FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Tactode Programming for Robotics and Other Targets

César Alexandre da Costa Pinho

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Armando Jorge Miranda de Sousa

July 21, 2021

Tactode Programming for Robotics and Other Targets

César Alexandre da Costa Pinho

Mestrado Integrado em Engenharia Informática e Computação

July 21, 2021

Abstract

The demand for graduates in Science, Technology, Engineering, and Mathematics (STEM) areas is currently quite visible. Thus, educational curricula are being adjusted. In schools, subjects are being introduced to develop more logical techniques and reasoning, the so-called Computational Thinking (CT).

The original version of Tactode is a tangible block programming system aimed at children. It focuses on multi-target programming (robots, simulations, or high-level language programs) by creating behaviors/programs with tangible pieces that fit like a puzzle. Then a picture is taken of the set of grouped pieces, and it is introduced into the Tactode system application so that the program can be compiled (transposed) into the target application. If a user desires, the Tactode application also allows the program's creation by assembling the parts in the application but with many limitations.

Tactode allows the compilation of the program for some physical platforms, but manufacturers' applications are required to run the program, and no debug information at all in the Tactode app. These additional steps cause a low-quality User Experience (UX) in the child, making learning difficult.

O Scratch is a visual and block programming tool, top-rated in education and used by several schools. However, it lacks support to allow multi-target programming, especially of physical robotic agents.

Scratch's popularity and previous studies show that the best is incorporating Tactode into the Scratch ecosystem, making it more attractive and promoting better educational goals.

This dissertation's broad goal was to obtain a programming language for physical or virtual multi-targets. The solution was to merge Tactode's features with the Scratch tool, replacing Tactode's application with Scratch itself. A Tactode extension was created for Scratch to add the Tactode functionalities to Scratch. Moreover, a Tactode Link application was developed, which establishes Scratch's connection with external platforms and allows the program's execution directly on the chosen platform. The platform of choice was a miniature robot R@FL of the PRO (*Portuguese Robotics Open*) of open software and hardware developed at FEUP.

This solution allows children to improve the UX of the Tactode system so that they can complete the whole process from the creation of the program to the execution by the external target. The acquired usability raised children's interest and improved learning in several areas of the STEM spectrum, including robotics and CT. Validation of the system was done by performing a set of test challenges and then by experimentation by students solving some challenges.

Keywords: Robotics, Teaching Programming, STEM Education, Technologies for Education, Computational Thinking

CCS concepts:

- Computer systems organization →Embedded and cyber-physical systems →Robotics
- Applied computing →Education
- Social and professional topics →Professional topics →Computing education
- Computer systems organization →Embedded and cyber-physical systems →Embedded systems →Embedded software

Resumo

Atualmente, a procura por formados nas áreas da Ciência, Tecnologia, Engenharia e Matemática (STEM) é bastante visível. Assim, os currículos educacionais estão a ser adaptados. Nas escolas estão a ser introduzidas matérias para que os alunos desenvolvam técnicas e raciocínios mais lógicos, o chamado Computational Thinking (CT).

A versão original do Tactode é um sistema de programação tangível por blocos destinado a crianças e que permite programação multi-target (robôs, simulações ou programas em linguagem de alto nível) através da criação de comportamentos/programas com peças tangíveis que se encaixam como um puzzle. Depois é tirada uma fotografia ao conjunto de peças agrupadas e é introduzida na aplicação do sistema Tactode para que o programa possa ser compilado (transposto) para a aplicação alvo. Se um utilizador assim desejar, a aplicação Tactode permite também criar o programa através da montagem das peças na aplicação mas com muitas limitações.

O Tactode inclui a compilação do programa para algumas plataformas físicas, mas são necessárias aplicações dos fabricantes para que estas executem o programa e não apresenta nenhuma informação de debug na aplicação Tactode. Isto causa uma User Experience (UX) de baixa qualidade na criança, o que dificulta a aprendizagem.

O Scratch é uma ferramenta e uma linguagem de programação gráfica e por blocos, muito popular na educação e usada por várias escolas. No entanto, ele carece de suporte para permitir a programação de multi-alvos, especialmente de agentes robóticos físicos.

A popularidade do Scratch e estudos previamente realizados, mostram que o melhor é incorporar o Tactode no ecossistema Scratch, tornando mais atrativo e promovendo melhores objetivos educacionais.

O objectivo geral desta dissertação era obter uma linguagem de programação para múltiplos alvos físicos ou virtuais. A solução foi fundir as características do Tactode com a ferramenta Scratch, substituindo a aplicação Tactode pelo próprio Scratch. Foi criada uma extensão Tactode para o Scratch para adicionar as funcionalidades do Tactode ao Scratch. Além disso, foi desenvolvida uma aplicação Tactode Link, que estabelece a ligação do Scratch com plataformas externas e permite a execução do programa directamente na plataforma escolhida. A plataforma de eleição foi um robô miniatura R@FL do *Festival Nacional de Robótica* de software e hardware aberto, desenvolvido na FEUP.

Esta solução permite melhorar o UX do sistema Tactode por parte das crianças de modo que possam completar todo o processo desde a criação do programa até à sua execução por parte do alvo externo. A usabilidade adquirida suscitou o interesse das crianças e melhorou a aprendizagem em diversas áreas do espetro STEM incluindo robótica e CT. A validação do sistema foi feita através da realização de um conjunto de desafios de teste e depois pela experimentação por parte de alunos, que resolveram também alguns desafios.

Keywords: Robótica, Ensino de Programação, Educação STEM, Tecnologias para Educação, Pensamento Computacional

Conceitos CCS:

- Organização de sistemas informáticos →Sistemas integrados e ciberfísicos →Robótica
- Computação aplicada →Educação
- Temas sociais e profissionaiss →Temas profissionais →Educação em informática
- Organização de sistemas informáticos →Sistemas integrados e ciberfísicos →Sistemas embarcados →Software embutido

Acknowledgements

I would like to thank Professor Armando Jorge Sousa, professor at the Faculty of Engineering of the University of Porto and supervisor of this dissertation, for his monitoring, support, and dedication during the development of this work.

To my family, parents and siblings, for their patience, support, and consideration in not disturbing me during working hours.

To my colleagues, Rui Guedes and João Barbosa, for their friendship and help throughout my academic journey.

And finally, but most important of all, to my girlfriend for the love, affection, and dedication that made me face this challenge and for the encouragement that made me reach the end of this work.

César Pinho

"What is education but a process by
which a person begins to learn how to learn?"

Peter Ustinov

Contents

List of Figures

List of Tables

Abbreviations

STEM	Science, Technology, Engineering and Mathematics
CT	Computational Thinking
UX	User Experience
TUI	Tangible User Interface
GUI	Graphical User Interface
PRO	Portugal Robotic Open
VM	Virtual Machine
XML	Extensible Markup Language
IP	Internet Protocol
IDE	Integrated Development Environment
OTA	Over-The-Air
PWM	Pulse Width Modulation

Chapter 1

Introduction

Contents

This chapter explains this dissertation's context, motivation, broad goal, and contribution and describes the document's structure.

1.1 Context

Over the years, with the evolution of technology, it has increasingly fit into all actions of our lives. Children are accustomed to new technologies and their use from an early age, but they do not realize what is behind them and what is necessary to build. In this way, they become consumers without even realizing their evolution and appearance.

Computational Thinking (CT) has recently received much attention. It has become a focus of many research pieces that intend to highlight its framework viability in an educational environment. Unfortunately, CT does not have a definition that has a common consensus. Wing [30] has defined CT as a set of thinking skills, habits, and approaches necessary to solve complex problems using the computer. Wing also says that CT covers much more than programming itself. It also incorporates a range of mental tools that reflect the fundamental principles and concepts of computer science, such as abstracting and decomposing problems, identifying recurring patterns, and generalizing solutions. According to Fagerlund et al. [8], CT is an umbrella term that embodies an intellectual foundation necessary to understand the computational world and employ multi-dimensional problem-solving skills within and across disciplines. Thus, CT is not restricted

to computing only and can be applied to the areas of Sciences, Technologies, Engineering, and Mathematics (STEM).

The European Commission and member states are working together to stay at the forefront of artificial intelligence. To this end, each country has presented its strategy for action under this goal. In Portugal, the strategy presented in 2019 [18] and with a deadline of 2030 describes specific objectives and actions for the different areas to intervene. In particular, in Education, some of the specific goals are disseminating STEM knowledge and promoting the early acquisition of coding skills.

The demand for qualified people in STEM areas has increased in the industry. According to the United States Department of Commerce [13], between 2005 and 2015, employment in STEM positions grew at a rate of 24.4% versus non-STEM jobs that grew only 4%. It also highlights that in the decade 2014-2024, an 8.9% increase in STEM jobs and 6.4% in non-STEM jobs is expected. This information translates into the continued growth of STEM jobs, although at a lower rate. This high demand means a lower unemployment rate in these areas and a higher salary in Portugal and most countries.

To meet the industry's needs, the educational curricula are being adjusted, not only in Portugal but also in Europe and worldwide. Teaching programming in primary a secondary education (K-12)[1] is being implemented by several countries [8]. The remaining global communities are being encouraged to do the same by being included in programming events for students. Examples of these events are the Hour of Code, a global movement that attracts millions of students, and Code Week, an event at the European level. These events are intended to uncloak programming and demonstrate that everyone can learn to program [7]. The teaching of programming uses robots, programming frameworks for children, and other tangible alternatives to increase students' interest. Portugal also has the Portugal Robotic Open (PRO) event, an initiative of the Portuguese Robotics Society. It provides the dissemination of Robotics and related areas throughout the country, including students from primary and secondary schools and seniors in robotics.

Cardoso [5] created a tangible block programming system, Tactode, which allows several platforms' programming using a tangible interface. The tangible interface is made up of blocks that fit together like a puzzle. The system consists of creating a program by joining the blocks and taking a picture of the final program with a device. This photo is later inserted in the Tactode application, compiling the program and running it on the target platform. The target platform is external to the application, and it can be a physical device connected to the device where the application runs or a platform like Scratch[2] or Python. The Tactode application was created to be used on mobile devices (smartphones, tablets) and computers, so there are two applications: mobile and web. However, the original Tactode is imperfect as there are no debug features, and the program execution in external devices is complex. It does not allow the program's execution directly on the external platform; it only allows the execution in the integrated simulator or the

[1] K-12 education - https://en.wikipedia.org/w/index.php?title=K%E2%80%9312&oldid=1025099416 (accessed Jun. 9, 2021)

[2] Scratch - https://scratch.mit.edu/ (accessed Jun. 6, 2021)

code file export destined to the chosen platform. Moreover, the tangible programming language does not allow the debug of the program created.

Scratch is a free web-based programming tool that allows creating media projects such as games, interactive stories, and animations. The projects are created through a visual language block-based that prevents students from making syntactic errors because only co-applicable blocks are combined into algorithmic sets of instructions. The combination of graphic blocks produces actions and behaviors on digital characters ("sprites"). The combinations of blocks, sprites, and scenarios constitute a project in Scratch.

Scratch is a top-rated educational tool for students and teachers and is used in several schools and initiatives to promote computer literacy. It also allows the creation of projects with external devices (robots) through extensions, such as LEGO® EV3[3], LEGO BOOST[4], and micro:bit[5].

1.2 Motivation and Broad Goal

In a study created by the author of Tactode [6] itself, it was possible to observe that the system did not reach the necessary acceptance by the target public. This study was created to experiment with Tactode and compare it with Scratch, being observed a preference for Scratch by most students, arguing for its speed and ease of execution.

As previously mentioned, the original Tactode application is not perfect. It depends on external applications to allow one of the most notorious functionality, executing the program on external platforms. For this reason, the application was abandoned. To replace the application and reconcile the tangibility and programming benefits of Tactode physical platforms with Scratch's preference, the objective of this dissertation is to merge Tactode with the Scratch tool. This fusion was accomplished by creating a Tactode extension for Scratch that includes all functionalities of the Tactode system, including the insertion of images of the programs created with the physical parts and the programs' direct execution on external platforms preferably physical.

Beyond the advantages associated with the inclusion of Tactode in Scratch, the proposal of this dissertation already included the use of Scratch due to its benefits in the programming tool business. Besides this choice, the physical platform used during this dissertation's development was also chosen by the proponent. The physical platform in question is a robot whose parts are printed on 3D printers, with open-source hardware and software, and developed at the Faculty of Engineering of the University of Porto (FEUP). The robot was developed within the "Robot at Factory Lite" competition and was adapted with the ESP32 microcontroller. The ESP32 allows the robot to be equipped with Bluetooth and Wi-Fi connection. The wireless link is usable for remote programming and continuous link from the programming interface to the physical robotic platform.

[3]Mindstorms | Official LEGO® Shop - https://www.lego.com/en-gb/themes/mindstorms (accessed Jun. 6, 2021)

[4]BOOST | Official LEGO® Shop - https://www.lego.com/en-gb/themes/boost (accessed Jun. 6, 2021)

[5]micro:bit | Micro:bit Educational Foundation - https://microbit.org/ (accessed Jun. 6, 2021)

This dissertation's broad goal is to obtain a programming language for physical or virtual multi-targets, captivating to increase students' interest and reliability for a wide range of ages so that it can be included in educational institutions. Since public institutions have a budget, expenditures must be reduced in order to allow the accessibility of the system to all social classes.

This dissertation encompasses technical, scientific, and pedagogical challenges. In terms of the technical challenge, we can mention the abstraction and generalization necessary to allow the programming of several targets, both physical and virtual. This challenge has significant importance in the economic impact of the inclusion of this system in public schools. The benefit of a diversity of possible targets of the system allows the institution to choose a physical robot economically viable according to the available budget or even opt for a virtual platform if the budget is more restricted. In terms of scientific and pedagogical challenge, it is essential to highlight and preserve the Tactode system's objective transmitting knowledge and developing programming skills and logical reasoning to children and young people, thus, triggering learning and interest in STEM areas.

1.3 Contributions

In virtue of the broad goal of obtaining a programming language for physical and virtual multi-targets, the contributions of this dissertation are focused on education, which is the medium of application of the intended language. Thus, the main contributions include:

- **Helping children to understand the technology and its development.** The focus of this work is to provide a tool for application in schools that allows children to understand technology, learn to program, develop CT and develop an interest in STEM areas.
- **Provide a multifunctional tool.** Like the original Tactode and fulfilling the objectives of this dissertation, the system created is multi-target, being possible to program the Scratch simulator or external platforms, such as the R@FL robot. The system is also compatible with different software making it more generic.

With the solution obtained in this work, it is possible to do robotics activities with children and young people, being them programming robots developing CT, and learning and gaining interest in STEM areas.

The system resulting from this dissertation allows Scratch to program external platforms of open structure, which was not the case until now. Before, Scratch only let the programming of some external LEGO platforms and the micro:bit. With the extension created for Scratch, it is possible to program diverse external platforms, with several compositions of sensors and actuators, being generic enough to allow the programming of platforms, namely robots, that may arise in the future.

During the context of this dissertation, a system was also developed to connect the R@FL robot to devices via Wi-Fi, whose configuration of this connection is done via Bluetooth (see section 4.2). This development allowed to solve a problem of simultaneous use of Bluetooth and

Wi-Fi technologies and was applied in the Junior University project, besides the application in this dissertation. The solution found for setting up the Wi-Fi connection using Bluetooth solved an existing problem that was often circumvented through unpleasant solutions that were not recommended for children and young people due to their complexity. In other words, without this solution, the goal of allowing the programming of a robot by a child, from the initialization of the robot to the program execution in the robot, would be compromised since an adult intervention might be necessary for the configuration of the Wi-Fi network.

An abstract was also submitted to the *14th annual International Conference of Education, Research and Innovation*, ICERI 2021[6], to present the system created.

