minus, times and division.

And finally, the "MathOperator" rule generates the following symbols: PLUS, MINUS, TIMES and DIVISION. These symbols corresponds to math operators plus, minus, times and division, respectively.

4.1.1 *Example*

The implementation below has the objective of making the robot move forward until it detects an obstacle. If an obstacle is detected, then the robot must turn and move forward again, until it finds another obstacle. This action is repeated three times. Finally, after the robot detects the last obstacle, the robot turns again. This example problem is solved using the language defined by the grammar above:

```
: VARIABLE distance

: INIT
: UPDATE distance TO 20
: DO 3 TIMES [
:     DO WHILE (SENSOR GREATER_THAN distance) [
:         FORWARD 250
:     ]
:     TURN_RIGHT 1200
: ]
: TURN_LEFT 1200
```

Figure 5 shows the visual representation in the block programming language for the example described above.

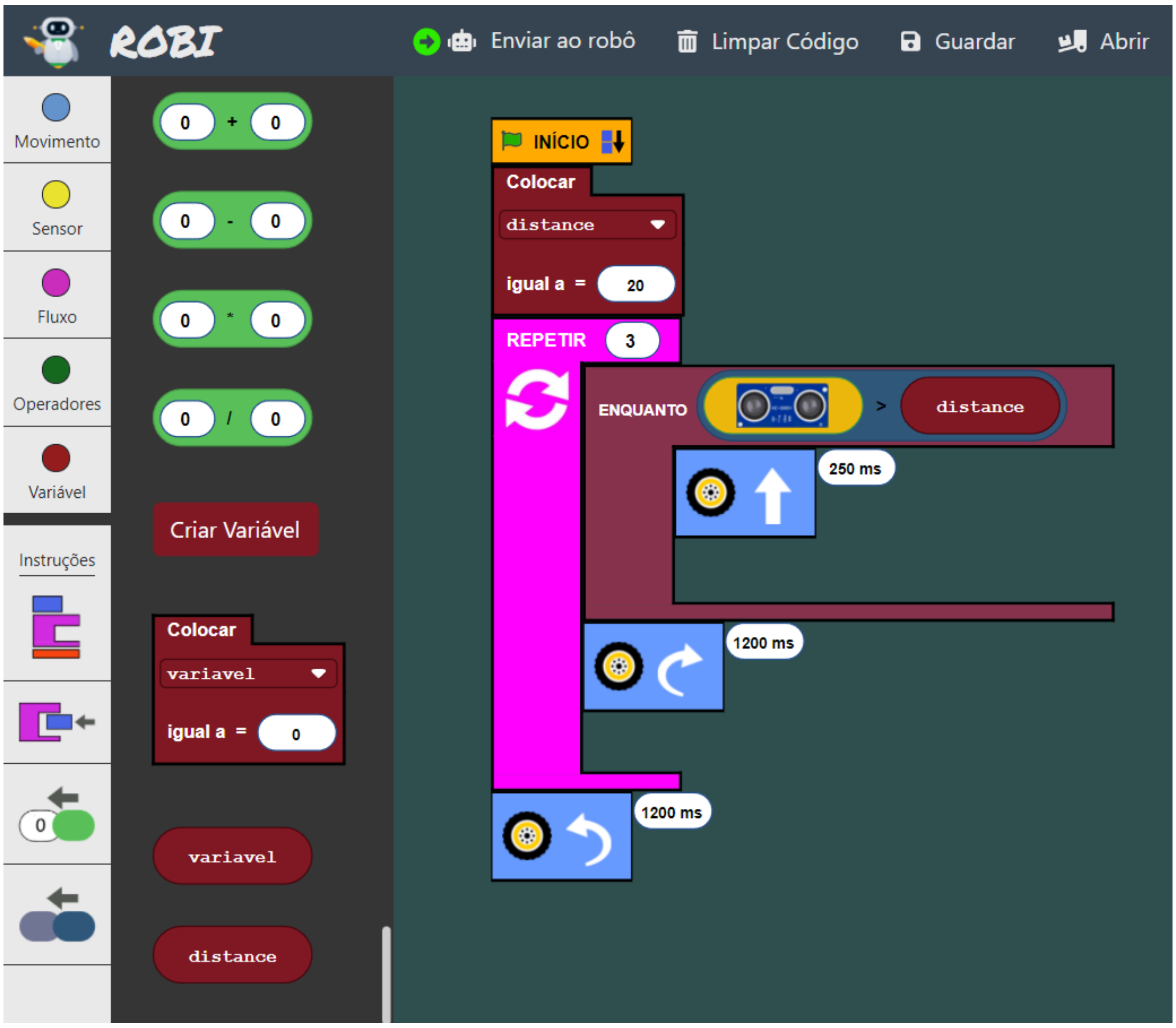


Figure 5: An example of a visual program created in the Web Interface

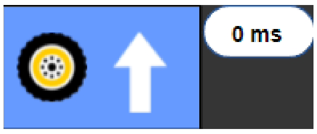
4.2 BLOCKS

The blocks corresponding to the terminal symbols above are divided into 5 categories: Movement, Sensor, Flux, Operators and Variable.

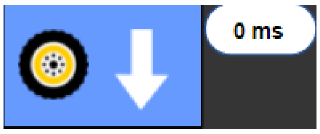
4.2.1 *Movement Category*

That set of blocks is responsible for the robot movement. At the right side of this block there is an input. That input corresponds to the amount of time in milliseconds that the motors will be turned on. On clicking this input, the user can either write a number from 0 to 9999 ms, or add a math operator block, a variable block, or the ultrasonic sensor block.

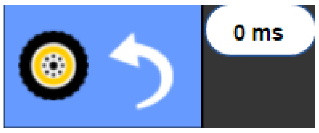
MOVE FORWARD Block responsible to move the robot forward.



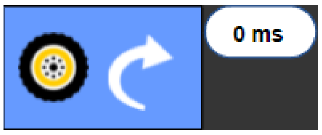
MOVE BACKWARDS Block responsible to move the robot backwards.



TURN LEFT Block responsible to make the robot turn left.



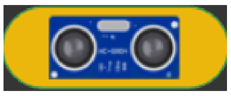
TURN RIGHT Block responsible to make the robot turn right.



4.2.2 *Sensor Category*

The robot contains only one sensor, that is the ultrasonic sensor block. This sensor is responsible to retrieve the distance that it is from an obstacle.

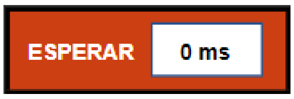
ULTRASONIC SENSOR The image inside this block reflects the sensor presented in the robot.



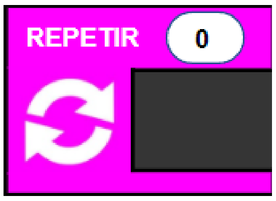
4.2.3 *Flux Category*

The blocks contained in this category are related to the flux that the robot does, such as iterations, delays and conditions. The blocks that contains a blue ellipsoid space with the same format as the white ellipsoid that contains a number, has the purpose of being the space in which the user can add the conditional operators, represented by the blue color. The white ellipsoid space with a number in it, on clicked, allows the user to write a number between 0 and 999. As in the movement blocks, the user can either add a math operator block, a variable block, or the ultrasonic sensor block.

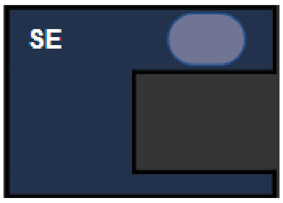
WAIT Block responsible to make the robot stop and wait the given value by the user. The range goes from 0 to 9999. The rectangular input found in this block does not allow the user to add another block in it.



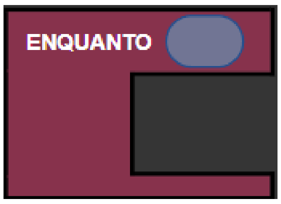
REPEAT Corresponds to the for loop. This block will repeat the code that is inside of it for a given value by the user.



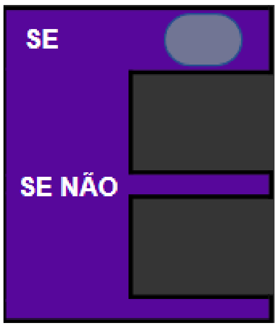
IF Corresponds to the if conditional. If the condition given by the user is true, then the code inside of it will run.



WHILE Corresponds to the while loop. While the condition given by the user is true, then the code inside of it will run.



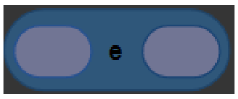
IF ELSE Corresponds to the if else conditional. If the condition given by the user is true, then the code inside of the upper space will run, or else, the code inside the lower space will run.



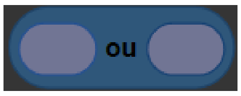
4.2.4 *Operators Category*

In this category there are three subsets of blocks, meaning that each subset contains different behaviors. One subset corresponds to the AND operator and the OR Operator, in which contain two blue ellipsoid spaces, one in each side. In other words, this subset of blocks accepts in these spaces all the blue operator blocks, such as nested adding AND operators and OR operators or the conditionals greater than, less than and equals. The second subset is formed by the conditionals mentioned before: greater than less than and equals. These three blocks are categorized by two white ellipsoid inputs with a number in it, as found in the movement blocks and the repeat block. This means that this subset accepts in those inputs the math operator blocks, variable blocks, or the ultrasonic sensor block. The third subset corresponds to the math operators. The math operators blocks contains the following math operations: plus (+), minus (-), times (*) and division (/). This subset is categorized by having two white ellipsoid inputs, one in each side, and in the middle is the math operation symbol. As in the repeat block, as well as in the second subset of operator blocks mentioned in this section, is possible to nested add math operator blocks, as well as the variable block and the ultrasonic sensor block.

