UNIVERSIDADE FEDERAL DE VIÇOSA

Robotics Studies with PINEL: A Platform for Interactive Navigation in Education and Learning Contexts

Guilherme Serra Francisco Pinel
Magister Scientiae

VIÇOSA - MINAS GERAIS
2025

GUILHERME SERRA FRANCISCO PINEL

Robotics Studies with PINEL: A Platform for Interactive Navigation in Education and Learning Contexts

Dissertation submitted to the Computer Science Graduate Program of the Universidade Federal de Viçosa in partial fulfillment of the requirements for the degree of *Magister Scientiae*.

Adviser: Alexandre Santos Brandao

Co-adviser: Rejane W. S. de C. Faria

VIÇOSA - MINAS GERAIS
2025

Ficha catalográfica elaborada pela Biblioteca Central da Universidade Federal de Viçosa - Campus Viçosa

T
P651r
2025

Pinel, Guilherme Serra Francisco, 1999-
Robotics studies with PINEL: a Platform for Interactive Navigation in Education and Learning contexts / Guilherme Serra Francisco Pinel. – Viçosa, MG, 2025.
1 dissertação eletrônica (86 f.): il. (algumas color.).

Texto em inglês.
Orientador: Alexandre Santos Brandão.
Dissertação (mestrado) - Universidade Federal de Viçosa, Departamento de Informática, 2025.
Referências bibliográficas: f. 79-86.
DOI: https://doi.org/10.47328/ufvbbt.2025.721
Modo de acesso: World Wide Web.

1. Programação (Computadores) - Estudo e ensino (Ensino fundamental). 2. Robótica na educação. 3. Redes neurais (Computação). I. Brandão, Alexandre Santos, 1982-. II. Universidade Federal de Viçosa. Departamento de Informática. Mestrado em Ciência da Computação. III. Título.

CDD 22. ed. 005.1071

Bibliotecário(a) responsável: Euzébio Luiz Pinto CRB-6/3317

GUILHERME SERRA FRANCISCO PINEL

Robotics Studies with PINEL: A Platform for Interactive Navigation in Education and Learning Contexts

Dissertation submitted to the Computer Science Graduate Program of the Universidade Federal de Viçosa in partial fulfillment of the requirements for the degree of *Magister Scientiae*.

APPROVED: October 2, 2025.

Assent:

Guilherme Serra Francisco Pinel
Author

Alexandre Santos Brandao
Adviser

ACKNOWLEDGMENTS

The journey to this point has been marked by many struggles and achievements, but after walking all the way, today I can look back and recognize the strong hand of the Lord guiding and supporting my every step and that of my family. These achievements were also only possible thanks to the support of my mother, Sandra Alexandra Pinel, who supported and guided me every step of the way; my sister, Carolinha Serra Rodrigues, who even from a distance believed in me and motivated me to keep going; and my fiancée, Leticia Pereira de Almeida, who was by my side throughout the journey.

In addition to dedicating my most sincere thanks to my family, I extend my gratitude to the friends who walked with me on this journey: Heleny Ribeiro, Aleff Oliveira and Estevão Marinato; brothers in Christ who provided support when needed and never left me alone. I would like to thank my friend and advisor, Alexandre Santos Brandão, who from the beginning believed in my potential and supported me like an older brother. I would also like to thank my co-supervisor, Rejane Waiandt Schuwartz de Carvalho Faria, for her help with academic activities, for complementing the theoretical basis of this work and, of course, for the project grant that enabled me to carry out this research.

I would like to thank the Graduate Program in Computer Science for its excellent teaching staff, who are dedicated to offering courses that provide both conceptual and practical training for the knowledge that is consolidated here. I would like to thank the Federal University of Viçosa for all the opportunities over the years, for the courses, internships, scholarships and the learning environment.

This work has been sponsored by the following Brazilian research agencies: Coordination for the Improvement of Higher Education Personnel (CAPES; Financing code 001), Minas Gerais State Foundation for Research Aid (FAPEMIG) and National Council of Scientific and Technological Development (CNPq).

Não te mandei eu? Sê forte e corajoso; não temas,
nem te espantes, porque o Senhor, teu Deus,
é contigo por onde quer que andares.
(Bíblia Sagrada, Josué 1:9)

ABSTRACT

PINEL, Guilherme Serra Francisco, M.Sc., Universidade Federal de Viçosa, October, 2025. **Robotics Studies with PINEL: A Platform for Interactive Navigation in Education and Learning Contexts**. Adviser: Alexandre Santos Brandao. Co-adviser: Rejane Waiandt Schuwartz de Carvalho Faria.

The rapid evolution of technology has driven changes in education, demanding new strategies for teaching programming, robotics, and computational thinking. This dissertation presents PINEL (**Platform for Interactive Navigation in Education and Learning**), a block-based programming platform integrated with the *Robot Operating System* (ROS 2) and a three-dimensional simulation environment. Designed as a modular, accessible, and extensible solution, PINEL bridges high-level visual abstractions and low-level control through an intermediate JSON representation, connecting a Blockly frontend, a Python/Flask backend with ROS 2 and closed-loop control, and a simulation connector based on the AuRoRA/MATLAB framework. Validation proceeded progressively: (i) in simulation, through experiments involving geometric figures, conditional logic, gesture recognition, and repeat loops; (ii) on a physical Pioneer 3-DX robot, confirming practical feasibility in classical navigation tasks; and (iii) in a multimodal scenario using a neural network–based LIBRAS gesture recognition module, applied to a tic-tac-toe activity for laterality training. Finally, a case study with ninth-grade students highlighted engagement, collaboration, and learning of mathematical and logical concepts. The results indicate that PINEL lowers entry barriers for beginners while preserving technical rigor for advanced applications, integrating simulation, real-robot control, and visual programming within a single environment. As such, it contributes to disseminating computational thinking and supporting reproducible, scalable pedagogical practices.

Keywords: educational robotics; computational thinking ; block-based programming ; gesture recognition (LIBRAS) ; ROS 2

RESUMO

PINEL, Guilherme Serra Francisco, M.Sc., Universidade Federal de Viçosa, outubro de 2025. **Estudos de Robótica com PINEL: A Platform for Interactive Navigation in Education and Learning Contexts**. Orientador: Alexandre Santos Brandao. Coorientadora: Rejane Waiandt Schuwartz de Carvalho Faria.

A rápida evolução tecnológica tem impulsionado mudanças na educação e exigido novas estratégias para o ensino de programação, robótica e pensamento computacional. Nesta dissertação, apresenta-se a PINEL (***Platform for Interactive Navigation in Education and Learning***), uma plataforma de programação em blocos integrada ao *Robot Operating System* (ROS 2) e a um ambiente de simulação tridimensional. Concebida como solução modular, acessível e extensível, a PINEL conecta abstrações visuais de alto nível a mecanismos de controle de baixo nível por meio de uma representação intermediária em JSON, articulando um *frontend* em Blockly, um *backend* em Python/Flask com ROS 2 e controle em malha fechada, e um conector de simulação baseado no AuRoRA/MATLAB. A validação ocorreu progressivamente: (i) em simulador, com experimentos de figuras geométricas, condicionais, reconhecimento de gestos e laços de repetição; (ii) no robô físico Pioneer 3-DX, confirmando viabilidade prática em tarefas clássicas de navegação; e (iii) em um cenário multimodal com reconhecimento de gestos em LIBRAS (rede neural), aplicado a um jogo da velha para treinamento de lateralidade. Por fim, realizou-se um estudo de caso com estudantes do 9º ano, evidenciando engajamento, colaboração e aprendizagem de conceitos matemáticos e lógicos. Os resultados indicam que a PINEL reduz barreiras de entrada para iniciantes, mantém rigor técnico para aplicações avançadas e integra, em um único ambiente, simulação, controle de robôs reais e programação visual, contribuindo para a difusão do pensamento computacional e para práticas pedagógicas reprodutíveis e escaláveis.

Palavras-chave: Robótica Educacional; pensamento computacional; programação em blocos; reconhecimento de gestos (LIBRAS); ROS 2.

List of Figures

Contents

1 Introduction

In recent decades, the rapid evolution of technology has profoundly transformed educational processes, highlighting the need for new methodologies capable of promoting engagement, autonomy, and critical thinking. In this context, educational robotics has been consolidated as one of the most promising tools for integrating science, technology, engineering, and mathematics (STEM) at different educational levels, by providing practical experiences that stimulate creativity, problem-solving, and collaboration (ÇETIN; DEMIRCAN, 2020; ATMATZIDOU; DEMETRIADIS, 2016). Several studies indicate that activities with robots foster significant gains in cognitive and socio-emotional skills, as well as increasing students' intrinsic motivation by linking abstract concepts to concrete real-world applications (MORAITI; FOTOGLOU; DRIGAS, 2022).

The introduction of block-based programming languages, such as Scratch and Blockly, represented a milestone in democratizing the teaching of computing, reducing entry barriers and enabling beginners to explore algorithms and logical structures in an intuitive way (DÚO-TERRÓN, 2023). This visual paradigm has been widely used in programming and robotics education initiatives, as it abstracts the syntactic complexity of textual languages and allows learners to focus their efforts on algorithmic reasoning (KUHAIL et al., 2021). Recent research confirms that block-based programming supports the development of computational thinking, including skills such as abstraction, decomposition, and pattern recognition, which are essential for learning in technological and scientific areas (CHEN; CHUNG, 2024; SOCRATOUS; IOANNOU, 2021). Moreover, when combined with robotics activities, block-based programming enhances engagement and supports the understanding of mathematical, physical, and logical concepts in educational settings (COŞKUNSERÇE, 2023; ÇOBAN et al., 2020).

Despite these advances, challenges remain regarding the integration between visual programming tools and real robotic systems. Many initiatives remain restricted to proprietary educational kits or to inflexible simulation environments (BACHILLER-BURGOS et al., 2020), which limits their scalability and applicability in diverse contexts. Another recurring obstacle is the lack of platforms that simultaneously combine the simplicity required for basic education and the technical robustness demanded in advanced training scenarios. In this sense, the literature highlights the need for solutions that articulate different levels of abstraction, connecting accessible high-level interfaces to low-level control mechanisms capable of operating real robotic systems (CHIKURTEV, 2022; MARTÍN et al., 2021).

The adoption of distributed architectures, with a clear separation between frontend and backend, has proven to be a promising path to overcoming these limitations.

The frontend, running in web browsers, can provide interactive graphical interfaces that generate intermediate representations of programs, often in standardized formats such as JSON. The backend, in turn, is responsible for interpreting these representations, translating them into commands in robotic middleware such as the Robot Operating System (ROS), and managing their execution in physical robots or simulators. This separation ensures modularity, reproducibility, and flexibility, allowing new features to be incorporated without compromising platform integrity (WANG et al., 2020; GONG et al., 2020; DUDJAK; MARTINOVIĆ, 2020). Furthermore, the integration with 3D simulators expands pedagogical possibilities, enabling remote experimentation, the re-execution of complex scenarios, and reducing costs associated with intensive hardware use (TZAFESTAS; PALAIOLOGOU; ALIFRAGIS, 2006; JARA et al., 2011; LEI et al., 2022).

It is within this context that the present dissertation is situated, bringing together, organizing, and expanding a series of studies developed around the creation and validation of the **PINEL** (Platform for Interactive Navigation in Education and Learning). The platform was designed to provide an accessible, modular, and extensible environment that connects block-based programming, ROS middleware, and simulation, enabling students at different educational levels to explore concepts of logic, algorithms, and mobile robot control in practical situations. By allowing the seamless execution of the same programs in both real robots and virtual environments, the proposal aims not only to reduce entry barriers for beginners but also to provide a solid infrastructure for advanced training activities, consolidating itself as a relevant contribution to the field of educational robotics and computing in education.

1.1 Educational Robotics

Understood as the articulation of physical artifacts (robots and sensors) with programming environments for instructional purposes, educational robotics is grounded in constructivist and project-based learning approaches, in which students build meaningful artifacts while developing algorithmic reasoning, modeling, and debugging skills (PAPERT, 2008). In line with this framework, block-based visual languages reduce the initial cognitive load and favor a focus on control, logic, and sequencing concepts, a particularly relevant aspect in computational literacy trajectories for beginners (DÚO-TERRÓN, 2023; KUHAIL et al., 2021).

In this context, the role of *computational thinking* (CT) is emphasized, as it involves processes such as abstraction, decomposition, pattern recognition, and algorithms, enabling students to develop logical and systematic reasoning (LIGEIRO; JACINTO; PIEDADE, 2024). The concept of CT, as proposed by Wing (WING, 2006), consists of reformulating a seemingly difficult problem into one that we know how to solve, whether

through reduction, incorporation, transformation, or simulation. Educational robotics constitutes a privileged space to exercise these competencies, as it places students in front of concrete problems that require both the formulation of high-level strategies (sequencing of actions and conditional logic) and their translation into instructions executable by machines. By programming and observing the robot's behavior, learners practice debugging, iterative refinement of solutions, and the construction of abstractions, consolidating in practice the pillars of computational thinking.

Empirical evidence refines the understanding of *what* and *how* is learned through robotics. In early years, activities with robots demonstrate gains in sequencing and self-regulation, surpassing approaches based on non-programmable physical blocks and especially benefiting children with lower initial control (YANG; NG; GAO, 2022). In middle and secondary education, gradual progress is observed in dimensions of computational thinking—decomposition, abstraction, conditional logic, and debugging—with different rhythms according to age and class profile, which recommends scaffolded curricula and adaptive pedagogical interventions (ATMATZIDOU; DEMETRIADIS, 2016; GROVER, 2011). In parallel, studies highlight positive impacts on creativity, critical thinking, and collaboration in classroom contexts, particularly when tasks are structured as projects and articulated around authentic problems (MORAITI; FOTOGLOU; DRIGAS, 2022).

The literature also highlights pedagogical design factors that mediate such results. The integration of concept maps into block-based programming can enhance problem-solving and conceptual reasoning by making explicit the relationships between software components and activity goals (CHEN; CHUNG, 2024). From a curricular design perspective, more structured sequences tend to reduce errors and support debugging practices for beginners, while less structured proposals favor exploration and co-authorship—a balance between both is desirable over time (SOCRATOUS; IOANNOU, 2021). Affective aspects also matter: perceptions of self-efficacy in block-based programming correlate with positive attitudes and willingness for further engagement, although barriers such as cost and access to hardware may limit opportunities (ÇOBAN et al., 2020).

On the side of implementation conditions, teacher training is a determining factor for the critical appropriation of robotics in the curriculum. Professional development initiatives that combine technical foundations with active methodologies (for example, the ROBOESL project) have proven to be more effective in sustaining consistent and inclusive pedagogical practices (ALIMISIS, 2019; SCHINA; ESTEVE-GONZÁLEZ; USART, 2021). In terms of infrastructure, virtual and remote laboratories have emerged as complementary alternatives to the face-to-face use of robots: comparative studies indicate that, even with less physical contact, virtual/remote environments provide adequate support for medium- and advanced-level competencies, expanding access and optimizing laboratory time (TZAFESTAS; PALAIOLOGOU; ALIFRAGIS, 2006; JARA et al., 2011).

Altogether, these findings outline a clear technical-pedagogical requirement for educational platforms: to provide a continuous bridge between high-level abstractions (block-based programming) and low-level control mechanisms, with equivalent support for real robots and simulation, reproducibility of experiences, and scaffolded curricular trajectories. It is precisely this space that motivates the adoption of modular architectures, an intermediate representation in JSON, and integration with robotic middleware such as ROS, elements that will be mobilized throughout this dissertation to support teaching and learning activities at different levels of education.

1.2 Platforms and Architectures

The development of digital platforms for educational purposes requires software architectures that combine accessibility, modularity, and scalability. In this context, the separation between *frontend* and *backend* has become a widely adopted paradigm, both in commercial systems and in academic environments, as it promotes a clear division of responsibilities. The frontend, running on web browsers or mobile devices, is responsible for providing interactive graphical interfaces in which users manipulate programming blocks, organize sequences of instructions, and monitor program execution. The backend, in turn, concentrates the processing logic, translating the received representations into commands suited to the application domain—in the case of robotics, messages published in middleware such as ROS. This separation allows interfaces to evolve independently of the control logic while facilitating maintenance and integration of new modules (WANG et al., 2020; GONG et al., 2020; DUDJAK; MARTINOVIĆ, 2020; CHIKURTEV, 2022).

A key element in this communication is the use of an *intermediate representation* (IR), which acts as a bridge between the visual level and execution in robotic systems. Standardized representation languages such as JSON (*JavaScript Object Notation*) offer advantages such as human readability, ease of parsing, and compatibility with multiple programming languages. In the context of block-based programming, each visual construct can be mapped to a JSON object containing its action and associated parameters, resulting in a sequential list of commands to be interpreted by the backend. This model ensures determinism, facilitates debugging, enables reproducibility of executions, and simplifies integration with both simulators and real robots (BACHILLER-BURGOS et al., 2020; LEI et al., 2022).

Communication between frontend and backend largely relies on the HTTP protocol, which is widely used in web applications. `POST` requests allow the transmission of JSON commands for execution, while `GET` requests can be used to query states such as odometry or the result of a gesture classification. This technological choice brings educational robotics closer to an ecosystem of established standards, lowering adoption barriers and enabling integration with external tools such as online simulation environments or

distance learning platforms (RICHARD et al., 2024; MARTÍN et al., 2021). Moreover, the use of well-defined API contracts—in line with practices such as *API-first*—ensures consistency and predictability, which are essential conditions for maintaining scalable learning environments (DUDJAK; MARTINOVIĆ, 2020).

From a pedagogical perspective, adopting this client-server architecture in educational robotics enables students to work at different levels of abstraction. At the highest level, they interact with visual blocks representing algorithmic intentions; at an intermediate level, these blocks are translated into human-readable JSON; and at the lowest level, the backend converts them into ROS messages that command motors, sensors, and simulators. This hierarchy not only enhances understanding of how computational systems are structured but also reinforces fundamental computer science concepts such as modularity, encapsulation, and communication protocols, while keeping the experience accessible and meaningful for learners at different stages of training (TZAFESTAS; PALAIOLOGOU; ALIFRAGIS, 2006; JARA et al., 2011).

1.3 Research Objectives, Question, and Hypotheses

Based on the theoretical and practical context presented so far, this dissertation aims to design, implement, and validate a block-based programming platform integrated with ROS to support educational robotics, with seamless execution on both physical robots and simulation environments. The central purpose is to lower entry barriers for learners, enabling the development of computational thinking, logic, and problem-solving skills through interactive experiences in real classroom contexts and controlled laboratory scenarios. By offering a modular architecture that connects frontend interfaces, backend processing, and robotic *middleware*, the platform seeks to combine accessibility with technical robustness, making robotics education more scalable, reproducible, and inclusive.

In summary, this dissertation is guided by the following research question:

How can a block-based programming platform integrated with ROS foster computational thinking and support scalable, accessible, and effective educational robotics practices in scenarios with physical robots and simulations?

To address this question, the following hypotheses were formulated:

- **H1:** A block-based programming environment that abstracts low-level ROS commands into high-level visual constructs will significantly reduce beginners' cognitive load, improving engagement and understanding of computational concepts.
- **H2:** The integration of backend processing with ROS *middleware*, using JSON as an intermediate representation, will ensure robust execution of commands on physical

robots and simulations, demonstrating scalability and reproducibility of learning activities.

- **H3:** The use of block extensions for gesture recognition and logical conditionals will enable the creation of interactive and multimodal activities, fostering the development of computational thinking dimensions such as abstraction, decomposition, and algorithmic reasoning.
- **H4:** The availability of equivalent execution paths (physical robot and simulation), integrated into the same interface and command flow, will facilitate classroom adoption, hybrid activity planning, and experiment re-execution without compromising educational outcomes.

The alignment between the research question and the hypotheses provides the framework for the design, implementation, and evaluation of the platform, contributing to the advancement of educational robotics by offering an accessible, reproducible, and pedagogically grounded tool, ready for use in real teaching contexts and simulation laboratories.

1.4 Organization of the Dissertation

This dissertation contributes to the field of Educational Robotics by presenting the development, validation, and application of **PINEL**, a block-based programming platform integrated with the ROS and a three-dimensional simulation environment. Conceived as a modular, accessible, and extensible solution, **PINEL** bridges high-level visual abstractions and low-level control mechanisms, enabling seamless execution of the same programs in both real robots and simulated environments.

The initial stage of this research focused on validating the platform exclusively in a virtual environment using the **AuRoRA** simulator in *MATLAB*. The goal was to ensure execution flow consistency and test fundamental control and logic elements, such as closed-loop feedback, repeat blocks, conditionals, and gesture recognition. This preliminary phase enabled a safe and controlled evaluation, guaranteeing robustness before integrating the system with physical hardware.

Building upon this consolidated foundation, **PINEL** was then extended to practical experiments with the Pioneer 3-DX mobile robot, employing a distributed integration between ROS 2 running on a central computer and on an embedded Jetson Nano. At this stage, the platform was applied to planar geometry activities, validating its use in classical educational robotics tasks such as constructing geometric figures and monitoring real-time telemetry, thus confirming its versatility in real-world applications.

Subsequently, the platform was enhanced with a vision module based on neural networks for LIBRAS gesture recognition, allowing the design of multimodal activities that integrate visual perception, conditional logic, and robot navigation. This feature was demonstrated in a laterality training scenario through a tic-tac-toe game, in which the robot moves according to gestures performed by the user, exemplifying the educational potential of combining computer vision with robotic control.

Finally, **PINEL** was applied in a real educational setting with ninth-grade students. In this case study, participants engaged in activities involving physical block planning, digital programming, and practical execution with the physical robot. These experiences provided evidence of the platform's potential to foster computational thinking, peer collaboration, and the learning of mathematical and logical concepts.