1.4 Dissertation Structure

This document is divided into five chapters. In the current chapter, Chapter 1, the context, motivation, broad objective, and contributions of this dissertation and its structure are explained. In Chapter 2, the state of the art of STEM education and teaching of CT using physical or tangible or block programming tools is presented. Also, the actual Tactode status and the technical issues related to Scratch and the physical robot to be used are referred. Chapter 3 contains the composition of the problem, the goals, and the requirements. Chapter 4 describes all the work realized during this dissertation, referring to the solution's architecture, the description of each part of the system, and the experiments and results. Chapter 5 includes the conclusions drawn and the future work.

[6]ICERI 2021 - https://iated.org/iceri/ (accessed Jun. 20, 2021)

Chapter 2

Fundamentals and Related Work

Contents

In this chapter, the state of the art is presented, and works related to this dissertation are described. Thus, it is discussed the teaching of Computational Thinking through programming, mainly associated with physical objects, physical programming (e.g., robots) and tangibility, or graphic programming tools, Scratch. The benefits and improvements of the approaches presented are also evaluated.

2.1 Tactode

In this section, the original Tactode system is presented in detail, describing its evolution and limitations.

2.1.1 History

The Tactode was created by the engineer Ângela Cardoso as part of her master's thesis in 2018/2019. Tactode [5] is a tangible block programming system designed to help children develop CT skills and interest in STEM fields. It is a comprehensive and diverse programming system. It allows programming through a tangible interface or a graphical interface, combining TUI and GUI advantages [22].

The tangible interface consists of joining physical blocks similar to jigsaw puzzle pieces that fit according to their compatibility. The junction of the blocks forms a program. The program is taken a picture with a device. This photo is introduced into the web application, compiling the program and running in the application's simulator. When the photograph is inserted in the application, the program is replicated in its graphical interface, with pieces equal to the tangible ones. Thus, it is also possible to build programs in the application using the graphical interface and the drag and drop of parts.

In the application, it is possible to export the program to physical devices (e.g., Ozobot, Cozmo) or virtual platforms (e.g., Scratch). After this export, it is necessary for the program to manually enter the file containing the program on the target platform for the program to execute on these platforms.

The Tactode system allows the aggregation of tangible and physical programming benefits.

2.1.2 Limitations

According to the creator of Tactode, Cardoso [5], the application should communicate with the platform chosen and run the program directly on that platform. Unfortunately, this objective was not achieved, and the application only allows the export of a file with the program. The steps necessary to run the program take the autonomy from programming to the younger children who may need monitoring in the various stages. This situation causes a low UX in the children and can cause demotivation and consequent discontent with the system itself and decrease learning.

The tangible programming language of Tactode does not allow debugging the program created. That is, the tangible language does not allow the detection of errors in the program. Thus, Tactode makes visual learning difficult and requires an additional effort for error correction since it is necessary to compile in the web application to detect the program errors.

Tactode contains a closed architecture since it only allows programming a small list of platforms, consisting of four robots and two virtual platforms. On the other hand, during the search performed, no system was found that allowed for a more extensive list of platforms or generic enough for several platforms.

2.2 Pedagogical Aspects

This section presents studies that demonstrate the importance and necessity of including programming and CT development in children. Systems created within the scope of research in these areas

and the experimentation results of these systems with users are also described. Mainly physical and tangible programming systems are mentioned.

2.2.1 Computational Thinking

"CT covers far more than programming itself; it also includes a range of mental tools that reflect fundamental principles and concepts of computer science..." [11]. As Malizia, Turchi, and Olsen said [11], computational thinking encompasses much more than programming. CT contains concepts such as problem-solving, abstraction and decomposition, pattern detection, and solution generalization. The ideas and intellectual foundations acquired can help solve multidisciplinary problems [8].

Several articles and publications deal with the subject and defend the high importance of CT and its teaching in various educational degrees (K-12).

Although CT is more than programming, programming is one of the best ways to develop such skills. Programming or coding power an environment of teaching and learning computational thinking since it makes the concepts concrete [1]. The content of programming and the involvement in programming activity by students stimulate skill and interest in areas involving CT and provide learning in areas such as mathematics, science, and engineering.

Programming can be practiced in several ways using diverse platforms and frameworks that allow it. There are platforms for tangible programming, graphic or visual programming, that can be associated with physical programming. Tangible programming is based on physical parts that represent programming instructions and fit together like a puzzle. It can be combined with the use of computers or other similar devices or not. This type of programming ends up defining a tangible programming language. Graphic or visual programming (e.g., Scratch) commonly uses the block-oriented programming paradigm. Each block represents a code instruction and allows the construction of programs in a virtual environment through compatible blocks combination. Unlike a high-level programming language (e.g., Python), the block programming paradigm reduces the errors committed during the program's development. Physical programming consists of physical programming devices such as robots, using a programming language that can be tangible or virtual as visual programming or other high-level programming languages. All the types of programming presented follow the imperative programming paradigm that stands out for its simplicity in understanding how it works since it is similar to the imperative behavior of natural languages that express commands.

The development of CT can be applied in various degrees of education. Roussou et al. [20] investigated the development of computational thinking in kindergarten, and Benvenuti et al. [1] addressed its development in elementary school. Besides this, some other studies and articles support this statement, reporting results from different education levels. Such is the case of [16], which presents a test study of robot programming for CT development by students from 10th to 12th grade.

Roussou et al. [20] conducted a study to observe the impact of programming activities on CT skills. The effects on the ability to perceive the cause and effect relationship were observed in

formulating hypotheses, understanding logical order and sequencing, and the localization and correction of code errors, developing problem-solving skills. Since children still cannot read or write, it is not possible to use conventional programming instructions. The alternative found involves optical and haptic programming. Colby from Learning Resources TM, a mouse-like educational robot, Figure 2.1, and Code & Go Robot Mouse Activity Set, Figure 2.2, were used.The trial lasted nine weeks and included an experimental group of 13 pupils (six boys and seven girls) who had contact with the robot programming and a control group of 7 pupils (three boys and four girls) from another class. The evaluations were carried out through individual interviews at the beginning and the end of the trial. The interview consisted of 36 questions that the pupils had to answer. Of these 36 questions, ten were about finding the cause of a problem, ten were about formulating hypotheses, eight were about logical figure ordering, and eight measured problem-solving ability. The results obtained demonstrate a positive impact; the experimental group improved the pre-test scores to the post-test, while the control group maintained the scores. The experimental group increased the right answers in a range from 11% (4 responses) to 42% (15 responses). In comparison, the control group only recorded 0 or 1 additional correct answer in the post-test. The interpretation of the results also leads to promising conclusions. During the verbal communication to answer the post-test questions, there was a more remarkable ability to reason, improvement in the use of conditional sentences, and the care to formulate more complex and realistic hypotheses. Due to the small size of the population under study, the results cannot be generalized but serve as indications.

Benvenuti et al. [1] report the work of the fifth grade class of an Italian elementary school where the Italian Ministry of Education introduced the program of initiation to programming for the development of computational thinking. Scratch was used in the introduction to programming. The class with 24 students (7 female and 17 male) used Scratch for one year in the computer lab. Initially, during half a year, the students worked on individual projects. As a result, the 24 students shared 248 individual projects on the Scratch website, most shared between 5 and 15 projects. In the second half of the year, the teacher formed six groups of 4 students with a theme associated with each group to create multimedia stories, including design steps, Scratch implementation, peer review, review, and presentation. The students answered a self-evaluation questionnaire, where it was possible to verify that, in general, it was a positive experience. At the end of the projects, an interview was held with each group to analyze the teamwork. They concluded that collaboration allowed better stories and improved Scratch knowledge but generated conflicts that decreased creativity compared to individual work.

Another study was created by Miller et al. [12] to examine the efficiency of the introduction of computing and robotics concepts. For this, the students used a robotic arm programmed by a tangible user interface (TUI) without using the computer. The authors created lessons to allow the integration of educators and students to compute without requiring expensive technical knowledge or resources. Miller et al. paid particular attention to comparing different socioeconomic status groups. The study population included 148 middle and high school students (grades 6-10), grouped in pairs or trios. Lessons were planned to acquire knowledge of sequencing, debugging,

Figure 2.1: Colby robotic mouse with haptic programming.

Figure 2.2: Code & Go Robot Mouse Activity set (Learning Resources ™).

and decision-making by building code for the robot to move a block. In the population under analysis, 43% of the students were the first contact with engineering lessons. The lessons were given in schools (55% of students) and an engineering summer camp (45% of students). They evaluated the interests and attitudes towards engineering before and after the lessons. The results of the summer camp group of students showed no change between the pre and post-analysis. But in the school group, there was an increase in students' interest in engineering, especially by students with lower socioeconomic status. This study concludes that students with lower socioeconomic status have more to gain from accessing engineering lessons. Hence the interest in including CT programming and development in public schools, where this population is more concentrated.

2.2.2 Tangible Programming

For initiation to programming in a school environment, a programming language should be used that captivates and awakens interest in the student since the intention is not to teach programming to create professionals but rather to develop computational thinking. Many computer programmers start programming with high-level programming languages that allow the creation of professional programs. Some of the most common languages for beginners are *Python, C/C++, Java,*

JavaScript, and *Ruby*. These languages require a knowledge of the language in order to understand its syntax. Block-oriented languages have the benefit of reducing the risk of syntax errors since the programmer can combine only compatible blocks. Therefore, they are an additional advantage for students of primary and secondary schools because they reduce the time of introduction to the language and also reduce the loss of interest due to errors made.

Tangible programming languages allow programming by manipulating physical parts instead of using a computer screen. They are similar to block-oriented or visual programming languages, and physical pieces represent the various programming elements, commands, and flow control structures. Most tangible programming languages consist of blocks or pieces that the user unites to obtain the program. In this way, they take advantage of the block-oriented programming benefits. Instead of virtually grasping or selecting the blocks, they are physically grasped and placed in the desired place.

According to Malizia et al. [11], tangible user interfaces (TUIs) are based on physical manipulation and consist of a digital interaction paradigm designed to provide more user-friendly interfaces. They argue that using our innate dexterity of object manipulation can efficiently help users acquire the abstract concepts of coding by developing CT skills.

Sapounidis et al. [22] say that face-to-face interaction and physical manipulation with TUIs make programming more "enjoyable, attractive, collaborative, and usable" than if graphical user interfaces (GUIs) are used. Tangibility is also easier for younger children to use and is considered more attractive. It is also considered more enjoyable for all children. To confirm their suspicions, they have developed a study to explore the preferences for TUIs or GUIs, measuring attractiveness, collaboration, enjoyment, easy-to-use, convenience, understanding, explainability, and easy-to-remember variables. The V-ProRob, Figure 2.3, and T-ProRob, Figure 2.4, programming tools were used, both allowing programming the NXT LEGO® robot. The participants, who were 148 (57.4% boys and 42.6% girls), were students from two public schools, aged between 6 and 13. The pupils formed pairs of the same age and worked for about 1 hour and 15 minutes. The couples were divided into two groups in order to alternate between the use of the GUI and TUI. In the first step, each pair performs two tasks on the interface that is assigned to their group — group 1 performs the first tasks on TUI and group 2 on GUI — and after finishing, performs two more tasks on the opposite interface — group 1 on GUI and group 2 on TUI. Then, using each system in turn, the pairs could program the robot twice more without any task assigned. In the end, each child filled out an individual questionnaire. As results obtained, most of the participants were in favor of tangible. Younger children showed a strong preference for TUI. The choice of older children is divided by gender; boys prefer GUI, and girls prefer TUI.

Caceres et al. [4] created a mechatronic tangible programming interface called FYO (Follow Your Objective), consisting of a programmable board, puzzle-based tangible blocks, and a mobile robot. Tangible pieces are inserted into the board to form the program. The board later connects with the robot in order to command the execution of the program. The available language is very restricted at the block level. There are only four motion command blocks and a "go" (redirects to a function), parameter blocks (numbers and function identifiers), and function definition blocks. The

Figure 2.3: V-ProRob.

Figure 2.4: T-ProRob.

system was designed to be a low-cost alternative for introducing programming and CT concepts to children. Still, it is restricted to the robot itself and with functionality limited to basic movement.

In addition to the examples of tangible programming languages shown, there are others such as AlgoBlock [9], Figure 2.5, Cubelets [23], Figure 2.6, Fisher-Price® Think and Learn Code-a-Pillar Twist™ [17], Figure 2.7, T-Maze [29], Figure 2.8, and Osmo Coding Family [14], Figure 2.9.

There are some tangible programming languages, but most are not available in the market. Besides, existing ones are usually expensive due to electronic components or limited capabilities, as we saw earlier. According to [6], there are no options for tangible programming languages that use different paradigms and robots.

2.2.3 Physical Programming

Physical programming encompasses interactive systems that collect information and respond to the world around them. These interactive systems are often robots with sensors and actuators. The most diverse programming languages can be used to program the systems. This type of programming, robot programming, is applied in the school context due to its benefits for students. In the school environment, programming languages are selected to be more attractive to students, tangible programming languages, graphical, or even programming languages of high level.

The teaching of robotics in schools allows creating an active and collaborative learning environment. Another advantage is the instantaneous response that students receive about the robot's

Figure 2.5: AlgoBlock.

Figure 2.6: Cubelets.

Figure 2.7: Code-a-pillar Twist.

behavior when executing the program and the expected behavior [21]. This response allows students to experiment and learn from mistakes. Robotic programming can be challenging for students and teachers who have no programming experience [3]. However, it can be introduced at various education levels, from kindergarten to high school, through tools adapted to the desired education degree.

Bers in [2] describes two programming tools for kindergarten and early elementary school children who do not have literary skills. ScratchJr, Figure 2.10 provides a screen-based programming experience used in tablets and smartphones. It has a target audience aged 5-7 years. ScratchJr allows the creation of programs through the combination of blocks representing actions executed in

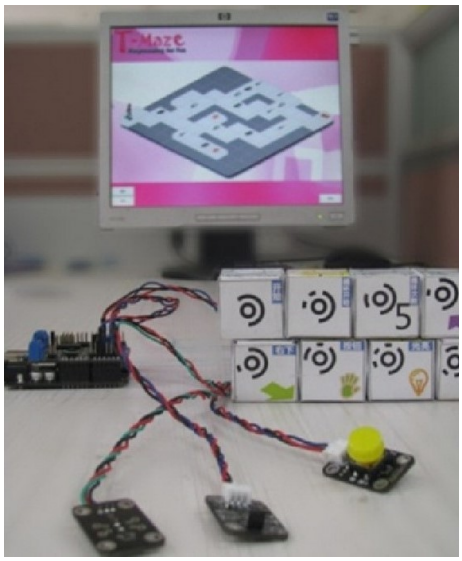


Figure 2.8: T-Maze.

Figure 2.9: Osmo Coding Family.

the simulator in the application itself. The tool follows the block-oriented programming paradigm, but the blocks have been carefully simplified for the target audience. The other system that this article addresses is a robot, the KIBO, Figure 2.11, adapted for children between 4 and 7 years. KIBO allows a screen-free programming experience and makes children's ideas physical and tangible. The KIBO kit includes the robot body, wheels, sensors, lights, motors, and the tangible programming language formed by a set of wooden blocks. The KIBO robot has a scanner that allows reading the blocks' bar codes to receive the program to run. Besides allowing children to find computer science ideas and develop computational thinking, it also allows attracting children to be problem solvers, engineers, or even artists, dancers, or writers. KIBO's language syntax supports the development of sequencing capabilities that will be essential to future academic success.

Figure 2.10: ScratchJr.

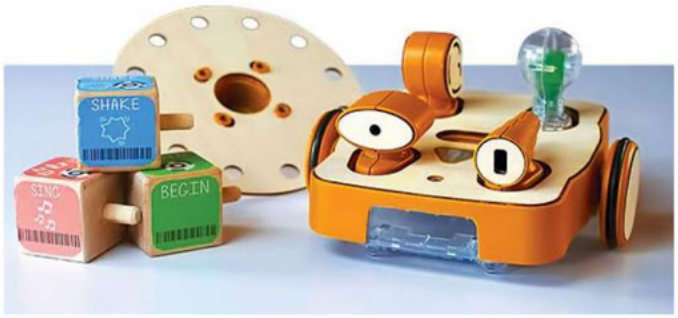


Figure 2.11: KIBO robot and programming blocks.