AND OPERATOR Corresponds to the AND operator. This block can be placed inside the inputs found in the blocks that accepts condition.

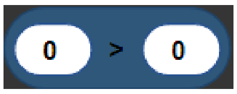


OR OPERATOR Corresponds to the OR operator. This block can be placed inside the inputs found in the blocks that accepts condition.

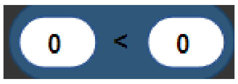


CONDITIONAL GREATER THAN Corresponds to the condition greater than. This block can be placed inside the inputs found in the blocks that accepts condition, including the AND

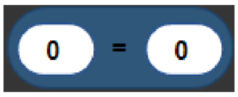
and OR operators.



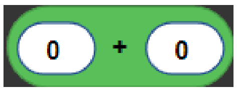
CONDITIONAL LESS THAN Corresponds to the condition less than. This block can be placed inside the inputs found in the blocks that accepts condition, including the AND and OR operators.



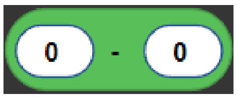
CONDITIONAL EQUALS Corresponds to the conditional equals. This block can be placed inside the inputs found in the blocks that accepts condition, including the AND and OR operators.



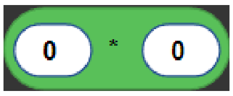
PLUS It corresponds to the math operation plus. It sums the two numbers or expressions found in each side of the block.



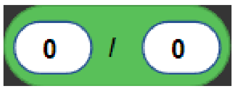
MINUS It corresponds to the math operation minus. It subtracts the two numbers or expressions found in each side of the block.



TIMES It corresponds to the math operation times. It multiplies the two numbers or expressions found in each side of the block.



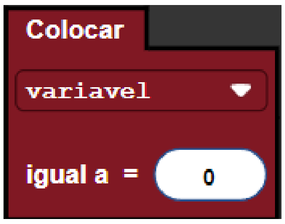
DIVISION It corresponds to the math operation division. It divides the two numbers or expressions found in each side of the block.



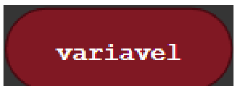
4.2.5 *Variable*

This category contains two blocks and one button. The button is responsible to create a variable with a maximum of twelve letters or numbers. The variable name chosen by the user must start with a letter. The set variable contains a dropdown. This dropdown is populated as the user creates a variable. When the user chooses a variable in the dropdown, he can set its value in the white ellipsoid input just below. This white ellipsoid input contains the same behavior found in the movement blocks, regarding the adding of a block. The range in this input goes from 0 to 9999.

SET VARIABLE Block responsible to set the value of a variable that the user selects in the dropdown.



VARIABLE Corresponds to a variable. When this block is created, its initial value is set to 0.



5 ROBOT

The aim of building a robot and integrating it into Robi environment was to provide an extra challenge to children, proposing the use of a physical material in addition to the computer. Having a robot, the didactic and dynamic increases. It's a material that children could use in any environment suitable for it, allowing them or a teacher or even parents to build any obstacle they desire in the physical world, with the goals they choose. Teamwork could also be leveraged with the inclusion of a robot. One or more person can work in the programming blocks, and one or more person can be responsible for putting the robot in the correct place and analysing it.

5.1 COMPONENTS

The robot was built with the aid of an Arduino. The Arduino not only has extensive web support but also enables the creation of electronic and interactive objects. Arduino also contains a high amount of modules and sensors that can be integrated into it easily. Below is the list of components was used to build the robot:

- 1 Acrylic Robot Chassis
- 2 DC motors
- 2 gears
- 2 Rubber wheels
- 1 HC-SR04 Ultrasonic Sensor
- 1 Arduino Uno
- 1 Arduino Sensor Shield
- 1 L298N Dual H-Bridge Motor Driver
- Male and female Jumpers Wire Set

- 1 AA Battery Holder
- Motor Mounting Brackets
- ON-OFF Switch
- A 360° back-wheel

Figure 6, by Banggood (2021), shows the set of components used to build the robot.

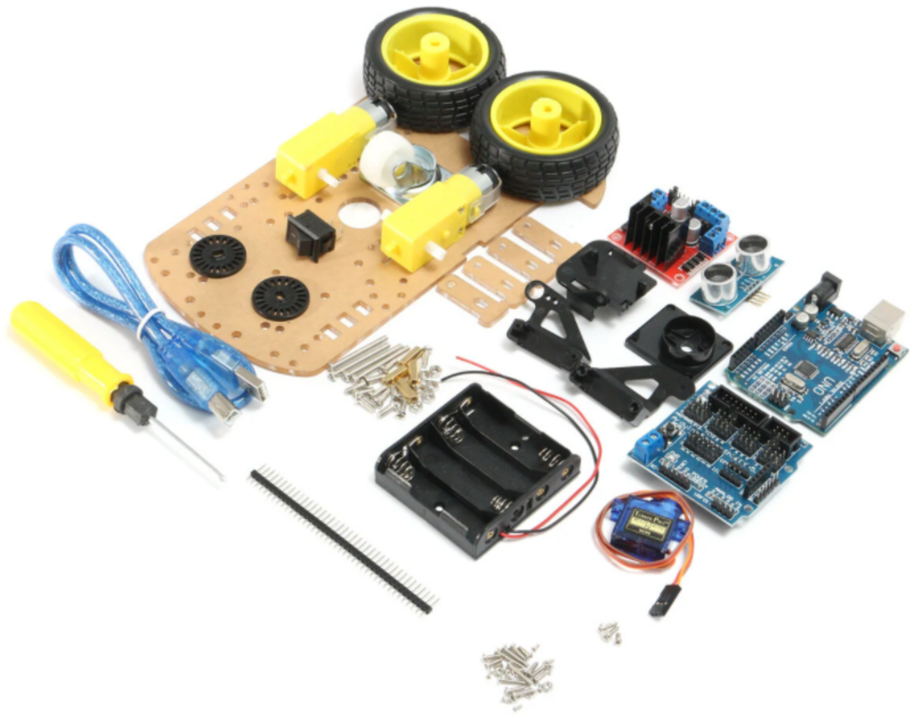


Figure 6: Robot components

Since the robot is a three-wheel car with 2 motors, and the motors are DC motors, the precision in which the robot has is not perfect. The DC motors works as soon as it has current going through them. It doesn't have any precision mechanism which makes them spin in relation to degrees.

The robot contains two different places which needs battery. One place has a 6V battery dedicated to the motors. That battery is connected to an ON-OFF switch, which easier the process of turning on and off the DC motors. That 6V battery is connected also to the L298N Dual H-Bridge Motor Driver. The other place contains a 9V battery dedicated to the Arduino. The ultrasonic sensor and the L298N Dual H-Bridge Motor Driver are directly connected to the Arduino.

Figure 7, by Tech (2018), corresponds to the circuit diagram used as basis to do the robot electronic part.

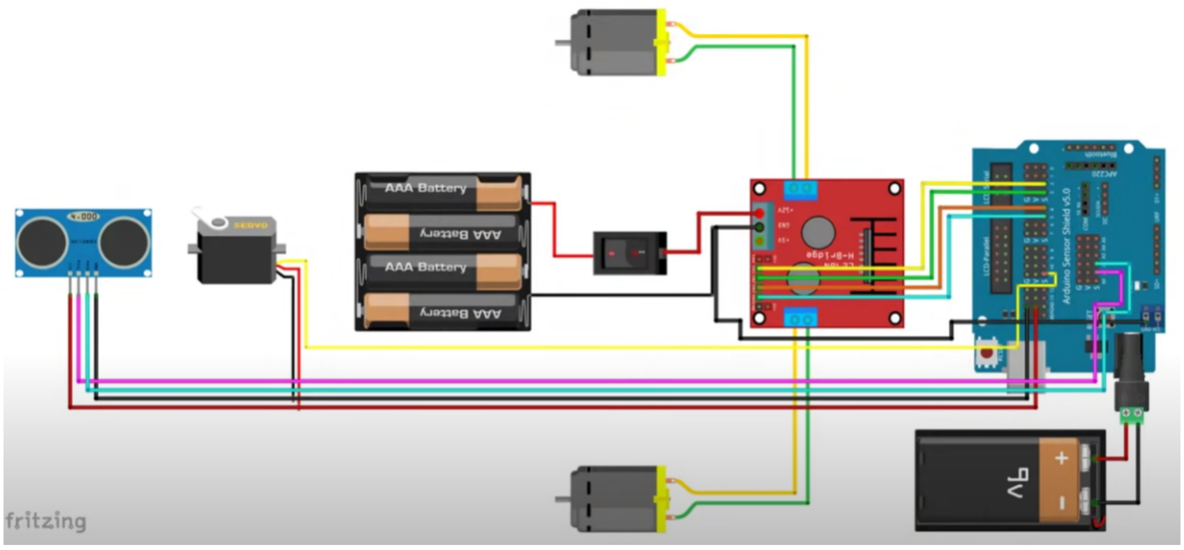


Figure 7: Circuit diagram

Is possible to upload code to the Arduino using the Arduino Integrated Development Environment (IDE). This IDE is a cross-platform application that is written in functions from C and C++, which makes easier implementing code to Arduino compatible boards.

Figure 8 corresponds to Robi robot after built.