Given these different stages of development and application, the structure of this dissertation was designed to provide readers with both a comprehensive understanding of the platform and a flexible navigation path through its contents. Readers seeking an overview of the research context, Educational Robotics principles, and the motivations underlying this work may begin with Chapter 1, which establishes the theoretical and conceptual foundations of the study and presents the main goals and hypotheses.

Chapter 2 introduces the overall architecture and development process of the **PINEL**, detailing its components, communication layers, and integration with ROS 2. This chapter serves as the technical foundation upon which all subsequent experiments and educational applications are constructed.

Chapter 3 presents the validation of the platform through experiments involving geometric figures, demonstrating its operation in both simulation and real environments using the Pioneer 3-DX robot. This validation confirms the reliability of the platform and its ability to reproduce identical logical flows across different execution contexts.

Chapter 4 extends this validation to a multimodal scenario by integrating a neural-network-based module for LIBRAS gesture recognition. The experiments described in this chapter include both simulated and real-robot implementations, showcasing the interaction between perception, reasoning, and robotic motion.

Next, Chapter 5 reports the educational case study conducted with ninth-grade students, in which the platform was used in collaborative activities that combined physical and digital programming tasks. This chapter emphasizes how the platform supports the development of computational thinking, teamwork, and mathematical reasoning in authentic learning environments.

This organization allows readers to follow the dissertation sequentially—from conceptual design to educational validation—or selectively, combining Chapter 2 with the application chapters (3, 4, or 5) according to their particular research or pedagogical

focus.

Finally, Chapter 6 consolidates the main contributions, discusses the limitations, and outlines perspectives for future developments and pedagogical applications. The analysis of the research hypotheses introduced in Section 1.3 unfolds across Chapters 2 to 5, with special emphasis on Chapter 5, where the case study provides qualitative evidence supporting the evaluation of these hypotheses.

2 PINEL: Platform for Interactive Navigation in Education and Learning

This chapter introduces the **PINEL** (Platform for Interactive Navigation in Education and Learning), a block-based programming solution integrated with the ROS middleware and validated in both real robots and a three-dimensional simulation environment. The platform is designed to lower entry barriers in programming and robotics education, combining accessibility for beginners with the technical rigor needed for advanced applications. Its architecture connects a Blockly-based frontend, which generates an intermediate JSON representation, to a Python/ROS backend responsible for command interpretation and robot control. The evaluation included simulation experiments involving geometric figures, gesture-based conditionals, and repetition loops, demonstrating consistent and reliable execution. As a core contribution, **PINEL** unifies high-level visual abstraction with low-level robotic control, providing a modular and extensible environment that supports reproducibility, student engagement, and application across diverse educational scenarios.

PINEL was developed to bridge the gap between visual programming tools and real robotic systems, enabling the same block-based program to be executed in both virtual environments based on the **AuRoRA** library (PIZETTA; BRANDÃO; SARCINELLI-FILHO, 2014) and on real robots. The platform maintains execution consistency and supports gesture recognition as an interaction modality, expanding its potential for multimodal educational applications. Its design integrates accessibility for beginners with the robustness required for advanced experimentation in mobile robotics, fostering scalability, flexibility, and pedagogical adaptability.

The main objective of this work is to document the design, implementation, and validation of **PINEL**, demonstrating its applicability in diverse educational contexts. The main contributions include: (i) a modular architecture based on frontend/backend with an intermediate JSON representation, (ii) seamless integration with ROS for real and simulated execution, and (iii) the inclusion of advanced blocks for loops, conditionals, and gesture classification, which expand the pedagogical possibilities of the platform.

This chapter is organized as follows: Section 2.1 presents the conceptual and technological foundations underlying the platform's development. Section 2.2 details the architecture, covering the frontend, backend, and simulator integration. Section 3.3 discusses the experiments conducted and the results obtained. Section 2.3 explores perspectives for application at different educational levels and in teacher training. Finally, Section 2.4 presents the conclusions and outlines future directions for continuing the research.

2.1 Preliminaries

The teaching of programming and robotics has been consolidated as a strategic axis for developing digital skills and computational thinking at different educational levels. Educational robotics promotes an active learning environment in which students not only consume technology but also use it to solve problems and create solutions, fostering logical reasoning, creativity, and collaboration (ÇETIN; DEMIRCAN, 2020). In this context, the introduction of accessible interfaces, such as visual languages and block-based programming environments, has expanded the reach of robotics in classrooms, reducing entry barriers and making the teaching process more inclusive (NASCIMENTO et al., 2021; DÚO-TERRÓN, 2023).

Several initiatives have sought to combine block-based programming with real or simulated robotic platforms. Works such as LearnBlock (BACHILLER-BURGOS et al., 2020) highlight the importance of flexible, hardware-independent architectures, while recent research explores virtual and remote laboratories to increase the availability of practical resources (TZAFESTAS; PALAIOLOGOU; ALIFRAGIS, 2006; JARA et al., 2011). Other studies demonstrate gains in the development of computational thinking, self-regulation, and engagement through visual programming applied to robotics (YANG; NG; GAO, 2022; ATMATZIDOU; DEMETRIADIS, 2016). Despite these advances, many solutions remain restricted to specific environments, with low interoperability between simulators and real robots, and few offer expansion capabilities to integrate sensors, control algorithms, and more sophisticated interaction modules.

The development of educational platforms for robotics requires an integrated understanding of software architecture, control abstractions, and visual programming paradigms. In this context, the distinction between **frontend** and **backend** plays a central role, as it separates the responsibilities of user interaction and logical execution. In educational applications, the frontend provides an intuitive graphical interface, while the backend concentrates the control logic and communication with the robot or simulator. This separation, derived from client-server architectures, has already proven effective in different domains, such as embedded cognitive systems (MARTÍN et al., 2021), digital twin-based teleoperation environments (KAARLELA et al., 2022), and educational chatbots focused on scalability (RICHARD et al., 2024).

In the field of robotics, another relevant conceptual axis is the distinction between low-level and high-level integration. The level closest to the hardware, represented by middlewares such as ROS, ensures message exchange through topics responsible for physical variables like linear velocity, angular velocity, and odometry. This mechanism is essential for direct control of sensors and actuators, providing reliability and responsiveness. In contrast, high-level environments, such as visual languages or graphical interfaces, allow users to issue abstract commands without having to deal with the complexity of commu-

nication topics. This layered hierarchy has been explored, for example, in service-oriented architectures for modular robots, where the high-level layer hides technical details and exposes reusable services (CHIKURTEV, 2022).

For such platforms to be sustainable, architectural solutions must prioritize flexibility, scalability, and reproducibility. The use of **micro frontends** and microservices in educational platforms follows this direction, allowing specialized modules—such as gesture recognition, controllers, or simulator integration—to evolve independently, as long as they maintain clear communication contracts (WANG et al., 2020; GONG et al., 2020). This logic is reinforced by **API-first** methodologies, which prioritize defining interfaces before actual development, ensuring consistency between services and clients (DUDJAK; MARTINOVIĆ, 2020). Tools such as LearnBlock illustrate how these practices can be applied to educational robotics by proposing a loosely coupled architecture that guarantees hardware independence and enables the same block-based program to run on different robotic platforms (BACHILLER-BURGOS et al., 2020).

Thus, block-based programming has become a strategy capable of connecting computational logic concepts with practical applications in robotics, making learning more accessible and engaging. Environments such as Scratch and Blockly are widely recognized for their positive impact on the development of computational thinking (NASCIMENTO et al., 2021), and their adoption in robotics contexts further increases student engagement. The use of an intermediate JSON representation to translate visual blocks into simple semantic structures strengthens this process, as it maintains readability, facilitates debugging, and ensures compatibility across different execution layers (XU et al., 2020; LEI et al., 2022).

Recent research has deepened this discussion by highlighting how different resources and methodologies enhance the use of block-based programming. Beginners' self-confidence in programming tends to increase when activities are conducted with visual interfaces (COŞKUNSERÇE, 2023), and the use of concept maps has proven effective in fostering problem-solving and critical thinking in this context (CHEN; CHUNG, 2024). The curriculum structure also exerts significant influence: well-structured curricula reduce errors and facilitate debugging, while more open approaches encourage collaboration and creativity (SOCRATOUS; IOANNOU, 2021). Studies combining educational robotics with block-based programming have also reported improvements in creativity, critical thinking, and collaborative work (MORAITI; FOTOGLOU; DRIGAS, 2022), while investigations into teacher training highlight above-average self-efficacy perceptions among future technology educators, although cost and accessibility issues remain significant barriers (ÇOBAN et al., 2020).

This scenario is reinforced by investigations in different educational contexts. Studies integrating visual programming packages into the teaching of Design and Tech-

nology in primary schools report significant gains in creativity and problem-solving when interactive and hybrid methodologies are adopted (ISMAIL; ZAMAN; MOHAMMAD, 2022). Moreover, using robots as educational tools from early childhood helps integrate technology and engineering into STEM curricula, enabling young children to act as creators of technological solutions (ÇETIN; DEMIRCAN, 2020).

Similarly, recent bibliometric analyses reveal that Scratch has established itself as the primary visual programming language studied over the last two decades, highlighting its contribution to the development of computational thinking and its integration into multidisciplinary STEAM projects (DÚO-TERRÓN, 2023). Broader systematic reviews also characterize different visual programming paradigms, confirming that flowcharts, blocks, and direct manipulation reduce cognitive barriers, bring programming closer to end-users, and create environments for exploration and expression (KUHAIL et al., 2021).

From this combination of technical and pedagogical perspectives, a framework emerges in which educational robotics benefits simultaneously from advances in software architecture and teaching methodologies based on visual programming. This convergence creates favorable conditions for developing platforms such as **PINEL**, capable of integrating different levels of abstraction, coordinating communication among distributed components, and, at the same time, offering users an accessible and engaging learning experience. It is at this intersection of technological robustness and educational intentionality that the proposal presented in this work is grounded.

2.2 PINEL Architecture

Platform for Interactive Navigation in Education and Learning (**PINEL**) is an integrated solution that combines a block-based graphical interface with a ROS 2 execution engine and a connector for simulation in MATLAB. This section describes the platform's architecture, detailing its main modules (frontend, backend, and simulator), their responsibilities, and the communication flow among them.

2.2.1 Architecture overview

The platform is organized into three main layers and communicates through an intermediate JSON representation (IR). The **frontend**, running in the browser, offers a block-based programming interface (Blockly) that abstracts ROS 2 low-level details and produces the IR. The **backend**, implemented in Python, interprets this IR and converts it into concrete actions, publishing commands on `/cmd_vel` and monitoring the pose on `/pose` with closed-loop control. Finally, the **simulation** layer, developed in MATLAB with the **AuRoRA** library (PIZETTA; BRANDÃO; SARCINELLI-FILHO, 2014), reproduces the robot's behavior in a virtual environment, exchanging data with

the backend via HTTP requests.

Figure 1 presents this overview: the frontend generates JSON-structured commands ("JSON Navigation Reference"), which are forwarded to the backend (Flask + ROS 2); the backend, in turn, coordinates execution and consolidates *sensor data* from both the real and virtual robots. The execution flow can be summarized as *blocks in the browser → JSON commands → backend server → ROS 2 or simulator*. Such separation ensures portability between real and simulated environments and modularity for the evolution of each component.

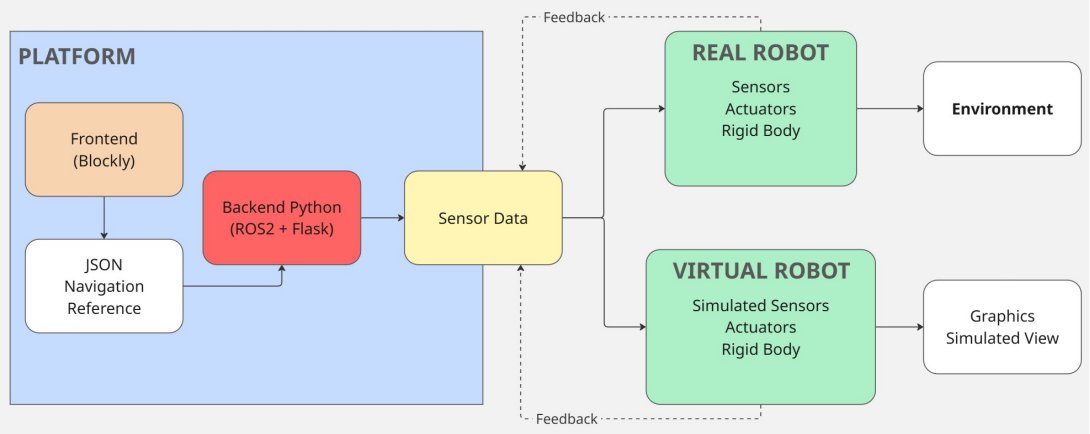


Figure 1 – Overview of the architecture: the frontend (Blockly) generates JSON, the backend (Python/ROS 2) orchestrates execution, and the simulation layer (**AuRoRA**) replicates the robot's behavior in a virtual environment.

2.2.2 Frontend

The frontend of the **PINEL** platform was developed using *Blockly*, a JavaScript library that enables the creation of visual programming environments based on interlocking blocks. This component provides an intuitive and accessible interface that allows users, especially students and beginners in programming, to construct robot control programs visually without requiring prior knowledge of textual coding. The workspace (Figure 2) is organized into thematic categories such as Pioneer-3DX (movement, rotation), gesture perception, and logic. Each block corresponds to a specific robot command or logical structure, such as *Move Forward*, *Turn*, *Gesture Detected*, *If*, and *Repeat*.

The platform's workspace supports essential interaction features, including drag-and-drop, copy/paste, undo/redo, zoom, and automatic layout organization. It also includes input validation mechanisms for numerical fields (e.g., ensuring distances and angles fall within allowed ranges), highlighting inconsistencies directly on the blocks to prevent invalid commands from being sent to the backend. This validation process contributes to a smooth and educational programming experience, reducing errors and allowing users to focus on logical reasoning rather than syntax.

Figure 2 – Workspace of the PINEL based on Blockly.

Each custom block was designed using the *Block Factory*[1] tool (Google Developers, 2025), where its graphical structure, input/output parameters, and code generation logic were defined. The blocks are implemented in JavaScript and integrated into the interface's toolbox. For example, a block like *Move Forward* is configured to accept a distance parameter, which is then processed into a specific JSON command. The detailed process for creating and implementing new blocks is explored in Section 2.2.3.

Once a program is built, Blockly's internal function `workspaceToCode` translates the visual blocks into JavaScript code. The frontend then converts this code into an intermediate representation (IR) in JSON, where each block corresponds to a structured command. This IR maintains consistency between the browser interface, the backend processing, and the ROS 2-based execution engine. Example commands include:

```
{"action":"move_forward","distance":1.0}
{"action":"turn","angle":90}
{"action":"conditional_execution","condition":"result == 'A'",
"commands":[{"action":"move_forward","distance":0.5}]}
```

The complete operational flow of the frontend is presented in Algorithm 1, covering all stages from block creation to JSON serialization and communication with the backend.

During execution, the frontend dynamically highlights the currently executing block and provides real-time feedback, including progress messages and error alerts, without resetting the workspace. Accessibility standards are followed, including support for keyboard navigation and visual contrast.

When gesture-based commands are included in the user program, the frontend

[1] <https://blockly-demo.appspot.com/static/demos/blockfactory/index.html>

Algorithm 1 PINEL Frontend Workflow.

Ensure: Block-based programming interface and command transmission
Load Blockly libraries and define the toolbox structure
Initialize the workspace with configuration options
Bind buttons and execution events (*Run*, *Stop*)
while the platform is active **do**
  while the user has not clicked *Run* **do**
    User drags and connects blocks to compose the program
    Validate input fields and highlight inconsistencies
  end while
  Convert visual blocks into JavaScript code (`workspaceToCode`)
  Translate the code into an intermediate JSON representation (IR)
  Send JSON commands sequentially to the backend via POST
  Display responses, highlight the executing block, and show progress/errors
end while

temporarily activates the classifier module connected to the camera stream, initiating a controlled recognition cycle before proceeding with commands dependent on the gesture label. This cycle ensures synchronization between gesture perception, logical conditions, and robotic action, as outlined in Algorithm 2.

Algorithm 2 Gesture classification cycle in the frontend.

Ensure: Obtain gesture label for conditional logic
Activate the camera stream and display classifier feedback
Wait for label stabilization or timeout
Retrieve the current gesture label from the classifier
Send label update to the backend
Continue with subsequent command execution (conditionals may use label)

Finally, the frontend was designed for extensibility: new blocks can be added by defining their appearance, input parameters, and code generation rules without altering the platform's execution contract. This modular design keeps the client-side lightweight and ensures compatibility with both simulated and real-robot executions. Consequently, the frontend of the **PINEL** platform effectively bridges user interaction and robotic control, providing a unified environment for experimentation, learning, and multimodal human-robot interaction. The modularity that allows for this extensibility is based on the creation of custom blocks, and their development process is detailed in the following section.

2.2.3 Block Creation

As mentioned previously, the frontend is designed to be extensible through the addition of custom blocks. Block creation is handled in JavaScript via the Blockly API. Each block combines a visual definition (how it appears and connects in the workspace)

with a generator (how the block is translated into the intermediate JSON representation). Two semantic roles are maintained: action blocks, which produce a self-contained JSON object, and expression blocks, which produce text strings used in conditionals. Loops are unrolled on the client side, and conditionals encapsulate both the expression and the list of internal commands, preserving the order consumed by the backend and, where applicable, the simulator. The recommended procedure for adding a new block is presented in Algorithm 3.

Algorithm 3 Defining and Generating Blocks on the Frontend.

Ensure: Visual structure and IR generation rule for each block

1: Choose the semantics: **action** (command) or **expression**
2: Define the identifier, displayed labels, and category in the *toolbox*
3: Select connections: *previous/next* (command) or *output* (expression)
4: Declare fields (number, list, text) with default values and validators (range/type)
5: Implement the generator, returning JSON (action) or a text string (expression)
6: Register the block and its generator in the block engine
7: Test in the *workspace*: generate IR, check limits, execute, and observe telemetry

To exemplify the process with elements used in the platform, the *classify gesture* block is presented (Listing 2.1). The visual definition introduces a simple command, and the generator produces the `"classify_gestures"` action in the IR. At runtime, the frontend (Section 2.2.2) uses this action to trigger the classification routine, which queries the backend and provides the resulting label to the program, maintaining the semantics of the label acquisition cycle.

Listing 2.1 – Gesture classification block (definition and generator).

```
Blockly.Blocks['classify_gestures'] = {
  init: function () {
    this.appendDummyInput()
        .appendField("Classify Gesture");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(160);
    this.setTooltip("Classifies gesture using the webcam and
       returns the result.");
    this.setHelpUrl("");  }
    };

javascript.javascriptGenerator.forBlock['classify_gestures'] =
   function (block) {
  return JSON.stringify({
    action: "classify_gestures"  });
    };
```

The IR sequence containing `"action":"classify_gestures"` is interpreted by the frontend *runtime*, which triggers the routine below to make an HTTP call to the backend and receive the label in the `"result"` field (Listing 2.2).

Listing 2.2 – Gesture classification routine in the frontend (backend call).

```
function classifyGests() {
  return fetch('/classify_gestures', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' }
  })
  .then(response => response.json())
  .then(data => {
    if (data.result) {
      return data.result; // detected class
    } else {
      throw new Error("No class detected.");} }); }
```

Next, the *move forward* block is detailed (Listing 2.3), as implemented on the platform. The visual definition exposes a numeric field with range validation (in meters), and the generator returns a self-contained IR object, which will be consumed directly in the execution process.

Listing 2.3 – Movement block: move forward (definition and generator).

```
Blockly.Blocks['move_forward'] = {
  init: function () {
    this.appendDummyInput()
        .appendField("Move forward")
        .appendField(new Blockly.FieldNumber(1, 0, 100), "
           DISTANCE")
        .appendField("meters");
    this.setPreviousStatement(true, null);
    this.setNextStatement(true, null);
    this.setColour(230);
    this.setTooltip("Moves the robot forward by the specified
       distance.");
    this.setHelpUrl("");  }};

javascript.javascriptGenerator.forBlock['move_forward'] =
   function (block) {
  const distance = parseFloat(block.getFieldValue('DISTANCE'));
  return JSON.stringify({ action: "move_forward",
    distance: distance });};
```

An instance of this block generates an IR command such as:

```
{"action":"move_forward","distance":1.0}
```

This command is sent to the backend, which publishes velocities on the `/cmd_vel` topic, monitors odometry on `/pose` until the target is reached, and finally triggers a stop command (as will be detailed in the next subsection). By connecting the visual definition, the generator, and the execution effect, the frontend's determinism is maintained, server-side validation is simplified, and compatibility is preserved with both the real robot and the simulator.

2.2.4 ROS Middleware

ROS serves as the core middleware platform, facilitating communication between the backend system and physical robotic hardware. For experimental validation, we employed the Pioneer 3DX mobile robot platform. The middleware architecture handles the following key topics:

- `/RosAria/cmd_vel`: Receives linear and angular velocity commands.
- `/RosAria/pose`: Provides odometry information, including the robot's position and orientation.

To ensure rotational precision in edge cases exceeding the $[-\pi, \pi]$ range, we implemented an angle normalization module. This module utilizes quaternion-to-Euler angle transformations for robust orientation interpretation of robot pose data. Figure 3 illustrates the system's interaction flow, which operates through the following sequence:

1. The user visually creates a program in the frontend by combining blocks and configuring parameters.
2. The Blockly-generated code is sent to the backend in JSON format.
3. The backend processes the commands and publishes them to the appropriate ROS topics.
4. The ROS middleware transmits the commands to the robot, which executes the physical actions.
5. Feedback is collected from the robot and returned to the backend, which informs the frontend about progress.

This modular architecture ensures flexibility for future expansions, such as adding new blocks, supporting different robots, and integrating additional sensors.