Cleto et al. [7] present a system, CodeCubes, Figure 2.12, that allows the physical programming and virtual representation of a program. Programming is done by manipulating paper cubes with an increased reality marker on each side associated with a programming instruction. The cubes' physical manipulation allows simultaneous visualization of the result, making it possible to change the program while running and viewing the changes. The authors conducted a study with nine students from the robotics club in a school between 9 and 13 years old to test this system's acceptance. The experimentation took place in 6 sessions and consisted of performing the same task using several programming platforms. The students used Scratch for visual block programming (programming with software), LEGO WeDo robot programming (programming with hardware), and CodeCubes programming. It was also performed a programming task using Code.org furthermore to compare this platform. In the last session, the participants were interviewed individually. The participants show a preference for CodeCubes programming. In 2nd place was the programming with the robot, and in 3rd the Scratch. We can conclude that students are more attracted to physical and tangible programming, within the visual programming, Scratch or Code.org, prefer Scratch.

Figure 2.12: CodeCube programming.

As mentioned in the previous section, Caceres et al. [4] have created a tangible mechatronic programming interface, FYO, Figure 2.13, which includes a tangible programming language and a mobile robot. The communication between the robot and the board containing the program is established via Bluetooth. The system, including the robot, was tested with a population of 15 students between 5 and 8 years. After four steps—first contact, general training, rules training, and final evaluation—with some associated tasks and the measurement of each task's time, they concluded that no child could complete the tasks in the estimated time due to mistakes with the directions. Furthermore, the children were able to find alternative solutions.

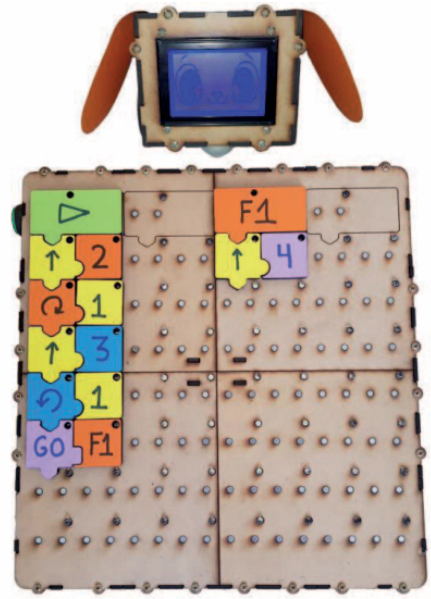


Figure 2.13: FYO system: robot and programming board.

According to Saleiro et al. [21], the inclusion of robots in classroom teaching in the long term increases the general learning, motivation, and performance of students. Elementary school children, in particular, show a higher interest in robots. The authors present a system composed of 5 small robots, the "Infantes", Figure 2.14, and a single multi-robot controller. The robots' programming is done in a web application that communicates with the controller via Wi-Fi, and this in turn with the robots via Bluetooth. Thus the controller serves as the communication interface

Figure 2.14: Infantes robots.

Figure 2.15: Infantes programming interface based on Blockly.

between the programming application and the robots. The programming application consists of a graphical programming interface by blocks based on Blockly [1], Figure 2.15. The authors created the system to reduce the setup time and to have an effortless use. To test this system's adaptation, the authors conducted two experiments, one with a 4th graders class developing mathematical reasoning while learning geography. The other experiment was with a 3rd graders class teaching how to recycle and reuse electronic waste using robots. In both cases, the students worked in groups of 5 or 6 students. In the first experiment, an impressive motivation was verified once they finished the tasks more quickly and autonomously. The same motivation was also found in the second experiment with creating new tasks by the children, after completing the defined tasks, to continue programming until the class finished. The children learned quickly to use the system and, during the tasks, unconsciously, developed "mathematical reasoning, deductive, abductive as well as inductive" [21]. This system's disadvantage is the limitation of actions because it only contains three possible commands—go forward, rotate to the left, and rotate to the right.

Komm et al. [10] describes his approach and experience accompanying a group of students in a LEGO EV3 robot programming competition using Python. In the article, the authors describe the work of a group of 24 high school students. The group consisted of 6 girls and 18 boys, and the students worked in pairs. The activity in which the students participated was divided into tasks to be solved with the EV3 robot and lasted one week. Especially the girls showed more interest in creating their challenges. In the end, the students answered six questions to realize the impact of the activity. Most students (between 79% and 93%) said:

[1] Blockly | Google Developers - https://developers.google.com/blockly

- They learned a lot about robotics;
- That the workshop aroused interest in computer science;
- Found coding exciting;
- Interest in repeating the activity the following year.

Phetsrikran et al. [16] created a platform for introducing CT concepts to children called "Arducation Bot". This platform includes a physical robot, a playing board with lines where the robot moves, and an application for programming the robot. The platform was tested in *Thailand-Japan Student Science Fair* with 32 groups of 5 to 6 Japanese and Thai students from 10th to 12th grade. Each group had one hour to complete ten incremental difficulty puzzles. The students solved the easy puzzles within the expected time. The averages took a little longer than expected. The tricky puzzles took a much longer time to solve. The latter involved solving conditional problems. According to the authors, a possible justification was the inadequate introduction of the concept. Despite the somewhat unexpected results, the authors demonstrate the feasibility of combining physical programming in CT development.

Pedersen et al. [15] present an educational robot, the BRICKO, Figure 2.16, for children from the 1st to 3rd grade. BRICKO consists of a robot and a LEGO-based programming-board where the LEGO-shaped parts are embedded to form the program. The article describes an experiment with 108 pupils between 5 and 8 years old to test the system. The experiment was divided into seven iterations with different tasks. During this experiment, the authors verified the efficiency of the system in the development of social interactions.

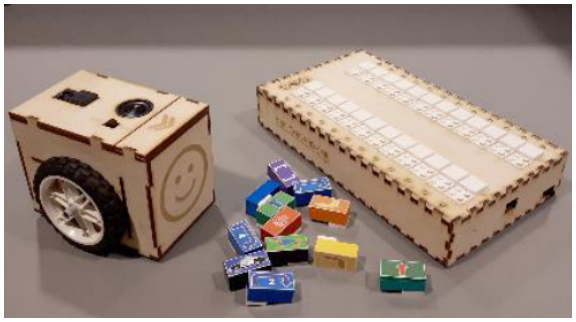


Figure 2.16: BRICKO educational robot and its programming board.

The inclusion of physical programming, especially robotics, in the educational curriculum brings several benefits beyond those mentioned so far [3, 10, 16, 21]:

- Increases the motivation of students;
- Awakens the interest of students in the STEM area;
- Helps young programmers to understand the machine and what programming is for;
- Teaches basic programming concepts;

- Increases creativity in creating projects;
- Allows the development of CT skills
- Increases student's confidence in their abilities;
- Helps develop student communication and collaboration;
- It can be taught to students from different backgrounds or age groups.

2.3 Technical Issues

This section describes and presents the tool to be used in this work, Scratch, as well as the external platform, which in this case is the R@FL robot.

2.3.1 Scratch

Scratch is a free visual block-oriented programming language. It is mainly intended for children between 8 and 16 years old [19]. Scratch can be used online in your web application or offline by installing the application locally. The application is available for offline use for the main desktop operating systems, Microsoft Windows 10, and Apple's macOS 10.13. At the time of this work, the Scratch version is Scratch 3.0 and was released in January 2019 [25].

2.3.1.1 Scratch Extensions

Starting with Scratch 2.0, extensions were made available to add extra blocks and other features for projects. In Scratch 2.0, all extensions were hardware-based, adding blocks to control physical devices like robots or receive external devices' inputs. In Scratch 3.0, the Scratch team added software-based extensions. Examples of extensions are [26]:

- Hardware-based:
  - LEGO Mindstorms EV3 — allows to control motors and receive information from sensors of the LEGO EV3 robot;
  - LEGO BOOST — allows to control the motors and receive information from the sensors of the LEGO BOOST robot;
  - LEGO WeDo 2.0 — allows to control the engines and receive information from the sensors of the LEGO WeDo 2.0 robot;
  - Makey Makey — allows using Makey Makey to control projects;
- Software-based:
  - Music — plays digital instruments;
  - Pen — draws on stage;
  - Text to Speech — converts words into text to speech;

2.3.1.2 Scratch Link

Scratch Link [27] is a program that establishes Scratch communication with devices in the physical world. This program is necessary when Scratch is used to program physical devices such as the LEGO EV3. To program a robot like the LEGO EV3, the corresponding extension is needed to add the device's control blocks and the Scratch Link to communicate between the programming environment (e.g., computer) and the physical device. Communication is established through Bluetooth.

2.3.1.3 ScratchX

ScratchX is a website developed by the Scratch team to provide a test environment for experimental Scratch extensions. Any programmer can create a JavaScript extension for Scratch and test its operation in ScratchX. However, with the release of Scratch 3.0, ScratchX was discontinued. The Scratch team decided to incorporate the development of experimental extensions into Scratch 3.0 itself. Although the ScratchX website is still operational, the extensions developed in this environment are not supported by the current Scratch version. And the conversion of the extensions is not always possible [28, 24]. Thus, the development of extensions for the current Scratch version is done locally, with a development environment that includes some Scratch 3.0 modules.

2.3.2 Open Robotic Platform R@FL

The R@FL robotic platform is a small robot created at FEUP for the "Robot at Factory Lite" competition. It is open-source hardware and software robot. The robot fits in a rectangle of 25×20 cm and has a maximum height of 20 cm. The structural parts of the robot were made with 3D printers. The robot has two motorized wheels, a line sensor, an electromagnet, and a chargeable battery.

The "Robot at Factory Lite"[2] competition is one of several competitions held at PRO, and the rules are based on the rules of the "Robot@Factory" but with mechanical and hardware simplifications.

The original robot only contains one Arduino Nano microcontroller that is responsible for all processing of the robot. This microcontroller does not support any wireless communication. The robot was changed during this dissertation to add new features such as the wireless communication needed for the inclusion of the R@FL with the Tactode system.

[2]Robot at Factory Lite Competition Material - https://github.com/P33a/RobotAtFactoryLite (accessed Jun. 19, 2021)

2.4 Summary

The interest in STEM areas can be induced by the development of computational thinking in schools. And the best way to develop CT skills is through programming. Several studies mentioned above claim using tangible programming technologies and physical programming to capture the children's attention. The range of such technologies available is diverse, but sometimes with limited functionality, a restricted target audience, or too expensive.

In addition to CT skills development, tangible programming and physical programming have great results in educating students. They are, therefore, an added advantage for their inclusion in public schools.

Given the positive effects of integrating programming into K-12 education, it can be seen that the market does not have sufficient supply for such a need. As discussed, most of the systems described are not available on the market, or those available have a limited range of functionality or high costs. The Tactode system completes these existing gaps, allowing the programming of various external platforms. It is possible to find the most suitable to each institution's budget besides that it encompasses the benefits of tangible and physical programming.

Chapter 3

Problem Statement

Contents

The current chapter formalizes the problem solved in this dissertation, the associated goals, the proposed solution, and the execution plan. It is also referred the work previously prepared.

3.1 Problem

The importance of the STEM areas, the CT, and the introduction of programming in the educational curriculum leads to a need for tools to be applied in schools to address these issues. Besides the tools, teachers and educators need to know the tools in order to transmit knowledge to students and help them understand the mechanism. As mentioned in the previous chapter, the tools whose students show greater affinity and interest have a limited target audience. For a school with several degrees of education, this would imply acquiring several different programming systems and, for teachers, the need to know several tools.

Tactode covers a wide range of ages as it allows tangible programming for the youngest, visual programming for the oldest, and physical programming for both. This system is a system that can be applied to various degrees of education. But Tactode is imperfect. The application does not allow direct programming of a robot, requiring the use of external software. This caused a low UX and discontent in students who preferred Scratch over Tactode [6].

Although it does not have tangible programming, Scratch allows the programming of physical robots such as Lego EV3, Lego BOOST, or micro:bit. The disadvantage is the acquisition price of these robots that makes their acquisition very expensive by a school.

To allow the inclusion of the Tactode system in a school, it is necessary to make it more appealing to students, with a better UX, and provide the programming of low-cost robots. Costs with tangible parts are already reduced and thus do not add any problems.

3.2 Goals

This dissertation's broad goal is to obtain a programming language for physical or virtual multi-targets, captivating to increase students' interest and reliability for a wide range of ages so that it can be included in educational institutions.

The broad initial goal also included tangible programming for physical and virtual multi-targets, but the time needed to complete this goal was too long. So it was decided to include this functionality as future work.

As a pedagogical objective, we must consider the teaching of robotics concepts, computer sciences, and programming to develop CT skills and increase interest in STEM areas.

In addition to the general objectives, taking into account the research done, the system should:

- Make programming debugging easier in external platforms like robots;
- Allow graphic programming;
- Allow the programming of several physical or virtual platforms;
- Have a good UX for a wide range of ages for learning improve.

3.3 Proposed Solution

The proposed solution to the problem presented is creating an extension for Scratch that includes the existing Tactode application features. From these functionalities, we can state the ability to develop programs for external platforms and developing programs in the application. This last functionality is trivial in Scratch, but we intend to add the necessary blocks for external programming platforms with response capability to commands not existing in the current Scratch.

The solution also includes the addition of running the program directly on the external platform from Scratch. As seen previously in section 2.3.1.2, Scratch allows communication with external devices through an additional application, Scratch Link. Thus, to allow a greater diversity of possible means of connection (Bluetooth, Wi-Fi, USB cable), an application, Tactode Link, similar to Scratch's one, is created.

3.4 Requirements

In this section, the functional and non-functional requirements of the system are presented, taking into account the user needs and user experience.

3.4.1 Non-Functional

The system must be intuitive, guarantee performance, high reliability, and a low life-cycle cost.

It must be **intuitive** so that users can easily understand how it works without an instruction manual and also so that attention is focused on developing programs and not on understanding how they work.

Performance is a priority since high waiting times decrease the user's attention and interest and cause poor UX.

Reliability must be ensured to prevent errors from occurring. If they do, ensure that the user can easily detect their cause by using visual elements to give feedback on what is happening.

The **cost**, as already mentioned, should permanently be reduced to a minimum, both the maintenance cost and the acquisition or production cost, in order to make the tool accessible to a broader range of economic classes and educational institutions.

3.4.2 Functional

Since the goals of the original Tactode application must be maintained, so must the requirements of the application. Thus, many of the functional requirements of the original application are practically assured by the Scratch functionalities. For example, creating programs by combining blocks, clicking on the green flag to start the program and the red button to stop, naming the program, exporting or saving it, and even changing the language are all functionalities that belong to the Scratch application.

On the other hand, some requirements must be met to ensure the system's proposed and described functionality in this dissertation. The main requirement that must be fulfilled is establishing the connection between Scratch and the external target. The user wants to connect the device where he uses the Scratch application to the external platform that he intends to program so that he can program the platform directly and also receive information, namely the sensor values.

In the same way that the program runs and stops in the Scratch simulator when the green flag and the red button respectively are clicked, it is necessary to ensure that the same happens with the program's execution on the external platform. That is, the user wants to click the green flag or the red button and see the external platform run or stop the program created in the Scratch application.

3.5 Summary

Given the existing problem of the need to include programming in schools, easy-to-use, attractive, and feasible tools are needed.

The proposed solution was to integrate the Tactode system with Scratch to allow programming of external multi-targets using the Scratch platform. This solution includes creating an extension for Scratch and a communication program with external platforms through Wi-Fi, Bluetooth, or USB cable. The system must meet specific requirements in order to be a feasible tool.

