The Visual Programming Environment ROBI for Educational Robotics

Gustavo Galvão
Centro ALGORITMI, Departamento de Informática, Universidade do Minho, Braga, Portugal

Alvaro Costa Neto
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Barretos, Brazil

Cristiana Araújo
Centro ALGORITMI, Departamento de Informática, Universidade do Minho, Braga, Portugal

Pedro Rangel Henriques
Centro ALGORITMI, Departamento de Informática, Universidade do Minho, Braga, Portugal

Abstract

This paper presents the outcomes of a research project focused on the training of Computational Thinking, resorting to a block-based visual programming language created to program an Arduino Uno based robot. To support the design and implementation of the visual programming environment Robi, we start discussing the relevance of Educational Robotics to motivate and engage children in programming activities. Students usually face great difficulties to learn computer programming and it is nowadays accepted that young people shall be trained in Computational Thinking to acquire the skills necessary to easily solve problems within and beyond the realm of Computer Science and Engineering. The resolution of obstacles imposed by the costs and reduced availability of typical Educational Robotics kits, in combination with the benefits of existing block-based programming languages, like simplicity and intuitiveness, motivated the project here reported and analyzed. We aim at showing that Robi, a visual block-based programming language and robot programming environment, provides an easy, accessible and intuitive platform to learn how to solve problems programming a computer and support the training of Computational Thinking.

2012 ACM Subject Classification Software and its engineering → Compilers; Social and professional topics → Computing education

Keywords and phrases Programming Languages, Visual Languages, Computer Programming, Educational Robotics

Digital Object Identifier 10.4230/OASIcs.SLATE.2022.14

Supplementary Material *Software (Web Application)*: `https://robi.di.uminho.pt/`

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Units Project Scope: UIDB/00319/2020.

1 Introduction

Computational Thinking was presented by Wing in 2006, stating that it is a method for solving problems, or designing systems and understanding human behavior, based on the fundamental concepts of Computer Science [25]. According to Wing, training in Computational Thinking is not just for Computer Scientists but for all people, and should be applied from an early age. Computational Thinking promotes the development of problem solving strategies, logical reasoning, problem decomposition, knowledge relationship, abstract thinking, among others [25]. It is believed that if children develop these skills from an early age, they will be able to solve problems more easily, whether in mathematics, physical chemistry, computer science or even in their daily lives.

One way to develop Computational Thinking skills is to solve problems using block-based programming environments. By using these environments, in addition to solving problems, children also acquire Computer Programming abilities. Block-based programming environments leverage a programming primitive-as-puzzle-piece metaphor that provides visual cues to the user on how and where commands can be used as their means of constraining program composition. This type of visual programming consists of dragging blocks into a scripting area, and fitting them together to form scripts [24]. Some examples of block-based programming environments platforms used to introduce programming are: *Scratch*[1], *mBlock*[2], *VEXcode VR*[3], *Alice*[4]. They provide a fun and engaging introduction to programming concepts without the need to deal with complex syntactic constructs, commonly considered obstacles to students that recently started learning text-based programming [14]. The *Scratch* and *mBlock* Programming environments allow integration with robot kits, such as *LEGO Mindstorms* and *mBot - Makeblock*, respectively.

The integration with robot kits provides more motivation and greater student engagement. However, despite the benefits of block-based programming environments, their integration with robot kits is absent or too expensive and complex for younger children.

Motivated by the benefits that robotics tools bring, which lead students to acquire not only Computational Thinking skills but also other necessary skills such as collaboration, cognitive capabilities, self-confidence, spatial perception and understanding [18], among others, we decided to develop a new block-based programming language integrated with an Arduino Uno[5].

Arduino Uno is an open source micro-controller that can be easily programmed and reprogrammed at any time. The Arduino platform was launched in 2005 for hobbyists, students or professionals to create devices that interact with their environments using sensors and actuators in a simple and low-cost way [19]. One of the reasons why the Arduino platform is a low-cost product is due to the independence of the suppliers of parts and components. This is in contrast to robot kits like *LEGO*, *Makeblock* and other manufacturers, which have their own standard components, making them more expensive [17]. The programming language created aims to be simple, intuitive and as iconic as possible.