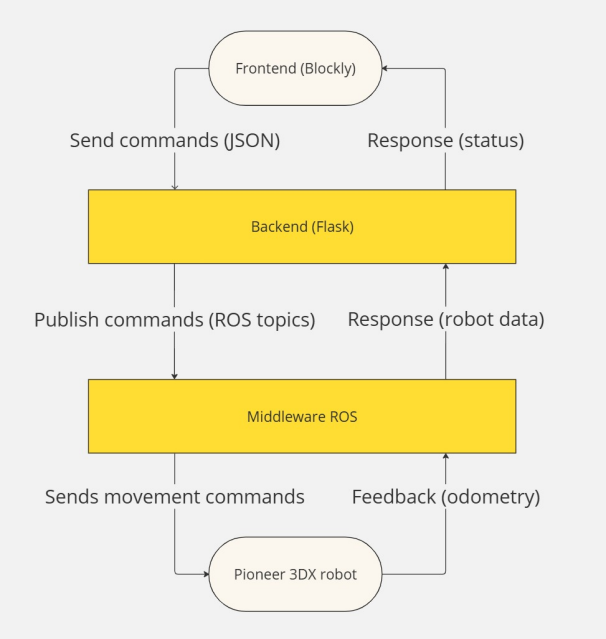


Figure 3 – Interaction diagram of the platform with the robot.

2.2.5 Backend

The backend was implemented in Python using the Flask microframework. It serves as an HTTP service responsible for processing requests from the frontend, validating data, and mediating between the block language and the execution on ROS 2. During initialization, the Flask application is created, and in parallel, a ROS node (`rclpy`) runs in a dedicated *thread*. This architecture prevents network operations from blocking control *callbacks* and timers, ensuring that sensor readings and command publications maintain low latency.

A key component of the backend is an intermediate layer that converts the Blockly-generated JSON commands into direct calls to specific Python functions developed to control the robot. The primary functions in this layer are:

- **move_forward(distance)**: Controls the robot's linear movement in meters.
- **turn(angle)**: Performs a rotation based on a given angle.
- **stop()**: Sends a command to immediately halt any ongoing movement.

These functions interact with the robot through ROS 2. Linear and angular velocities are published to the `/cmd_vel` topic (using `geometry_msgs/Twist`), and odometry is subscribed from the `/pose` topic (using `nav_msgs/Odometry`). With each message received from `/pose`, the backend's internal state is updated with position (x, y) and *yaw* orientation. This state is crucial for the movement and rotation controllers. At the end of each command, the `stop()` function is called, which publishes a null `Twist` and confirms immobility.

The HTTP contract is deliberately streamlined. The `POST /execute` endpoint receives a *batch* of commands, processes them sequentially, and returns a summary. The execution of a command *batch* follows a three-step process: pre-processing of gesture results, evaluation of conditionals, and finally, the execution of primary actions like `move_forward` and `turn`. Algorithm 4 summarizes this cycle.

Algorithm 4 PINEL Backend Workflow.

Ensure: Execution of commands and robot control via ROS 2

Initialize ROS node (`/cmd_vel` and `/pose`) and HTTP server
while server is active **do**
  Receive list of JSON commands at `/execute`
  Apply gesture results and evaluate conditionals
  for each command in the resulting sequence **do**
    if `move_forward` **then**
      Control v based on distance error, call `stop()`
    end if
    if `turn` **then**
      Control ω based on angular error, call `stop()`
    end if
    if `stop` **then**
      Publish a null `Twist` message
    end if
  end for
  Log telemetry and respond with status
end while

The `@app.route` decorator in Flask implements this contract, as shown in the implementation of the `/execute` endpoint (Listing 2.4). This function reads the IR list, applies the three-step flow, and calls a generic executor for each command.

Listing 2.4 – Endpoint `/execute` implementation in Flask.

```
@app.route('/execute', methods=['POST'])
def execute():
    ir_list = request.get_json(force=True)
    results = []

    # Simplified flow: pre-process, evaluate conditionals, then execute
    processed_commands = preprocess_and_evaluate(ir_list)

    for cmd in processed_commands:
        results.append(execute_generic_command(cmd))

    return jsonify({"status": "ok", "n_executed": len(results)}), 200
```

Within the command execution logic, movements like `move_forward` and `turn` are managed by a closed-loop PID (Proportional–Integral–Derivative) controller. This controller uses the robot's state from the `/pose` topic to calculate the error (either distance or angle) and generate the appropriate velocity commands. The completion of each movement is governed exclusively by the error falling below a predefined tolerance, ensuring precise and reliable execution.

2.2.6 Integration with AuRoRA

The simulation layer was developed in MATLAB based on the **AuRoRA** library, which provides specific classes for modeling differential drive mobile robots, including the Pioneer 3-DX. This library offers a 3D representation of the robot, with differential kinematics, virtual sensors, and rendering capabilities, allowing experiments designed for the real robot to be reproduced in a virtual environment without modifications to the block programming logic.

Communication between the simulator and backend is carried out through a periodic HTTP *loop*, which plays a role equivalent to message exchange in ROS. At each iteration, the simulator queries the `/bridge/cmd_vel` endpoint, obtaining the latest pair of linear and angular velocities published by the backend. These velocities are applied to the robot's kinematic model in **AuRoRA**, which updates the position and orientation in 3D space. Next, the new integrated pose is sent to the backend through the `/bridge/pose` endpoint so that the simulated robot's state is incorporated into the odometry log. The termination logic is governed by the activity signal at `/bridge/status`, ensuring that the simulation continues only while there is active command execution. To prevent premature shutdowns, the simulation only ends when the status is inactive and actual movement has occurred.

The process described is summarized in Algorithm 5, which illustrates the continuous integration cycle between the simulator and backend. In this dynamic, control updates occur at a higher frequency (30 Hz), while 3D graphical rendering is updated at a lower rate (around 3.3 Hz), ensuring visual smoothness with low computational cost.

This strategy allows the same sequence of blocks used in the frontend to be tested both on the real robot and in the virtual environment, without additional adaptations. In addition to facilitating the validation of new blocks and control strategies, the simulator provides a safe environment for educational activities, where programming errors or incorrect commands do not pose risks to physical equipment.

Algorithm 5 Integration of the MATLAB/AuRoRA simulator with the backend.

Ensure: Coherent execution in a virtual environment
Initialize AuRoRA model of the Pioneer robot
Configure control and visualization parameters
while simulation active **do**
  Query backend at `/bridge/cmd_vel`
  Apply (v, ω) to the robot model
  Update position and orientation in the 3D environment
  Send updated pose to the backend at `/bridge/pose`
  Update graphical visualization
  Query `/bridge/status`
  if status = inactive **and** movement occurred **then**
    End simulation
  end if
end while

2.3 Perspectives and Applications

By integrating block-based programming, ROS middleware, and simulation, the proposal described here opens multiple application possibilities in educational contexts. In elementary education, its use can introduce initial notions of computational thinking through the construction of simple movement sequences or the use of conditional blocks, bringing children closer to algorithmic reasoning in a playful, visual environment. Studies indicate that combining robot programming with concrete manipulation activities fosters sequencing ability, self-regulation, and the development of computational thinking competencies from early childhood. In particular, evidence shows that programming with robots outperforms traditional physical block activities in developing sequencing skills, computational thinking, and self-regulation, benefiting even children with lower initial control (YANG; NG; GAO, 2022).

In high school, the environment can serve as a bridge to more abstract concepts of logic, control, and sensor integration. At this level, robotics as a pedagogical tool has shown significant gains in dimensions of computational thinking such as task decomposition, conditional logic, and debugging, expanding the language and technical vocabulary employed by students (GROVER, 2011). Additional evidence indicates that the development of these competencies is progressive and benefits from continued activities, with pace variations associated with age and class profiles; such findings reinforce the importance of scaffolded curricula and adaptive pedagogical interventions to sustain advances in abstraction, decomposition, and modularity (ATMATZIDOU; DEMETRIADIS, 2016). Thus, the set of blocks and connectors can be explored in technology, physics, and mathematics courses, as well as in interdisciplinary STEM workshops, enabling students to progress from simple constructions to more complex control structures.

In higher education, the system applies both to computer science and engineering

programs and to teacher preparation degrees, serving as a resource to explore distributed architectures, communication protocols in ROS, and control strategies in mobile robotics. By allowing the same block-based programs to operate real robots and simulations, reproducibility and flexibility are ensured—qualities that are fundamental for training, rapid prototyping, and remote experimentation. This feature is especially relevant in shared or hybrid lab contexts in which physical access to the robot is not always feasible.

In addition, it creates room for remote lab scenarios in which students can access real or simulated robots through the Internet. The literature shows that such environments complement in-person classes by offering hands-on programming and control experiences without the need for direct physical contact with equipment; comparisons among in-person, remote, and virtual modalities indicate that, although differences may arise in lower-level skills, virtual and remote environments provide adequate support for intermediate and advanced competencies (TZAFESTAS; PALAIOLOGOU; ALIFRAGIS, 2006). Moreover, experiences in automation and robotics courses demonstrate that the combined use of in-person, virtual, and remote labs enhances engagement, supports comparison between simulated and real results, and increases efficiency in the use of lab time (JARA et al., 2011).

Another crucial aspect for consolidating this initiative is teacher training. Research shows that teacher preparation in educational robotics is decisive for the successful integration of technology in the classroom. Initiatives such as the ROBOESL project have developed specific curricula to enable teachers not only to master technical tools but also to apply constructivist and project-based methodologies aimed at engaging students at risk of dropping out (ALIMISIS, 2019). Likewise, recent reviews point to characteristics and recommendations for training programs, highlighting the importance of ongoing support, sound pedagogical practices, and critical integration of technology into the curriculum (SCHINA; ESTEVE-GONZÁLEZ; USART, 2021). In this sense, the platform should be viewed not only as a tool for students but also as a support resource for teacher training, strengthening its insertion into sustainable and inclusive educational ecosystems.

Finally, by combining accessibility, technical robustness, and pedagogical relevance, the framework shows potential to act as a bridge between the introduction to block-based programming and advanced training in robotics and engineering. The perspective is to support continuous learning pathways, sensitive to differences in pace and maturation observed in real contexts (ATMATZIDOU; DEMETRIADIS, 2016), while expanding into hybrid and remote scenarios with support for both simulation and physical robots.

2.4 Concluding Remarks

This chapter presented the development of **PINEL**, a block-based programming platform integrated with the ROS middleware and validated in a three-dimensional simulation environment. The proposed architecture combines a high-level interface based on Blockly with a Python/ROS backend capable of interpreting an intermediate JSON representation and executing locomotion commands, conditionals, loops, and gesture recognition.

The platform's architecture was described in detail, highlighting the separation between frontend and backend as a central design principle. The frontend, based on Blockly, allows users to build programs through a visual interface that generates an intermediate JSON representation. This representation is then interpreted by a Python/ROS backend responsible for processing commands and managing their execution in real or simulated environments. The use of this intermediate layer ensures modularity, reproducibility, and compatibility across different contexts. In addition, the chapter presented how **PINEL** integrates with the **AuRoRA** simulation environment, enabling programs developed in the visual interface to be executed in a 3D virtual scenario without modification. This capability allows educators and students to work in flexible and low-risk learning settings, where the same program can be applied both in simulation and later on real robots.

The central contribution of this study lies in offering an accessible, modular, and extensible platform that connects the fundamentals of block-based programming to the practice of controlling mobile robots. While it simplifies interaction for beginning students, the solution remains compatible with advanced robotics concepts, creating a bridge across different levels of training. In this way, it is consolidated as a support tool for education in computer science and engineering, while also establishing a solid foundation for research in hybrid environments, remote laboratories, and integration with new forms of human–robot interaction.

To encourage reproducibility and the extension of this work, the platform's source code (including frontend, backend, and simulation scripts) is available in a public repository[2]. This availability aims to support the academic and educational community, enabling researchers and educators to adapt and extend the solution in different contexts.

[2] <https://github.com/GuilhermePinel25/PINEL-Platform-for-Interactive-Navigation-in-Education-and-Learning>

3 First Validation: Geometric Trajectories

This chapter presents the first set of functionalities of **PINEL**, highlighting its ability to execute basic movement and rotation commands via a block-based programming interface integrated into ROS. By using a visual programming environment built with Blockly and a Python backend, the platform allows the creation of customized movement sequences that are interpreted directly by the robotic system. To demonstrate these capabilities, the platform was applied to the execution of geometric trajectories, specifically a square, an equilateral triangle, and a lemniscate. These experiments serve as an initial validation of the platform's operation, illustrating how users can easily program and control robotic movements in a structured and intuitive way, particularly in educational and experimental contexts.

3.1 Introduction

Robotics has become a fundamental field in several domains, including education, research, and industry (PINEL; ARAÚJO; BRANDÃO, 2025; TEIXEIRA et al., 2022; PEREIRA et al., 2023; FAGUNDES-JUNIOR et al., 2023; BARCELOS et al., 2023). However, the complexity of robotic programming often represents a barrier for beginners and enthusiasts. In this context, block-based programming platforms have emerged as a promising alternative, offering an intuitive interface that eliminates the need for advanced knowledge of syntax (KARVOUNIDIS; LADIAS; DOULIGERIS, 2023).

In recent years, visual programming tools have grown significantly in education, especially at introductory learning levels. Scratch, Open Roberta, and TinkerCAD platforms have played an important role in this scenario, allowing users with no previous programming experience to develop interactive projects and understand fundamental computational logic concepts (MOREIRA, 2025).

In particular, Scratch, developed by the MIT Media Lab, is a free platform designed to help children and beginners learn programming. In 2015, Google collaborated with the Scratch team to create a new version of the platform's blocks using Blockly technology (Coding Nemo, 2025). The current version of Scratch, the result of this partnership, was launched in 2019. Although widely used in primary education to introduce basic programming and robotics concepts, Scratch has limitations when it comes to integrating with more complex systems, such as the Robot Operating System (ROS) middleware and controlling physical robots (Scratch Wiki Community, 2025; AFFONSO, 2025). On the other hand, Blockly stands out as a more flexible tool, capable of adapting to advanced platforms and integrating with physical and robotic systems more effectively. Its ability to

generate code in languages such as Python and JavaScript allows users to progressively explore more advanced programming concepts (PASTERNAK; FENICHEL; MARSHALL, 2017; CHEN et al., 2021).

The integration between visual programming and physical systems represents one of the most significant advances in this field. Unlike Scratch, which is mainly geared towards simulations and interactive projects, Blockly allows the creation of programs that interact directly with real hardware, such as motors and mobile robots, facilitating the practical teaching of programming and motion control (PLAZA et al., 2018). This ability to connect programming to the physical world is an important differentiator, especially in educational and research contexts, where interaction with real elements increases engagement and understanding of programmed concepts. Application examples include the use of Blockly in hardware platforms such as Arduino (PATAKY; FEELPAK, 2019) and in educational robots such as Metabot (PASSAULT et al., 2016), demonstrating its effectiveness in both teaching and more sophisticated projects.

In addition, the development of block-based programming platforms for educational robotics has received increasing attention in the literature, with initiatives aimed at making these tools more accessible and adaptable (CHRONIS; VARLAMIS, 2022). In this sense, solutions such as RoBlock exemplify the potential of combining visual programming and robotics, allowing students and researchers to explore advanced concepts in an intuitive way (AZMI et al., 2021).

In this context, **PINEL** was designed to support accessible, modular, and adaptable control of mobile robots. The choice of Blockly as the main interface is motivated by three main aspects: (1) its flexibility for integration with external systems, (2) its compatibility with advanced robotic architectures, and (3) its ability to be customized for users with different levels of expertise. Therefore, the following sections present the first experiment to validate this platform, focused on programming and executing geometric trajectories with a mobile robot. This practical scenario illustrates how visual programming can be directly connected to robotic motion control, providing an engaging entry point for educational and experimental applications.

3.2 The PINEL ROS 1 Setup

This section presents the configurations of the **PINEL** used to carry out simulations and real-world experiments. Initially, the platform was validated in the MATLAB/AuRoRA simulator using three complementary experiments designed to evaluate its functionality under different programming conditions. The aim of this stage was to check that the simulator could coherently reproduce the functionalities implemented in the physical robot, ensuring consistency and correspondence between the virtual and real

environments. The first experiment, which is the focus of this chapter, consisted of executing geometric trajectories (a square, an equilateral triangle and a lemniscate), serving as a basis for demonstrating the execution of commands and the platform's behavior when faced with different block configurations. A video recording of these simulation demonstrations is available at <https://youtu.be/ZmbWoeE4G_A>.

After the simulation phase, the same set of experiments was reproduced with the physical Pioneer 3DX robot, in order to validate the platform in a practical robotics scenario. This stage sought to show how the integration of a block-based programming environment with the ROS middleware makes it possible to control and execute geometric trajectories in an intuitive, accessible and reproducible way. As in the simulation, the three trajectories (square, equilateral triangle and lemniscate) were implemented and executed in a controlled physical environment. A video recording of the demonstrations with the real robot is available at <https://youtu.be/z762lM7W3w4>.

The experimental setup employed a Pioneer 3DX robot controlled via a distributed ROS architecture. A Windows 10 notebook hosted the **PINEL** interface via Ubuntu 20.04.6 on WSL, connecting to ROS Master running on an Ubuntu 20.04.6 desktop PC with ROS 1 Melodic. The Pioneer robot was connected via a Raspberry Pi 3 (64-bit Quad Core, 1.2 GHz and 1 GB RAM) running Ubuntu 18.04 and ROS 1 Melodic. Communication between all the components followed the architecture illustrated in Figure 4.

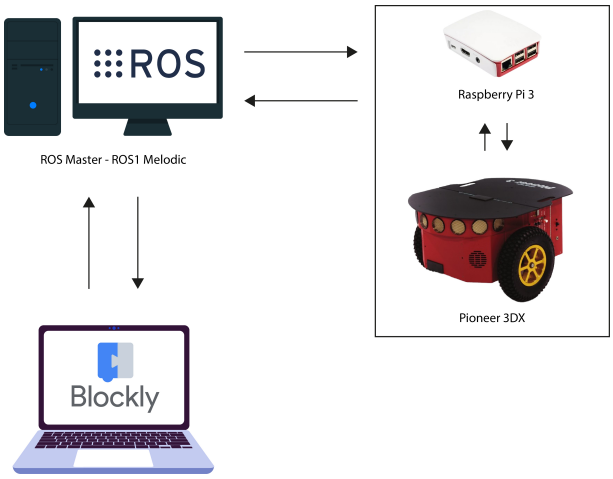


Figure 4 – Experimental setup.

Together, the simulation and real-world executions provide a comprehensive view of the platform's main functionalities, showing how the visual programming blocks allow robotic behaviors to be defined and reproduced in both environments. This integration provides the basis for the analysis and discussion presented in the following sections of this chapter.

3.3 In Action: Simulation

For the simulation scenario, three different programs were implemented in the front-end **PINEL**: a square, an equilateral triangle and a discretized lemniscate. Figure 5 shows the sequences of blocks used in each case, made up of movement and rotation instructions, which were converted into the intermediate JSON representation and sent to the back-end for execution in the simulator.

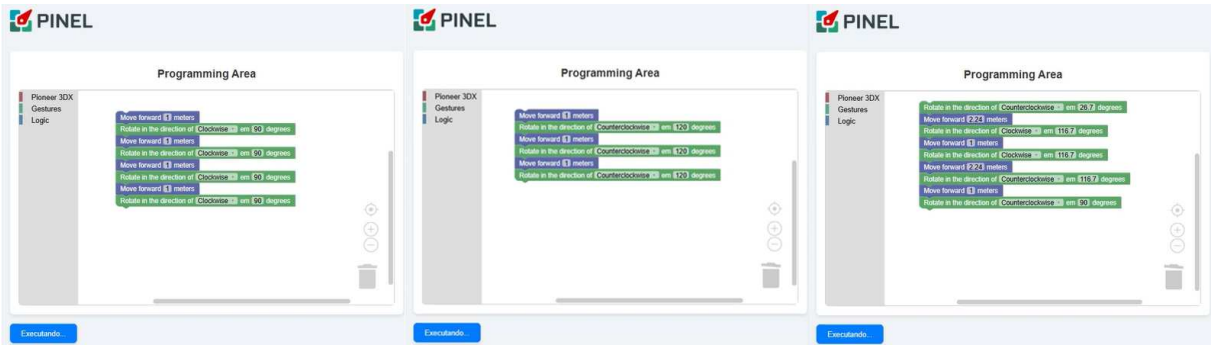


Figure 5 – Block programs used in the **PINEL** frontend for the geometric figures: square (left), equilateral triangle (center), and discretized lemniscate (right).

In the square execution (Figure 6), the robot traveled four one-meter segments, interleaved with 90° clockwise rotations. The plots show the correspondence between the desired position and the obtained trajectory, indicating the accuracy of the integration among frontend, backend, and simulator. The orientation ψ exhibits the characteristic steps of discrete rotations, while the control signals highlight the alternation between linear and angular velocity, as expected for "move-and-turn" motions.