Chapter 4

Tactode Multi-target Extension for Scratch – TaMuES

Contents

This chapter exposes the development of our programming system, the Tactode multi-target extension for Scratch - TaMuES. The system includes the extension for Scratch, the application for communicating with external devices, the Tactode Link, and the firmware developed for the robot. The architecture of each of the constituent parts of the system is presented, followed by a description of the implementation, the alternatives considered, and the operation of each element. This chapter also contains the pedagogical plan and the results obtained in testing the system with users.

4.1 Architecture

The system does not follow a single architecture since it is divided into three different programs. Thus, each of them, the Scratch Extension, the Tactode Link, and the robot firmware, has its architecture.

Figure 4.1 shows the overall architectural organization of the solution, showing the interaction between all the components involved.

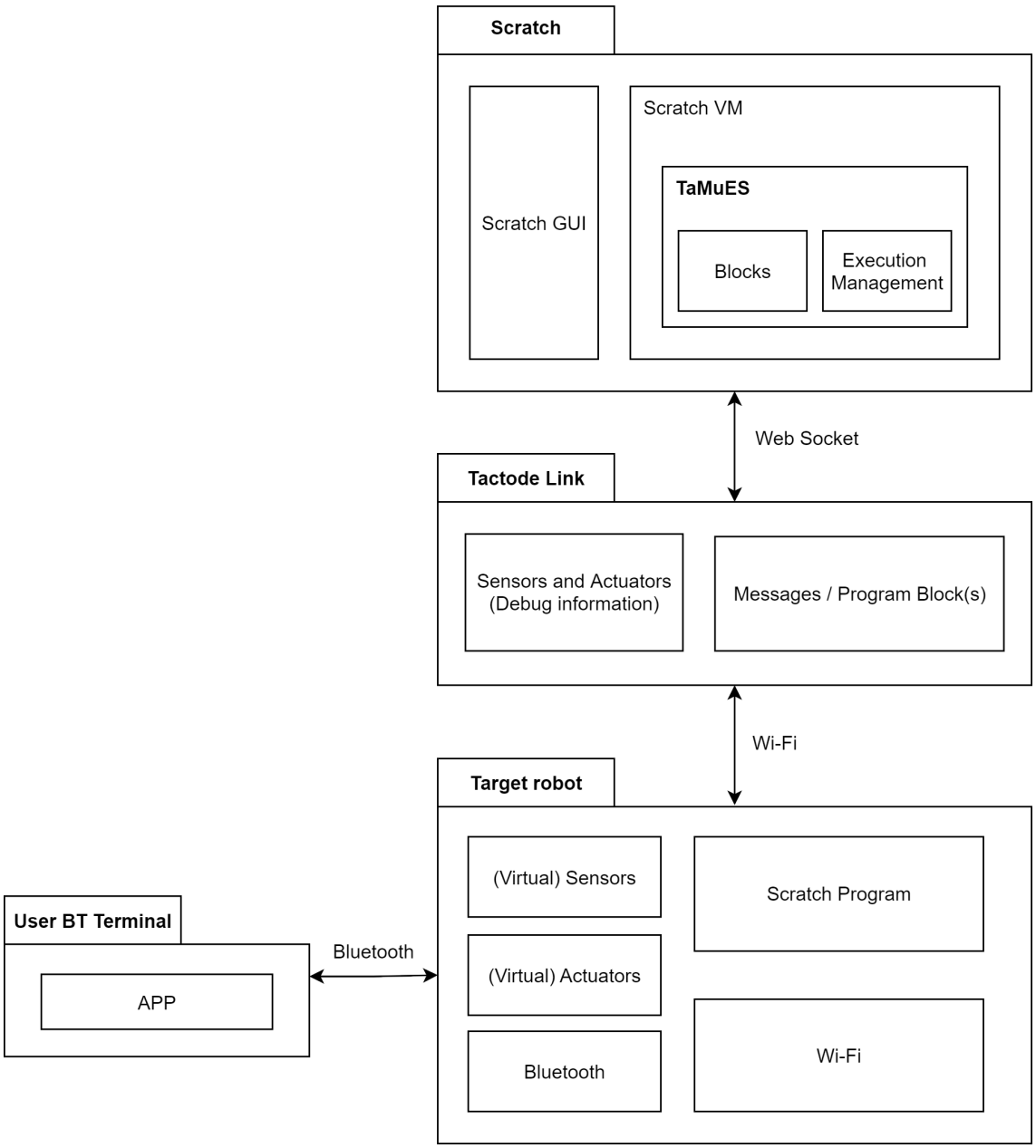


Figure 4.1: Solution architecture diagram.

4.1.1 Scratch Extension

The version of Scratch used in this dissertation was the latest version available to date, i.e., Scratch 3.0. The creation of an extension for Scratch 3.0 has to follow a template provided by Scratch to allow the addition of new functionality without compromising the functioning of the original system. Thus, the programming language used was `Javascript`, which is the language used throughout the Scratch system.

Scratch is divided into different modules, and for the development of the Tactode extension, the Scratch VM and Scratch GUI modules were needed. The first module is responsible for representing, executing, and maintaining the state of computer programs written using Scratch Blocks. Scratch GUI module is a set of React components that comprise the interface for creating and running Scratch 3.0 projects. While the creation and addition of the extension code are in the Scratch VM module, the Scratch GUI is used to test the use cases, validate the extension's correct functioning, and ensure a good UX.

The extension is created in the Extensions folder present in the Scratch VM module and is clustered in the folder named `scratch3_tactode` to follow the example of the existing extensions. In the extension folder, an index.js file was created, which contains the main class invoked by Scratch. This class is named `Scratch3Tactode` (Figure 4.2) and includes the constructor and the `getInfo` function required by the Scratch extension template. The `getInfo` function returns the extension metadata, an object with information about the extension, such as id and name, the list of blocks, and an object of menus used in the blocks. The `Scratch3Tactode` class also has to include a function for each block returned in the `getInfo` function, which is invoked when the block is executed.

In order to obtain an expandable and generalizable system and following the Scratch architecture and the existing extensions, the communication with the external platform is encapsulated in classes, being the extension organized so that each type of platform has a class that will be created when connecting to the platform, abstracting the main class from the management of peripherals and communications. Also, the code for linking to the external platforms was grouped in a specific class named `Peripheral`. This class is responsible for scanning and connecting peripherals, external platforms by various means, Wi-Fi or Bluetooth. Thus, the `Scratch3Tactode` class only has one `connector`, an object of type Peripheral that supports the connection, and one `peripheral` used in each function associated with a block in order to invoke the corresponding function and execute the code referring to the type of platform connected. This way, the inclusion of new platforms is done by creating a new class that includes the implementation of the functions invoked in the `Scratch3Tactode` class.

In terms of communication, still within the extension, a `WiFi` class was created that follows the same criteria as the `BT` and `BLE` existing classes, Bluetooth and Bluetooth Low Energy, respectively. These classes encapsulate the communication via web sockets between the peripheral and the extension, using the JSON-RPC 2.0 protocol.

The external platform used was the R@FL robot, so a corresponding class was created which

implements all the functions invoked in the extension's main class. In addition, the `RAFLRobot` class also includes the protocol for transmitting the program implemented in the Scratch interface to the R@FL robot via XML-encoded messages. The transmission of the program to the robot via XML-encoded messages can contain two blocks, the current and the next to be executed, or three blocks when the current block has branches. This strategy allows the delay of the messages not to affect the robot's execution since it has the necessary information to decide what to execute in conditional blocks.

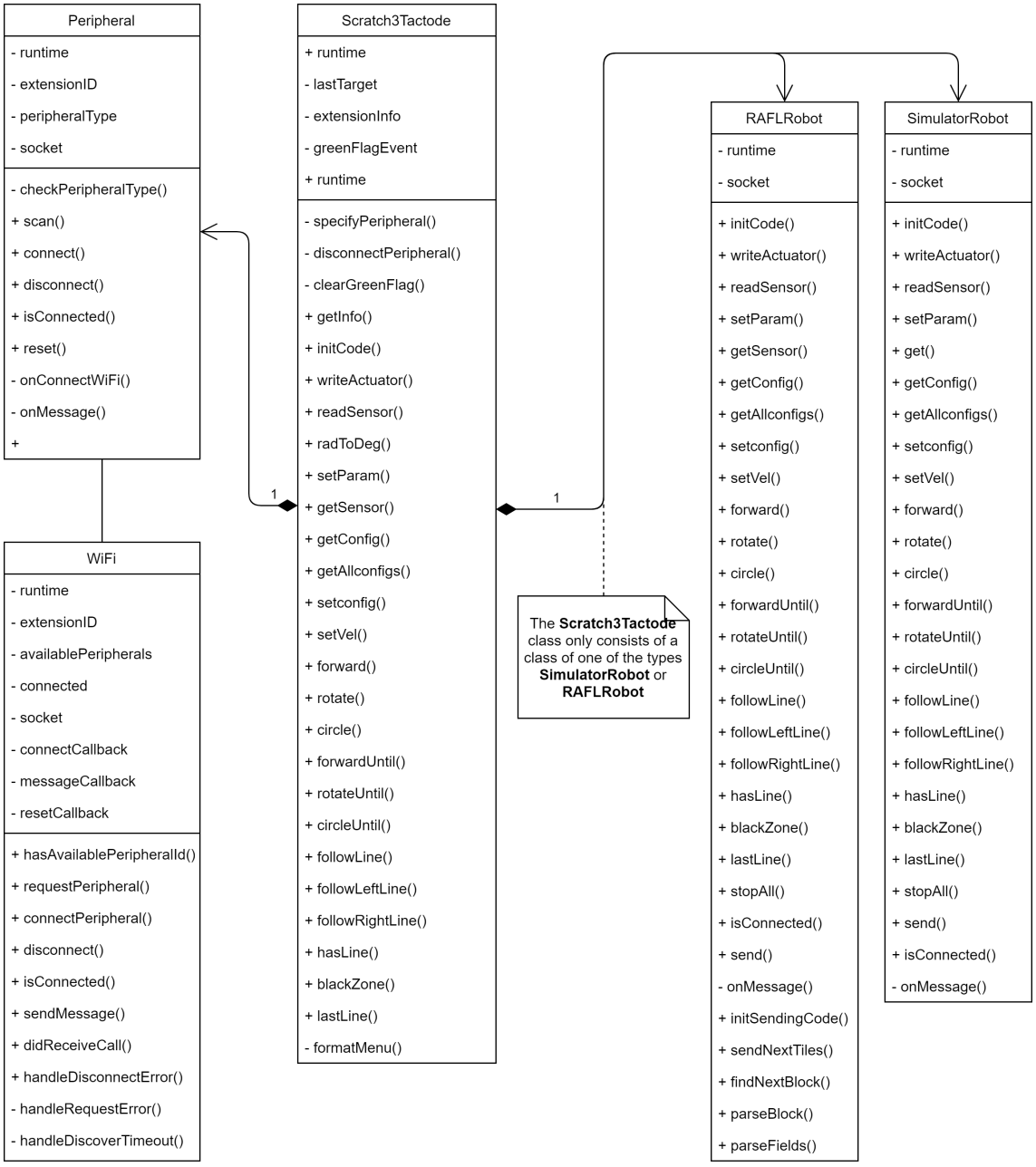


Figure 4.2: UML class diagram of the extension.

4.1.2 Robot

The robot used in this dissertation, the R@FL robot, was modified to consist of an Arduino Nano microcontroller and a Firebeetle ESP32 microcontroller. The Arduino Nano is connected to the robot's sensors. The Firebeetle ESP32 is connected to the motors, thus controlling the robot's movement, and to the Arduino Nano to receive data from the sensors.

In the firmware implementation, the PlatformIO IDE[1] was used, which is a C/C++ embedded development platform with the Arduino framework. The architecture of the robot firmware is divided into three communications classes: ESPComms, ESPBTComms, and ESPWebTerminal. The division allows you to separate the various types of communications established by the ESP32 and separate communications from the execution of the received program.

The ESPComms class encapsulates the communication between the ESP Firebeetle and the Arduino Nano, from where the values of the robot's built-in sensors, such as the line sensor and the distance sensor, are obtained.

The ESPBTComms class integrates the Bluetooth communication enabled by the robot to configure the Wi-Fi connection, that is, to select the Wi-Fi network to which the robot should connect and provide the password that may be required for the Wi-Fi connection to be established. The Bluetooth connection is described later in this document (see section 4.2).

The ESPWebTerminal class encompasses all communication with external devices via web sockets, namely communication with the Tactode Link application or the robot's web terminal, and also Over-the-Air Programming (OTA).

The `main.cpp` file includes the main Arduino functions, setup and loop, and all the functions corresponding to the Scratch and extension blocks, processing incoming messages and encoding information into JSON for sending to the extension.

4.1.3 Communication

The `TaMuES` system is composed of the extension for Scratch, the Tactode Link application, and the firmware for the external platform, which in this work was the R@FL robot. The Tactode Link application mediates the communication between the extension and the external platform. In the diagram in Figure 4.3 it is possible to see the flow of creating and executing a program from the starting point of the `TaMuES` system. The activity starts with adding the Tactode extension to Scratch, and all steps are followed until the program is executed. The diagram also allows us to analyze the communications between the various components of the system. Some actions are not covered in the chart, triggered by the other messages that the robot can receive from the extension, such as the "stop" message.

[1] PlatformIO IDE - https://platformio.org/platformio-ide (accessed Jun. 19, 2021)

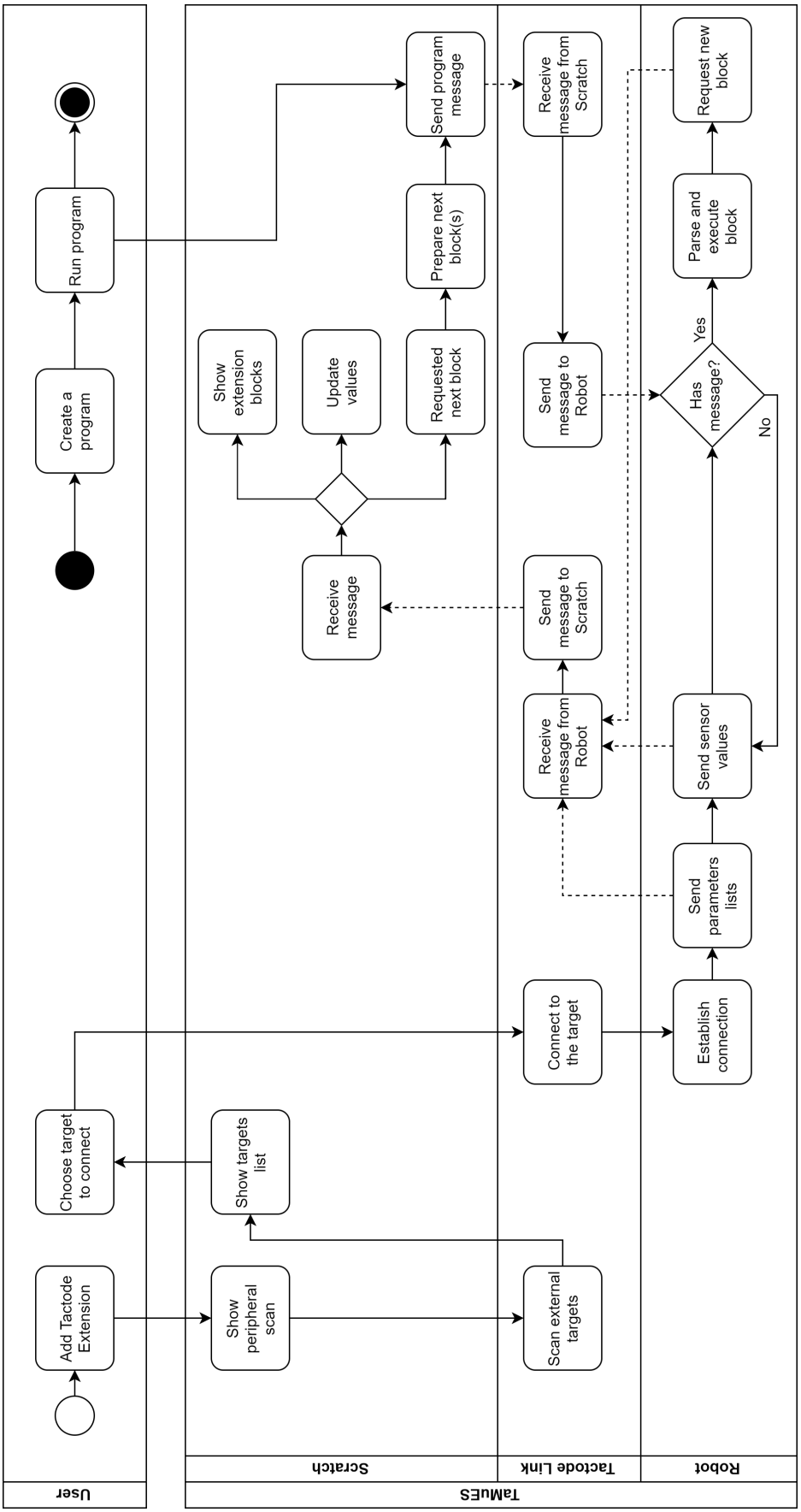


Figure 4.3: TaMuES system swimlane diagram.