This article has six sections. In Section 2, Computational Thinking and Educational Robotics are presented. An overview of Robi Environment is provided in Section 3. Section 4 presents the visual programming language of Robi Environment. Section 5 discusses how to use robot Robi and make it to work controlled by our programming environment. Section 6 presents the system assessment. Finally, in Section 7, conclusions and future work are discussed.

2 Computational Thinking and Educational Robotics

Advances in technology made computing devices present in almost every aspect of daily lives. Essential to this development, Computer Programming presents itself both as a main topic and as one in which students face several difficulties to properly learn. By leveraging Computational Thinking (CT) it is possible to better prepare younger students and consequently improve their academic success later in life [22]. Being an important subject

[1] Accessible at: `https://scratch.mit.edu/`
[2] Accessible at: `https://mblock.makeblock.com/en-us/`
[3] Accessible at: `https://vr.vex.com/`
[4] Accessible at: `https://www.alice.org/`
[5] Accessible at: `https://store.arduino.cc/`

of national education programs in many countries, to the extent of receiving classifications as national objectives [16], CT is recognized as a fundamental competency for success in the fields of Science, Technology, Engineering and Mathematics (STEM).

The International Society for Technology in Education & Computer Science Teachers Association [10], define Computational Thinking as a process of problem solving that includes the following characteristics: (1) formulate problems in a way that allows us to use a computer or similar to help solve them; (2) logically organize and analyze the data; (3) represent the data through abstractions such as models and simulations; (4) automate solutions through algorithmic thinking; (5) identify, analyze and implement possible solutions more effective; and (6) generalize and transfer this process to a wide variety of problems.

Teaching CT from an early age benefits students not only to overcome difficulties in programming courses, but also to improve skills that can be crucial in other areas, such as the development of critical thinking, abstract thinking, logical reasoning, problem solving strategies and persistence [15, 1]. To this intent, block-based programming environments are commonly used to keep children both interested and motivated when learning computer programming. As an example, the Hour of Code campaign, which provides online introductory programming activities, reaches millions of students through the use of several tools, including visual block-based programming environments such as Scratch [8].

A resource that has stood out to train students in Computational Thinking is Educational Robotics. Educational Robotics is a powerful and flexible teaching and learning tool, encouraging students to build and control robots using specific programming languages. The robot is a tangible object that students can interact with [9]. Recent reviews of the literature on Educational Robotics in the mandatory stages distinguish its potential for training CT and understanding concepts related to STEM areas [7, 12, 6, 2, 3, 4]. The use of robots in the classroom, in addition to being motivating, allows the design of activities that promote both CT and skills related to scientific and mathematical skills. In addition, it also enhances the development of skills: cognitive, digital, social, collaborative and teamwork, creativity [13, 5, 3, 4].

3 Robi, the Environment

Robi environment is composed of two web applications: one in the cloud, and one local to the user's computer[6]. The cloud-based web application is responsible for providing the visual programming environment to the user and for providing downloads to the local web application. The local web application contains a compiler service, that translates the intermediary code developed translated by the cloud-based web application to an Arduino compatible binary using the AVRDUDE tool, the same used by the official Arduino IDE. Figure 1 shows the system architecture as explained above.

3.1 System Requirements and Implementation Details

Some requirements were identified for the proper design of the visual programming language, as well as to develop the system architecture, both cloud-based and local. These included: an IDE for programming Robi using the block-based language, the hardware upload mechanism to send the downloaded script to an Arduino Uno based robot, and the definition of both the block-based language and its intermediary textual representation.

[6] The official Robi website may be found at the following address: `https://robi.di.uminho.pt/`