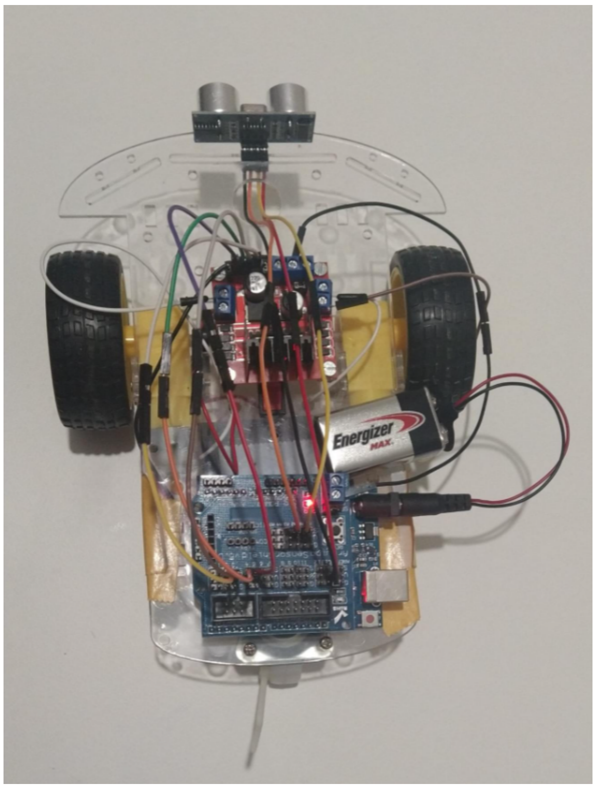


Figure 8: Robi robot

5.2 ARDUINO PROGRAMMING CODE

As explained before, using the Arduino IDE, the user can program an Arduino using functions from C and C++ languages. In order to the code be compiled successfully, the code should contain two mandatory methods, the setup and loop methods. Some things to take into consideration when programming an Arduino:

- All variables created outside the setup and loop methods will be declared/initialized.
- The setup method will be called only once. This method will be the first method to be called.
- The loop method will be called repeatedly, in an infinity loop, unless the user power off the Arduino, or press the Arduino reset button.

As an example of an Arduino implementation, the Listing 5.1 presents an example got from the Arduino IDE itself, that makes a LED blink for periods of one second.

```
const int led_digital_port = 3;

void setup() {
  pinMode(led_digital_port, OUTPUT);
}

void loop() {
  digitalWrite(led_digital_port, HIGH);
  delay(1000);
  digitalWrite(led_digital_port, LOW);
  delay(1000);
}
```

Listing 5.1: LED blink program

Figure 9 shows the result from the code present in the Listing 5.1.

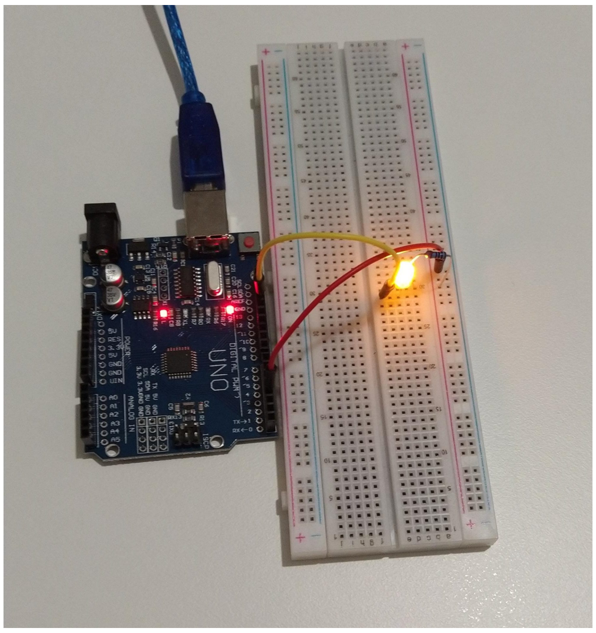


Figure 9: LED blinking

5.3 CODE UPLOAD PROCESS

It's possible to implement code using functions from C and C++, but the microcontroller present in the Arduino board doesn't receive/understand that language. In order to compile and upload the code to the Arduino, the Arduino IDE makes use of an open source software called AVRDUDE, an command-line driven user interface open source software which uses the avr-gcc compiler to compile the Arduino IDE code and upload it to the board.

The avr-gcc compiler makes use of the standard AVR libc libraries, which are open-source C libraries, specifically written for Atmel hardware, in which corresponds to the Arduino's microcontroller brand.

6

ROBI PROGRAMMING SYSTEM

As investigated and written in the State of the Art chapter from this thesis, children with special educational needs, or more precisely, children diagnosed with Autism Spectrum Disorder, tends to have preferences in technologies with visual interfaces. Robi programming system was developed to be as much visual and intuitive programming language and robot-oriented as possible. The robot-oriented blocks contain robot images by using those icons to carry on information; programming-oriented blocks contain as less text as possible. The system was developed to be simple, containing only the necessary instructions to program the robot Robi, making use of the programming statements or commands commonly found on programming languages, like loops, conditions, variables and math or conditional expressions.

6.1 VISUAL LANGUAGE INTERFACE

The system contains three main areas. The header, the left side bar, and the script area. The header contains four buttons, aligned horizontally, and an input at the right end of the header. The first button is to send the implemented code to the robot; the second button is to clear the script area, with the exception of the Init block; the third button is to save the implemented code and the last button is to load an implemented code. The input presented in the header corresponds to the COM port number that the Arduino is connected. For the computers with the Windows operational systems, the COM port number can be found in the Computer Management window, in the Ports (COM & LPT) section.

The left side bar contains all the blocks available in Robi environment that can be dragged and dropped in the script area. Since the left side bar contains more the 20 blocks, and in order to improve the user experience, a category selection section was implemented at the left side where the blocks are present. Below the category selection section, there are four instructions images.

The script area will always contains the Init block, even if the user clicks on the clear code button available in the header. This Init block is the only block that cannot be erased. The user can add as many blocks in the script area as he wants, but the code that will be sent to the robot, are the blocks that are connected to the Init block. The Init block only allow connection below it, which means that code created by the blocks will be interpreted from top to bottom. Figure 10 presents Robi programming environment.

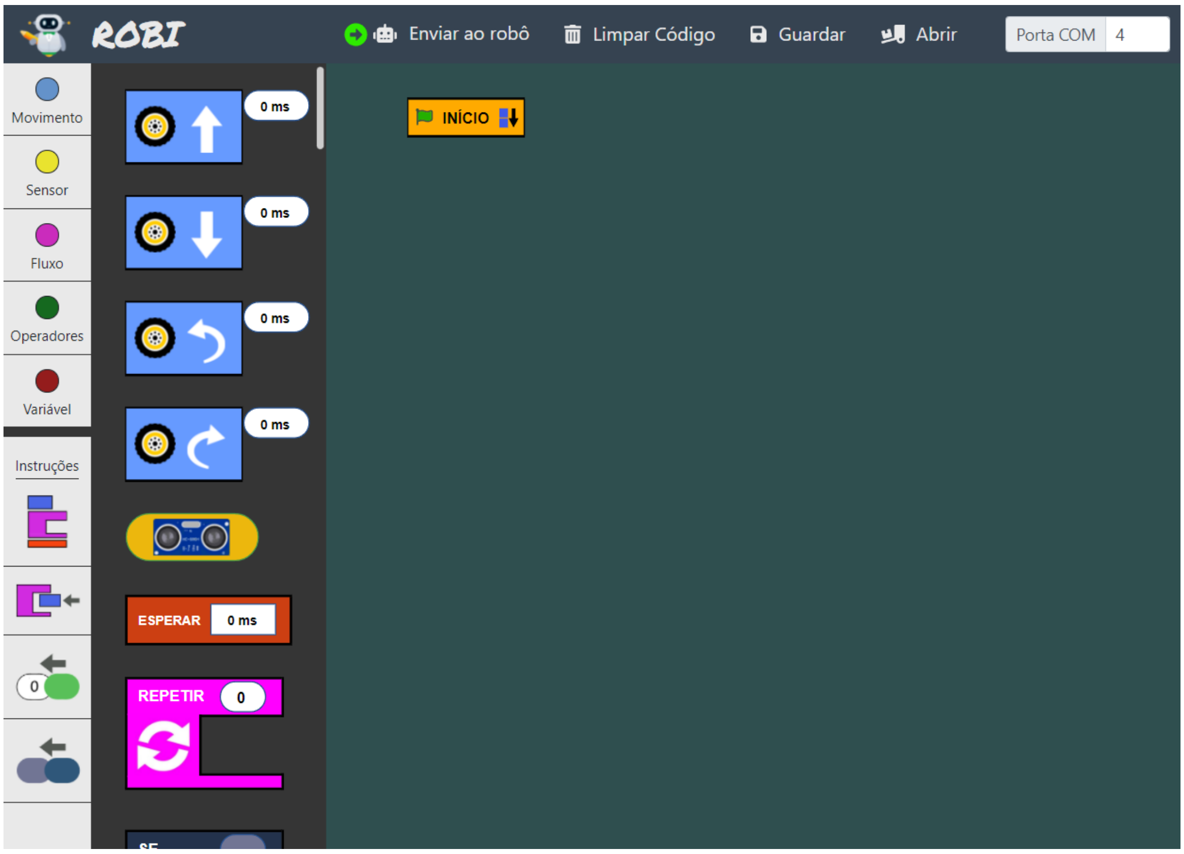


Figure 10: Robi programming environment

6.1.1 *Blocks Connections*

Every programming block present in Robi environment contains information about its position (vertical and horinzontal), action, identifier, type, size (width and height) and the blocks identifier that are plugged on it. Each block contains its own identifier and its own action. The blocks type contained in Robi system are the following:

- Init
- Movement
- Sensor
- Wait

- Flux
- If else
- Nest conditional operator
- Conditional operator
- Math operator
- Variable
- Set variable

With the exception of sensor, math operator, conditional operator, nest conditional operator and variable types, all the blocks connect into each other vertically. In other words, the blocks that are connectable vertically, are the ones with square corners. The blocks in an ellipsoid format connects in ellipsoid spaces. The blue ellipsoid blocks can be connected on blue ellipsoid spaces and every other ellipsoid blocks can be connected on white ellipsoid spaces with a number in it. If a block that the user is dragging is close enough to a connectable area from another block, the border that will link both blocks will be highlighted.