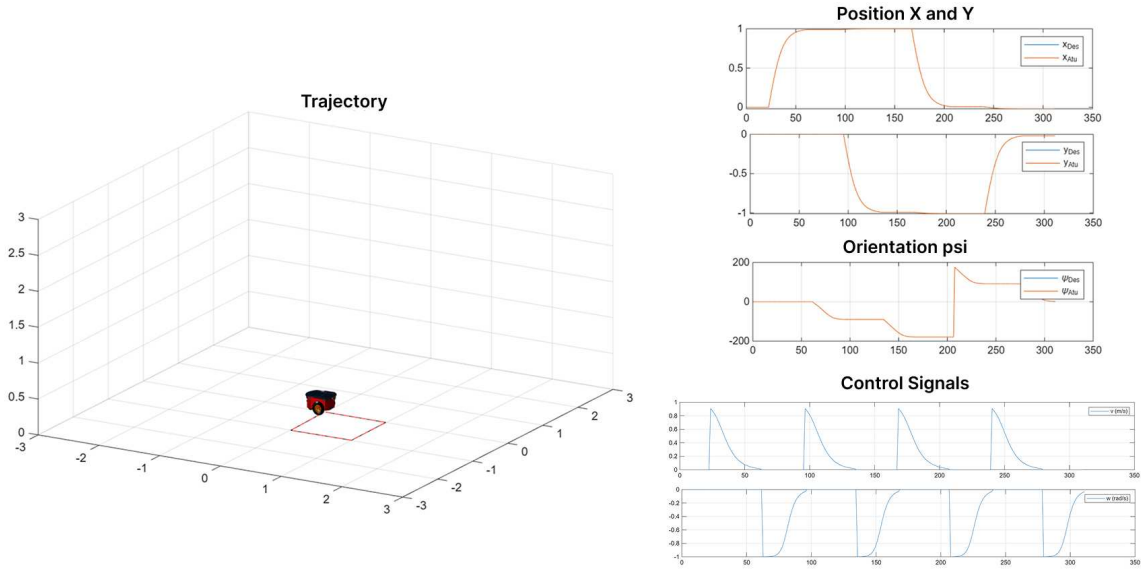


Figure 6 – Results of the square trajectory execution: spatial trajectory, desired and obtained positions, orientation, and control signals.

Complementarily, Figure 7 shows the backend terminal during the execution of one of the rotation commands in the square. In this log, the desired rotation angle, the robot's current orientation, the orientation error, and the applied angular velocity are updated at each iteration. This process illustrates how the backend continuously processes and transmits motion commands based on the robot's state, maintaining consistent communication and control throughout the execution, even in simulation scenarios.

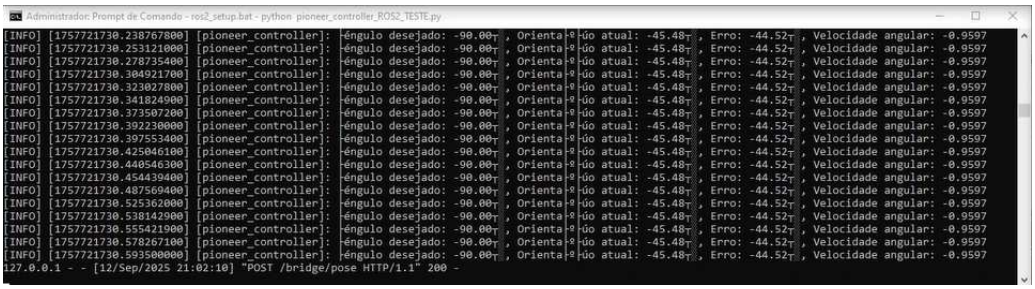


Figure 7 – Backend terminal during the execution of a square rotation command, showing current orientation, angular velocity, desired angle, and real-time updated error.

The equilateral triangle (Figure 8) consisted of three one-meter segments, interleaved with 120° counterclockwise rotations. The resulting trajectory confirms the figure's closure with minimal deviations between desired and obtained positions. The orientation plot displays three marked angular changes, consistent with the triangle's geometry, while the control signals confirm the precise alternation between straight segments and rotations.

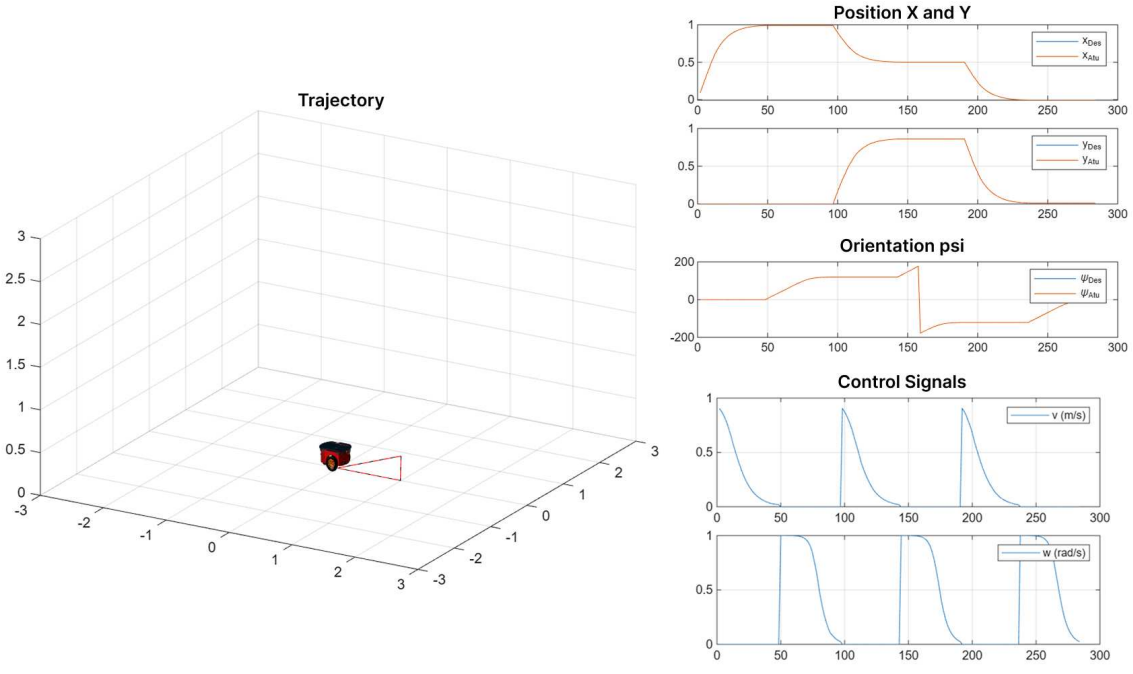


Figure 8 – Results of the triangular trajectory execution: spatial trajectory, desired and obtained positions, orientation, and control signals.

Finally, the discretized lemniscate (Figure 9) required a longer sequence of instructions, with linear displacements of different lengths and intermediate rotations (for example, around 116,7° and 26,7°). The "figure-eight" path was consistently reproduced on the plane, demonstrating that the simulator supports nontrivial trajectories. Good overlap is observed between the desired and obtained trajectories, even though the greater geometric complexity imposes more frequent alternation of commands. The orientation ψ varies throughout the entire path, while the control signals show the continuous transition between translation and rotation commands, validating the backend's stability in the face of long sequences.

After confirming the consistency of the system in the simulation environment, the same experiments were replicated using the real Pioneer 3DX robot to validate the platform under real-world conditions.

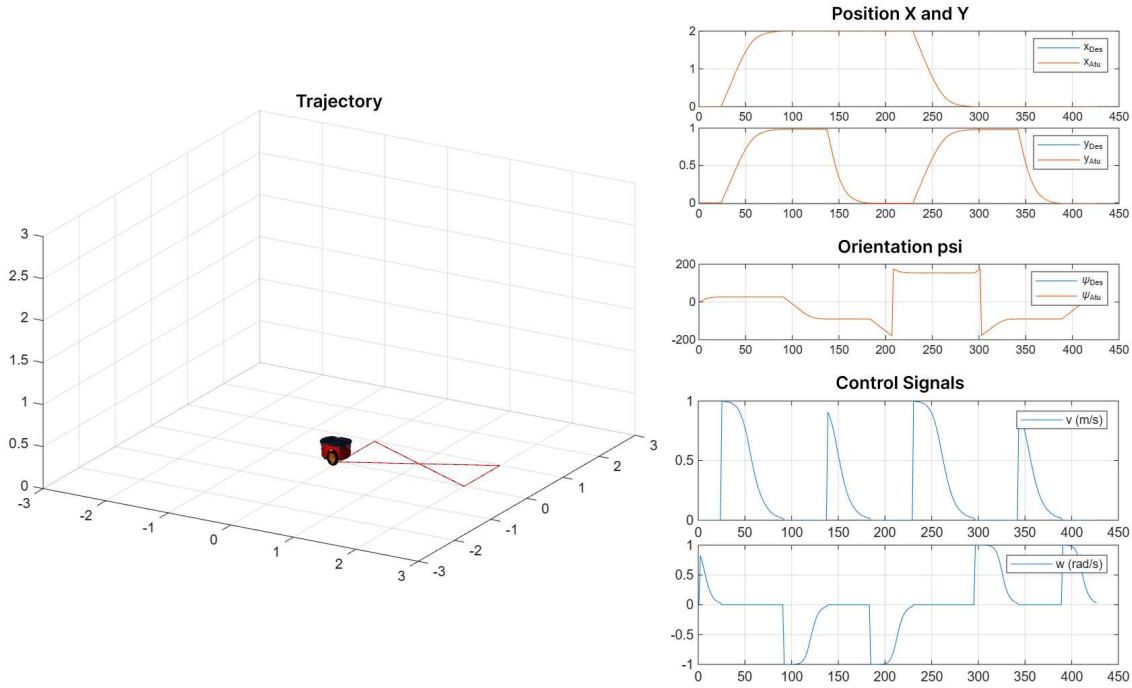
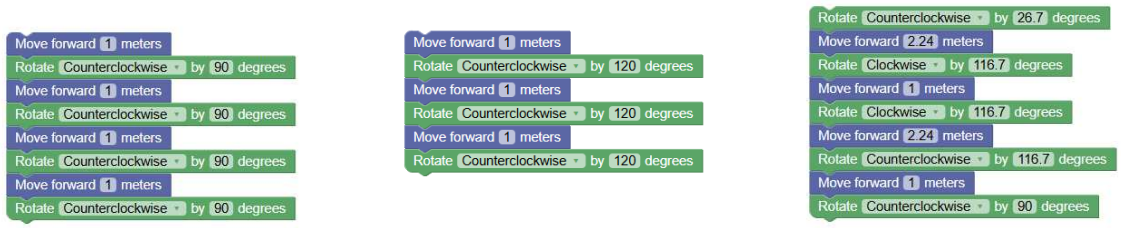


Figure 9 – Results of the discretized lemniscate execution: spatial trajectory, desired and obtained positions, orientation, and control signals.

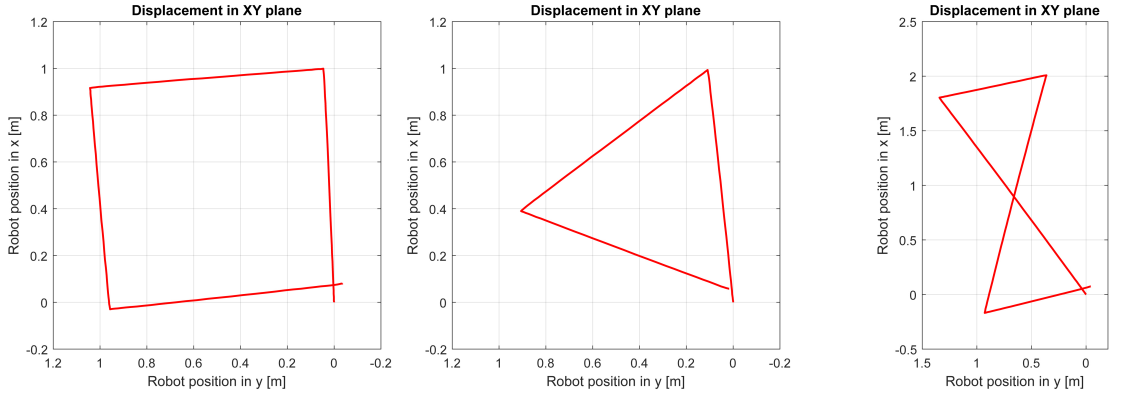
3.4 In Action: Pioneer-3DX

The experimental setup utilized a Pioneer 3DX robot controlled through a distributed ROS architecture. The same experimental setup described in Section 3.3 was used for the tests under real conditions.

The first activity involved constructing a square. The robot was programmed to follow a rectangular path, returning to the starting point after completing the trajectory. The sequence of blocks used for this experiment is shown in Figure 10a. Observing the



(a) Square code. (b) Triangle code. (c) lemniscate code.



(d) Square path. (e) Triangle path. (f) Lemniscate path.

Figure 10 – Blockly code implementation (top) and resulting paths (bottom) for three geometric paths: square (left), equilateral triangle (middle), and lemniscate (right), showing the robot's path-following.

execution of the command by the robot allows students to explore right angles and symmetry in practice. The performance graph, presented in Figure 10d, shows the robot's trajectory along the path.

The second activity consisted of constructing an equilateral triangle. With the Blockly platform, students can program the robot to follow a precise path, generating a triangle with three equal sides. This activity is useful for exploring the symmetry of geometric figures and the precision of the robot's movements. The sequence of blocks for this experiment is illustrated in Figure 10b. The performance graph for this experiment, shown in Figure 10e, details the accuracy of the robot in following the desired path.

The third activity involved constructing a more complex figure, the lemniscate, a curve resembling an inverted "eight". This activity provided an opportunity to explore more complex curves and their movement patterns. The sequence of blocks programmed for this experiment is shown in Figure 10c. The performance graph, presented in Figure 10f, details how the robot followed the desired path to draw the lemniscate.

3.5 Implemented Features

The platform's functionality encompasses basic robot motion control (linear and angular), looping operations via *Repeat* blocks, and ROS-based state feedback and command execution. In summary, the platform's core functionalities include:

- Linear movement (forward/backward) and angular rotation (in-place turns) of the robotic platform;
- Structured repetition of command sequences using the *Repeat* block for iterative task execution;
- Bidirectional communication with ROS for real-time robot state monitoring (sensor data, odometry) and command execution (velocity control).

3.6 Concluding Remarks

The conducted experiments demonstrate the feasibility of the block-based programming platform integrated with ROS for executing navigation tasks and drawing geometric figures with the Pioneer 3DX robot. The trajectory accuracy was assessed through generated graphs, highlighting the platform's efficiency in controlled environments. Additionally, the proposed activities show significant potential for teaching plane geometry interactively, making mathematical concepts more accessible and visually intuitive for students.

The developed platform successfully achieved its objectives by delivering an accessible and modular environment for mobile robot programming. The Blockly-ROS integration demonstrated both viability and effectiveness, as evidenced by the positive experimental results.

Future work will focus on four key enhancements: (1) implementing PID control to improve turning precision, (2) integrating LiDAR sensors for autonomous navigation capabilities, (3) expanding the block set with conditional and sensor-based loops, and (4) incorporating human-robot interaction through gesture and facial recognition blocks. These improvements will advance the platform's functionality while maintaining its accessibility for educational applications.

4 Second Validation: LIBRAS-Based Robotic Navigation with Emphasis on Laterality Training

This chapter introduces the second case study, which explores the integration of a gesture recognition module into the PINEL platform to enable human–robot interaction through Brazilian Sign Language (LIBRAS). By extending the block-based programming environment with conditional and loop structures connected to a real-time gesture detection system, the platform allows users to control a mobile robot through intuitive hand-signed commands. A tic-tac-toe scenario was adopted as the main experimental setting to demonstrate this functionality, in which the robot autonomously navigates the board based on the recognized gestures. This approach enhances accessibility, supports playful learning activities, and promotes the integration of computer vision and robotics in educational contexts, encouraging motor laterality training and inclusive pedagogical practices.

4.1 Introduction

Educational robotics has been increasingly consolidated as a pedagogical tool capable of promoting the development of cognitive, motor, and social skills, especially when integrated into playful and accessible learning environments (FERREIRA; LIMA, 2023; SALMA et al., 2025). This perspective aligns with the constructionist approach of Seymour Papert, whose ideas were expanded by authors such as Blikstein (2013), advocating for the use of digital fabrication technologies and robotics to democratize invention and foster hands-on learning. In this sense, low-cost tools and accessible platforms play a fundamental role in expanding access to robotics across diverse educational settings (WARDRIP; BRAHMS, 2016).

Among the strategies adopted to make robotics education more inclusive and intuitive, block-based programming platforms stand out. Such solutions allow learners to compose algorithms visually, eliminating the rigid syntax of traditional programming languages (KÖLLING; BROWN; ALTADMRI, 2015; BACHILLER-BURGOS et al., 2020). User-friendly interfaces like Scratch, for instance, have demonstrated great potential in engaging students by simplifying their entry into the programming world (RESNICK et al., 2009), reinforcing the educational value of similar block-based approaches. Among them, Blockly stands out for its ability to generate code in different languages and connect

to real hardware—from Arduino boards to mobile robots (PASTERNAK; FENICHEL; MARSHALL, 2017). Despite its pedagogical potential, few studies combine Blockly with professional robotic systems that integrate both advanced logical structures and accessible gesture-based interaction.

The Robot Operating System (ROS) provides the necessary abstraction to orchestrate sensors and actuators in mobile robots. The integration between Blockly and ROS, as demonstrated in our previous work (PINEL; BRANDÃO; FARIA, 2025), has the potential to simplify the control of mobile robots in educational settings, reducing configuration complexity and allowing students to focus on algorithmic concepts. As demonstrated, this combination enables the intuitive execution of geometric trajectories (square, triangle, or lemniscate). Such activities reinforce the teaching of mathematical, spatial, and computational concepts, highlighting the pedagogical value of the approach.

However, these prototypes still relied on the ROS 1 master node, lacked iterative blocks (*for*, *while*), and did not include a multimodal interface based on Libras (Brazilian Sign Language). In parallel, the literature indicates that interactive robotic environments enhance engagement and cognitive performance in children, including those with hearing impairments (LIU et al., 2023; CARDOZO, 2017). This accessibility can be further expanded through real-time gesture recognition, which connects to broader discussions about the role of artificial intelligence in educational contexts. Modern deep learning techniques have enabled significant advances in computer vision tasks such as gesture detection (GOODFELLOW; BENGIO; COURVILLE, 2016).

Models like YOLOv8n can perform detection and classification at around 30 FPS on embedded GPUs, maintaining a mean average precision (mAP) above 90% for static gesture datasets (REDMON et al., 2016; DIWAN; ANIRUDH; TEMBHURNE, 2023), making them suitable for educational applications that require fast and accurate responses (AVULA; R; PILLAI, 2022; LIMA; HUDSON; BRANDãO, 2025, submitted for publication). Furthermore, gesture recognition falls within the broader field of artificial intelligence, emphasizing how computational agents can proactively perceive and respond to their environment (RUSSELL; NORVIG, 2021).

Complementing this approach, Lima, Hudson e Brandão (2025, submitted for publication) proposed a system that combines computer vision and educational robotics through an interactive game in which Libras gestures are used as commands to control a graphical agent. This proposal highlighted the potential of computer vision as a tool for inclusion by expanding interaction possibilities with robotics-based educational systems, particularly in accessibility contexts. This work aligns with efforts toward more inclusive and equitable education, as emphasized by (DANIELA; LYTRAS, 2019), who advocates for educational robotics to promote accessible environments with low-cost tools adapted to student diversity.

In response to these gaps, this chapter validates the **PINEL** which incorporates loop and conditional blocks, integrates LIBRAS gesture recognition via YOLOv8n and eliminates dependence on the ROS 1 master node when migrating to ROS 2, thus reducing communication latency. The platform is demonstrated in a playful scenario: a tic-tac-toe game in which the robot moves autonomously around the board based on the user's gestures. It is hoped that the proposed platform will contribute to: promoting the training of motor laterality through the robot's spatial movement; encouraging the playful and interactive learning of LIBRAS, especially in inclusive educational contexts; and reinforcing the integration between computer vision and robotic control, offering an accessible, dynamic and technologically robust learning environment.

4.2 The PINEL ROS 2 Setup

In this new scope, two functionalities were added: the repeat block, which allows commands to be executed iteratively, and the gesture recognition block, which enables the robot's behavior to be adapted to conditional commands based on visual signals captured by a camera. These additions required modifications in both the interface layer (frontend) and the command interpretation and execution logic (backend), in addition to synchronization with the real-time detection logic in the ROS middleware.

Figure 11 illustrates the communication flow of the proposed system. The Blockly interface runs on a notebook with Windows 10 operating system and ROS 2 Humble, responsible for sending movement commands (*cmd_vel*) via the DDS domain to a Jetson Nano equipped with Ubuntu 20.04 and ROS 2 Foxy. The Jetson, in turn, converts these commands into velocity signals that control the Pioneer 3DX robot. In the opposite direction, the Jetson publishes odometry data, allowing the notebook to update the application's internal state and record the robot's displacement. In parallel, the webcam attached to the notebook transmits frames to the YOLOv8-based detection module, whose gesture classification result is interpreted in the backend and converted into new block commands that are reinserted into the ROS 2 topic cycle.

4.3 Repeat Block

The system interface was expanded with the creation of the *repeat_custom* block, implemented in the Blockly library. This block displays the phrase "repeat [N] times" with an interactive numeric field where the user defines the desired number of repetitions. The internal commands are nested within its body and repeated according to the specified value. The corresponding JavaScript code is generated through the *javascriptGenerator* function, which visually converts the blocks into an expanded list of JSON commands.

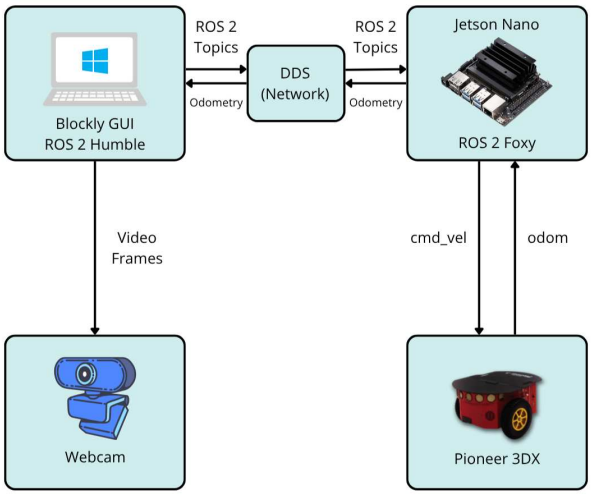


Figure 11 – Communication flow between the Blockly interface (ROS 2 Humble), the Jetson Nano (ROS 2 Foxy), and the Pioneer 3DX robot.