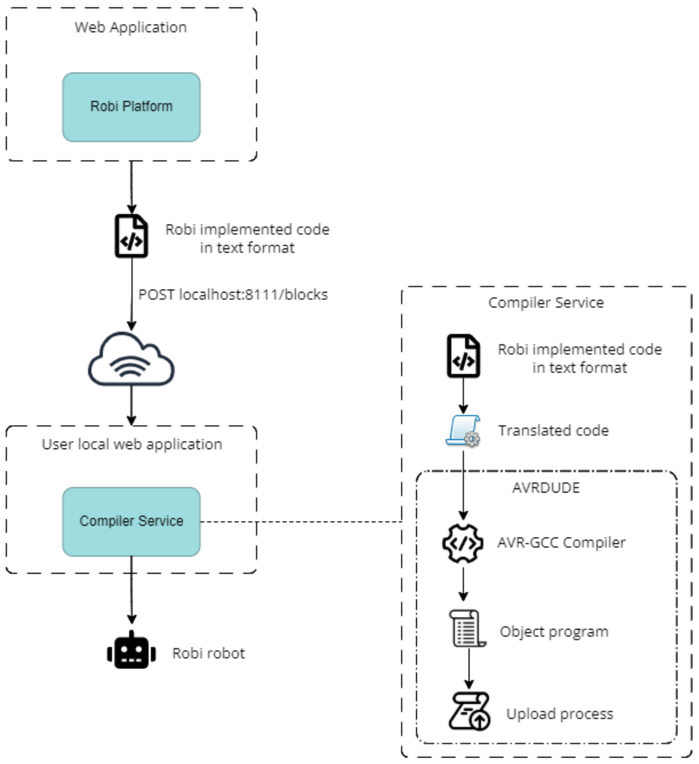


Figure 1 System architecture of the Robi platform.

Among the many languages, packages and tools that could be used to develop this project, given the authors' experiences in similar projects, the following were chosen:

- Angular for the development of the front-end;
- Java, Gradle and Spring boot for the development of the back-end;
- JUnit, WireMock and RestAssured for testing purposes;
- Spring Rest Docs and Asciidoctor for documentation;
- AVRDUDE for the final compilation and upload to the robot's hardware.

4 Robi, the Visual Programming Language

Robi programming language was designed with the purpose of programming an Arduino-based robot using only blocks that represent desired robot actions. For this reason, this language is mainly composed of robot movement and sensor blocks. Besides these robot-oriented blocks, Robi also contains classic programming-oriented blocks, such as repetition and conditional structures, math operators, variable declaration and assignment, and a delay operation. The intent was to represent the robot behavior programming in a high level language, directly mapping the actions that users would expect to be executed by the robot itself.

While there would be advantages in adopting a known visual language, such as Blockly[7], it was decided that Robi should have a visual identification specifically designed to its purposes. This decision was made based on the fact that Robi's language is directly aimed at controlling the robot and trying to reduce a general purpose language to fit its specifications could lead to a series of adaptation problems.

[7] Accessible at: `https://developers.google.com/blockly/`

Listing 1 Formal definition of the Robi Programming Language.

```
Program          : Variables INIT Block
Variables        : & | Variables VARIABLE
Block            : Instruction | Block Instruction
Instruction      : Condition | Assignment | Loop | MoveOp Expression
Conditional      : IF Condition THEN "[" Block "]"
                 | IF Condition THEN "[" Block "]" ELSE "[" Block "]"
MoveOp           : FORWARD | BACKWARDS | TURN_LEFT | TURN_RIGHT
Loop             : DO Expression TIMES "[" Block "]"
                 | DO WHILE Condition "[" Block "]"
Assignment       : UPDATE VARIABLE TO Expression
Condition        : Expression RelationalOp Expression
                 | Condition LogicOp Condition
LogicOp          : AND | OR
RelationalOp     : GREATER_THAN | LESS_THAN | EQUALS
Expression       : NUMBER | VARIABLE | SENSOR
                 | Expression MathOp Expression
MathOp           : PLUS | MINUS | TIMES | DIVISION
```

4.1 Formal Definition

Being an actual programming language (albeit a domain-specific one), the Robi programming language was formally defined using a list of terminal symbols and a Backus-Naur Form (BNF) description of its grammar. The terminal symbols mainly represent the blocks available to the user: `INIT`, `VARIABLE`, `IF`, `THEN`, `ELSE`, `FORWARD`, `BACKWARDS`, `TURN_LEFT`, `TURN_RIGHT`, `DO`, `TIMES`, `WHILE`, `UPDATE`, `TO`, `AND`, `OR`, `GREATER_THAN`, `LESS_THAN`, `EQUALS`, `NUMBER`, `SENSOR`, `PLUS`, `MINUS`, `TIMES` and `DIVISION`.

A program is formally defined by the first rule (for symbol 'Program') in Listing 1. The `INIT` symbol corresponds to the *Init* block, always present in the visual code. The other lines, after the first one referred, describe the rest of the grammar rules that define Robi Language (see Listing 1). The symbols, both terminal and non-terminal, are self-explanatory as to their uses. The rules form a lambda complete language that allows for the construction of entire programs that correctly control the robot with known statements, such as variable declaration, conditional and repetition structures and mathematical and logical operators.