The blocks contain properties that will support the action of connection between blocks. If a block is being dragged by the user, all other blocks present in the script area will have the "isAnotherBlockMoving" Boolean property set to true. The blocks containing that property with the value "true" will become connectable blocks. With regard to the connection between blocks with square corners, these blocks contain two connectable areas, one in the bottom and the other in the top, which means that either the bottom border or the top border from these blocks can become highlighted. With regard to the connection between an ellipsoid block and an ellipsoid space present either in another ellipsoid block or in a block with square corners, if the ellipsoid block is inside the connectable area from that ellipsoid space than the border from that space will be highlighted. If a border is highlighted, and the user drops the block he is dragging, then that dropped block will be automatically positioned to a place where it is aligned with the block it was connected. Figure 11 corresponds to how the connection between blocks works.

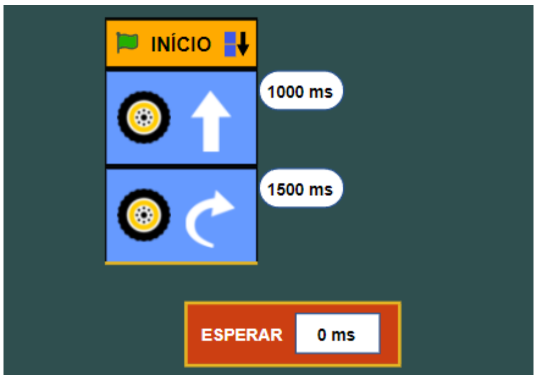


Figure 11: Connection between blocks

Within the square corners blocks there are blocks that are referred to in the system as aggregating blocks. The aggregating blocks are the ones that contain a rectangular space within them. The user can connect blocks inside that space. The Repeat block is an example of an aggregating block. The code that the user created with blocks he placed inside that space from the Repeat block will be repeated a given number of times that the he chose. Another example is the If block. If the condition from that If block is true, then the code the user created inside the If block space will be executed. Figure 12 presents an example of aggregating blocks being used in the script.

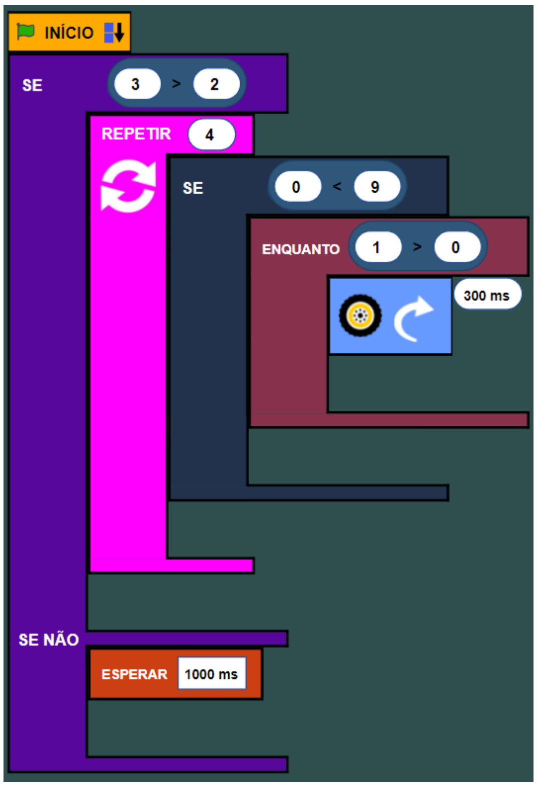


Figure 12: Connection between aggregating blocks

Every block contains a property called "pluggedBlocks". This property is an array of block identifiers. Whenever a block or multiple blocks are connected to each other, for the square corners block case, the uppermost block will add to the "pluggedBlock" array all the identifiers from the blocks below, in order, from top to bottom. For the ellipsoid blocks case is similar, the outermost ellipsoid block is analogous to the uppermost square corners block.

The square corners blocks that contains an input to add operators block contain a property called "operatorsBlockPlugged". If an ellipsoid block is connected to an ellipsoid space inside a square corners block, then that ellipsoid block identifier will not be added to the "pluggedBlock" property, but will be added to the "operatorsBlockPlugged".

6.1.2 *Blocks Movements*

If multiple blocks are connected to each other, and if the user drags the block in the top, then all the blocks connected to it will move together. In other words, if the user drags a block, then every block that has its identifier present either in the "pluggedBlock" property or in the "operatorsBlockPlugged" property from the block the user is dragging, will move along with it. All operators block connected to the blocks moving along will also move along.

Every block being dragged by the user, if dropped in the left side area, will be removed from the script area.

6.1.3 *Operator Blocks and Expressions*

Each operator block, such as math operators, conditional operators, variables or even the sensor block contains its own expression. Each time an operator block is inserted inside another operator, the outermost block expression will always be updated. Figure 13 corresponds to a math operator block with the plus operation.

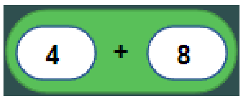


Figure 13: Plus operation from a math operator block

The math operator block shown in the Figure 13 above contains the following expression: (4+8). If another math operator block is inserted in the block above, then the expression from that block shown above will be updated. Figure 14 shows an example of nested math operator blocks.

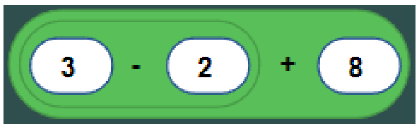


Figure 14: Nested math operator blocks

The expression from the outermost block would be the following: ((3 - 2) + 8). For the Conditional block operators, the expression is built similarly. The AND block correspond to the "&&" operator, and the OR block corresponds to the "| |" operator. Figure 15 presents an example of a nested conditional blocks.

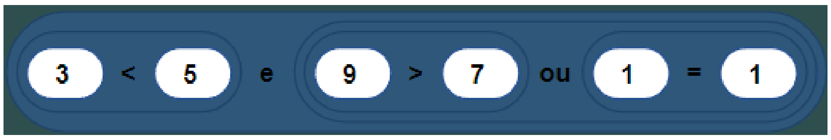


Figure 15: Nested conditional operator blocks

The expression for the condition above would be the following: ((3 < 5) & ((9 > 7) | (1 = 1))). After the user clicks in the send code to the robot button, and before the code is sent to the service that will compile the code and upload to the robot, the symbols "&" and "|" will be translated to "&&" and "| |" respectively. The expression correspondent the the sensor block is the following: (readPing()). The "readPing()" is a function that is already present in the code that will be sent to the robot, independent of the blocks used by the user. If a variable block is used in the script created by the user, then the expression correspondent to it would be the variable name chosen by the user between parenthesis. For example, if the user created a variable named "distance", then if that variable is used in the script, the expression correspondent to it would be: (distance).

6.2 CODE SETUP

For the robot to work as planned, besides the implementation done by the user, its programming comes with a configuration code, or a setup code. In that setup, there is code to define the ports from the motors and ultrasonic sensor, libraries import, ultrasonic sensor calibration and already implemented methods to support the code translation from the interface to the Compiler service.

The code needs to import two libs, one correspondent to the motors, called "Servo.h", and the other correspondent to the ultrasonic sensor, called "NewPing.h". The code responsible to import the libs is shown in the Listing 6.1.

```
#include <Servo.h>
```

```
#include <NewPing.h>
```

Listing 6.1: Libs import

As explained before, the robot contains two DC motors. Four Arduino ports are dedicated to these two DC motors, the port number 5, 4, 3, and 2. The port number 5 makes the robot left motor rotate forward; the port number 4 makes the robot left motor rotate backwards; the port number 3 makes the robot right motor rotate forward and finally the port number 2 makes the robot right motor rotate backwards. About the ultrasonic sensor, there are two Arduino analogical ports dedicated to it, the A1 and A2. The code responsible to the ports definition is shown in the Listing 6.2.

```
const int LeftMotorForward = 5;
const int LeftMotorBackward = 4;
const int RightMotorForward = 3;
const int RightMotorBackward = 2;

#define trig_pin A1
#define echo_pin A2
```

Listing 6.2: Arduino ports definition

Other variables definition to be used throughout the code, NewPing lib initialization, Servo lib initialization, and the setup method that every Arduino program contains, are shown in the Listing 6.3.

```

#define maximum_distance 200

boolean goesForward = false;

int distance_from_ultrasonic_sensor = 100;

NewPing sonar(trig_pin, echo_pin, maximum_distance);

Servo servo_motor;

void setup() {

    pinMode(RightMotorForward, OUTPUT);
    pinMode(LeftMotorForward, OUTPUT);
    pinMode(LeftMotorBackward, OUTPUT);
    pinMode(RightMotorBackward, OUTPUT);

    servo_motor.attach(11);
    servo_motor.write(90);
    delay(2000);
    distance_from_ultrasonic_sensor = readPing();
    delay(100);
    distance_from_ultrasonic_sensor = readPing();
    delay(100);
    distance_from_ultrasonic_sensor = readPing();
    delay(100);
    distance_from_ultrasonic_sensor = readPing();
    delay(100);
}
```