These commands are then transmitted to the backend via an HTTP request, already in their repeated format, with no need for additional interpretation.

The integration of the loop block with action blocks—such as movement and rotation—enables users to create more complex command sequences in a straightforward manner. For example, in Figure 12, the robot is programmed to draw a square by repeating four times the sequence of moving forward 1 meter followed by a 90-degree counterclockwise rotation.

This visual logic is automatically converted into a list of commands in JSON format, as illustrated in Figure 13. This list is sent to the backend via an HTTP POST request, typically to the `/execute` endpoint, where each command is interpreted individually. The backend processes the commands and sends them to the ROS2 middleware through the `rclpy` library and `Twist` messages, which activate the robot's motors according to the defined parameters (distance and angle).

It is worth noting that although the frontend generates and displays the full

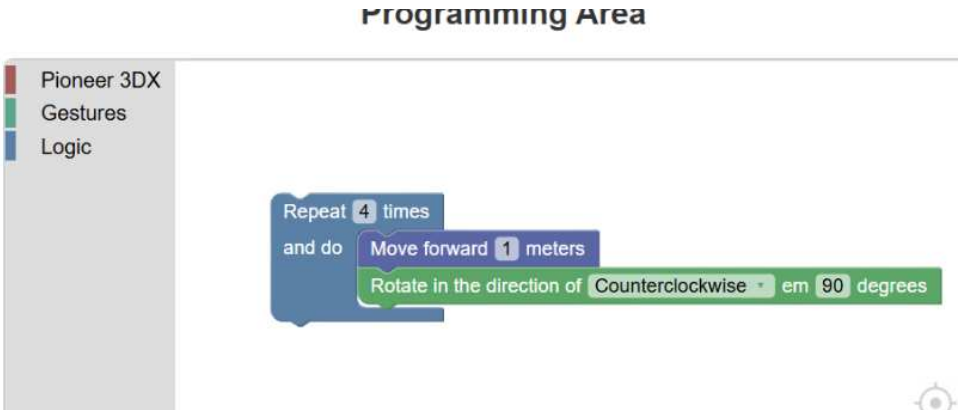


Figure 12 – Grouped blocks for constructing a square.

Figure 13 – List of extracted commands.

command list (as seen in the 'Extracted Commands' field), the ROS system processes only one command at a time, waiting for its completion before starting the next. Consequently, the browser console displays only the currently executing command (as shown in the 'Received Command' field), a behavior expected due to the absence of real-time feedback mechanisms during operation.

4.4 LIBRAS Gesture Recognition System

4.4.1 Image Acquisition

The image acquisition for LIBRAS gesture recognition was carried out using a conventional RGB camera capable of capturing real-time video. The images used during the model training phase were manually extracted from videos recorded under varying lighting conditions and angles, ensuring greater robustness of the system against environmental variations. The resulting dataset consisted of 1,735 images representing static hand gestures for letters of the LIBRAS alphabet. The camera was strategically positioned to center the user's hands within the field of view, facilitating gesture detection and framing in the captured images.

4.4.2 Processing and Recognition

All the steps related to image processing, gesture classification and performance analysis were not developed as part of this work. Instead, the entire recognition pipeline was reused from previous research carried out at the Robotics Specialization Center

(NERo) at the Federal University of Viçosa, particularly Lima's study (LIMA; HUDSON; BRANDãO, 2025, submitted for publication). Therefore, in this work, the YOLO-based classifier developed in that study is only integrated and consumed by **PINEL**, with no modifications to its architecture, training process or data set.

After capturing the image, an annotation process was carried out using the Roboflow platform, allowing the precise definition of the gesture bounding boxes. The recognition model was built on the YOLOv8n architecture, which unifies detection and classification in a single step, supporting real-time inference. Feature extraction was performed directly by the YOLOv8 backbone. The model was trained specifically to recognize static letters from the LIBRAS alphabet, with each detected letter mapped to a corresponding movement command, such as forward movement or rotation. The training results, as reported in (LIMA; HUDSON; BRANDãO, 2025, submitted for publication), indicate an overall accuracy of 89.5% and an average accuracy of 98.9%.

4.4.3 Integration with the Blockly Platform

To enable gesture-based interaction within the platform, the backend of the system was adapted to interpret commands resulting from gesture classification. A new action,`"classify_gestures"`, was incorporated into the `executar()` function, allowing the server to retrieve the most recent classification stored in a global variable (`last_gesture_result`). This variable is updated asynchronously via the `/classify_stream` route, which activates the webcam and uses the YOLOv8-based classifier to recognize static LIBRAS gestures in real time. Once detected, the gesture result is saved and remains available for subsequent evaluation in conditional and iterative structures, preventing synchronization issues during execution.

An additional ROS node is responsible for running the inference process using the Ultralytics library, processing the video stream for short intervals to update the classification result. The value stored in `last_gesture_result` is accessed whenever needed by the `execute` and `classify_stream` endpoints, enabling smooth integration between computer vision, decision-making logic and movement commands.

On the Blockly side, a custom *gesture_block* was developed to represent this functionality within the visual programming interface. When included in a sequence of blocks, it requests the currently recognized gesture through a REST call to the backend and reflects the result directly in the flow of the program. Thus, each gesture is automatically mapped to its associated command, ensuring that robot actions correspond consistently to the user's manual interaction. In addition, execution events such as inference status, recognized gestures and action progress are transmitted to the frontend as JSON responses, maintaining real-time communication and enhancing the interactivity of the system.

4.5 In Action: Simulation

4.5.1 Integration of gesture classification and conditional logi

The second experiment aimed to validate the integration between gesture recognition blocks and decision-making logic structures. To this end, gesture-classification blocks were combined with *if* conditional blocks so that different geometric trajectories would be executed depending on the detected gesture. Figure 14 presents the block sequence used: in the first condition, if the recognized gesture was equal to "B", the robot should traverse a square; next, a new classification was performed and, if the detected gesture was equal to "A", the robot should execute the trajectory of an equilateral triangle.

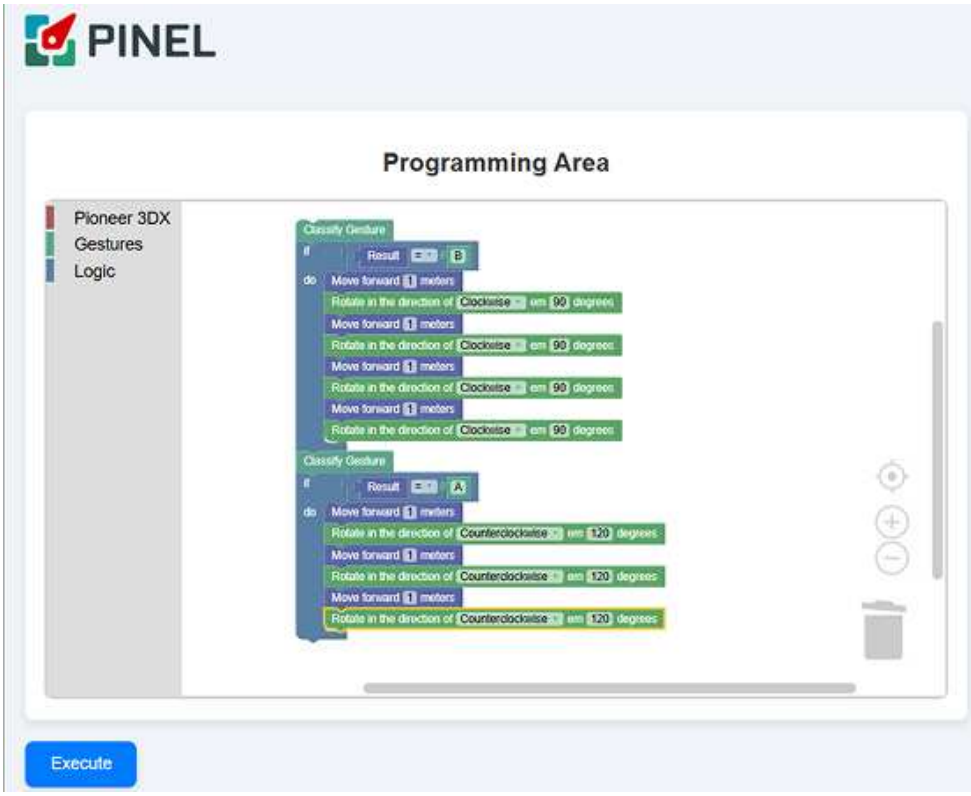


Figure 14 – Block program combining gesture classification and conditional logic. Each condition leads to the execution of a distinct geometric trajectory.

During execution, the neural network–based vision module correctly detected the gestures performed by the user in real time (Figure 15), returning labels with high confidence. The classification result was propagated to the frontend and associated with the `Result` variable, which served as input to the conditional blocks. This mechanism demonstrated the platform's ability to associate external inputs with the logical flow of visual programming, allowing decisions to be made dynamically according to the recognized gestures.

Execution in the simulator confirms the correct functioning of the logical chaining (Figure 16). The plots first show the square trajectory, followed by the triangular one,

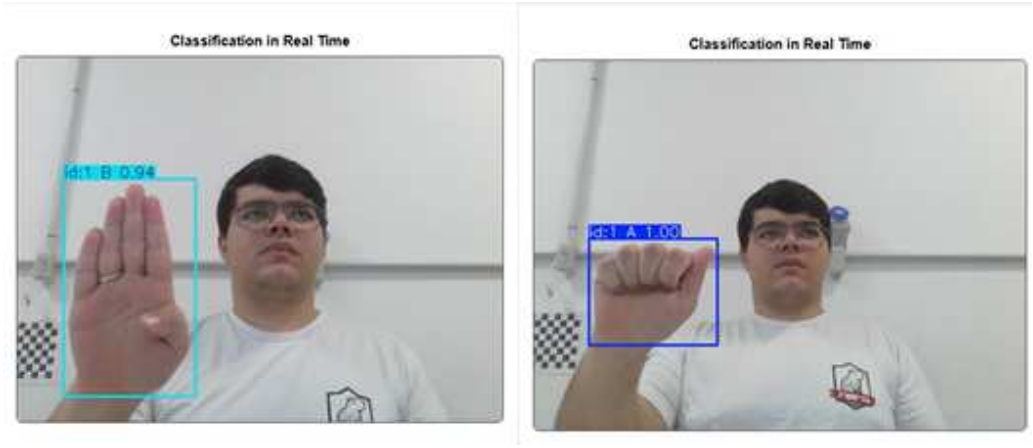


Figure 15 – Real-time classification of gestures performed by the user. Gesture "B" triggers execution of the square trajectory, while gesture "A" starts the triangular trajectory.

each with its respective orientation variations and control signals. It is observed that the backend processes the commands sequentially, respecting the order defined by the frontend. One command is completed at a time, and only then is the next sent for execution, remaining in an intermediate state similar to a cache until it is released. This approach ensures predictable behavior and prevents overloading the control node.

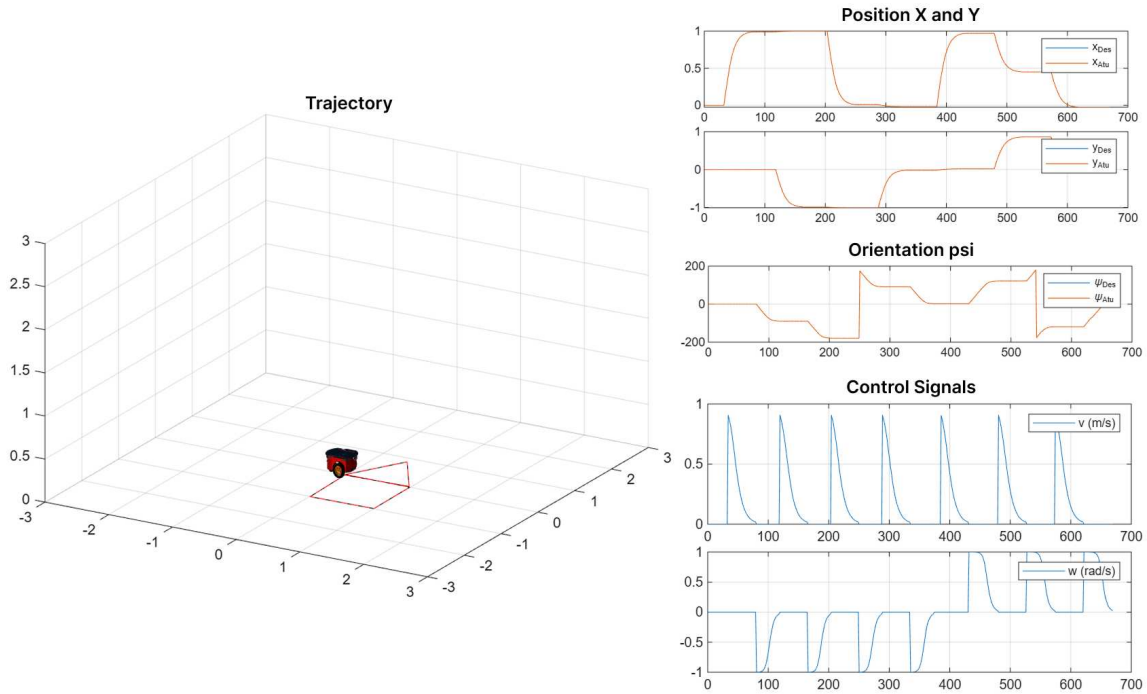


Figure 16 – Results of the simulator execution of the program based on gesture-classification blocks and conditional logic. The first part corresponds to the square (gesture "B"), followed by the triangular trajectory (gesture "A").

When the user assembles the block program and starts execution, the frontend traverses the block tree in the *workspace* and produces an ordered list of JSON objects (Listing 4.1). This list encapsulates both the result of the last recognized gesture and the

logical condition to be evaluated, along with the subsequent commands. In the example below, the first object updates the gesture label stored in the backend, while the second describes a conditional execution that is only triggered if the declared expression is true. This arrangement preserves deterministic order and makes the parameters of each action explicit (distance in meters, angle in degrees):

Listing 4.1 – JSON generated by the frontend for the conditional block with gesture classification

```
[{
    "action": "set_gesture_result",
    "result": "B",
    "status": "Gesture result updated in backend"},
  {
    "action": "conditional_execution",
    "condition": "result == \"B\"",
    "commands": [
      { "action": "move_forward", "distance": 1 },
      { "action": "turn", "angle": -90 },
      { "action": "move_forward", "distance": 1 },
      { "action": "turn", "angle": -90 },
      { "action": "move_forward", "distance": 1 },
      { "action": "turn", "angle": -90 },
      { "action": "move_forward", "distance": 1 },
      { "action": "turn", "angle": -90 }]
  }
]
```

On receiving this list, the backend confirms that the gesture label has been updated and then processes the declarative condition (`result == "B"`). In this case, the expression has been evaluated as true and the internal commands have been queued. Execution is sequential: the backend releases only one command at a time, keeping the others in memory until the previous one is completed. This mechanism is evident in the execution logs shown in Listing 4.2.

Listing 4.2 – Excerpt from the backend log during execution of the conditional block

```
Commands received: [{'action': 'set_gesture_result', 'result': 'B'}]
[PRE-EXEC] Processing set_gesture_result...
last_gesture_result updated to: B
No odometry data collected.

Commands received: [{'action': 'move_forward', 'distance': 1}]
[NORMAL] Processing action: move_forward
[INFO] [1757995676.992758100] [pioneer_controller]: Distance traveled: 0.0 meters,  Error:
    1.0
```

The first excerpt confirms the update of the gesture result (`B`), which becomes available for condition evaluation. Next, the server processes the `move_forward` motion command, publishing velocities on `/cmd_vel` and monitoring the odometry error on `/pose`. The command's execution only ends when the error falls below the configured threshold. Only then is the next command in the queue released, ensuring synchronization between what was defined in the blocks and the behavior observed in the simulator.

These results show that **PINEL** can integrate gesture recognition blocks, conditional logic, and motion control in a single execution flow, both on real robots and in the simulator, reinforcing its applicability in scenarios that require autonomy and dynamic interaction. A video recording of these simulation demonstrations is available at <https://youtu.be/ZmbWoeE4G_A?t=213>.

4.5.2 Repeat block integrated with gesture recognition

The third experiment aimed to validate the functioning of the repeat block, verifying its interaction with the robot's motion commands. For this purpose, repeat blocks were connected sequentially in the frontend, each containing rotation or linear displacement instructions. Figure 17 shows the program built in the **PINEL** frontend, highlighting the cyclic structure used.

Figure 17 – Block program combining repetition loops with rotation and linear displacement commands.

During execution, the gesture classifier was also activated, but in this case, it did not function as a trigger to start specific trajectories as in the previous experiment. Its use was limited to assisting the user in identifying the stage of execution and confirming the correct functioning of the recognition module (Figure 18). Thus, the main evaluation focused on the behavior of the repeat block and the consistency of the command sequence in the backend.

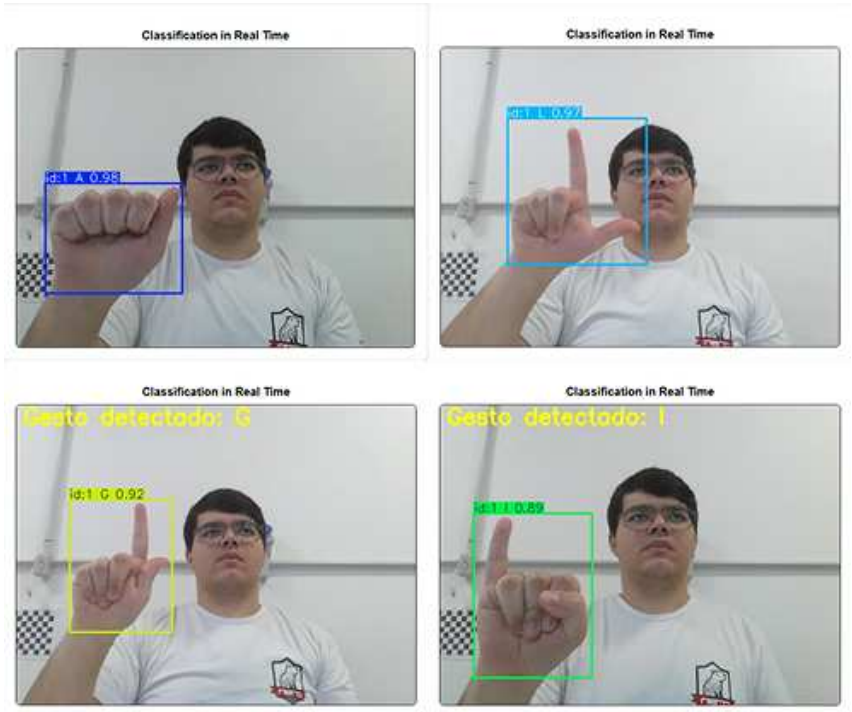


Figure 18 – Real-time gesture classification, used only as a resource for user location and classifier validation.

The results obtained in the simulator confirm the correct execution of the programmed cycles (Figure 19). The spatial trajectory shows the accumulated displacement from the repeated execution of movement and rotation commands, while the position and orientation plots highlight the periodic behavior resulting from the loops. The control signals indicate the alternating application of linear and angular velocities in successive blocks, validating the backend's ability to process extensive sequences of loop commands and complete each cycle before moving on to the next. This result proves that **PINEL** supports the execution of programs that use iteration structures, maintaining coherence between the frontend, backend, and simulator. A video recording of these simulation demonstrations is available at <https://youtu.be/ZmbWoeE4G_A?t=327>.

The analysis of the results confirms that the simulator, integrated with **PINEL**, consistently reproduced planned trajectories and elementary motion commands. In the first experiment, the execution of geometric figures showed correspondence between desired and obtained trajectories, reinforcing the reliability of the backend in generating velocities and monitoring odometry. Each movement was completed only when the ori-

entation or position error was reduced to values within the established limit, providing predictability even in longer sequences.

In the second experiment, the combination of gesture-classification blocks with conditional structures highlighted the platform's ability to integrate external inputs into the logical execution flow. The simulator correctly executed different trajectories in response to classifier results, validating communication between the frontend, backend, and vision module. The way the backend processes commands sequentially, storing them until each step is completed, proved essential to preserving execution consistency and avoiding system overload.

Finally, the third experiment demonstrated that repetition structures can be integrated into the execution flow without compromising stability. The programmed loops were fully executed, and the control signals indicated coherence in the alternation between linear and angular velocities. Although gestures were used only as an auxiliary feature, the experience reinforces **PINEL's** versatility in combining different types of blocks in the same program. Together, the three experiments demonstrate that the platform, combined with the simulator, provides a robust, reproducible, and accessible environment for developing and testing programs in educational robotics, establishing itself as a support tool for both teaching and research.

These results not only confirm the technical feasibility of the platform but also reveal its potential for expansion in different directions. Based on what was validated in this section, there is room to discuss future perspectives for evolution, both regarding new control and interaction functionalities and the expansion of its use in educational contexts. These possibilities are presented in the following section.