Listing 2 presents an example of a program formally derived from the formal grammar to move the robot in a specific pattern: move forward until an obstacle is detected, then turn right; repeat three times. Listing 2 resembles an ordinary source code written in a high level programming language, with the use of variables, execution flow statements and expressions. While it wasn't the purpose to have users to program directly using the formal definition of Robi's language, it was important to construct this definition in order to formally validate the structure of the visual language, which follows the same rules. The visual programming interface for the Robi environment and the block-based programming language equivalent of the example described in Listing 2 are shown in Figure 2. The interface itself and the graphic symbols (blocks) used to represent the language statements make heavy use of iconography, in order to facilitate the understanding and use of Robi by younger children.

The blocks are divided into five categories. The visual programming interface aims to facilitate its use by grouping blocks through similar colors (which may be seen on the left side of Figure 2). This technique provides visual cues to children as to what the block represents in a general sense. The five categories for the blocks are:

Listing 2 Example of a concrete program formally written in Robi's formal language.

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

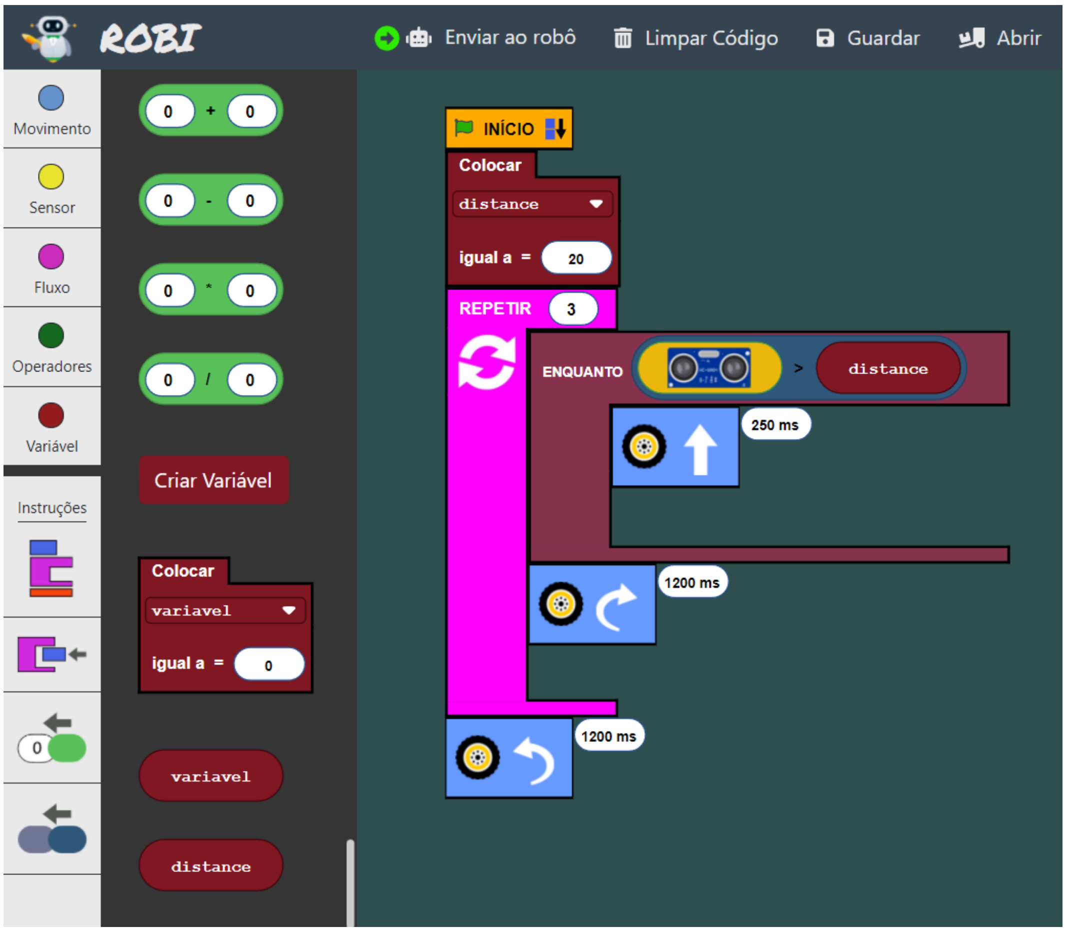


Figure 2 Program created in the Robi visual programming interface, equivalent to Listing 2.