The Bluetooth communication developed is represented in Figure 4.4, and the triggered actions and processes can be analyzed. The operation of this communication is described in section 4.2, and the description of the menu and the possible actions. Bluetooth is initialized upon robot startup and is available to connect only for 5 seconds, otherwise it is deactivated. If during those 5 seconds a connection is established, the Bluetooth connection remains active for 30 seconds which are refreshed with each incoming message.

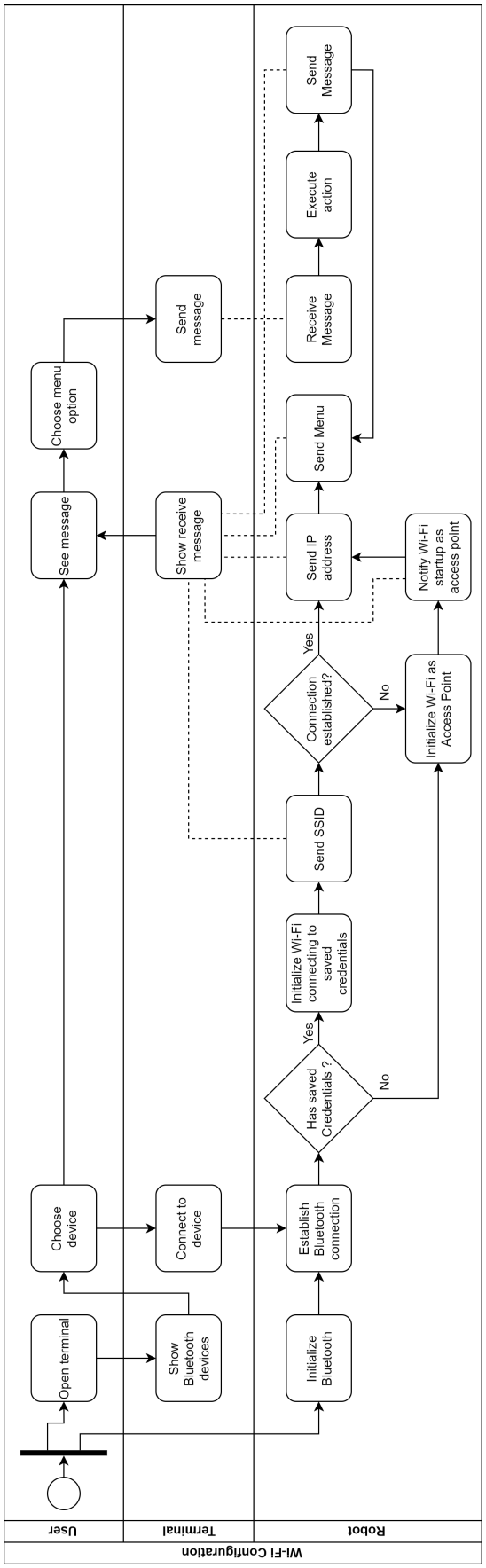


Figure 4.4: Representation of Bluetooth communication for Wi-Fi configuration.

4.2 Case Study Robot - R@FL

During this dissertation, the R@FL robot was improved to be equipped with new functionalities and fulfill the requirements for its use with the new Tactode system. The robot's improvement was made together with the Junior University's team from the electrical engineering area.

For the development of all the firmware, the PlatformIO IDE was used, which is a C/C++ embedded development platform, and the Arduino framework.

The main hardware change included a new microcontroller, the Firebeetle ESP32, with a higher flash and RAM memory capacity and wireless communication technology, Wi-Fi, and Bluetooth. Thus, the robot was left with two microcontrollers:

- The Arduino Nano connected to the robot sensors and the electromagnet, being used for processing the sensor values and sending them to the Firebeetle ESP32;
- The Firebeetle ESP32 is connected to the motors to control the robot's movement and to the Arduino Nano to receive the data from the sensors.

In this distribution, the Firebeetle ESP32 becomes the main microcontroller with all responsibility over the robot.

The communication between the ESP and the Arduino Nano proved to be a challenge given the need for the communication to be synchronous and as effective as possible for the ESP to receive the sensor values and get the robot to act accordingly, with no mismatch between receiving the information and the action taken.

The addition of the ESP32 Firebeetle to the robot allowed the implementation of OTA programming and improved debugging since it is possible to establish a continuous connection between the robot and a device (e.g., computer, smartphone, tablet) without the use of cables.

The first approach to enable a continuous connection between the robot and a device was to initialize the ESP's Wi-Fi as an Access Point to which the device connects to establish communication. This approach blocked the device from connecting to the Internet wirelessly since it is only possible to be connected to a Wi-Fi network, and a cable connection is required for Internet use. To solve this problem, we considered having the robot work as a Wi-Fi repeater, i.e., the robot would connect to the internet router via Wi-Fi, and the user device would connect to the robot in the same way. For this, it was necessary to implement all the Internet request routing, Network Address Translation, IP forwarding, and to take into account that the ESP only allows a minimal number of TCP connections which would also imply the implementation of a TCP/IP stack. We found some libraries that allowed us to implement the microcontroller as a Wi-Fi repeater. However, they used the ESP-IDF framework, which was different from the one we used and was for the ESP8266 microcontroller, an older version.

That said, it was necessary to change to another strategy. Instead of the ESP working as an Access Point, we configured it to connect to the local network and implement communication between devices on the same network. However, it was necessary to find a solution for configuring the network to which the robot should connect, without the need for a cable connection to configure

the Wi-Fi access credentials every time the robot is used in a different network. Thus, it was decided to implement a Bluetooth connection to configure the Wi-Fi credentials and obtain the robot's IP address.

When implementing the solution of connecting via Bluetooth to configure the Wi-Fi credentials, a hardware limitation was detected. The Firebeetle ESP32, although it has both technologies, Bluetooth and Wi-Fi, only has one antenna that is shared by both, but the Bluetooth connection has priority over Wi-Fi in the use of the antenna. When both technologies are in use and receiving messages simultaneously, the ESP gives an error and reboots. This could happen in the case of establishing a connection via Bluetooth with the robot while the Wi-Fi connection was being used. To solve this new problem, it was decided that Bluetooth connection is only available for connection in the first 5 seconds of robot startup. After that time, it only uses Wi-Fi communication.

Bluetooth communication with the ESP can be established through a device with Bluetooth technology using a free terminal application, a Bluetooth Serial Terminal, available for all operating systems (e.g., Android, IOS, Windows, MAC OS).

When the connection via Bluetooth is established in the first 5 seconds of the robot's operation, the ESP starts connecting to the Wi-Fi network by attempting to connect with the credentials it has in memory (Figure 4.4). If credentials are in memory, the SSID is sent to the terminal, and the connection attempt is started. If the connection is unsuccessful or no credentials exist, the ESP initiates Wi-Fi in Access Point mode. When the Wi-Fi connection is created, the IP address is sent to the terminal. Then the menu of available options is sent to the terminal and printed, Figure 4.5, which allows choosing a new network, viewing the IP address, deleting the credentials in memory, restarting the ESP, reconnecting to the Wi-Fi network, and exit. If no Bluetooth connection is established within 5 seconds, the Wi-Fi connection process is carried out in the same way, except that messages are not sent to the terminal.

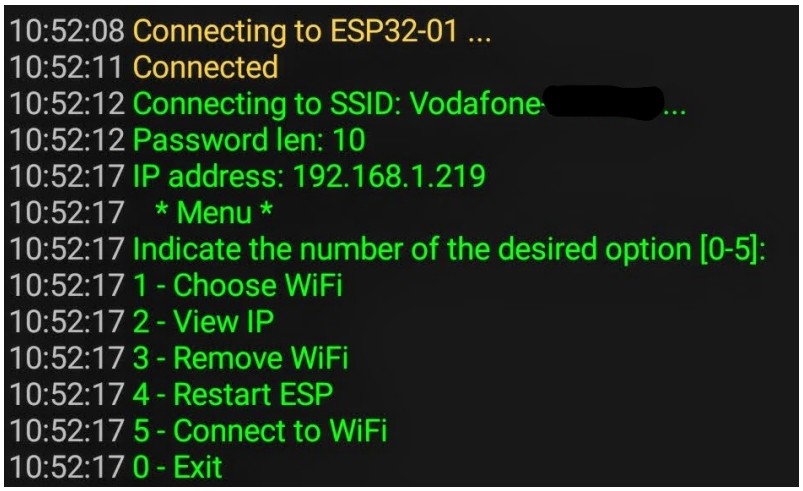


Figure 4.5: Wi-Fi connection setup menu.

Wi-Fi network choices are made by selecting one of the networks captured by the ESP. When the option choose Wi-Fi is selected, the ESP scans the available networks, presents the list, and

asks the user to select the desired one. After selection, the SSID is stored in memory, and if it is a protected network, the user is asked for the password and then saved.

Wi-Fi credentials are stored in memory so that they persist after restarting the ESP. To clear these credentials, it can be through the Bluetooth connection menu or by three consecutive clicks on the ESP reset button. The ability to erase Wi-Fi credentials in memory by restarting the ESP three times consecutively allows forcing the ESP to start Wi-Fi in Access Point mode without the need for a Bluetooth connection, making it faster and more convenient.

For storing data in permanent memory, the `Preferences.h` library is used, which allows storing it in non-volatile flash memory. It was common to use the *EEPROM* library for this purpose, but it has been discontinued, and `Preferences.h` is its successor. Unlike *EEPROM*, `Preferences.h` allows to specify the type of data to store, which is stored in a `key:value` pair inside a previously created dictionary, named "esp_wifi", Figure 4.6. To store Wi-Fi credentials, the namespace "esp_wifi" is used, which contains four pairs of data: "reset_count" is the ESP reset counter so that when three resets are recorded the credentials are deleted; "wifi" is a boolean indicating whether there are credentials stored or not; "wifi_ssid" is the SSID of the Wi-Fi network; and "wifi_pass" is the password of the Wi-Fi network which can be null if the network is opened.

```
esp_wifi {
  reset_count: 0
  wifi: false
  wifi_ssid: "ssid"
  wifi_pass: "pass"
}
```

Figure 4.6: Data structure stored in memory.

The counting of consecutive resets is implemented using the algorithm 1. The "reset_count" value is incremented and kept in memory. If the ESP is reset during the waiting period, the value that remains in memory is the incremented value that will be read at the next startup. If the reset is done after 5 seconds of ESP initialization, the value read from "reset_count" is 0.

Algorithm 1: Consecutive resets counting algorithm

```
initialize ESP;
begin setup function;
initialize Bluetooth;
get reset_count value;
increment reset_count;
if reset_count >= 3 then
    erase credentials;
else
    store reset_count;
    wait 5 seconds;
    store reset_count with value 0;
end
```

The solution created for setting up Wi-Fi via Bluetooth connection turned out to be a solution to an existing problem solved with unpalatable and impractical makeshift solutions. Many times to configure a robot's Wi-Fi connection, the user needed to change the robot's firmware to enter the desired network credentials manually, and the programming environment was required to change the firmware. A strategy was also needed to get the IP address visible only after the connection. To get the IP address, we could either add a display to the robot, increasing costs unnecessarily or cable the robot to a computer to monitor debug messages. Now, setting up the internet connection becomes accessible and intuitive and can be done by any user without a programming environment. The only requirement is a Bluetooth-enabled device and a terminal obtained for free from the store.

The Junior University team has also developed a debug console for web browsers, the Web Terminal, that allows communication with the robot. This terminal prints out all the messages received from the robot and also allowed for sending messages. It can be accessed by entering the robot's IP in the browser.

Due to the memory and processing limitations of the microcontroller, the incoming and outgoing messages have a maximum value that cannot be exceeded. Thus, sending the extension program to the robot in order for it to execute it cannot be done all at once in a single message that includes the entire program. There was the possibility of sending only small commands with the robot's actions, but this implied that the processing and decision-making would be done by Scratch and not by the robot. This way, the communication delay between Scratch and the robot could lead to wrong decisions. The solution to this problem was to send a message that includes the extension blocks to the robot, coded in XML. Thus the processing and decision making becomes the robot's responsibility. The messages that send the blocks are composed of the current block and the next one(s) to be executed. The robot processes the current block and then requests the next block from the extension indicating its identifier, and the extension sends the requested block and the next one. If there are blocks with branches, namely cycles or conditions, the message is made up of three blocks instead of two, the first block being the conditioning block, and

then the subsequent two blocks are the two execution possibilities that depend on the veracity of the condition. After processing the conditioning block, the robot sends the request for the next chosen block. The execution of the current block by the robot is performed in a new thread to allow the block execution and the messages exchange simultaneously and the request and receipt of the next block while the current one is executed.

Appendix A describes the Extensible Markup Language (XML) used to encode the blocks sent to the robot, and examples of messages sent are shown. We can find messages with two and three blocks in the example messages due to the "repeat" block present in the code and blocks with inputs that are also a block. The latter case is due to the inclusion of the result of a division as input for the rotation value.

Sending the program from the Scratch extension to the robot could be done all before the robot executes the program. Even with the limitations of the size of the messages, we could divide them into several messages taking into account this limit. However, this strategy is not feasible due to the high message traffic imposed before the program execution and the risk of microcontroller memory overflow if the program were extensive. Thus, the strategy used allows splitting the communication traffic throughout the program execution and keeps the microcontroller memory under control since the message size is limited.

In this dissertation, the case study was the R@FL mobile robot, but the software was developed to be generic and applicable to several types of targets.

4.3 Firmware

The developed firmware includes the implementation of the functions corresponding to the Scratch blocks, the handle, and the sending of messages, in addition to the Wi-Fi and Bluetooth connections described in the previous sections.

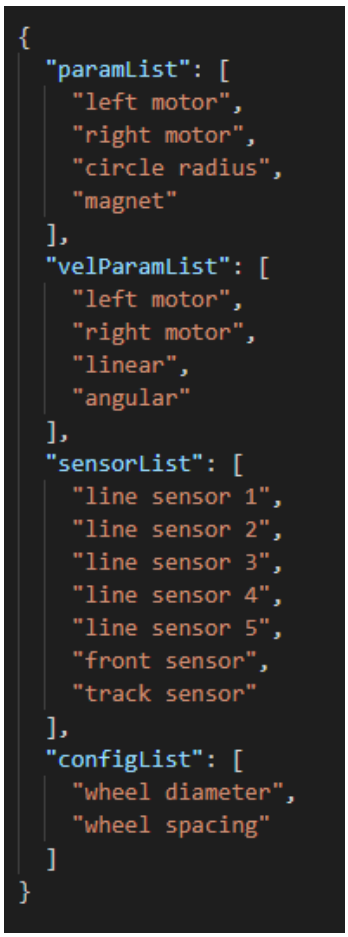
When the connection via Wi-Fi is established between the robot and the extension, the robot sends an initial message with the lists of editable parameters, sensors, and settings that the user can change and access via the extension. This message was created to allow the extension to be used by a more significant number of targets and with a great diversity of functionalities since the message indicates the components and sensors available by the target, which can be more or less equipped without any prejudice to the extension user.