Listing 6.3: Arduino code setup

The setup method shown in the Listing 6.3 above is a method that normally should be sent to the Arduino. This method is the first to be executed, and is executed only one time. Inside this method is implemented the digital ports used by the motors being declared as OUTPUT, and the ultrasonic sensor calibration. Below this setup method, there are methods that are implemented to support the translation process. The methods implemented are shown in the Listing 6.4.

```
int readPing() {
    delay(70);
    int cm = sonar.ping_cm();

    if (cm == 0) {
        cm = 250;
    }

    return cm;
}

void moveForwardCustom(int delayTime) {
    goesForward = false;
    moveForward();
    delay(delayTime);
    moveStop();
}

void moveStop() {
    digitalWrite(RightMotorForward, LOW);
    digitalWrite(LeftMotorForward, LOW);
    digitalWrite(RightMotorBackward, LOW);
    digitalWrite(LeftMotorBackward, LOW);
}
```

```

void moveBackwardsCustom(int delayTime) {
    moveBackward();
    delay(delayTime);
    moveStop();
}

void moveForward() {
    if (!goesForward) {
        goesForward = true;
        digitalWrite(LeftMotorForward, HIGH);
        digitalWrite(RightMotorForward, HIGH);
        digitalWrite(LeftMotorBackward, LOW);
        digitalWrite(RightMotorBackward, LOW);
    }
}

void moveBackward() {
    goesForward=false;
    digitalWrite(LeftMotorBackward, HIGH);
    digitalWrite(RightMotorBackward, HIGH);
    digitalWrite(LeftMotorForward, LOW);
    digitalWrite(RightMotorForward, LOW);
}

void turnRight(int delayTime) {
    digitalWrite(LeftMotorForward, HIGH);
    digitalWrite(RightMotorBackward, HIGH);
    digitalWrite(LeftMotorBackward, LOW);
    digitalWrite(RightMotorForward, LOW);
    delay(delayTime);
    digitalWrite(LeftMotorForward, HIGH);
    digitalWrite(RightMotorForward, HIGH);
    digitalWrite(LeftMotorBackward, LOW);
    digitalWrite(RightMotorBackward, LOW);
}

void turnLeft(int delayTime) {
    digitalWrite(LeftMotorBackward, HIGH);
    digitalWrite(RightMotorForward, HIGH);
    digitalWrite(LeftMotorForward, LOW);
    digitalWrite(RightMotorBackward, LOW);
    delay(delayTime);
    digitalWrite(LeftMotorForward, HIGH);
    digitalWrite(RightMotorForward, HIGH);
    digitalWrite(LeftMotorBackward, LOW);
    digitalWrite(RightMotorBackward, LOW);
}
```

Listing 6.4: Implemented methods to support the translation process.

Below the code shown in the Listing 6.4 is all the code from Robi interface that will be translated by the Compiler service.

6.3 TRANSLATOR

After the user has implemented the desired code, and clicks on the send code to the robot button, the visual instructions the user chose for the robot to replicate will generate a JSON object, consisting of all the relevant information from each block, and also the COM port defined by the user, and all the variables the user created, counting with the default variable already created. That JSON object will then be sent to the service that will translate it to a language that the avr-gcc compiler will understand, in order to then compile that translated code to a language the Arduino understands.

Using the Figure 5 as an example, and assuming the user chose the *COM Port number 4*, and didn't create any variable, the JSON object that will be generated and sent to the Compiler service is shown in the Listing 6.5:

```
"blocks": {
    "0": {
        "action": "set_variable",
        "operatorExpression": undefined,
        "value": 20,
        "variableName": "distance"
    },
    "1" : {
        "action": "repeat",
        "blocksInside": {

        "0": {

        "action": "while",
        "blocksInside": {
        "0": {
            "action": "move_forward",
            "operatorExpression": undefined,
            "value": 250
        },
        "operatorExpression": "((readPing())>(distance))",
        "value": "0"
        }

        },
        "1": {
            "action": "turn_right",
            "operatorExpression": undefined,
            "value": 1200
        }

        }
    },
    "2" : {
        "action": "turn_left",
        "operatorExpression": undefined,
        "value": 1200
    }
},
"portCOM": "4",
"variablesInScript": ["variavel"]
```

Listing 6.5: Generated JSON object

The blocks with the "operatorExpression" property with an undefined value correspond to blocks with no operator block inserted in the input. For these cases, the "value" property will be the property that will be taken into consideration for these particular blocks. For the case that a block only accept conditional input, and the user doesn't insert any conditional operator, then the default value will be taken into consideration, that is the "false" value. If a variable is created by the user, and the user doesn't add the Set variable block, to set a value for that created variable, then the default value will be taken into consideration, that is the number 0.

With regard to the JSON object, every block present in it contain an action. That action is the identifier that the Compiler Service will use to identify the block and then map it to the correct code translation. About the output from the code translation, besides the code the user implemented, there will be already functions added automatically. The functions already existing in the code are the following:

- **moveForward():** function responsible to turn on both motors making them rotate in the same direction, making the robot to move forward.
- **moveForwardCustom(int delayTime):** function responsible to make the robot move forward for a specific number of time in milliseconds given by the user. Whenever the user uses the Move forward block, the block will be translated to this function here described.
- **moveStop():** function responsible to turn off both motors, making the robot stop. Whenever the user uses the Wait block, the block will be translated to this function here described, followed by a delay in the milliseconds the user chose.
- **turnLeft(int delayTime):** function responsible to turn on both motors in opposite directions, making the robot turn left for a specific number of time in milliseconds given by the user. Whenever the user uses the Turn left block, the block will be translated to this function here described.
- **turnRight(int delayTime):** function responsible to turn on both motors in opposite directions, making the robot turn right for a specific number of time in milliseconds given by the user. Whenever the user uses the Turn right block, the block will be translated to this function here described.
- **readPing():** function responsible to return the value correspondent to the distance in centimeters between the ultrasonic sensor and a obstacle in front of it. Whenever the user uses the Sensor block, the block will be translated to this function here described.

Besides the functions above, the code also have a code setup implemented, such as the import of libraries, variables being initialized, Arduino ports being defined and more.

About the "operatorExpression" property, during the translation process in the Compiler Service side, that property will be calculated. If the expression corresponds to a set of conditional operator blocks, then the value calculated and used in the correspondent instruction, would be either "true" or "false". In case the expression corresponds to a set of math operator blocks, sensor blocks and variables, then the value calculated and used in the correspondent instruction, would be a number. The sensor block would be translated to the number returned by the **readPing()** function, and the variable block would be translated to the value set by the user, or by the default value zero, in case the user didn't set a value for that specific variable. In case the user didn't add any expression to a block, then the value used in the instruction correspondent to that particular block would be the value set by the user, or in case it's a block that accepts only condition instead of a number, would be the default value, that is the "false".

After the translation process is done, then the code will be compiled and uploaded to the *COM Port* chosen by the user in the interface. If the Arduino from the robot is connected in the right *COM Port*, then code will be uploaded successfully.

The output from the translation, in addition with all the code configuration automatically added by the Compiler service correspond to the final code that will be sent to the robot. So, about the example shown in the Figure 5, the final code is shown in the Listing A.1 that is shown in Appendix A.

For every block sent to the Compiler service, two lines code will be added right after the correspondent instruction. The Listing 6.6 present these two lines:

```
moveStop();
delay(200);
```

Listing 6.6: Two lines code added after every block instruction

The Compiler service adds at the end of the loop method a while loop that will iterate infinitely with the "moveStop()" method inside. This is shown in the Listing 6.7.

```
while(1) {
    moveStop();
}
```

Listing 6.7: Stop instruction inside an infinity loop

Having a program developed by an user in Robi language, having it being translated to a code that can be sent to the compiler so it can be successfully uploaded and executed by Robi Arduino-based robot is process that defines Robi environment. The next step is to do a comparison between Robi environment and Scratch environment.

7

COMPARISON BETWEEN ROBI AND SCRATCH

Robi is a platform that was made to be simple, intuitive and to be as most visual as possible. In the other hand, Scratch is a more textual platform than Robi programming environment, but more powerful containing a higher amount of block categories, and consequently, more blocks. Figure 16 and Figure 17 present an exercise example written in both environments.

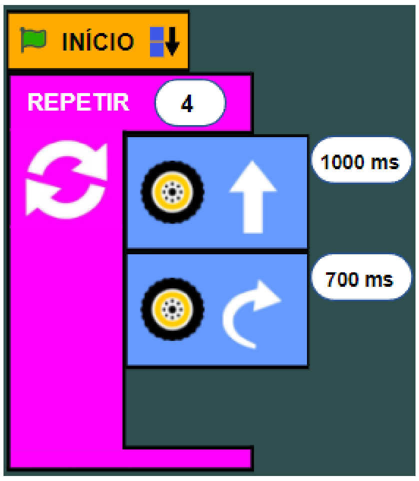


Figure 16: Square exercise implemented in Robi language

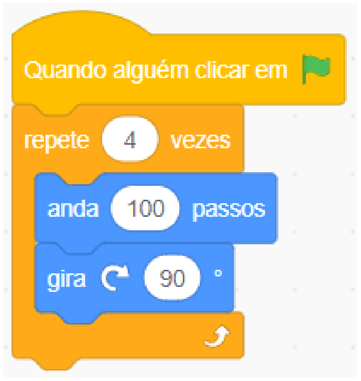


Figure 17: Square exercise implemented in Scratch language

Figure 16 is less textual than Figure 17, but both contain the same objective. Another noticeable difference is the blocs size. Robi platform contains bigger blocks than the Scratch blocks.

With regard to the left side bar found on both systems, where the user can select a category or drag a block to place in the script area. Figure 18 and Figure 19 present that case.