Movement contains blocks for controlling the movement of the robot, such as moving forward or turning right (blue boxes with wheels in Figure 2);

Sensor the only block present in this category reads data from the ultrasonic sensor (yellow pill shaped block in Figure 2);

Flow controls program execution flow with encompassing blocks, such as the pink and dark rose containers in Figure 2;

Operators contains blocks for common mathematical, relational and logical operators, such as the dark blue pill shaped block in Figure 2;

Variables deal with declaration and assignment of variables, which may be named with a limit of twelve letter or digits (red blocks in Figure 2).

The interface works via drag and drop operations to add blocks to the programming are, while connections are made between blocks by snapping actions whenever they are in the proximity. In order to remove a block, a simple drag and drop operation from the programming area back to the list of blocks is necessary.

5 Robi, the Robot

The intention behind building a robot and connecting it to the programming interface is to provide children with a physical representation of the outputs that their programs actually produce. This tactile, real-world feedback is crucial in the learning process for younger children and greatly improves understanding for older students, specially when mediated by an instructor or teacher [20, 23].

5.1 Components and the Platform

As said above, the robot was built with on an Arduino Uno platform, which not only has extensive on-line documentation, but also enables the easier creation of electronic and interactive objects. The Arduino ecosystem also contains a high amount of modules and sensors that can be easily integrated. The components used to build the robot are available in the official Robi website.

The set of electronic and mechanic components was purchased as a low-cost kit in a wholesale website both for economic and logistic reasons. By basing the robot around a readily available kit instead of building a custom platform, the whole project becomes more affordable and easier to complete, since it may be bought worldwide. Any compatible kit would also be suitable to use, even a custom built, as long as the same connections and configurations are made to the individual components or the low-level definitions are adapted to the new platform.

Figure 3, copied from [21], corresponds to the circuit diagram used as basis to create the robot electronics and hardware. The robot uses two different batteries: one set of four 1.5V batteries for the motors and a 9V battery for the Arduino Uno. The platform is robust, easy to build, and works almost immediately. One caveat of its current construction is the precision loss of simple DC motors, which could be improved using stepper or servo motors. This choice was made to simplify construction and lower the total cost of the robot, since more precise motors would probably be the most expensive part of the build.

The main component of the robot system is the Arduino Uno which receives the program from the computer and controls the rest of the circuit. The DC motor driver board is connected to both motors and the Arduino Uno (GPIO pins 2, 3, 4 and 5) and allows the rotation of each motor in both directions independently. The ultrasonic sensor is connected

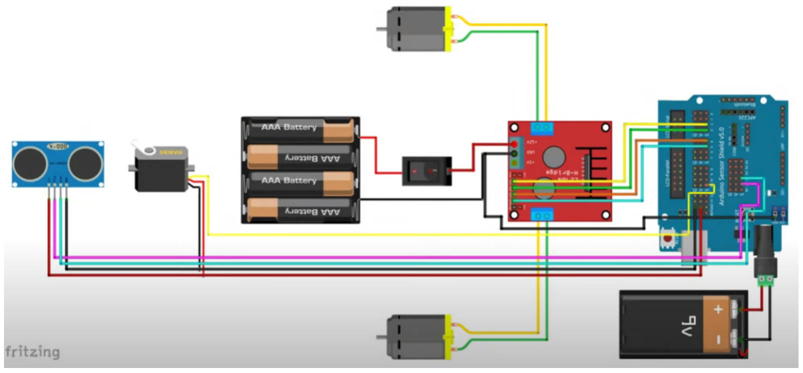


Figure 3 Electronic connections diagram for the Robi robot.

directly to the Arduino Uno through ports A1 and A2. These components and ports definitions are important for the configuration code that is used by the Robi environment to program the robot behavior.