Considering the case under study in this dissertation, the R@FL robot, the initial message (Figure 4.7a) includes:

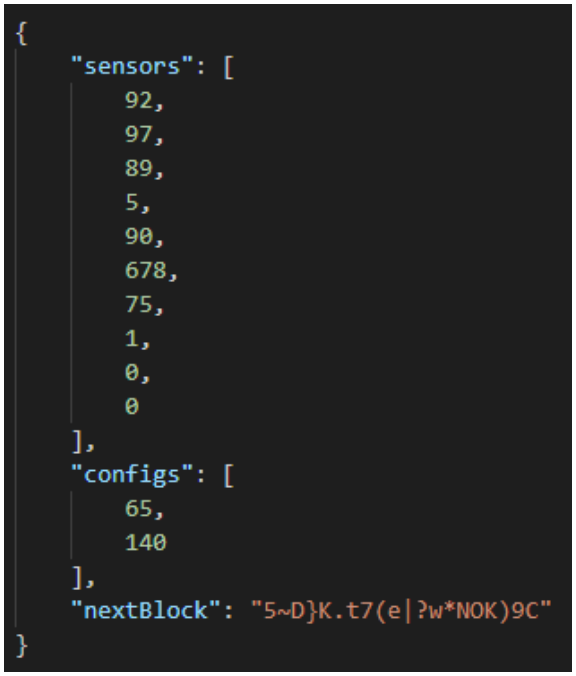
- `paramList` - the list of editable parameters in open loop, being changed with a value in percent;
- `velParamList` - list of closed-loop editable parameters, essentially velocities assigned in m/s units;
- `sensorList` - list of the sensors present in the robot;

- `configList` - list of robot configuration parameters that can change its motion, namely wheel diameter, and wheel spacing.

After the initial message at connection startup, the robot sends every three executions of the `loop` function the values that can be queried through the blocks (see section 4.5.1), namely the values of the sensors and the robot configurations to the extension (Figure 4.7b). In addition to these values, the message also includes the ID of the next desired block, if any.



(a) Initial message.



(b) Message with sensor values and requested block.

Figure 4.7: Messages sent to Extension.

There are three types of messages that the robot reads. These types of messages are shown in table 4.1.

To process the blocks received from the extension encoded in XML, a *TinyXML-2* library is used. The advantages of this library that led to its election are its simplicity, size, efficiency, and easy integration.

In order to organize and facilitate the mapping between the opcodes of the extension blocks and the implemented functions, a data structure was created that matches the opcode of a block to

Table 4.1: Types of Messages

Message	Description
`stop`	Stops all robot motion and processing.
`CMD:`*`op`*`:`*`v1`*`:`*`v2`*	It's a command message that executes the *op* using the values *v1* and *v2*. *op* can take the values: • *write* - to write the value *v2* to pin *v1*; • *read* - to read a value from pin *v1*; • *setParam* - change the *v1* parameter to the *v2* value; • *setConfig* - change the *v1* configuration parameter to the *v2* value; • *setVelParam* - change the *v1* velocity parameter to the *v2* value. Message Example: `CMD:setParam:0:50`
`XML:`*`code`*	It is a code message with the blocks encoded in *code*. The blocks are parsed and stored for processing.

the pointer of the desired function. This data structure presents the opcodes of all the blocks implemented by the robot: the Scratch blocks of the extension, the Control section, and the Operators section.

Processing the incoming XML message from the extension involves creating a new task if the opcode of the first received block exists in the data structure or sending a warning message to the extension (Listing 4.1), followed by requesting the next block if the message includes more than one block. Creating a new task for executing the function of a block allows the use of the microcontroller's multitasking advantage so that the next block is received while processing the previous one. This improves communication and allows the continuous sending of sensor information to Scratch, enabling debugging of program execution.

```
1  if (opcodeMap.count(firstBlock->FindAttribute("opcode")->Value()) == 1) {
2    auxBlock = firstBlock;
3    xTaskCreate(
4      opcodeMap[firstBlock->FindAttribute("opcode")->Value()],
5      firstBlock->FindAttribute("opcode")->Value(),
6      2000,
7      NULL,
8      2,
9      &taskHandle);
10   vTaskDelay(10/portTICK_PERIOD_MS);
11 } else {
12   snprintf(message, MAX_SIZE, "{\"warning\":\"Block '%s' not executed\"}",
         firstBlock->FindAttribute("opcode")->Value());
13   terminal.print(message);
14 }
```

Listing 4.1: Block processing

During the robot's operation, a task responsible for changing the speed of the motors is also executed. When the robot's speed is altered through the parameter change blocks, the power sent to the engines varies gradually to avoid abrupt starts and stops of the motors.

4.4 Tactode Link

The connection between the Scratch extension and the external platform is mediated by the Tactode Link application, similar to the existing Scratch Link application. However, Scratch Link only allows connection to platforms via Bluetooth. In order to cover a broader range of platforms, Tactode Link supports communication via Wi-Fi.

The Tactode Link was implemented in Javascript like the extension and is a web socket server that the extension connects.

The connection between the extension and the Tactode Link is made by creating a socket in the extension that connects to local port *8080*, the port where the Tactode Link server is located and includes the desired connection type. When the type of connection is "WiFi", the application scans the devices connected on the same network and tries to establish a connection via web socket with port *1337* of each device found. Port *1337* is the port used by the robot's web socket server, so if communication is established after 5 seconds, the device is considered elected as a possible external platform. Thus a "didDiscoverPeripheral" message is sent with the results to the extension so that it can present them to the user. Besides this device scanning, when it comes to Wi-Fi connection, the application also scans the available Wi-Fi networks and, in this case, looks for networks beginning with "RAFL", also sending the valid networks to the extension. This search for wireless networks is because when the robot does not connect to a Wi-Fi network, it starts in Access Point mode creating a private network (see section 4.2).

When the user chooses the platform to which they want to connect, the information needed to establish the connection is sent to the application. The message includes the IP address and the port to create the web socket between the application and the external platform, and a boolean indicating if it is an Access Point that, if so, make the message also include the network name and password. The socket created with the external platform forwards all incoming messages to the extension under the JSON-RPC protocol with the method "didReceiveMessage". All messages sent by the extension are disassembled and sent to the peripheral in plain text.

The Tactode Link application runs only in the background, and only user intervention is required for its initialization. During execution, it neither requires nor enables user intervention.

At the conclusion point of this dissertation, the Tactode Link application only allows the connection to devices via Wi-Fi, being left for future work the improvement to include the link to devices via Bluetooth.

4.5 Extension

The extension developed for Scratch allows the programming of external platforms using the block programming method present in Scratch. For users of the original Scratch, the adaptations required to initiate external platform programming are practically nil. In the Scratch programming environment, adding the extension is done through the "Add Extension" button present in the lower-left corner, which leads to a list of extensions where the "Tactode" extension can be found and selected, Figure 4.8. By selecting the extension, the search for external platforms through the Tactode Link application will start, listing the targets located and allowing selecting the desired platform, Figure 4.9. After selection, the connection is established, Figure 4.10, and the block list is updated.

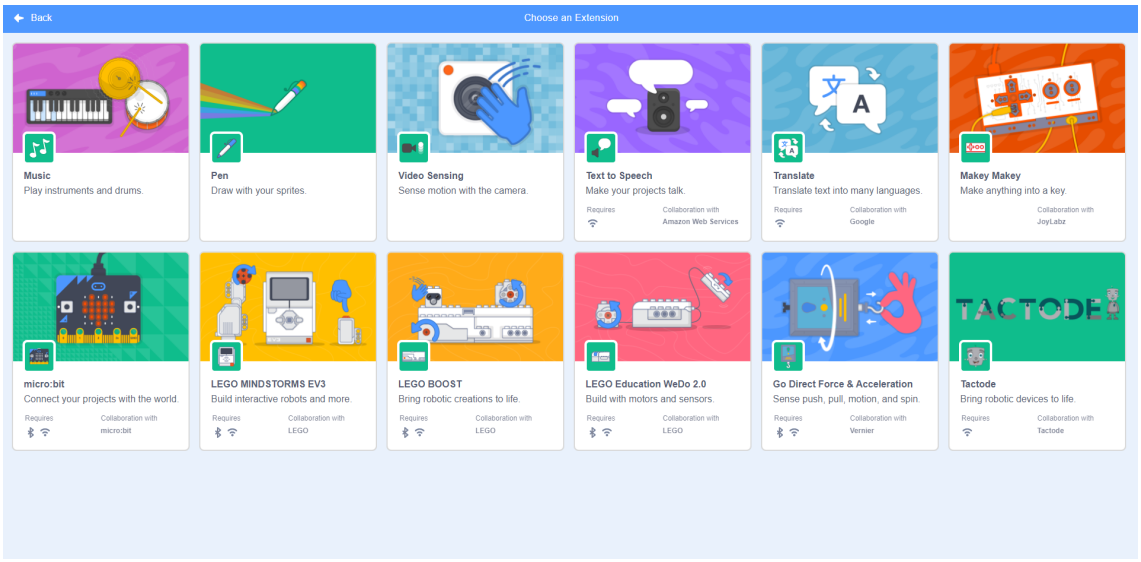


Figure 4.8: Menu to add an extension to Scratch.

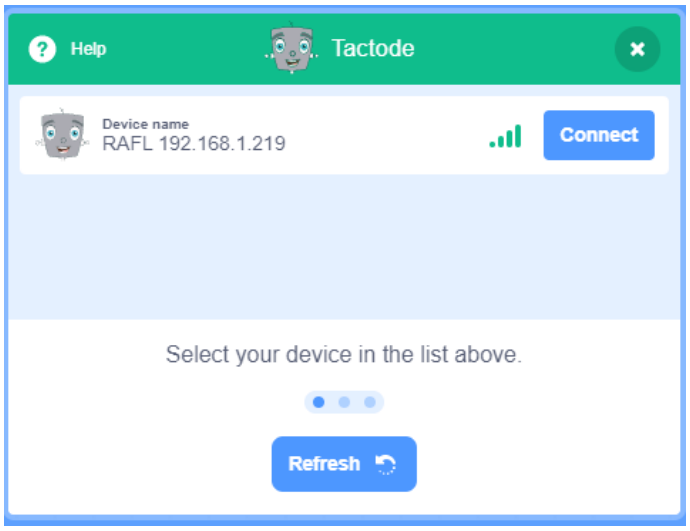


Figure 4.9: Result of scanning by external targets.

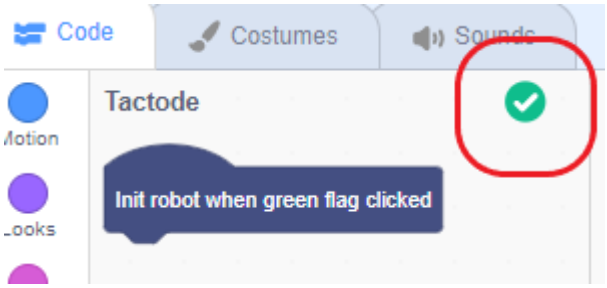


Figure 4.10: State of the connection between the extension and the target.

The constituent blocks of the extension can be divided into three levels of abstraction.

- Abstraction level 0 has only two blocks for writing and reading information from the microcontroller pins, which is the lowest programming level.
- Abstraction level 1 already includes getting preprocessed and well-identified information, such as through the blocks in Figures 4.19 and 4.20, and changing parameters using percentage values.
- Abstraction level 2 includes more easily understood programming blocks for younger users, including more assertive control instructions, such as "robot forward 10 cm until *condition*" or "robot follow line until it loses it", and changing parameters using values in units of the International System of Units.

The three levels of abstraction allow for different types of robot control. The first allows generic programming without dependence on the physical settings of the robot. The second allows open-loop control of the robot, changing the settings and obtaining the value from the sensors for visualization and debugging. The last level allows closed-loop control of the robot, requiring encoders on the robot for the correct execution of the blocks.

In addition to the blocks created in the extension, the robot programming can and should include other blocks that already exist in Scratch. Only the blocks that do not have a direct action on the Scratch simulator should be used. That is, the blocks in the Control and Operators sections are generic and can be used without having an effect on the actor in the Scratch simulator. If other blocks are used, what can happen is that these blocks are not executed. In the case of R@FL, in addition to the blocks that the robot has no corresponding implementation not being performed, a warning message is shown indicating the block that was not executed, Figure 4.11.

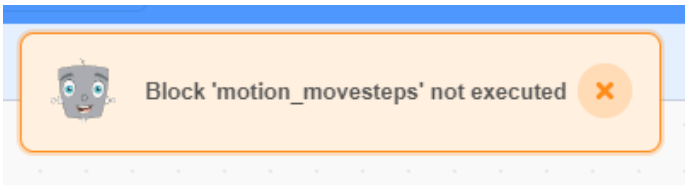


Figure 4.11: A warning message that `motion_movesteps` block was not executed.

The extension makes it possible to program external platforms using Scratch programming regardless of their origin, making Scratch more inclusive. Scratch was limited to programming its simulator and programming some LEGO robot models and Microsoft's micro:bit. With this extension, the range of target possibilities is much broader since all that is needed is for the external platform to have Wi-Fi communication, a limitation that can be eliminated in future work. The extension does not depend on the target platform. It is so generic that the platform can contain several sensors and actuators, and it's still possible to program all of them using the extension without any changes in their operation. In this way, we can say that the extension allows the programming of robots that may still be created in the future, ensuring their longevity in future times.

4.5.1 Blocks

In total, 22 blocks were created that make up the Tactode extension. As already stated, these blocks can be divided into three levels of abstraction that encompass 20 of these blocks. The two remaining blocks are:

- Figure 4.12 is radian to degree conversion block since the unit requested in the blocks is degrees;
- A code initiation block, Figure 4.13, has the same effect as the `GreenFlag` block already in Scratch but functions as the initiation of the transfer of the program to the external platform.

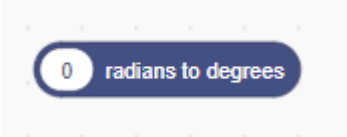


Figure 4.12: Radian to Degree conversion block.

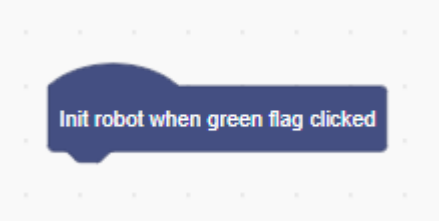


Figure 4.13: Code initiation block.

Abstraction level 2 includes a total of 10 blocks. This level allows closed-loop control and consists of blocks that allow:

- Change of velocity with values in units of the International System of Units, Figure 4.14;
- Move the target strictly forward, rotating around itself and in an arc of a circle until the indicated measure or until a condition is reached, Figure 4.15;

- Follow a line straight ahead, always to the left or always to the right at intersections, until the line is lost, Figure 4.16;
- To know if the target is on the line or if it is in a black zone and whether it has lost the line to the right or not, Figure 4.17.

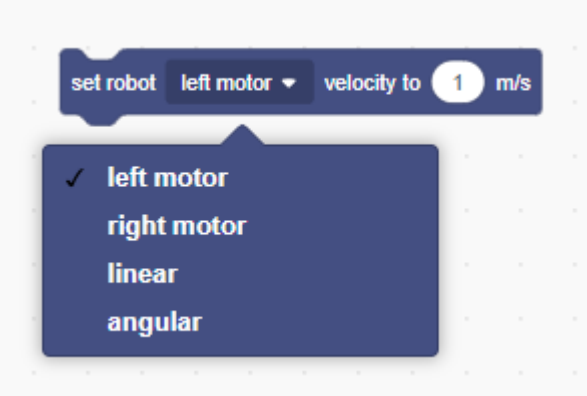


Figure 4.14: Set velocity block with the list of editable velocities.