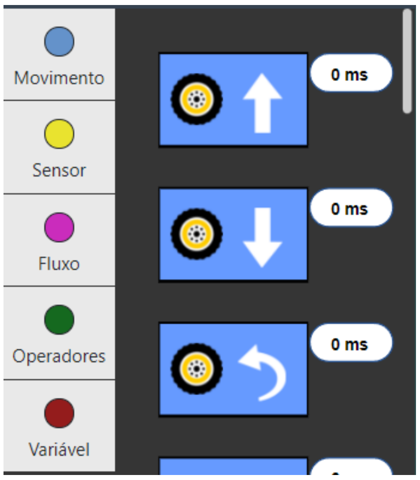


Figure 18: Categories existing in Robi platform

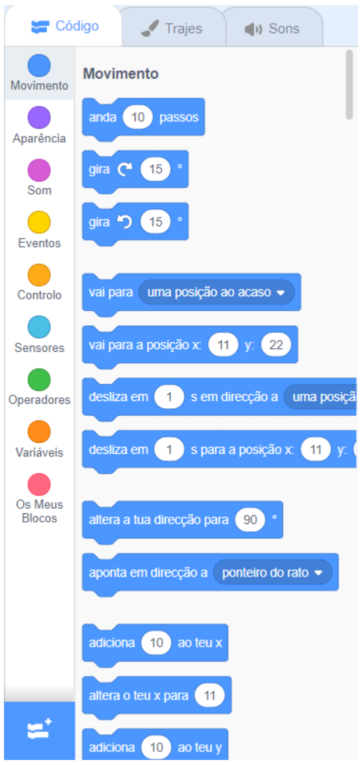

Código
Trajes
Sons
Movimento
Aparência
Som
Eventos
Controlo
Sensores
Operadores
Variáveis
Os Meus Blocos
Movimento
anda 10 passos
gira 15 °
gira 15 °
vai para uma posição ao acaso
vai para a posição x: 11 y: 22
desliza em 1 s em direcção a uma posiçã
desliza em 1 s para a posição x: 11 y:
altera a tua direcção para 90 °
aponta em direcção a ponteiro do rato
adiciona 10 ao teu x
altera o teu x para 11
adiciona 10 ao teu y

Figure 19: Categories existing in Scratch platform

While Robi contains 5 categories, Scratch contains 9 categories. With regard to the amount of blocks, Robi contains 21 blocks, while Scratch contains at least 107 blocks.

Since Scratch doesn't have any additional product, it uses a simulator at the right side of the system. Figure 20 shows the Scratch programming environment.

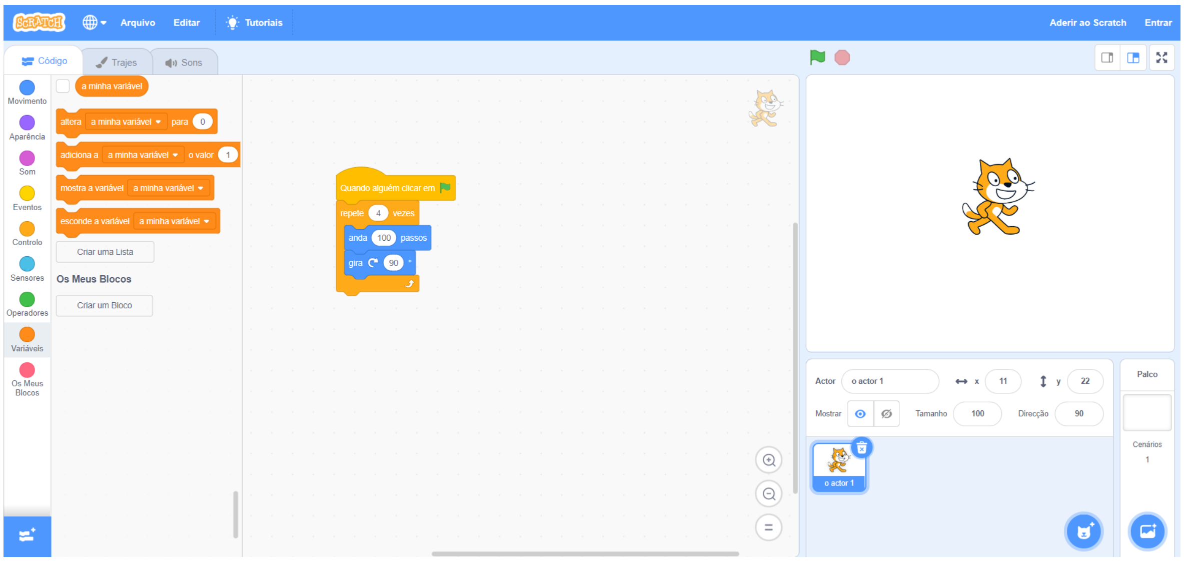


Figure 20: Scratch programming environment

Robi can provide a simpler interface, since it doesn't need a simulator to show the result of the implemented code, because Robi focus on programming a robot.

8

TESTING ROBI

Robi platform was tested by four children. Two of them are 7 years old, while one is 9 years old, and the other is 12 years old. Three exercises were proposed to them to implement in Robi platform. The exercises given to the children are the described in the following sections:

8.1 BASIC EXERCISE - MOVING ROBI ROBOT

The Basic Exercise has the objective of making the robot move forward, turn approximately 90 degrees, and then moving forward again. This exercise has the purpose of making the children get to know the basics of Robi environment, such as connecting blocks to each other, changing the input values from the movement blocks, and try-and-error to make the robot turns the approximately degrees asked in this exercise. The solution for this exercise is shown in the Figure 21.

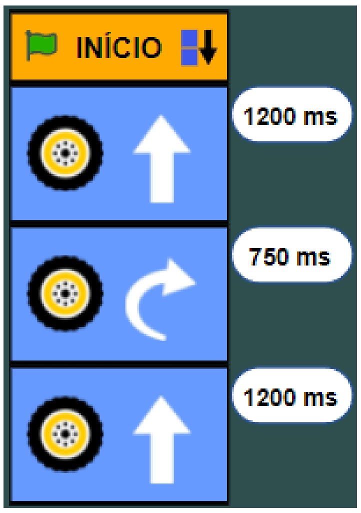


Figure 21: First exercise solution

The final code generated by the Compiler service, correspondent to the solution of the exercise here described, is shown in the Listing 8.1.

```
int variavel = 0;

void loop() {

moveForwardCustom(1200);
turnRight(750);
moveForwardCustom(1200);
moveStop();
delay(200);

while(1) {
    moveStop();
}

}
```

Listing 8.1: Code generated from solution of the first exercise

The code configuration that is automatically added by the Compiler service is omitted in the Listing 8.1. A 7 years old child, since he didn't have computer skills, had difficulties at connecting the blocks. The other children didn't have any problems at solving this exercise.

8.2 INTERMEDIATE EXERCISE - LOOPING THROUGH THE ROBOT MOVEMENT

The Intermediate Exercise consists in command the robot in order to repeat the movements of the Basic Exercise ten times. The purpose of this exercise is to make them know the importance of loops, and consequently, the advantage of programming instructions. The solution for this exercise is shown in the Figure 22.

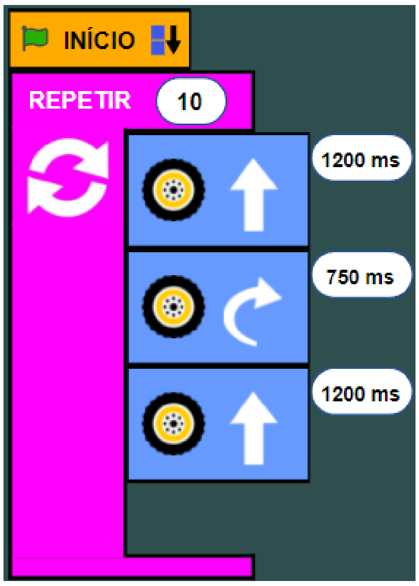


Figure 22: Second exercise solution

The final code generated by the Compiler service, correspondent to the solution of the exercise here described, is shown in the Listing 8.2.

```
int variavel = 0;

void loop() {

for (int i = 0; i < 10; i++) {
    moveForwardCustom(1200);
    turnRight(750);
    moveForwardCustom(1200);
    moveStop();
    delay(200);
}
moveStop();
delay(200);

while(1) {
    moveStop();
}

}
```

Listing 8.2: Code generated from solution of the second exercise

The code configuration that is automatically added by the Compiler service is omitted in the Listing 8.2. All the children tried adding the same blocks used in the solution from the first exercise ten times, as asked in the exercise. During the adding, they were told that they could use another block present in the system, that is the Repeat block.

8.3 RECTANGULAR MOVEMENT EXERCISE

The Rectangular Movement Exercise has the objective of making the robot to move in order to draw a rectangle. This exercise has the purpose of making the children use the knowledge acquired from the past exercises. The solution for this exercise is shown in the Figure 23

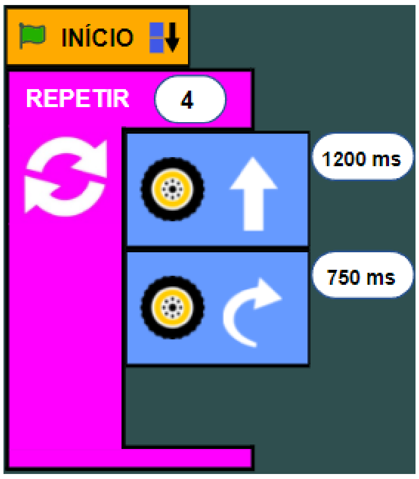


Figure 23: Third exercise solution

The final code generated by the Compiler service, correspondent to the solution of the exercise here described, is shown in the Listing 8.3.

```
int variavel = 0;

void loop() {

for (int i = 0; i < 4; i++) {
    moveForwardCustom(1200);
    turnRight(750);
    moveStop();
    delay(200);
}
moveStop();
delay(200);

while(1) {
    moveStop();
}

}
```

Listing 8.3: Code generated from solution of the third exercise

The code configuration that is automatically added by the Compiler service is omitted in the Listing 8.3. Although all the children were able to achieve the solution of this exercise, one of them, 7 years old, after implementing the solution, tried solving this exercise with an alternative approach. He added only the movement blocks, without using the Repeating block. He noticed that he was about to repeat the same values for the Movement blocks inputs, so he asked if there was a way of accelerating this action. That was an opportunity to introduce him the variable block. After the explanation, he came up the solution shown in the Figure 24.