5.2 Code Setup

There is a predefined configuration code for the robot to work as planned. In that code, some basic system initialization is done (Arduino ports are defined to connect the motors and the ultrasonic sensor, libraries are imported, and the ultrasonic sensor is calibrated) and some support functions are defined. Listing 3 presents the code that Robi adds to any program to bootstrap the Arduino system initialization and avoid having users worried about this part of the process.

In order to aid the translation process from the block-based programming language to standard Arduino source code, eight support functions are also predefined:

`readPing` returns a ping from the ultrasonic sensor measured in centimeters;
`moveStop` stops any movement;
`moveForward` moves forward indefinitely;
`moveBackwards` moves backwards indefinitely;
`moveForwardCustom` moves forward for a specific interval measured in milliseconds;
`moveBackwardsCustom` moves backwards for a specific interval measured in milliseconds;
`turnLeft` turns left for a specific interval measured in milliseconds;
`turnRight` turns right for a specific interval measured in milliseconds.

These functions assume a similar role to an instruction set of a classic target architecture in a compilation process. They are used in the translation service as basis for the intermediary textual representation of the block-based code. When compiled, some visual language blocks are translated to calls to these functions inside the main `loop` function of the Arduino platform, while others are translated to actual Processing directives, such as conditional and repetition statements.

5.3 Translation and Upload

The translation from the block-based code composed in the visual programming interface to the final binary instructions that can be uploaded to the the robot follows three steps:

Listing 3 Basic Arduino configuration code.

```
#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(LeftMotorForward,   OUTPUT);
    pinMode(LeftMotorBackward,  OUTPUT);
    pinMode(RightMotorForward,  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);
}
```

1. The block-based visual code is evaluated in the cloud-based web application and translated into a JSON object – the equivalent of an intermediary representation in classic compilation terms – which is then sent to the local web application;
2. The local web application receives the JSON object (see Listing 4 for an example) and translates it into Arduino compatible source code by mapping the action nodes to either equivalent statements or predefined support functions. It then aggregates it to the configuration source code shown in Subsection 5.2 for final processing;
3. The local web application calls its compiler service to compile the Arduino compatible source code from the previous step into bytecode using the AVRDUDE tools. The compiler service then uploads the bytecode to the robot's Arduino Uno via USB.

With regards to the JSON object, every block represents an action. That action is the identifier that the compiler service uses to identify the original visual block and then map it to the correct code translation. If the action corresponds to one of the predefined support

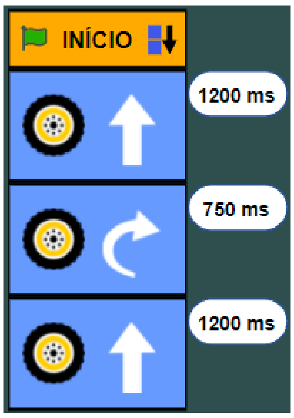
Listing 4 JSON object for the example in Figure 2.

```
"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"]
```

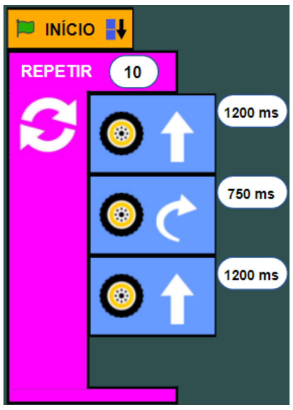
functions listed in Subsection 5.2 the resulting mapping is just a call to that function. If, otherwise, the block represents another kind of statement (flow control structures, variable declaration *etc.*) than an appropriate snippet of code is mapped, taking into account necessary grammatical analysis of expressions. The last two lines of the JSON object represent the communication port used to upload the bytecode to the Arduino Uno and the list of variables used in the program.

6 Robi, Assessment

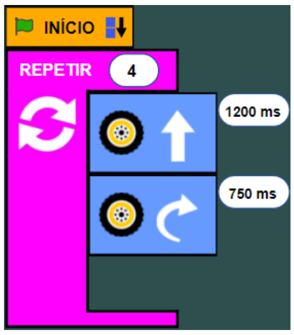
Robi's platform was tested by four children: two 7 years old, one 9 years old, and one 12 years old. Three exercises were proposed to be implemented by them in the Robi platform:



(a) First exercise solution.



(b) Second exercise solution.



(c) Third exercise solution.

Figure 4 Expected solutions for the three assessment exercises.