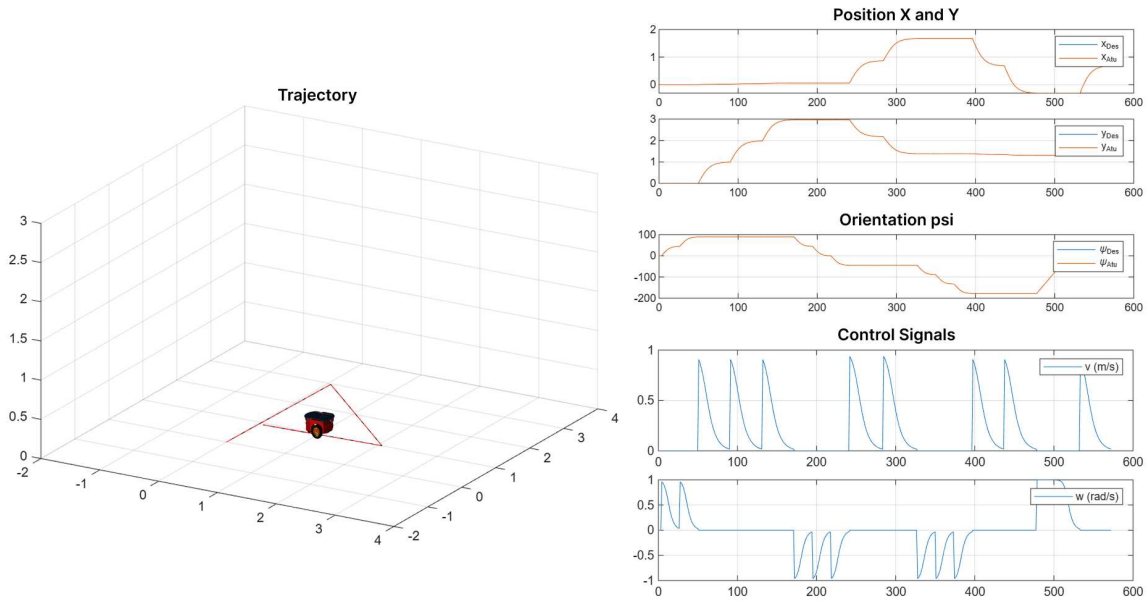


Figure 19 – Results of simulator execution with repeat blocks: spatial trajectory, positions, orientation, and control signals.

4.6 In Action: Pioneer-3DX in Tic-Tac-Toe

4.6.1 Grid Structure

To validate the implementation of the loop and gesture classification blocks, a playful scenario was designed based on the traditional tic-tac-toe game. The board was adapted into a 3×3 matrix composed of 9 cells labeled with the letters A, B, C, D, E, F, G, I, and L, arranged on a Cartesian plane marked on the floor of the experimental environment. The letters H and J were not included because the current gesture recognition model only supports static patterns (still images), whereas those two LIBRAS signs require motion.

Each cell corresponds to a specific coordinate on the Cartesian plane, with uniform spacing, where the robot can move linearly in 1-meter units (adjacent movement) or diagonally, covering a distance of approximately $\sqrt{2}$ meters. Figure 20 illustrates the layout of the tic-tac-toe cells in the experimental environment, along with the exact coordinates of each position.

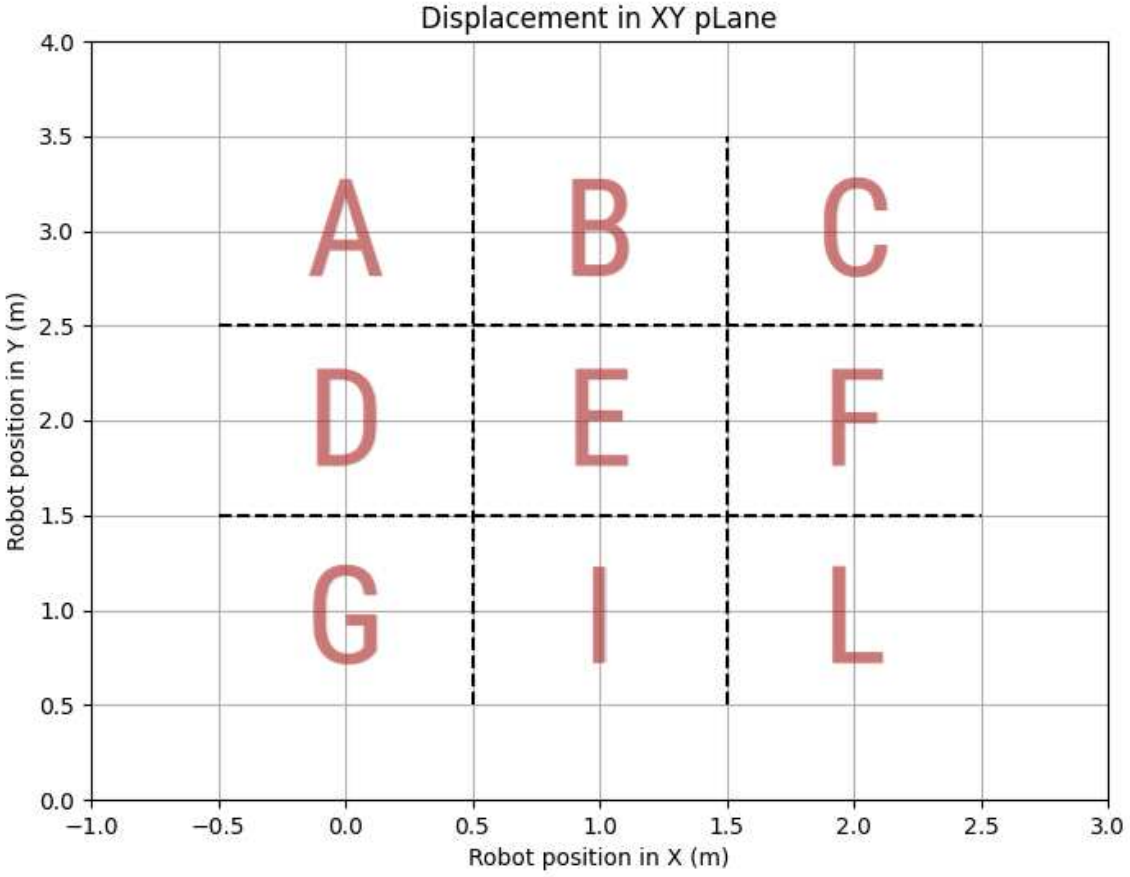


Figure 20 – Grid structure of the tic-tac-toe game with LIBRAS letters corresponding to positions on the Cartesian plane.

4.6.2 Initial Letter Assignment

For the experimental validation, a fixed letter assignment scenario was adopted, in which each board position corresponded to a specific static letter of the LIBRAS alphabet, ranging from A to L. Letters that require motion for proper representation in LIBRAS were not included. This fixed mapping was maintained to simplify observation during testing and to ensure consistent conditions for evaluating the system's performance. In the experiments, the robot was programmed to navigate to the positions associated with the letters A, L, G, and I, allowing for a focused assessment of the newly implemented loop and gesture recognition blocks.

4.6.3 Route Programming

The robot's route was programmed through the Blockly platform using the newly implemented loop and gesture recognition blocks. Figure 21 shows the exact sequence of commands visually assembled by the user, including frequent use of the loop block and conditional commands associated with gesture recognition.

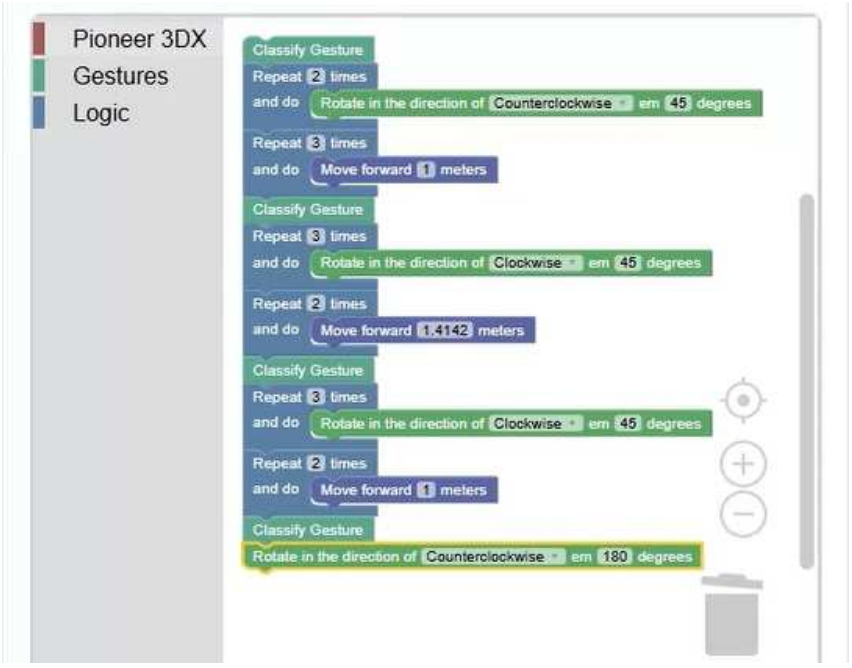


Figure 21 – Blockly command sequence used to program the route followed by the robot.

The robot began its path at the initial position (0,0), outside the game board, and moved to the positions previously defined by the selected letters (A, L, G, and I), performing the following actions for each cell:

- Capturing the letter through the gesture classification block, where the user performs the corresponding LIBRAS gesture;

- Executing the commands defined by the loop block, which include rotations in multiples of 45° and precise displacements (1 meter or $\sqrt{2}$ meters) until reaching the corresponding cell.

4.6.4 Game Objective

The objective of the game was to evaluate the platform's accuracy and robustness in recognizing LIBRAS gestures and correctly executing visually programmed iterative commands. Upon reaching the corresponding position, the robot would mark an "X" on the board. Then, the user would take their turn with an "O", alternating moves until the tic-tac-toe game was completed.

A graph of the complete trajectory followed by the robot, showing the visited and marked positions during the experiment, is presented in Figure 22, demonstrating the spatial and logical precision achieved through the use of the newly implemented blocks.

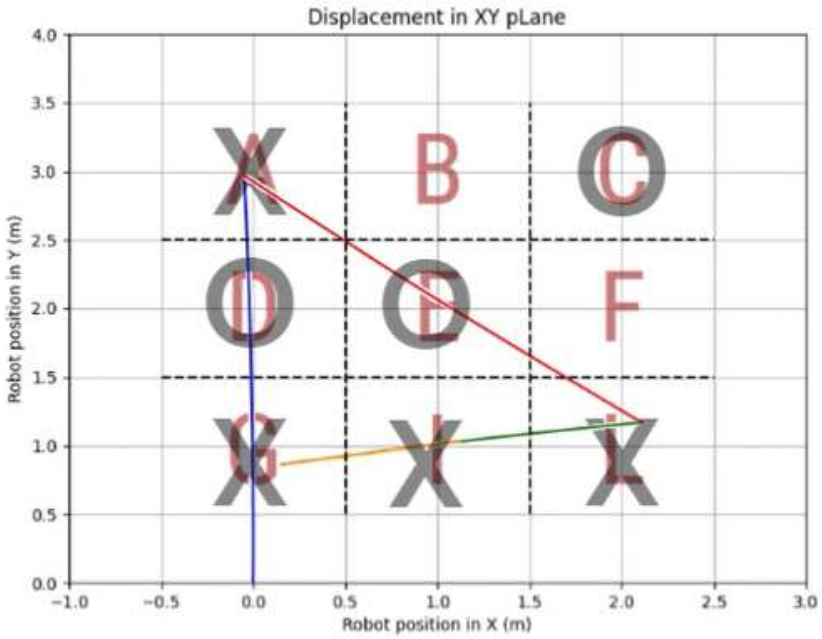


Figure 22 – Complete route of the robot during the tic-tac-toe match.

4.7 Results and Discussion

This section presents and discusses the experimental results obtained with the **PINEL** integrated with the LIBRAS gesture recognition module, validating both the loop and gesture classification blocks through the tic-tac-toe game scenario. A full video of the experiment can be accessed at <https://youtu.be/3610xcTbuLc>.

4.7.1 System Demonstration in Operation

During the experiment, the robot interacted with the user by visually recognizing the LIBRAS gestures performed. Figure 23 presents a mosaic with representative snapshots of the system operating in real time, showing the classification of the letters A, L, G, and I. It can be observed that the letter A is the only one without a bounding box. This occurs because, being the first capture after backend initialization, there is a delay in activating and starting gesture capture. Since the total capture time is fixed at 10 seconds, by the time the camera begins, the remaining time is already ending, preventing the bounding box from being generated in time.

Each classified gesture was automatically converted into specific commands sent to the robot, which then executed precise movements on the board.

4.7.2 Scenario: Fixed Letters (A–I)

In this experimental scenario, predefined letters (A, L, G, and I) were used, allowing for the evaluation of both gesture recognition accuracy and the robot's ability to follow the programmed trajectories. Figure 24 shows the individual paths taken by the robot for each selected letter.

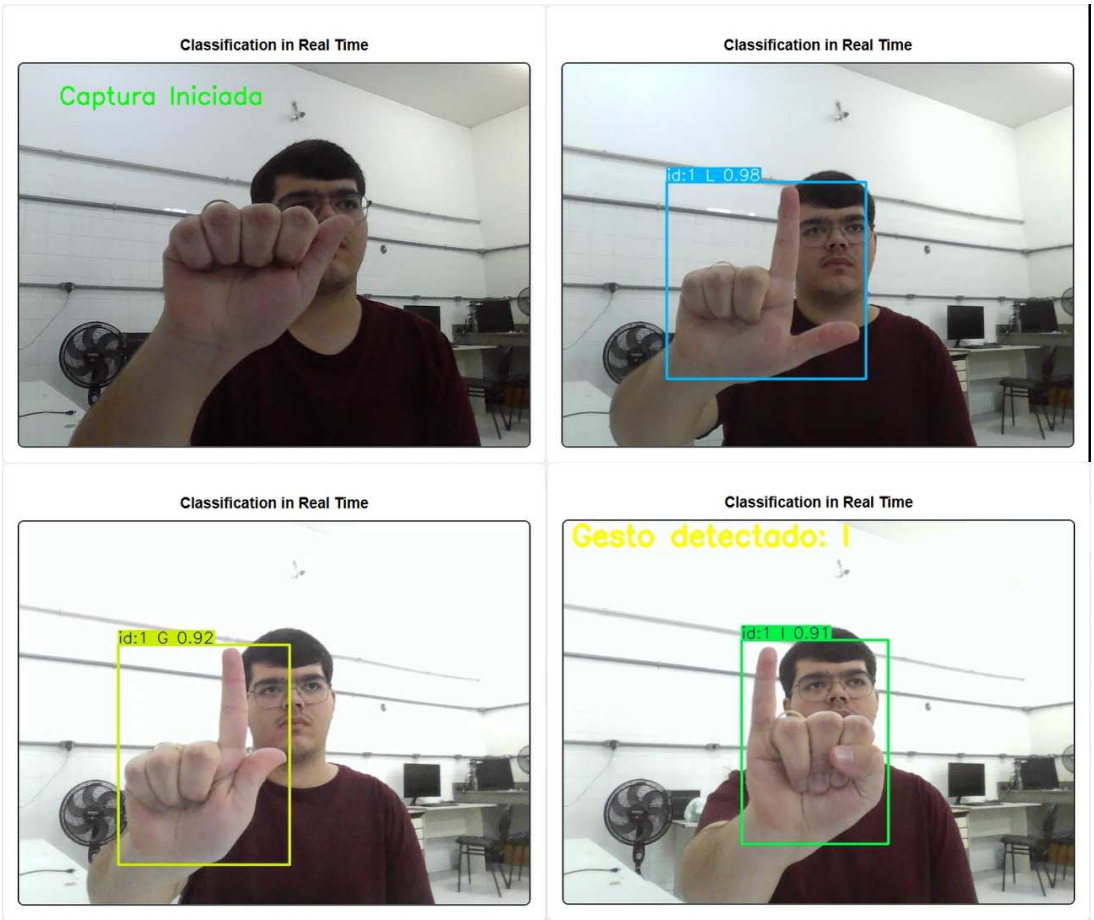


Figure 23 – Mosaic of real-time LIBRAS gesture recognition.

The complete trajectory, integrating all positions visited by the robot during the experimental game session, is shown in Figure 25, confirming the system's success in executing the programmed commands.

4.7.3 Qualitative Evaluation

The system demonstrated high accuracy in recognizing LIBRAS gestures, with an average rate exceeding 90%, confirming the robustness and effectiveness of the YOLOv8 model employed. This reliable classification ensured that user inputs were consistently interpreted as intended.

In terms of navigation, the robot exhibited precise movement control, consistently reaching the predefined coordinates with negligible positional errors. This level of accuracy guaranteed the correct execution of programmed trajectories on the game board, reinforcing the system's operational reliability.

The interaction between the user and the robot was smooth and intuitive, with prompt recognition of visual commands followed by their immediate execution. This performance confirms the effectiveness of the newly implemented loop and gesture classification blocks in playful educational scenarios, highlighting their potential for fostering interactive and inclusive learning experiences.

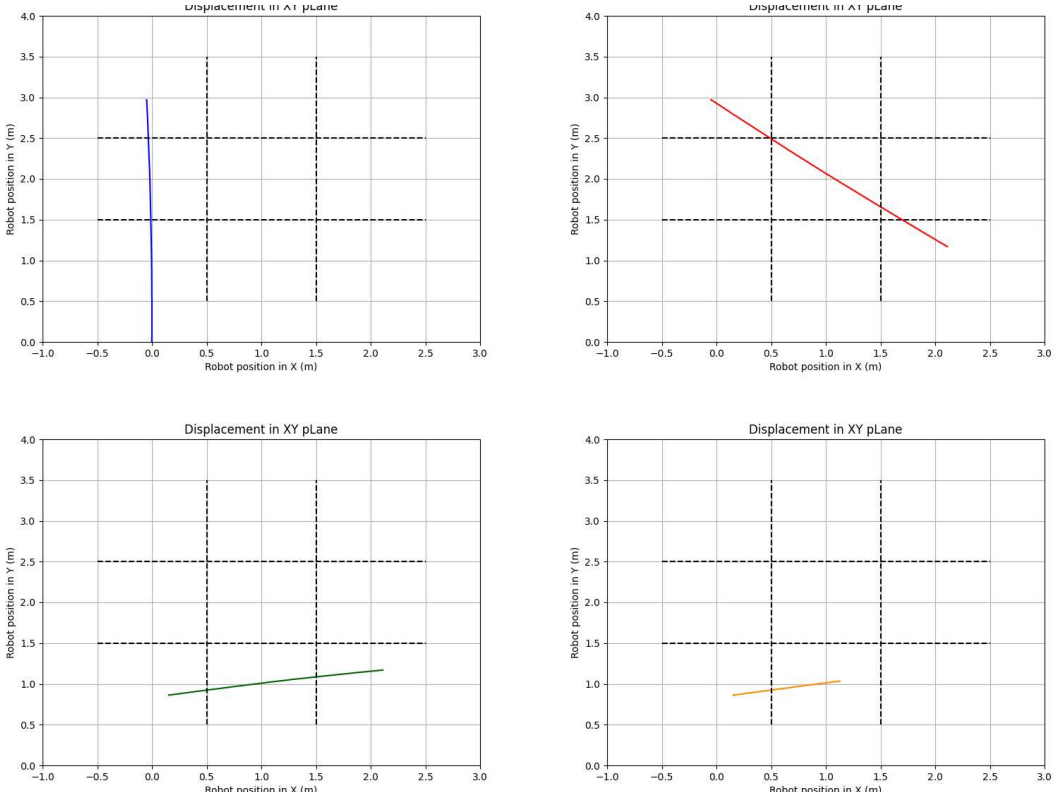


Figure 24 – Mosaic showing individual trajectories for letters A, L, G, and I.

4.8 Concluding Remarks

This work presented an expansion of the Blockly–ROS platform for mobile robot control, introducing loop blocks and LIBRAS gesture recognition, validated through a playful scenario based on the tic-tac-toe game. The results demonstrated that the system is effective both in identifying gestures and in executing the programmed trajectories with precision, reinforcing its potential as an educational tool.

It is worth noting that the command validation conducted in this study was carried out exclusively in a laboratory setting and has not yet been tested with real users in an educational context. Therefore, the current results represent a methodological proposal aimed at simulating and demonstrating the technical and pedagogical feasibility of the platform.

The experiments demonstrated the effectiveness of the approach: geometric figures were drawn accurately, conditional blocks based on gesture classification proved to be functional and repetition structures allowed long sequences of commands without loss of performance. These results confirm that integrating visual abstraction and execution in ROS is feasible and offers a balance between pedagogical accessibility and technical rigor. In addition, the odometry-based feedback control strategy ensured that movement commands were only completed when position and orientation targets were reached,

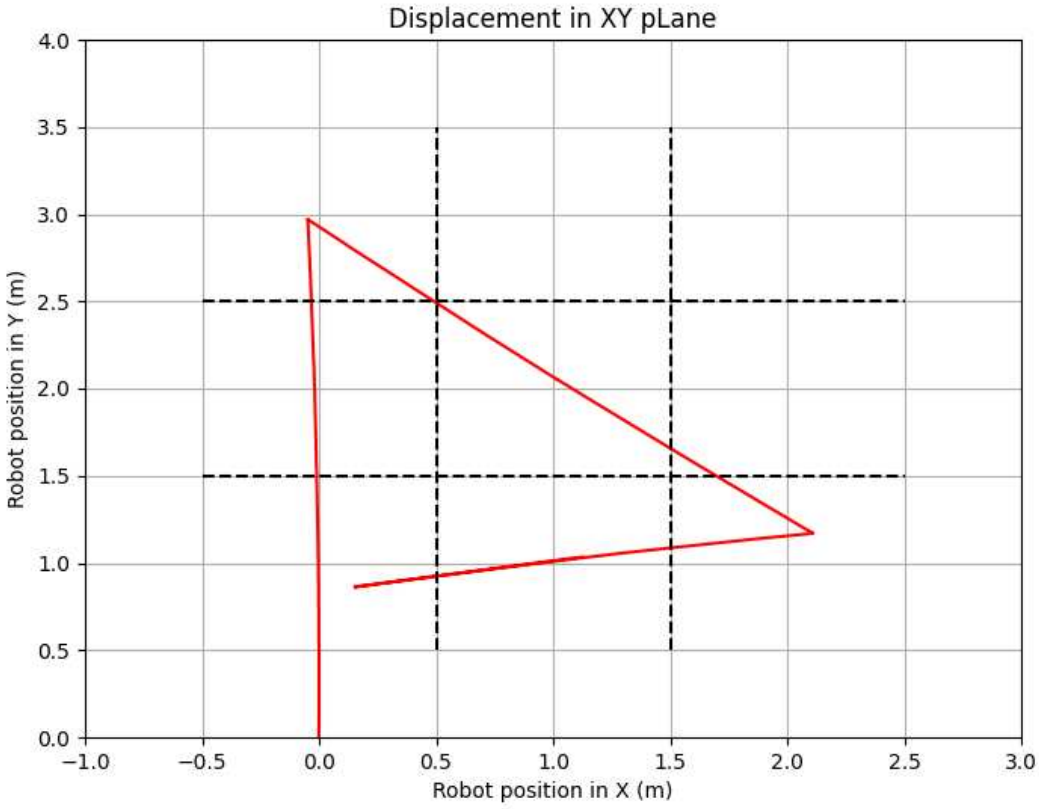


Figure 25 – Complete route followed by the robot during the game.

providing reliability to the process.