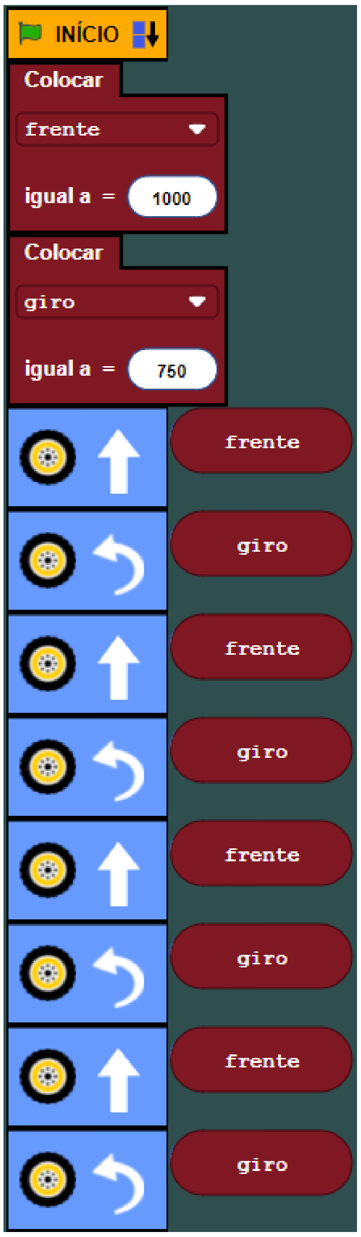


Figure 24: Exercise 3 with variables

The final code generated by the Compiler service, correspondent to the alternative solution of this exercise here described, is shown in the Listing 8.4.

```
int variavel = 0;
int frente = 0;
int giro = 0;

void loop() {

```

```
frente = 1000;
giro = 750;
moveForwardCustom((frente));
turnLeft((giro));
moveForwardCustom((frente));
turnLeft((giro));
moveForwardCustom((frente));
turnLeft((giro));
moveForwardCustom((frente));
turnLeft((giro));
moveStop();
delay(200);

while(1) {
    moveStop();
}
```

Listing 8.4: Alternative solution of the third exercise

The code configuration that is automatically added by the Compiler service is omitted in the Listing 8.4.

8.4 RESULT

All the children were able to find a solution for the basic exercise. One of the children, 7 years old, took more time than the others, since he didn't have practice using a computer. The other children were able to solve the exercise without difficulties-

Although the intermediate exercise was solved by all the children without any difficulties, none of them tried to look if there was another block that could simplify the script, so they delayed more than what is expected for this exercise. After explaining how loop works and after stating that throughout the blocks available in the platform there were blocks that could loop through the code implemented, they found and used the Repeat block. The 7 years old child that didn't have practice with computer delayed a little more than the others at trying to connect the Movement blocks inside the Repeat blocks.

The rectangular movement exercise was proposed in a way that the children could come up with different solutions. A 7 years old child, different from the one without computer practice, asked about a way of simplifying the act of writing the same values on all the blocks he added. So the variable concept was explained to him. The 9 years old child solved this exercise without the use of Repeat block, and and 12 years old child did use the Repeat block. Every child had their own solution. None of them delayed solving this exercise neither had difficulties finding a solution.

Based on the exercises given to the children, Robi environment proved to be an intuitive and simple environment. The children were able to learn the basics of educational robotics and consequently the programming itself. The children using the platform confirmed that the first proposed exercise showed that the language was developed intuitiveness and simplicity, since they could be able to identify the blocks responsible to move the robot, as well making it turn to a desired side, and use them in the script. The second and third proposed exercises were able to teach programming to them, as the children after finishing the exercises gained knowledge about variables, loops and the possibility to create different solutions for a single programming question.

9 CONCLUSION

Researches were made to decide if the creation of a new platform to support the educational robotics was necessary. After analyzing some platforms being used throughout the world, was noticed that the simplicity and intuitiveness could be improved. In that way, develop Robi language and programming visual platform would be important, not only to improve the simplicity and intuitiveness for a visual programming language, it is also an advantage regarding the flexibility to adapt or upgrade the system in any direction.

Since the aim of this Master's project was to develop a new programming language, the design of a grammar was necessary. The creation of a grammar has the purpose of supporting the development of the new programming language. The grammar rules and symbols helped with the decision of blocks that Robi system would have and the conditions that the system needed in order to connect the blocks.

The Arduino Uno is an open-source microcontroller board and since it is a well known component, it has a great amount of documentation and forums easily found throughout the internet. Besides the documentation, the Arduino Uno and the electronic components that can be connected to it, offer a cheap approach if the desire is to build a robot. Having knowledge of that, it was decided that the robot that Robi platform would be integrated with, would be the Arduino Uno and its chassis car kit. Arduino Uno can be programmed using the Arduino IDE, which uses the AVR-GCC compiler. In order to upload the code the user writes using the visual programming language, Robi makes use of that AVR-GCC compiler, so the system only needs to translate the code that will be sent to AVR-GCC compiler.

Robi interface was developed from scratch. All blocks were designed using the SVG technology, with the support of the Angular framework, which helped with the maintainability, readability and with the code reuse. The advantage of developing Robi from scratch is that the chosen approach provides flexibility to add, change, update or remove anything as the developer desires. Robi features require a flexible and powerful development, such as the visual interface reflecting the robot, or the interface that shall be as simpler as possible.

Different from the LEGO programming environment, Robi provides a web application solution, which opens opportunities to create a class management system or any other advantage that a cloud solution provides.

After concluding that Robi system would bring benefits as a new visual programming environment, the implementation was concluded after 7 months of hard work. Due to Covid and severe time constraints, it was not possible to gather a great number of people to test the system. So, the testing was done with 4 children, being two of them 7 years old. The system showed to be intuitive and simple as expected. After the exercises done in Robi environment, the children were asked to do the same in the Scratch environment. They had more difficulty to find out where to start the program in Scratch and felt the system less intuitive, to the fact of Scratch being less visual than Robi. For the children that tested Robi, they confirmed the intuitiveness in Robi environment intuitive, as well the simplicity. The experiment conducted was focused in the easiness of programming. In the future it is necessary to plan usability tests.f Robi platform is accessible through the following link: `https://robi.di.uminho.pt/`

9.1 FUTURE WORK

Robi has three main improvements that would bring even more benefits to the educational robotics lessons.

9.1.1 *Bluetooth Integration*

In order to upload the code to the robot, it is needed to connect a cable in the robot, and then click the correspondent button to send the code to the robot. Integrating a Bluetooth to the robot would avoid to always having taking the robot off the position he was located, and would improve the user experience. There is a good amount of documentation through the internet explaining how to integrate the HC-05 Bluetooth module to the Arduino.

9.1.2 *Improvements on Robi system*

The robot developed contains only one sensor, that is the ultrasonic sensor, responsible to return the distance in centimeters between the sensor and the closest obstacle. There is a high amount of sensors or modules that could be integrated in the Arduino based robot, such as color sensor, a sound emitter module, LED's to blink whenever the robot is done with the implementation given to it and more. As components are added to the robot, if necessary, the correspondent blocks should also be created and added to the system, so the user could make use of them.

9.1.3 *Class management*

Currently Robi has only a platform to program the robot. The improvement of the system to support the management of the students and the exercises they solved or needs to do would bring benefits such as the tracking of their learning improvements. Another benefit that this feature would bring is a path that the teacher could use to orient himself. He could decide the set of exercises suitable to the students he is teaching.