1. An introductory and tutorial exercise. The students should program the robot to move forward, turn approximately 90 degrees, and then move forward again. The purpose of this exercise was to introduce the basic concepts of block-based programming and the Robi visual programming interface. The expected solution for this exercise is presented in Figure 4a;
2. An intermediate exercise. The students should build upon the previous solution and order the robot to repeat the first exercise movement ten times. It was intended to teach students the importance of constructive and progressive learning, while introducing the concept of loops. The expected solution for the second exercise is shown in Figure 4b;
3. A final and more advanced exercise. It was requested from the students to teach the robot to draw a rectangle (or square). While it seems very similar to the previous exercise, considering only its solution, this exercise was described in a way to force the children to reason about the real world problem (moving the robot to draw something specific) and think about strategies to transform their solutions into software. The expected solution for the final exercise is presented in Figure 4c.

All four children were able to find a solution for the first exercise. One of the children (7 years old) took more time than the others, since he did not have practice using a computer. This fact reinforces the need for taking students background into account when teaching any kind of subject [11]. The other three children were able to solve the exercise without major difficulties.

Although the second exercise was solved by all children without any difficulties, none of them tried to search for a block that could simplify the script that they did not use in the first exercise. After explaining how loop works and showing that among the available blocks in the platform there were a few that could repeat a portion of code, they found and used the *Repeat* block. The 7 years old child that did not have practice with computer took a little more than the others at trying to connect the *Movement* blocks inside the *Repeat* blocks, meaning that the use of a computer graphical interface may pose a substantial obstacles to children that are not used to computers.

The last exercise was proposed in a way that the children had the possibility to come up with different solutions. A 7 years old child, different from the one without computer practice, asked about a way to simplify the act of writing the same values on all the blocks he added. The concept of variables was explained to him and he came up with the solution presented in

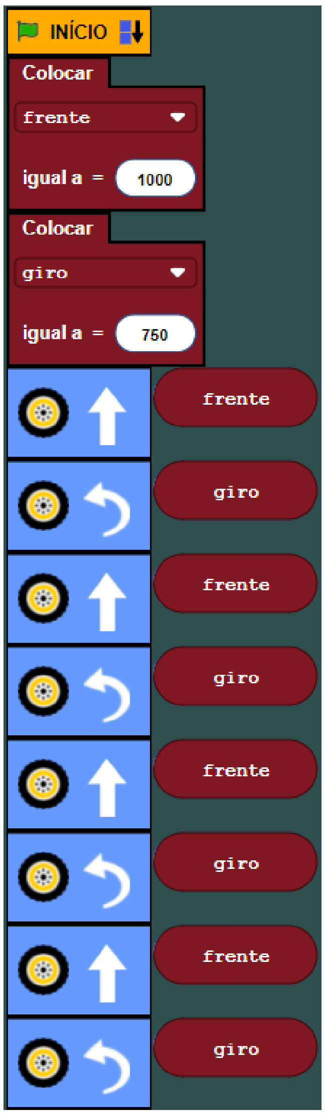


Figure 5 Third exercise alternative solution.

Figure 5. The 9 years old child also solved this exercise without the use of the *Repeat* block, and the 12 years old child did use the *Repeat* block implying that more complex subjects such as repetition structures may pose a heavier cognitive burden on younger children. Every child had his or her own solution and none of them delayed solving this exercise neither had major difficulties finding a solution.

Based on the solutions and feedback given by the children, the Robi environment seemed to be intuitive and simple to use. The children were able to learn the basics of educational robotics and consequently the main programming concepts themselves, while having fun in using and teaching a robot to drive around.

7 Conclusion

A research work based on literature review evaluated if a new platform to support Educational Robotics would contribute to the current state of the art. After analysing some platforms in use throughout the world, it was noticed that simplicity, intuitiveness of use, and cost-effectiveness could be improved. We then concluded that developing Robi's language and programming visual platform was worthwhile, both to increase the user-friendliness of the platform and the ease of controlling a cheap but effective Arduino-based robot, as well as to secure the flexibility to adapt or upgrade the system in any direction.

Since the aim of this project was to develop a new programming language, the design of a grammar was necessary to formally guide the process. 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.