From an educational standpoint, the platform is expected to offer important benefits, notably the training of motor laterality, improvement of motor coordination, and the playful and interactive teaching of LIBRAS. The combination of computer vision with robotic control provides a dynamic and accessible environment, encouraging natural interactions between users and robots.

For future work, several opportunities for expansion have been identified. These include the incorporation of numbers and facial expressions into the recognition module, further enriching the system's interaction capabilities. Additionally, the implementation of sensors integrated into the blocks is suggested, which could assist in more advanced control strategies. Another relevant contribution would be the creation of an interface for automatically translating block-based code into Python, take advantage of a key feature of Blockly that allows visual algorithms to be converted into code. Lastly, it is recommended to enhance basic command blocks such as *else*, *elif*, *for*, and *while*, further expanding the platform's pedagogical and functional potential.

5 Educational Validation: A Case Study with PINEL in the Final Years of Education Fundamental

This chapter presents the validation of the **PINEL** in a school setting, highlighting its integration with ROS 2 and a computer-vision module (YOLO) for recognizing LIBRAS hand gestures. The platform allows students to control a Pioneer 3-DX robot through intuitive gesture-based commands within a block-based programming environment. A qualitative study was conducted with twenty 9th-grade students working in groups over two class hours, with data collected through participant observation, field notes, audiovisual records, student artifacts, and a post-activity discussion. This validation explores how the platform supports collaborative learning, fosters the development of Computational Thinking skills (decomposition, abstraction, algorithm design, and debugging), and enables multimodal interaction between students and the robot. The chapter emphasizes the pedagogical potential of combining block-based programming, AI-driven gesture recognition, and ROS 2 to create engaging, inclusive, and educationally meaningful learning experiences.

5.1 Introduction

The rapid technological evolution of recent decades has led to profound social transformations, reshaping the job market and demanding new competencies for the full exercise of citizenship. In this scenario, the educational environment is challenged to go beyond traditional teaching models, making room for methodologies that foster active, hands-on learning aligned with the demands of the 21st century (NUNES; VIANA; VIANA, 2021). In this sense, Educational Robotics (ER) emerges as a prominent pedagogical tool capable of driving this transition by providing a dynamic and multidisciplinary learning environment.

Moreover, the educational field has also incorporated other emerging technologies, such as Artificial Intelligence (AI), a branch of computer science that simulates human capabilities like reasoning and decision-making (FUJIYOSHI, 2024), which has become a reality in schools, transforming pedagogical practices and learning processes (EKIN et al., 2025). Applications range from intelligent tutoring systems to adaptive platforms that personalize content to students' individual needs (SAHAR et al., 2025). These tools not only identify learning difficulties early but also provide immediate feedback, optimizing teachers' time and boosting student performance (GUO; ZHENG; ZHAI, 2024).

By integrating ER and AI into K–12 education, we aim to develop and enhance Computational Thinking (CT), which involves processes such as abstraction, decomposition, pattern recognition, and algorithms, enabling students to develop logical and systematic reasoning (LIGEIRO; JACINTO; PIEDADE, 2024). This approach is particularly relevant in educational contexts where the integration of CT with AI and ER—spanning subjects like mathematics—has shown potential to enrich learning by fostering connections between abstract concepts and practical applications.

Accordingly, the objective of this paper is to analyze the performance of a visual programming platform as an educational tool in a classroom activity, considering the user experience and the collaborative process of students in the final years of middle school. To address this goal, the section *Educational Robotics and Artificial Intelligence in the Educational Process* discusses the theoretical foundations and the potential of these technologies in schools. Next, the section *Computational Thinking* addresses the concept, associated competencies, and relevance for skills development. The *Methodology* section describes the investigative path, procedures adopted, and instruments used for data collection and analysis. In *Results and Discussion*, we present the study's main evidence and interpret it in light of the literature. Finally, we offer *Final Considerations*, summarizing the contributions of the research, pointing out limitations and perspectives, and suggesting future practices.

5.2 Educational Robotics and Artificial Intelligence in the Educational Process

ER and AI can play a significant role in the educational process of K–12 students. Far from merely applying technology in the classroom, ER is grounded in a solid pedagogical basis whose main pillar is Constructionism, a theory developed by Seymour Papert.

Papert—a mathematician and educator who collaborated with Jean Piaget—expanded his mentor's constructivist theory. While Piaget's constructivism holds that knowledge is built through the individual's interaction with the environment, Papert (PAPERT, 2008) argues that this process is significantly enhanced when the learner is engaged in building a public and meaningful artifact (SANTOS; OLIVEIRA, 2025). The central idea is not that the computer teaches the child, but that the child teaches the computer—that is, programs it. In doing so, the learner mobilizes mental structures, reflects on actions, and consolidates knowledge in a practical, concrete way. This "learning by doing" philosophy is the essence of Constructionism.

Historically, this approach originated with the LOGO programming language and the "turtle," a simple robot that allowed children to visualize the results of their commands

immediately and tangibly (SANTOS; SILVA; ANDRADE, 2024). When programming the turtle to draw geometric shapes, for example, children learned not only geometry and programming, but also about their own reasoning processes.

Contemporary ER realizes and expands this vision. Through assembling and programming robots, students cease to be mere recipients of information and take on a protagonist role in their learning (SANGALI; CATABRIGA; BOERES, 2024). The use of assembly kits, sensors, and actuators enables abstract concepts, especially in STEAM (Science, Technology, Engineering, Arts, and Mathematics), to be explored in practical, experimental ways (ATEŞ; GÜNDÜZALP, 2024).

One of the most promising areas in education is integrating AI with ER, which can elevate student–robot interaction to a new level. AI-equipped robots can adapt behaviors, respond to gestures, and even "see" through computer vision, making learning more dynamic and immersive (FUJIYOSHI, 2024). Technologies like YOLO (You Only Look Once), capable of detecting objects and gestures in real time with high precision (NGUYEN; NGO; NGUYEN, 2025), allow students to control robots with bodily movements, turning abstract programming concepts into tangible and intuitive experiences (HAN et al., 2025).

This integration not only makes learning more engaging but also develops cognitive and socio-emotional skills. By replacing traditional interfaces with physical and gestural interactions, AI in ER brings education closer to contemporary technologies, preparing students for a future in which human–machine collaboration will be increasingly essential (WANG et al., 2024). Thus, AI becomes an ally in promoting innovative, accessible educational experiences aligned with 21st-century demands.

This approach transforms classroom dynamics. The teacher shifts from being the sole holder of knowledge to acting as a mediator and facilitator of the learning process, while students are encouraged to investigate, formulate hypotheses, test solutions, collaborate in groups, and learn from mistakes (NUNES; VIANA; VIANA, 2021). As highlighted by Sangali, Catabriga e Boeres (2024), through robotics activities students learn to work in teams, plan and execute projects, test hypotheses, and correct mistakes, fostering active and meaningful learning.

In this context, the articulation between ER and AI, allied with constructionist foundations, proves to be a powerful strategy in students' educational processes. As argued by Azevedo (2022), education should not be seen as mere content acquisition but as a living, organic, and non-linear dynamic driven by creativity, agency, and social and intellectual engagement. When students build meaningful robotic artifacts, they not only mobilize interdisciplinary knowledge but also develop the ability to act in the world with responsibility and criticality. Education thus goes beyond technical–scientific mastery and is consolidated as an emancipatory process that integrates reason, sensitivity, and

transformative action, helping to form individuals capable of ethically, innovatively, and responsibly intervening in reality for the common good.

5.3 Computational Thinking

Incorporating CT into K–12 education requires pedagogical strategies aimed at students' cognitive development (NG; GONZÁLEZ-CALERO; JACINTO, 2024). As evidenced by Ligeiro, Jacinto e Piedade (2024), the use of exploratory tasks and accessible tools—such as block-based programming—facilitates the transition from concrete to abstract reasoning. This perspective aligns with the view that CT is not limited to mastering programming languages; rather, it represents a way of thinking that can be cultivated from the early years of elementary school, preparing students for the challenges of an increasingly digital world.

In this context, programming has become an essential skill, recognized not only as a tool for training technology professionals but also for broadly developing CT in students (CHENG et al., 2024; RIBEIRO; OLIVEIRA; DUSI, 2025). CT, understood as the ability to solve problems in a structured and logical way, is increasingly in demand in various fields. However, introducing programming into school environments, especially in K–12, faces significant challenges.

Traditionally, programming instruction is associated with text-based languages such as Python or C. Although powerful, these languages are complex for beginners, mainly due to rigid syntax and the abstract nature of their concepts. A single syntax error (like a missing semicolon) can prevent a program from running, causing frustration and disengagement (MARINHO, 2022). This initial barrier hinders understanding of programming's logical principles, as the student's focus shifts to memorizing complex syntactic rules.

To overcome these obstacles, visual programming languages (block-based programming platforms) emerged. Environments like Scratch and Blockly revolutionized instruction by abstracting syntactic complexity. In these platforms, students build algorithms by dragging and connecting graphical blocks that represent commands, loops, conditionals, and variables. This approach allows students to focus on logical reasoning and problem-solving rather than worrying excessively about the formal structure of the language (BILGIC; DOGUSOY, 2023; SANTOS et al., 2024).

Scratch, developed by the MIT Media Lab, is among the most popular tools in this segment. It offers a playful, interactive environment where students can create games, animations, and stories, making learning more engaging and meaningful (SANTOS et al., 2024). By manipulating blocks, children and teens learn fundamental programming concepts in an intuitive and experiential way.

Complementarily, Google's Blockly library offers even greater flexibility. While Scratch is a closed environment, Blockly is a toolkit that enables the creation of customized block-based programming editors for specific applications (PINEL; BRANDÃO; FARIA, 2025). Its main advantage lies in generating source code in text-based programming languages such as JavaScript, Python, and others (SUTHERLAND, 2022). This feature positions Blockly as an excellent bridge between visual and text-based programming. Students can begin with blocks and gradually view the corresponding generated code, facilitating a smoother transition to more advanced programming paradigms (MARINHO, 2022).

Studies show that using block-based programming (whether Scratch or Blockly) positively impacts student engagement and programming self-efficacy (CHENG et al., 2024; TANG; QIAN; PORTER-VOSS, 2024). The ability to obtain immediate, visual results—especially when programming is connected to physical artifacts like robots—makes learning more tangible and motivating. Seeing a robot execute a sequence of commands created by combining blocks reinforces students' understanding and confidence in their own abilities.

Therefore, teaching programming in K–12 is most effective when it adopts a progressive approach: starting with the abstraction and playfulness of block-based programming and evolving toward the complexity of text-based programming. Tools like Scratch and Blockly are fundamental in this process because they democratize access to basic programming concepts and prepare students for more complex challenges, enabling them to become not only consumers but also creators of technology.

5.4 Methodology

This research is characterized by a qualitative approach whose objective is to analyze the performance of the visual programming platform as an educational tool in a classroom activity, considering the user experience and the collaborative process of students in the final years of middle school. Qualitative research, according to Bogdan e Biklen (1994), values understanding phenomena in their natural settings, attending to process complexity and participants' perspectives.

The hands-on activity was carried out over two class periods at Escola Estadual Doutor Mariano da Rocha in Teixeiras, Minas Gerais. Twenty 9th-grade students participated. The class was organized into five groups of four students to foster collaboration and idea exchange. During the activity, we had technical and pedagogical support from the classroom teacher and six research assistants, members of the Robotics Specialization Center (NERO) and the Group for Attention to Technologies in Education (GATE), which conduct studies on Educational Robotics. Both groups are affiliated with the Federal

University of Viçosa (UFV), in which we actively participate.

The programming platform used in this study is a web application that employs the Google Blockly library to provide a visual development environment. In this interface, users build programs by manipulating and connecting graphical blocks representing commands and logical structures. The logic built with blocks is then converted into a structured representation in JSON (JavaScript Object Notation) format.

The JSON commands are transmitted over the local network to a control server running on a single-board computer (Jetson Nano) which, together with the Pioneer 3-DX mobile robot (Figure 26), composes the robotic system. The server interprets these commands and translates them into robot instructions. Communication between the server and the robot is managed by Robot Operating System 2 (ROS 2), which acts as middleware. The Pioneer robot belongs to NERO and thus represents a low-cost research apparatus.

Figure 26 – Robotic system used: Pioneer 3-DX robot and Jetson Nano single-board computer.

A distinctive component of the platform is the computer vision module for gesture recognition. This module uses an AI model based on the YOLO (You Only Look Once) architecture, trained to classify static gestures corresponding to letters in LIBRAS (Brazilian Sign Language), captured in real time by the system's camera (Figure 27). Identifying a specific gesture can be used as a condition to trigger sequences of robot actions.

Data collection instruments included participant observation, field notes, audio-visual records (photos and videos), student-produced artifacts (planning with physical blocks and implementation on the digital platform), and a group discussion to capture students' testimonials, perceptions, opinions, and evaluations about the activity and the

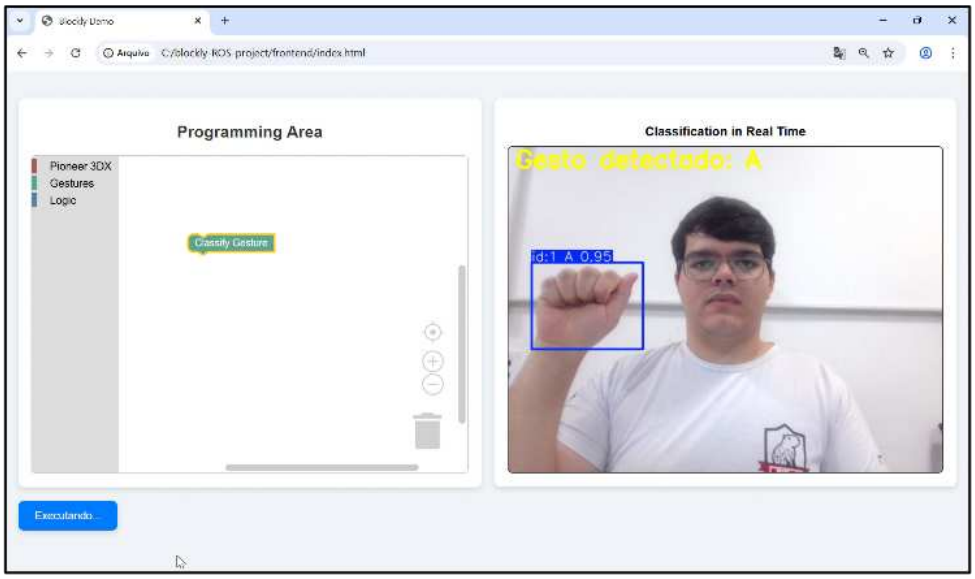


Figure 27 – Interface of the gesture recognition system identifying a letter in LIBRAS.

tool. With these records, we analyzed the data and identified emerging patterns related to student collaboration, CT strategies, platform usability, and the perceived pedagogical potential of using block-based programming for robotic control in a challenge-based learning context.

5.5 Results and Discussion

Data were collected in April 2025 during an intervention designed to progressively introduce concepts of robotics and programming logic and culminate in the practical application of this knowledge. We began with an introductory moment about robots and robotics, revisiting prior knowledge and situating programming as a language for communication with automated systems (Figure 28). In this initial dialogue, one student stated that "the robot vacuum runs on several commands," which served as an introduction to make explicit that robot behavior depends on clear, sequenced instructions.

Figure 28 – Students during the introductory class on robotics and programming.

Next, we presented the Google Blockly Programming Platform. We detailed the interface, the functionalities of each available command block (such as move forward,

rotate, classify gesture, if, result, and text), and demonstrated practical examples of its use for robot control (Figure 29).

Figure 29 – Presentation of the block-based programming platform interface.

Given that the platform's gesture recognition system was trained to identify letters of the alphabet in LIBRAS due to the nature of recognizing static image patterns, we conducted a teaching–practice session on that alphabet (Figure 30). This step aimed not only at technical preparation for the activity but also at raising awareness of LIBRAS as a means of communication and inclusion.

Figure 30 – Presentation and practice of the LIBRAS alphabet.

We then explained how the program logic would be executed from a gesture. For this, we presented an example in which, upon making the sign for the letter "B" (Figure 31), the robot should trace a path that draws that letter on the floor.

Figure 31 – Classifying the gesture within the program.

At that moment, students showed great interest and began to understand the relationship between input (gesture), processing (blocks), and output (robot movement). They were then asked about the classify-gesture block, and the following dialogue occurred:

Researcher: What will happen when you make the sign for the letter b?
Student 1: It will work.
Student 2: I don't know.
Researcher: Has the program already understood the code?
Student 3: No, it will ask to classify again.

This dialogue highlights the process of constructing knowledge and understanding the commands. According to Papert (2008), computers can support new forms of relationships with knowledge because learning is building knowledge in the student's mind, and the best way to do that is when the learner builds something outside themselves that is meaningful (PAPERT, 2008). During this activity, students had to construct the path to be traced by the robot using their understanding of distance units and angles, as well as the content taught in the lesson.

In short, the dialogue evidenced a shift from intuitive understanding to a procedural control model, in which the input–processing–output relationship ceases to be implicit and comes to depend on explicit, ordered instructions (invoke the classification block, wait for the result, then trigger movement). This shift is typical of the constructionism described by Papert (2008), as the student restructures concepts while designing and testing a meaningful artifact; at the same time, core CT practices are activated, such as problem formulation, condition definition, and flow debugging. Thus, the episode not only illustrates student engagement but also consolidates the understanding that the robot does not "understand" on its own: it executes what has been logically specified, validated, and adjusted at each iteration.

Each group received a kit containing physical representations of the programming blocks, laser-cut in MDF, to discuss, plan, and collaboratively assemble a logical sequence of commands. The challenge was to program the robot to draw, through its movements, a letter of the alphabet chosen by the group (Figure 32). While groups worked, research assistants followed the development and answered questions.

Different strategies emerged during planning. One group chose the letter "s" and asked to measure the classroom size to estimate how many meters the robot could travel, mobilizing notions of measurement units and scale. Other groups chose to sketch the path on paper before programming. One group chose the letter "a," represented a triangle, set 1 meter for each side, and used 120° as the exterior angle, anticipating the sequence of rotations and displacements (Figure 33). In this process, students identified the problem,

Figure 32 – Student groups planning with physical MDF blocks.

structured it, and planned the next steps, integrating plane geometry and algorithmic thinking.

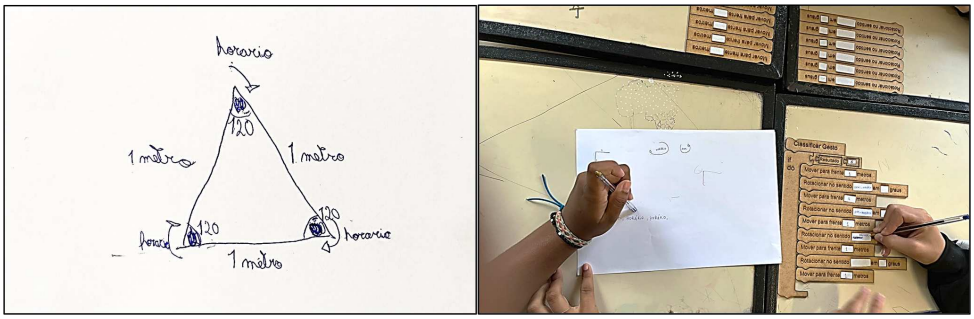


Figure 33 – Planning the path to be traced by the robot.

Discussion about order and dependency among commands also appeared in the dialogues. In one group, when debating whether they should place "move forward" after "turn," the following exchange occurred:

> **Student 1:** After turning it will stop. So it has to move first, then turn.
> **Student 2:** No need, because it turns and moves a bit.
> **Student 1:** No, it only turns.
> **Student 2:** I don't get it. Doesn't it turn and move?
> **Student 1:** No, it only turns. It doesn't move unless we tell it to.

From this discussion, progress in understanding explicit control became evident: turning and moving are independent actions and must be specified and ordered correctly in the sequence of blocks. In another presentation, when the robot executed the letter "a," students noticed an extra turn, and one concluded: "It's wrong—it's turning four times; we only needed three," showing evaluation and debugging of logic during execution.

The two dialogues above show moments when students noticed an error in their activity execution. Student 2 had assumed the robot was always moving forward, so only turning would be necessary. However, in talking with a peer, she realized each part of the robot's movement had to be better specified, following the order and dependency among commands leading to the expected outcome. According to Santos, Santos e

Javaroni (2021), the reorganization of thinking that occurs during the teaching–learning process—especially how technologies transform ways of thinking, as in the situations analyzed—produces new knowledge. As those authors argue, in the situations described, this reorganization proved to be a process by which mathematical, cognitive, and social practices are transformed when new instruments and media are integrated, producing changes in both teaching and learning.