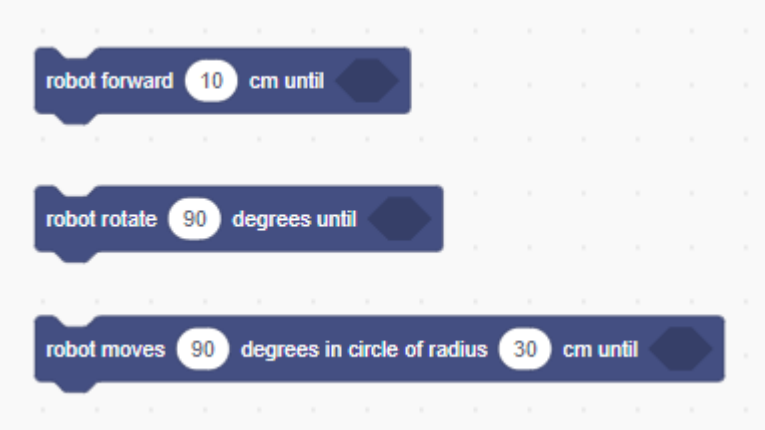


Figure 4.15: Block Move Forward Until at the top, block Rotate Until in the middle, and block Move in Circle Until at the bottom.

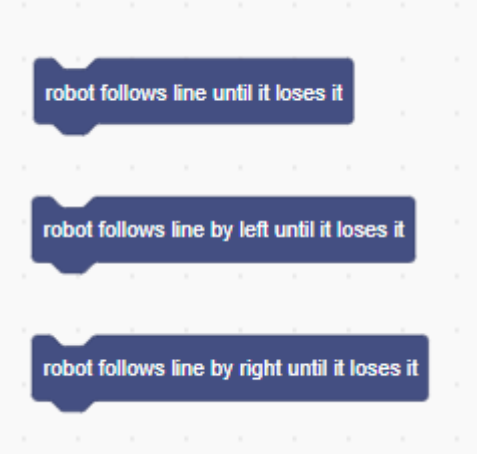


Figure 4.16: The three Follow Line blocks

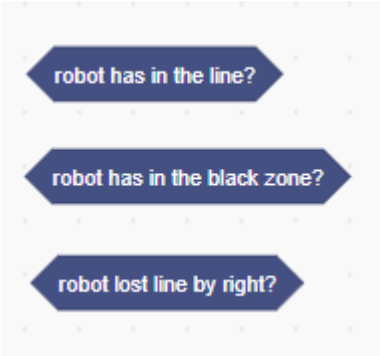


Figure 4.17: The three condition blocks

Abstraction Level 1 includes eight blocks that allow:

- Changing various parameters with a percentage value, Figure 4.18;
- Obtaining the value of the chosen sensor from the sensor list, Figure 4.19;
- Getting and changing the value of the chosen configuration from the configuration list, Figure 4.20;
- Strictly moving the target forward, rotating around itself and in an arc of a circle, Figure 4.21.

At abstraction level 0, it consists of two blocks, Figure 4.22, one for writing values and the other for reading information from pins. The pins can be the microcontroller's pins or virtual pins defined by the external platform. In the robot understudy, the R@FL, we can consider the ESP32 microcontroller pins and, as virtual pins, the Arduino Nano microcontroller pins. This level of abstraction allows programming equivalent to firmware creation since it is based on reading and writing information directly to the platform components connected to the pins.

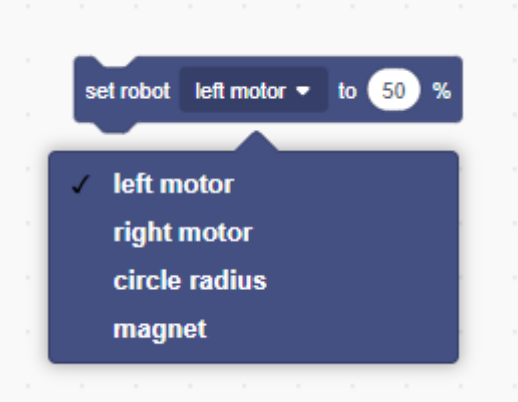


Figure 4.18: Set parameter block to percentage value and the list of editable parameters.

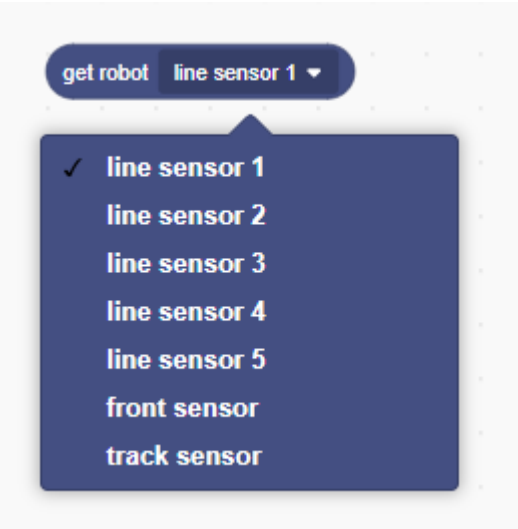


Figure 4.19: Get sensor value block and the list of available sensors.

The lists shown in the blocks are the lists received from the R@FL robot when the connection is established, so it is possible to change the robot, for example, adding a new sensor, without needing to change the extension, just changing the firmware.

The extension created differs from existing ones, as already mentioned, for being generic and allowing the programming of several external platforms. Besides this, we can add as a differentiating factor the levels of abstraction provided by the various blocks that let programs be created in several different ways and with varying degrees of difficulty. In this way, the extension opposes LEGO extensions that only allow the programming of the model for which they were created and using only one block per action while maintaining a constant level of difficulty and abstraction.

4.6 Acceptance Experiences

To validate and test the developed system, we decided to select a set of challenges that must be created and executed using the system. The challenges have increasing complexity and constitute a

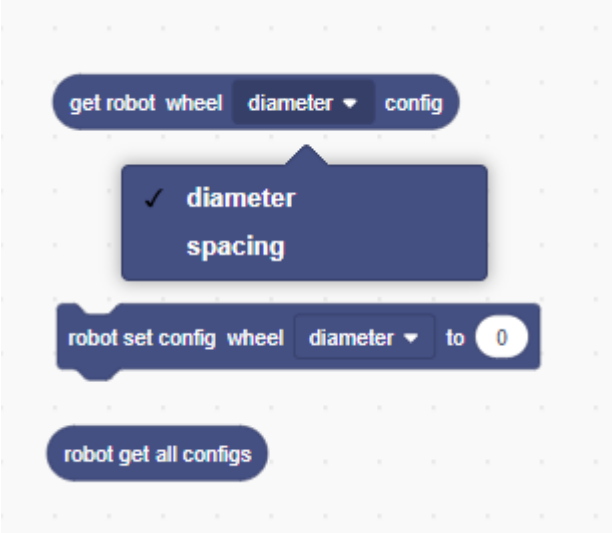


Figure 4.20: Get configuration block at the top, Set configuration block in the middle, and get all configurations block at the bottom.

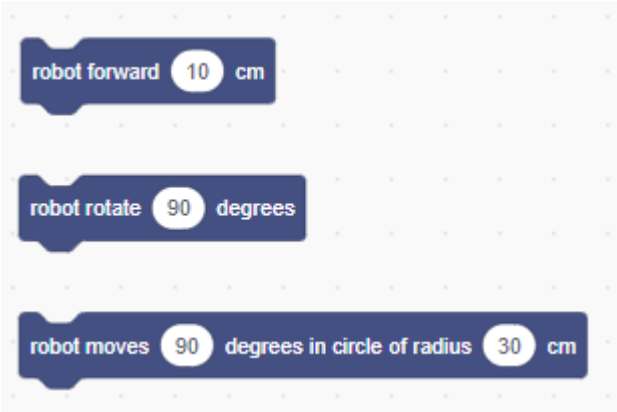


Figure 4.21: Move Forward block at the top, Rotate block in the middle, and Move in circle block at the bottom.

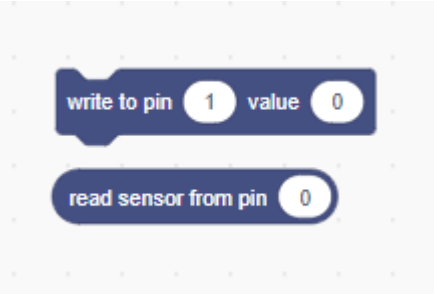


Figure 4.22: Write value to pin and read value from pin blocks.

goal that the target must meet. These challenges also act as acceptance tests for the `TaMuES` system and evaluate if the system meets the requirements previously defined. As already mentioned, the external platform used in the development of this work was the R@FL robot, and as such, it was

also the platform used to test the application.

The main challenges are:

- Get around an obstacle;
- Draw a regular polygon;
- Follow a line.

In order to test the system more gradually, smaller tasks were performed at first, increasing the complexity and testing the various blocks.

The first tasks performed were to move the robot forward, rotate, and move in a circle and were solved with the simple combination of two blocks, the start block and a movement block, Figures 4.13 and 4.21. With the completion of these simple tasks, we were able to move on to solving the first two challenges since their solution can be constituted by the combination of the blocks used previously. The solution to the first challenge consists of the combination of blocks used in the tasks. A possible solution is shown in Figure 4.23 and combines only the walking forward and spinning blocks. The second challenge follows the same path, although it can be optimized by introducing a loop block. In Figure 4.24, we can see the solution for drawing a regular polygon, in this case, a pentagon whose sides measure 30cm.

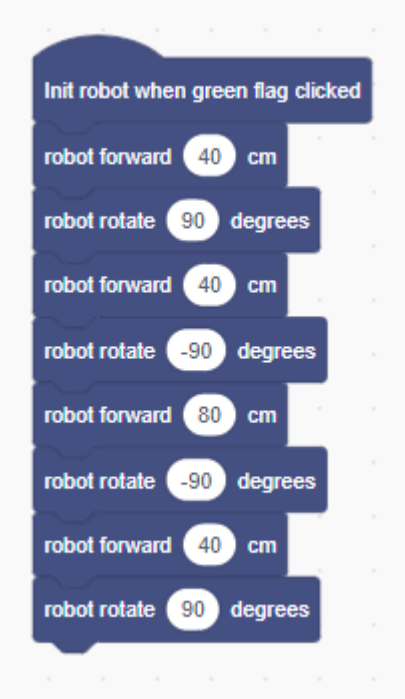


Figure 4.23: Program to go around an obstacle.

The third challenge can be implemented using only the Follow Line block that makes the robot follow the line until it loses it. The challenge was successfully completed in following straight lines and also in following lines with some slight curves. However, the following lines

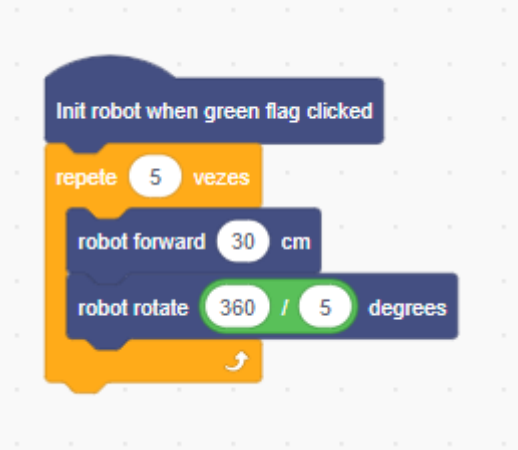


Figure 4.24: Program to draw a regular polygon.

with sharper curves were unsuccessful due to the line following algorithm implemented in the robot's firmware. As future work, we can consider improving the line following algorithm.

The strict motion of the R@FL robot is not correct, that is, the R@FL robot does not meet the exact measurements indicated in the strict motion blocks. This is due to the lack of encoders on the robot motors, making it impossible for the microcontroller to obtain the displacement made by the robot. Thus, the movement is implemented approximately, being this movement only the result of the activation of the motors during a time calculated by multiplying the distance to be traveled by a constant and divided by Pulse Width Modulation (PWM). The constant used in the calculations was obtained approximately theoretically through the relationship between speed in m/s and the PWM of the motors and adjusted empirically. Although the calculation was adjusted to correspond to reality, some factors can alter the result, namely the charge of the robot's batteries.

A pedagogical plan was created to be applied to the users, also to test the system. The plan includes five activities that the users should perform in order to test the system and evaluate the difficulties and interests of the users. The activities in the plan are:

1. Set the robot to move forward for 5 seconds and then stop;
2. Bypass an obstacle that is in the path between two points, Figure 4.25;
3. Draw a circle;
4. Draw a rectangle;
5. Draw a triangle.

It is considered to "draw" the robot's movement on the ground, so drawing a circle is to make the robot move in a circle.

The pedagogical plan was applied to two test users, two children aged 9 and 11. The limited number of users was due to the limitations and restrictions imposed by the Covid-19 pandemic.

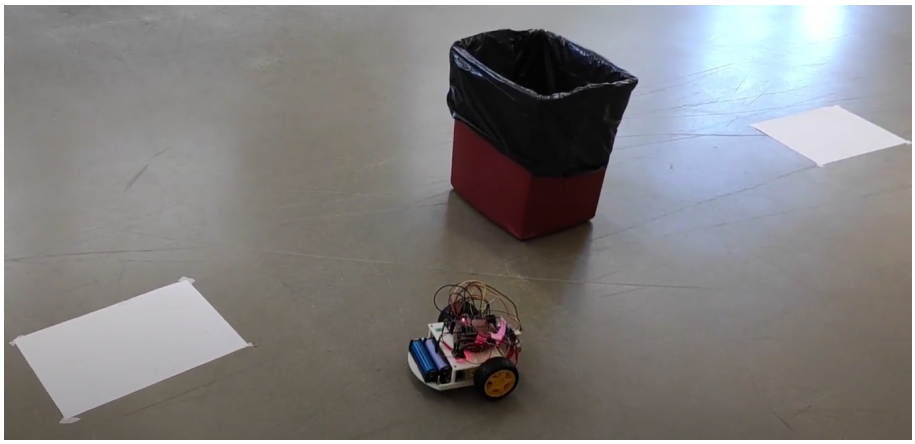


Figure 4.25: Challenge to go around an obstacle on the path between two points.

4.7 Results

For an hour and a half, the two children tested the system and worked out the five activities that made up the pedagogical plan. The elaboration of the activities was done in groups by the two children, with the computer control alternating between them at each new activity. The children managed to complete all the activities, not always on the first try, but they could easily find the error and correct it with a few small tips.

A survey with seven simple questions was prepared and asked to each of the children. The survey included the following questions:

1. Have you used Scratch before? If yes, did you like Scratch when you used it?
2. Have you programmed any robots before? If yes, did you learn anything new now?
3. Did you understand the purpose of the activity? (1-nothing, 5-completely)
4. Did you find the Tactode blocks easy to understand? (1-very difficult, 5-very easy)
5. Did you find the Tactode extension easy to use? (1-very difficult, 5-very easy)
6. Did you like the moving “Cat” or the real robot better?
7. Did you think the “Cat” looked like the real robot?

The answers obtained are shown in Table 4.2, where children 1 and 2 are the nine and eleven-year-old children, respectively.

By analyzing the answers, we realize that we have two different cases, one child who had already used the Scratch tool and another who had not, although both had already programmed robots. Both were able to understand the activity and found the Tactode extension easy to use. However, when comparing the Scratch simulator (the “Cat”) and the real robot, the choice was obvious, the children preferred the robot.