We decided to integrate the Robi platform with the Arduino Uno and its chassis car kit, because the Arduino Uno is a well-known open source microcontroller board and has a great amount of documentation and forums easily found all over 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. Arduino Uno can be programmed using the Arduino IDE, which uses the AVR-GCC compiler and the AVRDUDE tool. In order to compile the code the user writes using the visual programming language, Robi translates the visual script to adequate Processing source code and makes use of that AVR-GCC compiler. After the compilation is done, the bytecode is uploaded through AVRDUDE. It was decided early on that there would be no simulators for this project, because it would largely increase the project's scope and time requirements, and other similar systems already provide simulation tools.

Robi's 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, in the future, a class management system or any other advantage that a cloud solution provides.

The assessment of the Robi platform was carried out with 4 children, 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. Children said they found it more difficult to find out where to start the program in Scratch and felt the system was less intuitive, due Scratch being less visual and iconic than Robi. At the end of the platform assessment, the children confirmed that Robi was a very intuitive and simple environment to work with.

As future work, we consider that it is necessary to design and to carry out an experiment with a larger number of children and to create and apply questionnaires to evaluate the experiment. Since the experiment carried out was focused on ease of programming, it is important in the future to design formal usability tests. Another important aspect is to integrate Bluetooth in the Robot to make it more convenient and practical to load the code to the Robot (this avoids the cable connection). In addition, other sensors or modules must be integrated (example: color sensor, sound emitting module, display module, among others), so that the Robot is more complete and has more potential. Currently, Robi only has got the platform to program the Robot. We also consider relevant to develop in the future a support system for managing students and the exercises they solved to monitor their progress and identify their difficulties.

References

1 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, November 2019. doi:10.1109/SIIE48397.2019.8970131.

2 Soumela Atmatzidou and Stavros Demetriadis. Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, 75:661–670, 2016. doi:10.1016/j.robot.2015.10.008.

3 Nicholas Alexander Bascou and Muhsin Menekse. Robotics in k-12 formal and informal learning environments: A review of literature. In *2016 ASEE Annual Conference & Exposition*, New Orleans, Louisiana, June 2016. ASEE Conferences. https://peer.asee.org/26119. doi:10.18260/p.26119.

4 Fabiane Barreto Vavassori Benitti. Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education*, 58(3):978–988, 2012. doi:10.1016/j.compedu.2011.10.006.

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

6 Guanhua Chen, Ji Shen, Lauren Barth-Cohen, Shiyan Jiang, Xiaoting Huang, and Moataz Eltoukhy. Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education*, 109:162–175, 2017. doi:10.1016/j.compedu.2017.03.001.

7 Morgane Chevalier, Christian Giang, Alberto Piatti, and Francesco Mondada. Fostering computational thinking through educational robotics: a model for creative computational problem solving. *International Journal of STEM Education*, 7, August 2020. doi:10.1186/s40594-020-00238-z.

8 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.

9 Francesc Esteve, Jordi Adell, Mª Ángeles Llopis, Gracia Valdeolivas, and Julio Pacheco. The development of computational thinking in student teachers through an intervention with educational robotics. *Journal of Information Technology Education: Innovations in Practice*, 18:139–152, October 2019. doi:10.28945/4442.

10 International Society for Technology in Education (ISTE) & Computer Science Teachers Association (CSTA). Operational definition of computational thinking for K-12 education. http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf, 2011. Accessed: 2021-12-16.

11 P. Freire. *Pedagogia da Autonomia: Saberes necessários à prática educativa.* Paz e Terra, 2011.

12 Anaclara Gerosa, Víctor Koleszar, Leonel Gómez-Sena, Gonzalo Tejera, and Alejandra Carboni. Educational robotics and computational thinking development in preschool. In *2019 XIV Latin American Conference on Learning Technologies (LACLO)*, pages 226–230, 2019. doi:10.1109/LACLO49268.2019.00046.

13 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.

14 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.

15 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.

16 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.

17 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.

18 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.

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

20 J. Piaget, M. Piercy, and D.E. Berlyne. *The Psychology of Intelligence*. Routledge classics. Routledge, 2001.

21 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.

22 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. `doi:10.4230/OASIcs.ICPEC.2020.27`.

23 L.S. Vygotsky, E. Hanfmann, G. Vakar, and A. Kozulin. *Thought and Language*. The MIT Press. MIT Press, 2012.

24 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.

25 Jeannette M. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, March 2006. `doi:10.1145/1118178.1118215`.