A

CODE SENT TO THE ROBOT

In this appendix is listed the code that represents the final code that is sent to the robot when the program presses the 'Enviar ao robô' button.

```
#include <Servo.h>
#include <NewPing.h>

const int LeftMotorForward = 5;
const int LeftMotorBackward = 4;
const int RightMotorForward = 3;
const int RightMotorBackward = 2;

#define trig_pin A1
#define echo_pin A2

#define maximum_distance 200

boolean goesForward = false;

int distance_from_ultrasonic_sensor = 100;

NewPing sonar(trig_pin, echo_pin, maximum_distance);
Servo servo_motor;

void setup() {
    pinMode(RightMotorForward, OUTPUT);
    pinMode(LeftMotorForward, OUTPUT);
    pinMode(LeftMotorBackward, OUTPUT);
    pinMode(RightMotorBackward, OUTPUT);
    servo_motor.attach(11);
    servo_motor.write(90);
    delay(2000);
    distance_from_ultrasonic_sensor = readPing();
    delay(100);
    distance_from_ultrasonic_sensor = readPing();
    delay(100);
    distance_from_ultrasonic_sensor = readPing();
    delay(100);
    distance_from_ultrasonic_sensor = readPing();
    delay(100);
}

int readPing() {
    delay(70);
    int cm = sonar.ping_cm();

    if (cm == 0) {
        cm = 250;
    }

    return cm;
}

void moveForwardCustom(int delayTime) {
    goesForward = false;
    moveForward();
    delay(delayTime);
    moveStop();
}

void moveStop() {
    digitalWrite(RightMotorForward, LOW);
    digitalWrite(LeftMotorForward, LOW);
    digitalWrite(RightMotorBackward, LOW);
    digitalWrite(LeftMotorBackward, LOW);
}

void moveBackwardsCustom(int delayTime) {
    moveBackward();
    delay(delayTime);
    moveStop();
}

void moveForward() {
    if (!goesForward) {
        goesForward = true;
        digitalWrite(LeftMotorForward, HIGH);
        digitalWrite(RightMotorForward, HIGH);
        digitalWrite(LeftMotorBackward, LOW);
        digitalWrite(RightMotorBackward, LOW);
    }
}

void moveBackward() {
    goesForward=false;
    digitalWrite(LeftMotorBackward, HIGH);
    digitalWrite(RightMotorBackward, HIGH);
    digitalWrite(LeftMotorForward, LOW);
    digitalWrite(RightMotorForward, LOW);
}

void turnRight(int delayTime) {
    digitalWrite(LeftMotorForward, HIGH);
    digitalWrite(RightMotorBackward, HIGH);
    digitalWrite(LeftMotorBackward, LOW);
    digitalWrite(RightMotorForward, LOW);
    delay(delayTime);
    digitalWrite(LeftMotorForward, HIGH);
    digitalWrite(RightMotorForward, HIGH);
    digitalWrite(LeftMotorBackward, LOW);
    digitalWrite(RightMotorBackward, LOW);
}

void turnLeft(int delayTime) {
    digitalWrite(LeftMotorBackward, HIGH);
    digitalWrite(RightMotorForward, HIGH);
    digitalWrite(LeftMotorForward, LOW);
    digitalWrite(RightMotorBackward, LOW);
    delay(delayTime);
    digitalWrite(LeftMotorForward, HIGH);
    digitalWrite(RightMotorForward, HIGH);
    digitalWrite(LeftMotorBackward, LOW);
    digitalWrite(RightMotorBackward, LOW);
}

int variavel = 0;
int distance = 0;

void loop() {

distance = 20;

for (int i = 0; i < 3; i++) {
    while (((readPing()) > (distance))) {
        moveForwardCustom(250);
        moveStop();
        delay(200);
    }

    turnRight(1200);
    moveStop();
    delay(200);
}

turnLeft(1200);
moveStop();
delay(200);

while(1) {
    moveStop();
}

}
```

Listing A.1: Final code that will be sent to the Robot

BIBLIOGRAPHY

Azad Ali and David Smith. Teaching an introductory programming language in a general education course. *Journal of Information Technology Education: Innovations in Practice*, 13 (6):57–67, 2014.

Cristiana Araújo, Lázaro Lima, and Pedro Rangel Henriques. An Ontology based approach to teach Computational Thinking. In Célio Gonçalo Marques, Isabel Pereira, and Diana Pérez, editors, *21st International Symposium on Computers in Education (SIIE)*, pages 1–6. IEEE Xplore, Nov 2019. ISBN 978-1-7281-3182-5. doi: https://doi.org/10.1109/SIIE48397.2019.8970131.

Yusuf Abdullahi Badamasi. The working principle of an arduino. In *2014 11th international conference on electronics, computer and computation (ICECCO)*, pages 1–4. IEEE, 2014.

Banggood. DIY L298N 2WD Kit Automóvel Rôbo Moteur Rastreamento Ultra-sônico para Arduino. https://pt.banggood.com/Geekcreit-DIY-L298N-2WD-Ultrasonic-Smart-Tracking-Moteur-Robot-Car-Kit-for-Arduino-pr html?cur_warehouse=CN, 2021. Accessed: 2021-12-16.

Neil CC Brown, Jens Mönig, Anthony Bau, and David Weintrop. Panel: Future directions of block-based programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 315–316, 2016.

Christina Chalmers. Robotics and computational thinking in primary school. *International Journal of Child-Computer Interaction*, 17:93–100, 2018.

Tomáš Effenberger and Radek Pelánek. Towards making block-based programming activities adaptive. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, pages 1–4, 2018.

Menna Elshahawy, Mariam Bakhaty, and Nada Sharaf. Developing computational thinking for children with autism using a serious game. *IEEE*, 2020.

M Faisal, Rosihan Yuana, and Mr Basori. Comparative study between robomind and scratch as programming assistance tool in improving understanding of the basic programming concepts. In *International Conference on Teacher Training and Education 2017 (ICTTE 2017)*. Atlantis Press, 2017.

Mohamed Fezari and Ali Al Dahoud. Integrated development environment "ide" for arduino. *WSN applications*, pages 1–12, 2018.

Carina Soledad González-González. State of the art in the teaching of computational thinking and programming in childhood education. *Education in the Knowledge Society*, 20:1–15, 2019.

Shuchi Grover and Satabdi Basu. Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, pages 267–272, 2017.

Shuchi Grover, Satabdi Basu, Marie Bienkowski, Michael Eagle, Nicholas Diana, and John Stamper. A framework for using hypothesis-driven approaches to support data-driven learning analytics in measuring computational thinking in block-based programming environments. *ACM Transactions on Computing Education (TOCE)*, 17(3):1–25, 2017.

Ting-Chia Hsu, Shao-Chen Chang, and Yu-Ting Hung. How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education*, 126:296–310, 2018.

Afiza Ismail, Nazlia Omar, and Abdullah Mohd Zin. Developing learning software for children with learning disabilities through block-based development approach. In *2009 International Conference on Electrical Engineering and Informatics*, volume 1, pages 299–303. IEEE, 2009.

Asha K Jitendra, George J DuPaul, Fumio Someki, and Katy E Tresco. Enhancing academic achievement for children with attention-deficit hyperactivity disorder: Evidence from school-based intervention research. *Developmental disabilities research reviews*, 14(4): 325–330, 2008.

Luiz A Junior, Osvaldo T Neto, Marli F Hernandez, Paulo S Martins, Leonardo L Roger, and Fatima A Guerra. A low-cost and simple arduino-based educational robotics kit. *Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Robotics and Control (JSRC), December edition*, 3(12):1–7, 2013.

Eija Karna-Lin, Kaisa Pihlainen-Bednarik, Erkki Sutinen, and Marjo Virnes. Can robots teach? preliminary results on educational robotics in special education. In *Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06)*, pages 319–321. IEEE, 2006.

Özgen Korkmaz. The effect of lego mindstorms ev3 based design activities on students' attitudes towards learning computer programming, self-efficacy beliefs and levels of academic achievement. *Online Submission*, 4(4):994–1007, 2016.

Özgen Korkmaz. The effect of scratch-and lego mindstorms ev3-based programming activities on academic achievement, problem-solving skills and logical-mathematical thinking skills of students. *MOJES: Malaysian Online Journal of Educational Sciences*, 4(3): 73–88, 2018.

Jie Liu, Hayden Wimmer, and Roy Rada. " hour of code": Can it change students' attitudes toward programming? *Journal of Information Technology Education: Innovations in Practice*, 15:53, 2016.

Leo Louis. working principle of arduino and u sing it. *International Journal of Control, Automation, Communication and Systems (IJCACS)*, 1(2):21–29, 2016.

Matthew B MacLaurin. The design of kodu: A tiny visual programming language for children on the xbox 360. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 241–246, 2011.

John H Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. Programming by choice: urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 367–371, 2008.

Yoshiaki Matsuzawa, Yoshiki Tanaka, and Sanshiro Sakai. Measuring an impact of block-based language in introductory programming. In *International Conference on Stakeholders and Information Technology in Education*, pages 16–25. Springer, 2016.

Roberto Munoz, Rodolfo Villarroel, Thiago S Barcelos, Fabián Riquelme, Angeles Quezada, and Patricia Bustos-Valenzuela. Developing computational thinking skills in adolescents with autism spectrum disorder through digital game programming. *IEEE Access*, 6: 63880–63889, 2018.

Jelena Pisarov and Gyula Mester. Programming the mbot robot in school. In *MechEdu Conference & Workshop*, pages 1–4, 2019.

Marcos Román-González, Juan-Carlos Pérez-González, Jesús Moreno-León, and Gregorio Robles. Extending the nomological network of computational thinking with non-cognitive factors. *Computers in Human Behavior*, 80:441–459, 2018.

Kathryn T Stolee and Teale Fristoe. Expressing computer science concepts through kodu game lab. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 99–104, 2011.

Alexander Svendsen. Robomind. a platform for on-the-fly programming and inspection of behavior-based robot programs. Master's thesis, UiT Norges arktiske universitet, 2014.

Xiaodan Tang, Yue Yin, Qiao Lin, Roxana Hadad, and Xiaoming Zhai. Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, 148: 103798, 2020.

MERT Arduino & Tech. How to make Arduino Obstacle Avoiding Robot Car | Under $20. `https://www.youtube.com/watch?v=4CFO0MiSlM8&ab_channel=MERTArduino%26Tech`, 2018. Accessed: 2021-12-16.

Salete Teixeira, Diana Barbosa, Cristiana Araújo, and Pedro Rangel Henriques. Improving Game-Based Learning Experience Through Game Appropriation. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *OpenAccess Series in Informatics (OASIcs)*, pages 27:1–27:10, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. ISBN 978-3-95977-153-5. doi: 10.4230/OASIcs.ICPEC.2020.27. URL `https://drops.dagstuhl.de/opus/volltexte/2020/12314`.

Marjo Virnes, Erkki Sutinen, and Eija Kärnä-Lin. How children's individual needs challenge the design of educational robotics. In *Proceedings of the 7th international conference on Interaction design and children*, pages 274–281, 2008.

David Weintrop. Block-based programming in computer science education. *Communications of the ACM*, 62(8):22–25, 2019.

David Weintrop and Uri Wilensky. Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–25, 2017.

Rosihan Ari Yuana and Dwi Maryono. Robomind utilization to improve student motivation and concept in learning programming. In *Proceeding of International Conference on Teacher Training and Education*, volume 1, pages 962–966, 2016.