Table 4.2: Survey Responses

Question	Child 1	Child 2
1	No	Yes, I liked it.
2	Yes. Yes.	Yes. Yes.
3	5	5
4	3	3
5	5	4
6	Robot	Robot
7	No	No

It is noteworthy that the answers to question 4 did not correspond to the intended reality since the language present in the blocks, English, was not the children's native language, Portuguese. So it was necessary to translate what each of the pieces said.

It is essential to emphasize the impact of the results obtained in that the 9-year-old child had never programmed with Scratch before and yet could complete the challenges posed and understand the robot's results and actions.

In addition to the challenges, with high enthusiasm, the 11-year-old child came up with a new challenge of drawing the first letter of her name. Thus, after completing the activity challenges, she wanted to try by herself to solve this challenge. The letter in question was an "M" that was drawn with great precision and commitment. This attitude shows the high interest that the system arouses in children.

4.8 Summary

This chapter presents the system developed within this dissertation, the TaMuES system, which consists of the Tactode extension for Scratch, the Tactode Link application, and the firmware of the R@FL robot. The system is described from its architecture to its operation. Some of the tests done and the experiment carried out with two children to test the system are also presented. Although there are some flaws detected, the experience with the children showed that the system is promising and can be very beneficial for education, increasing interest in STEM areas.

Chapter 5

Conclusions

The Tactode programming system is designed to teach programming and help develop computational thinking in schools at various grade levels. In addition, the use of robots and other targets increases interest in the system and in STEM areas.

This dissertation has the broad goal of providing a block-based programming system for physical or virtual multi-target, captivating the interest of students of a wide range of ages to be included in educational institutions. In this way, the Tactode system has been reinvented. The new Tactode system, `TaMuES` (Tactode Multi-target Extension for Scratch), is an extension of Scratch that has merged the goals and benefits of the original Tactode and the Scratch tool.

`TaMuES`, after this work, includes a programming language consisting of several blocks that can be divided by levels of abstraction. It allows the programming of external platforms, such as the R@FL robot, directly from the Scratch tool and continuously receives and visualizes the values and information collected by the robot's sensors.

The solution's architecture allows to easily add new targets with different communication strategies without the need for significant changes in the extension. For example, if the new target follows the same communication strategy - connecting via Wi-Fi using web sockets and receiving the programming blocks encoded in XML - no change in the extension is required, it is only necessary to check and adjust the R@FL firmware to the new target. The extension developed is generic enough to allow programming for platforms that may be created.

5.1 Future Work

Although `TaMuES` is already functional and able to meet its objective, it is possible to point out the improvements needed to increase the system's value. There are generic and global improvements and others more specific ones that were detected during the work.

The `TaMuES` system does not include one of the advantages of the original Tactode system, the tangible programming. In order to make `TaMuES` more complete, the next development step for

this system is to create tangible pieces of programming similar to those presented in the graphical programming environment, Scratch, and add the functionality of adding the tangible program to the graphical environment for compilation and execution. This improvement would make `TaMuES` a visual and tangible programming system, increasing its advantage for educating the young.

One of the improvements needed to increase the range of targets possible to program using the system is enhancing the Tactode Link application to include the connection of targets via Bluetooth. This enhancement makes it possible to program external targets with only Bluetooth communications and no Wi-Fi.

In addition to these, it is possible to think of an improvement that would enable communication and interaction between students even at a distance. This consists of turning Tactode into a collaborative system, a programming environment that allows more than one user to change the program and see the changes made by the others. As an example, we can consider Google Docs or a different approach such as Git.

In the developed firmware, it is also possible to identify some necessary improvements, such as:

- allow to set the angular velocity as the linear velocity is set, through the setVelParam block and indicating the velocity value;
- equipping the robot with encoders to ensure that the strict motion blocks are executed precisely;
- improving the line following algorithm so that the robot travels along a line with sharp curves without deviating from the trajectory.

Appendix A

XML

This appendix describes the language used to encode the blocks sent to the robot.

A.1 Tags and Attributes

When coding the blocks, it was only necessary to create two tags:

- Block — `<block id="" opcode=""></block>`
- Input — `<input type=""></input>`

The `block` element contains the *id* and *opcode* attributes which are, as the name implies, the id and opcode of the coded block. This element can contain `input` elements if the block has any data fields, either by option selection or by typing.

The `input` element contains only one *type* attribute, which can take several values among them: *NUM, TEXT, OPERATOR, PARAM, block*. When the type is "*block*", it means that the input is another block. This happens in case use blocks of type *Reporter* — which return a numeric value such as the sum-of-two block — in a block input.

Since each message includes more than one block, a root element `code` was also created that encompasses several `block` elements.

A.2 Example

To show the use of the language in practice, the code presented below shows all the messages sent to the external platform during the execution of the code presented in Figure 4.24. Each message starts with "XML:". In total, 16 messages were sent to the target.

```
XML:<code>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
```

```
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>

XML:<code>
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="?(G4``51Hhc{HgkBBoqM" opcode="tactode_rotate">
        <input type="block">
            <block opcode="operator_divide">
                <input type="NUM">360</input>
                <input type="NUM">5</input>
            </block>
        </input>
    </block>
</code>

XML:<code>
    <block id="?(G4``51Hhc{HgkBBoqM" opcode="tactode_rotate">
        <input type="block">
            <block opcode="operator_divide">
                <input type="NUM">360</input>
                <input type="NUM">5</input>
            </block>
        </input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>

XML:<code>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>

XML:<code>
```

```
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="?(G4``51Hhc{HgkBBoqM" opcode="tactode_rotate">
        <input type="block">
            <block opcode="operator_divide">
                <input type="NUM">360</input>
                <input type="NUM">5</input>
            </block>
        </input>
    </block>
</code>

XML:<code>
    <block id="?(G4``51Hhc{HgkBBoqM" opcode="tactode_rotate">
        <input type="block">
            <block opcode="operator_divide">
                <input type="NUM">360</input>
                <input type="NUM">5</input>
            </block>
        </input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>

XML:<code>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>

XML:<code>
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="?(G4``51Hhc{HgkBBoqM" opcode="tactode_rotate">
        <input type="block">
            <block opcode="operator_divide">
                <input type="NUM">360</input>
                <input type="NUM">5</input>
            </block>
```

```
        </input>
    </block>
</code>

XML:<code>
    <block id="?(G4``51Hhc{HgkBBoqM" opcode="tactode_rotate">
        <input type="block">
            <block opcode="operator_divide">
                <input type="NUM">360</input>
                <input type="NUM">5</input>
            </block>
        </input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>

XML:<code>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>

XML:<code>
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="?(G4``51Hhc{HgkBBoqM" opcode="tactode_rotate">
        <input type="block">
            <block opcode="operator_divide">
                <input type="NUM">360</input>
                <input type="NUM">5</input>
            </block>
        </input>
    </block>
</code>

XML:<code>
    <block id="?(G4``51Hhc{HgkBBoqM" opcode="tactode_rotate">
        <input type="block">
            <block opcode="operator_divide">
                <input type="NUM">360</input>
```

```
                <input type="NUM">5</input>
            </block>
        </input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>

XML:<code>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>

XML:<code>
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="?(G4``51Hhc{HgkBBoqM" opcode="tactode_rotate">
        <input type="block">
            <block opcode="operator_divide">
                <input type="NUM">360</input>
                <input type="NUM">5</input>
            </block>
        </input>
    </block>
</code>

XML:<code>
    <block id="?(G4``51Hhc{HgkBBoqM" opcode="tactode_rotate">
        <input type="block">
            <block opcode="operator_divide">
                <input type="NUM">360</input>
                <input type="NUM">5</input>
            </block>
        </input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>

```

```
XML:<code>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
    <block id="5~D}K.t7(e|?w*NOK)9C" opcode="tactode_forward">
        <input type="NUM">30</input>
    </block>
    <block id="AKQS6dh+5{vs]v9.F9ek" opcode="control_repeat">
        <input type="NUM">5</input>
    </block>
</code>
```

Listing A.1: XML messages of execution of code in Figure 4.24.

References

[1] Martina Benvenuti, Augusto Chioccariello, and Giorgia Giammoro. Programming to learn in Italian primary school. In *ACM International Conference Proceeding Series*, WiPSCE'19, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3361721.3361732.

[2] Marina Umaschi Bers. Coding, playgrounds and literacy in early childhood education: The development of KIBO robotics and ScratchJr. In *IEEE Global Engineering Education Conference, EDUCON*, volume 2018-April of *IEEE Global Engineering Education Conference*, pages 2094–2102, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2018. IEEE. doi:10.1109/EDUCON.2018.8363498.

[3] Flor A. Bravo, Alejandra M. Gonzalez, and Enrique Gonzalez. A review of intuitive robot programming environments for educational purposes. In *2017 IEEE 3rd Colombian Conference on Automatic Control, CCAC 2017 - Conference Proceedings*, volume 2018-January, pages 1–6, 2018. doi:10.1109/CCAC.2017.8276396.

[4] Pablo Cardenas Caceres, Renato Paredes Venero, and Francisco Cuellar Cordova. Tangible programming mechatronic interface for basic induction in programming. In *IEEE Global Engineering Education Conference, EDUCON*, volume 2018-April, pages 183–190, 2018. doi:10.1109/EDUCON.2018.8363226.

[5] Ângela Cardoso. Tangible language for educational programming of robots and other targets. FEUP, University of Porto. Available at `https://hdl.handle.net/10216/119132`, February 2019.

[6] Ângela Cardoso, Armando Sousa, and Hugo Ferreira. Programming for Young Children Using Tangible Tiles and Camera-Enabled Handheld Devices. In *ICERI2018 Proceedings*, volume 1, pages 6389–6394, nov 2018.

[7] Bárbara Cleto, Cristina Sylla, Luís Ferreira, and João Martinho Moura. "Play and learn": Exploring CodeCubes. In Cristina Sylla and Ido Iurgel, editors, *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, pages 34–42. Springer Nature, 2020.

[8] Janne Fagerlund, Päivi Häkkinen, Mikko Vesisenaho, and Jouni Viiri. Computational thinking in programming with scratch in primary schools: A systematic review. *Computer Applications in Engineering Education*, 2020. doi:10.1002/cae.22255.

[9] Suzuki Hideyuki and Kato Hiroshi. AlgoBlock: a tangible programming language, a tool for collaborative learning. *Proceedings of 4th European Logo Conference*, June 2016:297–303, 1993.

[10] Dennis Komm, Adrian Regez, Urs Hauser, Marco Gassner, Pascal Lütscher, Rico Puchegger, and Tobias Kohn. Problem Solving and Creativity: Complementing Programming Education with Robotics. In *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, pages 259–265, 2020. doi:10.1145/3341525.3387420.

[11] Alessio Malizia, Tommaso Turchi, and Kai A. Olsen. Block-oriented programming with tangibles: An engaging way to learn computational thinking skills. In F Turbak, J Gray, C Kelleher, and M Sherman, editors, *Proceedings - 2017 IEEE Blocks and Beyond Workshop, B and B 2017*, volume 2017-Novem, pages 61–64, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2017. IEEE. doi:10.1109/BLOCKS.2017.8120413.

[12] Blanca Miller, Adam Kirn, Mercedes Anderson, Justin C. Major, David Feil-Seifer, and Melissa Jurkiewicz. Unplugged Robotics to Increase K-12 Students' Engineering Interest and Attitudes. In *Proceedings - Frontiers in Education Conference, FIE*, volume 2018-Octob of *Frontiers in Education Conference*, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2019. IEEE. doi:10.1109/FIE.2018.8658959.

[13] Ryan Noonan. STEM Jobs: 2017 Update. Technical Report 17, U.S Department of Commerce, Available at https://www.commerce.gov/sites/default/files/migrated/reports/stem-jobs-2017-update.pdf, Accessed last time in June 2021, 2017.

[14] Osmo. Osmo coding family. Available at https://www.playosmo.com/en/coding-family/, Accessed last time in June 2021.

[15] Bjarke Kristian Maigaard Kjar Pedersen, Kamilla Egedal Andersen, Anders Jorgensen, Simon Koslich, Fardin Sherzai, and Jacob Nielsen. Towards playful learning and computational thinking-Developing the educational robot BRICKO. In *ISEC 2018 - Proceedings of the 8th IEEE Integrated STEM Education Conference*, volume 2018-Janua, pages 37–44, 345 E 47TH ST, NEW YORK, NY 10017 USA, 2018. IEEE. doi:10.1109/ISECon.2018.8340502.

[16] Titiphan Phetsrikran, Wansuree Massagram, Thanathorn Phoka, and Antony Harfield. A feasibility study of arducation bot : An educational robotics and mobile application kit for computational thinking skills. In *2018 22nd International Computer Science and Engineering Conference, ICSEC 2018*, 2018. doi:10.1109/ICSEC.2018.8712671.

[17] Fisher Price. Think & learn code-a-pillar twist. Available at https://www.fisher-price.com/en-us/product/think-learn-code-a-pillar-twist-gfp25, Accessed last time in June 2021.

[18] INCoDe.2030 Program. Estratégia inteligência artificial 2030 - xxi governo - república portuguesa. Available at https://www.portugal.gov.pt/pt/gc21/comunicacao/documento?i=estrategia-inteligencia-artificial-2030, Accessed last time in June 2021, may 2019.

[19] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67, 2009. doi:10.1145/1592761.1592779.

[20] Evgenia Roussou and Maria Rangoussi. On the use of robotics for the development of computational thinking in kindergarten: Educational intervention and evaluation. In M Merdan,

W Lepuschitz, G Koppensteiner, R Balogh, and D Obdrzalek, editors, *Advances in Intelligent Systems and Computing*, volume 1023 of *Advances in Intelligent Systems and Computing*, pages 31–44, GEWERBESTRASSE 11, CHAM, CH-6330, SWITZERLAND, 2020. SPRINGER INTERNATIONAL PUBLISHING AG. doi:10.1007/978-3-030-26945-6_3.

[21] Mário Saleiro, Bruna Carmo, Joao M.F. Rodrigues, and J. M.H. Du Buf. A low-cost classroom-oriented educational robotics system. In G Herrmann, M J Pearson, A Lenz, P Bremner, A Spiers, and U Leonards, editors, *5th International Conference on Social Robotics, ICSR 2013*, volume 8239 LNAI of *Lecture Notes in Artificial Intelligence*, pages 74–83. SPRINGER-VERLAG BERLIN, 2013. doi:10.1007/978-3-319-02675-6_8.

[22] Theodosios Sapounidis, Dimitrios Stamovlasis, and Stavros Demetriadis. Latent Class Modeling of Children's Preference Profiles on Tangible and Graphical Robot Programming. *IEEE Transactions on Education*, 62(2):127–133, 2019. doi:10.1109/TE.2018.2876363.

[23] Global STEAM solutions. Cubelets – edtech, soluções educacionais. Available at https://www.globalsteamsolutions.com.br/cubelets/, Accessed last time in June 2021.

[24] Scratch Team. Scratch - faq. Available at https://scratch.mit.edu/info/faq#scratch-extensions, Accessed last time in June 2021.

[25] Scratch Team. Scratch 3.0 - scratch wiki. Available at https://en.scratch-wiki.info/wiki/Scratch_3.0, Accessed last time in June 2021.

[26] Scratch Team. Scratch extension - scratch wiki. Available at https://en.scratch-wiki.info/wiki/Scratch_Extension, Accessed last time in June 2021.

[27] Scratch Team. Scratch link - scratch wiki. Available at https://en.scratch-wiki.info/wiki/Scratch_Link, Accessed last time in June 2021.

[28] Scratch Team. Scratchx - scratch wiki. Available at https://en.scratch-wiki.info/wiki/ScratchX, Accessed last time in June 2021.

[29] Danli Wang, Cheng Zhang, and Hongan Wang. T-Maze: A tangible programming tool for children. In *Proceedings of IDC 2011 - 10th International Conference on Interaction Design and Children*, IDC '11, pages 127–135, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1999030.1999045.

[30] Jeannette M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006. doi:10.1145/1118178.1118215.