Executing geometric trajectories recontextualized measures and angles into hands-on learning, shifting reasoning from verbal commands to the materiality of robot motion. By estimating rotation amplitudes and correcting distances between attempts, groups evidenced a cycle of plan → execute → observe → debug, in which the robotic artifact functioned as a cognitive mediator to externalize hypotheses and adjust strategies. This pattern is consistent with constructionist perspectives on learning in ER, in which students build knowledge through interaction with artifacts (PAPERT, 2008; BLIKSTEIN, 2013).

After all teams assembled their sequences with the physical blocks, a representative transposed the logic to the digital platform by dragging and connecting the corresponding virtual blocks (Figure 34). With the program ready, a group member made the LIBRAS gesture in front of the camera; recognition then triggered execution, and the whole class watched the robot's movement to check alignment between planning and outcome. This moment also encouraged comparisons among solutions and real-time adjustments as groups identified possible improvements by observing their peers' presentations.

We observed productive tolerance of error: when the figure did not close, students justified parameter changes (e.g., "add 10° to the last turn" or "reduce linear velocity")

Figure 34 – Program execution with a student performing the LIBRAS gesture and the robot in motion.

before resubmitting the program. This public explanation of reasoning made quality criteria visible (path closure, symmetry, and return to origin) and favored solution transfer among groups, reinforcing the idea that ER promotes iterative reasoning and solution engineering in authentic contexts (PAPERT, 2008; BLIKSTEIN; WORSLEY, 2016).

The concept of CT, as proposed by Wing (2006), consists of reformulating an apparently difficult problem into another we know how to solve—by reduction, embedding, transformation, or simulation. Therefore, we consider that CT was mobilized organically throughout the activity. In paper planning, students explored problem formulation; in using blocks, they practiced pattern recognition and algorithm design; in public executions, they analyzed solutions and performed refinements. Thus, robotics functioned as an integrating resource, enabling learning to occur actively, collaboratively, and across disciplines by articulating programming, geometry, and measurement toward a tangible and inclusive objective (the LIBRAS letter).

Thus, these classroom episodes are understood, according to Ligeiro, Jacinto e Piedade (2024), as CT items in which students elaborate and follow detailed instructions, create or interpret algorithms to solve specific parts of a problem and also demonstrate the ability to decompose problems, organize thinking and carry out sequential and logical steps. Furthermore, by using blocks, students were anticipating the results they would obtain prior to running the activity on the computer.

At the end, a group discussion was held to systematize perceptions and gather student feedback (Figure 35). Students highlighted challenges (estimating angles, adjusting distances, synchronizing gesture and execution) and learnings (importance of block order, clarity of commands, incremental testing) from the activity. This moment reinforced the reflective dimension of the process and pointed to directions for adjustments in platform design and in teachers' planning of future activities.

Figure 35 – Group discussion with students at the end of the activity.

At the CT level, problem decomposition (segmenting the trajectory), abstraction (representing curves as sequences of micro-movements), and algorithm design (ordering blocks with distance and rotation parameters) emerged. Debugging occurred when discrepancies between the planned and the executed required tracing variables and adjusting reusable heuristics (e.g., "last turn with a negative correction if the robot overshot").

These indications align with classical CT references and the role of block-based interfaces in reducing syntactic barriers while keeping the focus on strategy and sequence (WING, 2006; RESNICK et al., 2009).

Integrating gesture-based AI (the detection module) introduced multimodal interaction, reducing the need to recall syntax and bringing the student's intention closer to the robot's behavior. When ambiguous classifications occurred, groups made decision criteria explicit (repeat/adjust the gesture, reposition the camera, combine the gesture with a conditional block), turning prediction errors into metacognitive opportunities. This movement valued critical reading of intelligent systems and the notion that results must be validated under varying lighting, occlusions, and distances.

As an educational tool, the platform showed a short learning curve: after the demonstration, most groups began assembling and adjusting programs with little teacher intervention. The block interface favored the explicitness of algorithmic reasoning (order, parameters, dependencies), while robot execution provided immediate feedback to verify hypotheses. Operationally, the front-end + ROS 2 integration kept the flow responsive to attempts, allowing fast test-and-refine cycles; pedagogically, it connected programming to concrete visual and motor effects, strengthening student engagement and agency (RESNICK et al., 2009).

The user experience was marked by spontaneous role distribution (who programs, who observes the trajectory, who controls the gesture), negotiation of decisions, and collaborative debugging. During tests, students justified changes, compared results, and collectively validated the path before proceeding. This ecosystem of practices developed evidence-based argumentation, decision-making, and shared responsibility for the final artifact, consolidating the platform as a space for active learning and collaborative solution building.

Thus, we conclude that integrating block-based programming, robotics, and CT in a single environment is a promising strategy for programming instruction and competency development. By bringing abstract concepts closer to concrete, socially meaningful situations, this approach contributes to building autonomous, reflective, inclusive learning that is connected to the demands of contemporary society.

5.6 Concluding Remarks

The study enabled an evaluation, in a real classroom context, of the potential of a block-based programming platform integrated with ROS and equipped with a LIBRAS gesture recognition module as a didactic resource for developing CT and promoting active, collaborative learning.

The proposed activity mobilized multiple cognitive and socio-emotional skills,

ranging from logical planning and problem decomposition to applying mathematical concepts—such as measures and angles—in a practical, meaningful context. The use of physical MDF blocks as an intermediate step favored the transition to the digital platform and the understanding of programming logic, allowing students to visualize, manipulate, and discuss their ideas concretely before executing them with the robot.

The results indicate that integrating ER and AI can broaden pedagogical possibilities, making programming more accessible, inclusive, and motivating. Physical interaction via LIBRAS gestures not only diversified input modalities but also introduced an element of awareness and appreciation of inclusive communication—highly relevant in contemporary schooling.

Group work dynamics encouraged idea exchange, argumentation, and solution negotiation, showing that collaborative learning is strengthened when combined with concrete problem-solving activities. We also observed that experimentation and command debugging during practical execution stimulated reflection, self-assessment, and strategy reformulation, consistent with the dimensions of CT (WING, 2006).

As a practical contribution, the experience demonstrated that the developed platform is technically feasible and pedagogically effective, and it can be adapted to different grade levels and curricular content. However, we recognize that the research was conducted within a specific scope (one school with a limited number of participants), suggesting the need for future studies that explore other contexts and investigate the impact of this approach on learning over longer periods.

6 Conclusions

This chapter summarizes the main contributions of this dissertation, discusses the primary challenges encountered throughout the research, and presents possible directions for future work.

6.1 Contributions and Achievements

The main objective of this dissertation was to design, implement, and validate **PINEL** (Platform for Interactive Navigation in Education and Learning), a block-based programming platform integrated with ROS 2 and a three-dimensional simulation environment. PINEL was conceived as a modular and accessible solution, connecting high-level visual abstractions, through blocks, to low-level control mechanisms, enabling the seamless execution of the same programs on both physical robots and simulators.

To achieve this objective, the work was developed progressively, moving through stages of conceptual design, controlled validation in the simulator, and practical experiments with physical robots, culminating in a real-world case study in an educational context. The main contributions of this research include: the creation of a robust architecture integrating a Blockly-based frontend, a Python/ROS 2 backend, and the AuRoRA simulator; the definition of an intermediate JSON representation ensuring compatibility between simulated and physical environments; validation in simulation through experiments involving geometric figures, conditional logic, gesture recognition, and repeat loops; real-world implementation with physical robots, confirming the practical feasibility of the platform; the incorporation of a LIBRAS gesture recognition module based on neural networks, expanding the possibilities for multimodal interaction; and, finally, the execution of a case study with ninth-grade students, demonstrating the pedagogical impact of PINEL on learning concepts related to robotics and computational thinking.

Each of these contributions aligns with the objectives and hypotheses established in this research, demonstrating the effectiveness of the proposed approach. The progressive evolution of the platform throughout the project was essential for validating the solution across different scenarios, from controlled environments to real educational situations, showcasing its versatility and potential for future expansion.

6.2 Limitations and Challenges

Despite the promising results, some limitations were identified during the development and validation of this research. Continuous motion control of the robot still presents

challenges, as executing more complex trajectories requires more advanced planning and navigation algorithms. The LIBRAS gesture recognition module, although effective in controlled conditions, is sensitive to environmental variations such as lighting, background noise, and occlusions, which can affect classification reliability.

Additionally, as the block library grows, the challenge of maintaining an intuitive and accessible interface arises, ensuring usability for both beginners and advanced users. Another point of attention is the dependence on specific hardware: so far, PINEL has only been validated with the Pioneer 3-DX robot and Jetson Nano, requiring broader validation to confirm portability across different platforms. Finally, the educational study was conducted with a relatively small group of students, which limits the generalization of the results regarding learning and collaboration.

These challenges highlight the importance of future research aimed at improving the technical robustness of the platform, as well as its pedagogical evaluation in varied contexts, ensuring scalability and adaptability of the solution.

6.3 Final Considerations

This dissertation presented **PINEL** as an innovative solution that integrates simulation, visual programming and real robot control in a single educational environment. The results confirm that the use of visual abstractions can significantly reduce the barriers to entry for beginners, while offering powerful features for advanced robotics applications.

Based on these results, the platform demonstrates a significant advance in the field of Educational Robotics, combining accessibility, modularity and scalability. However, some limitations have been identified, which opens up important avenues for further development. Future work should therefore prioritize improving control algorithms to enable continuous and autonomous navigation, strengthening perception modules through deep learning techniques and carrying out large-scale educational validation with diverse student profiles. In addition, the integration of cloud-based solutions and remote learning environments could further extend the platform's reach and strengthen its potential as a comprehensive tool for teaching programming, robotics and computational thinking.

Expanding on these perspectives, several additional opportunities for enhancement have been identified. These include incorporating numerical gestures and facial expressions into the recognition module to enrich human-robot interaction, as well as integrating sensors directly into the block structure to support more advanced control strategies. Another promising direction involves the development of an interface capable of automatically translating block-based programs into Python code, taking advantage of Blockly's native code generation capabilities. Future developments should also consider implementing more precise control methods to improve curve accuracy, integrating LiDAR sensors to enable

autonomous navigation and expanding the Blockly library with additional conditional and repetition structures (*else*, *elif*, *for*, *while*). Together, these improvements should enhance the platform's functionality while maintaining its pedagogical accessibility.

Bibliography

AFFONSO, G. **Scratch3 ROS VM**. 2025. Accessed: 2025-02-02. Disponível em: <https://github.com/Affonso-Gui/scratch3-ros-vm>.

ALIMISIS, D. Teacher training in educational robotics: The roboesl project paradigm. **Technology, Knowledge and Learning**, Springer, v. 24, n. 2, p. 279–290, 2019.

ATEŞ, H.; GÜNDÜZALP, C. A unified framework for understanding teachers' adoption of robotics in stem education. **Education and Information Technologies**, Springer, v. 29, n. 11, p. 1–27, 2024.

ATMATZIDOU, S.; DEMETRIADIS, S. Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. **Robotics and Autonomous Systems**, v. 75, p. 661–670, 2016. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0921889015002420>.

AVULA, H.; R, R.; PILLAI, A. S. Cnn based recognition of emotion and speech from gestures and facial expressions. In: IEEE. **2022 6th International Conference on Electronics, Communication and Aerospace Technology**. Coimbatore, India, 2022. p. 1360–1365.

AZEVEDO, G. T. d. **Processo formativo em matemática: invenções robóticas para o Parkinson**. Tese (Doutorado em Educação Matemática) — Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP), Instituto de Geociências e Ciências Exatas (IGCE), Rio Claro, SP, Brasil, 2022. 213 f. Disponível em: <http://hdl.handle.net/11449/236186>.

AZMI, M. M. et al. Uniten smart programming apps using combination of robotic and block programming (roblock). In: IEEE. **2021 IEEE International Conference on Computing (ICOCO)**. Kuala Lumpur, Malaysia, 2021. p. 204–207.

BACHILLER-BURGOS, P. et al. Learnblock: A robot-agnostic educational programming tool. **IEEE Access**, v. 8, p. 30012–30026, 2020.

BARCELOS, C. O. et al. Robot formation performing a collaborative load transport and delivery task by using lifting electromagnets. **Applied Sciences**, v. 13, n. 2, 2023. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/13/2/822>.

BILGIC, K.; DOGUSOY, B. Exploring secondary school students' computational thinking experiences enriched with block-based programming activities: An action research. **Education and Information Technologies**, Springer, v. 28, n. 8, p. 10359–10384, 2023.

BLIKSTEIN, P. Digital fabrication and 'making' in education: The democratization of invention. **FabLabs: Of machines, makers and inventors**, transcript Verlag, Bielefeld, p. 203–222, 2013. Disponível em: <https://doi.org/10.1515/transcript.9783839423820.203>.

BLIKSTEIN, P.; WORSLEY, M. Children are not hackers: Building a culture of powerful ideas, deep learning, and equity in the maker movement. In: TAYLOR AND FRANCIS. **Makeology**. Northwestern Scholars, 2016. p. 64–79. ISBN 9781138847767.

BOGDAN, R.; BIKLEN, S. **Investigação qualitativa em educação: uma introdução à teoria e aos métodos**. Portugal: Porto editora, 1994.

CARDOZO, G. D. **A robótica como ferramenta aplicada à educação**. Monografia (Licenciatura em Computação), Valença, BA, 2017.

CHEN, C.-H.; CHUNG, H.-Y. Fostering computational thinking and problem-solving in programming: Integrating concept maps into robot block-based programming. **Journal of Educational Computing Research**, v. 62, n. 1, p. 186–207, 2024. Disponível em: <https://doi.org/10.1177/07356331231205052>.

CHEN, T.-L. et al. Nnblocks: a blockly framework for ai computing. **The Journal of Supercomputing**, Springer, v. 77, n. 8, p. 8622–8652, 2021.

CHENG, M. et al. Children's programming environment acceptance: extending the boundary conditions to programming competition, computational thinking, and programming modality. **Education and Information Technologies**, Springer, v. 29, n. 1, p. 939–969, 2024.

CHIKURTEV, D. **Service-oriented architecture for control of modular robots**. Crete, Greece, 2022. 304-309 p.

CHRONIS, C.; VARLAMIS, I. Fossbot: An open source and open design educational robot. **Electronics**, v. 11, n. 2606, 2022.

ÇOBAN, E. et al. Attitudes of it teacher candidates towards computer programming and their self-efficacy and opinions regarding to block-based programming. **Education and Information Technologies**, Springer, v. 25, n. 5, p. 4097–4114, 2020.

Coding Nemo. **Blockly vs Scratch: Which One is Right for You?** 2025. Accessed: 2025-02-02. Disponível em: <https://codingnemo.com/blockly-vs-scratch/>.

COŞKUNSERÇE, O. Comparing the use of block-based and robot programming in introductory programming education: Effects on perceptions of programming self-efficacy. **Computer Applications in Engineering Education**, v. 31, n. 5, p. 1234–1255, 2023. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cae.22637>.

DANIELA, L.; LYTRAS, M. D. Educational robotics for inclusive education. **Technology, Knowledge and Learning**, v. 24, n. 2, p. 219–225, jun. 2019. ISSN 2211-1670. Disponível em: <https://doi.org/10.1007/s10758-018-9397-5>.

DIWAN, T.; ANIRUDH, G.; TEMBHURNE, J. V. Object detection using YOLO: challenges, architectural successors, datasets and applications. **Multimedia Tools and Applications**, v. 82, n. 6, p. 9243–9275, mar. 2023. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-022-13644-y>.

DUDJAK, M.; MARTINOVIĆ, G. An api-first methodology for designing a microservice-based backend as a service platform. **Information Technology and Control**, v. 49, n. 2, p. 206–223, 2020.

DÚO-TERRÓN, P. Analysis of scratch software in scientific production for 20 years: Programming in education to develop computational thinking and steam disciplines. **Education Sciences**, v. 13, n. 4, 2023. ISSN 2227-7102. Disponível em: <https://www.mdpi.com/2227-7102/13/4/404>.

EKIN, C. C. et al. Artificial intelligence in education: A text mining-based review of the past 56 years. **Education and Information Technologies**, Springer, p. 1–43, 2025.

FAGUNDES-JUNIOR, L. A. et al. Communication delay in uav missions: A controller gain analysis to improve flight stability. **IEEE Latin America Transactions**, v. 21, n. 1, p. 7–15, 2023.

FERREIRA, F. P.; LIMA, D. A. The use of educational robotics in the development of students' computational thinking and cognitive skills: a systematic literature review. **RENOTE**, v. 21, n. 1, p. 363–372, jul. 2023. Disponível em: <https://seer.ufrgs.br/index.php/renote/article/view/134378>.

FUJIYOSHI, M. R. d. S. Inteligência artificial e suas implicações no contexto educacional. **Revista Ilustração**, v. 5, n. 2, p. 41–52, abr. 2024. Disponível em: <https://journal.editorailustracao.com.br/index.php/ilustracao/article/view/299>.

GONG, Y. et al. The architecture of micro-services and the separation of frond-end and back-end applied in a campus information system. In: IEEE. **2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA)**. Dalian, China, 2020. p. 321–324.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. The MIT Press, 2016. <http://www.deeplearningbook.org>.

Google Developers. **Blockly Documentation**. 2025. Accessed: 2025-02-02. Disponível em: <https://developers.google.com/blockly/guides/overview>.

GROVER, S. Robotics and engineering for middle and high school students to develop computational thinking. New Orleans, Louisiana, 2011.

GUO, S.; ZHENG, Y.; ZHAI, X. Artificial intelligence in education research during 2013–2023: A review based on bibliometric analysis. **Education and information technologies**, Springer, v. 29, n. 13, p. 16387–16409, 2024.

HAN, L. et al. A wad-yolov8-based method for classroom student behavior detection. **Scientific Reports**, Nature Publishing Group UK London, v. 15, n. 1, p. 9655, 2025.

ISMAIL, R.; ZAMAN, H. B.; MOHAMMAD, U. H. A visual-based project production package for design & technology subject, based on computational thinking skills across-stem. **JOIV: International Journal on Informatics Visualization**, v. 6, n. 2-2, p. 445–454, 2022.

JARA, C. A. et al. Hands-on experiences of undergraduate students in automatics and robotics using a virtual and remote laboratory. **Computers Education**, v. 57, n. 4, p. 2451–2461, 2011. ISSN 0360-1315. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0360131511001515>.

KAARLELA, T. et al. Common educational teleoperation platform for robotics utilizing digital twins. **Machines**, v. 10, n. 7, 2022. ISSN 2075-1702. Disponível em: <https://www.mdpi.com/2075-1702/10/7/577>.

KARVOUNIDIS, T.; LADIAS, A.; DOULIGERIS, C. Assessment of data types and of the ways they are used in scratch using the solo taxonomy. In: IEEE. **2023 IEEE Global Engineering Education Conference (EDUCON)**. Kuwait, Kuwait, 2023. p. 1–6.

KÖLLING, M.; BROWN, N. C. C.; ALTADMRI, A. Frame-based editing: Easing the transition from blocks to text-based programming. In: **Proceedings of the Workshop in Primary and Secondary Computing Education**. New York, NY, USA: Association for Computing Machinery, 2015. (WiPSCE '15), p. 29–38. ISBN 9781450337533. Disponível em: <https://doi.org/10.1145/2818314.2818331>.

KUHAIL, M. A. et al. Characterizing visual programming approaches for end-user developers: A systematic review. **IEEE Access**, v. 9, p. 14181–14202, 2021.

LEI, Z. et al. Unified and flexible online experimental framework for control engineering education. **IEEE Transactions on Industrial Electronics**, v. 69, n. 1, p. 835–844, 2022.

LIGEIRO, C.; JACINTO, H.; PIEDADE, J. Desenvolvimento do pensamento computacional e do raciocínio geométrico no 7.º ano: Resultados de uma experiência de ensino. **Quadrante**, v. 33, n. 2, p. 151–192, dez. 2024. Disponível em: <https://quadrante.apm.pt/article/view/37328>.

LIMA, J. M.; HUDSON, T. M.; BRANDãO, A. S. Game interface for inclusive teaching through educational robotics. Work submitted for publication. 2025, submitted for publication.

LIU, Y. et al. Effects of robotics education on young children's cognitive development: a pilot study with eye-tracking. **Journal of Science Education and Technology**, v. 32, n. 3, p. 295–308, jun. 2023. ISSN 1573-1839. Disponível em: <https://doi.org/10.1007/s10956-023-10028-1>.

MARINHO, F. G. T. Uprobotics: Robótica educacional utilizando linguagem visual baseada em blocos. Universidade Federal do Amazonas, 2022.

MARTÍN, F. et al. Client-server approach for managing visual attention, integrated in a cognitive architecture for a social robot. **Frontiers in Neurorobotics**, Volume 15 - 2021, 2021. ISSN 1662-5218. Disponível em: <https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2021.630386>.

MORAITI, I.; FOTOGLOU, A.; DRIGAS, A. Coding with block programming languages in educational robotics and mobiles, improve problem solving, creativity amp; critical thinking skills. **International Journal of Interactive Mobile Technologies (iJIM)**, v. 16, n. 20, p. pp. 59–78, Oct. 2022. Disponível em: <https://online-journals.org/index.php/i-jim/article/view/34247>.

MOREIRA, T. A. **Ferramentas para ensino do Pensamento Computacional no Brasil: Uma Revisão Sistemática**. 2025. Accessed: 2025-02-02. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/40919/3/FerramentasParaEnsino.pdf>.

NASCIMENTO, L. M. d. et al. Sbotics-gamified framework for educational robotics. **Journal of intelligent & robotic systems**, Springer, v. 102, n. 1, p. 17, 2021.

NG, O.-L.; GONZÁLEZ-CALERO, J. A.; JACINTO, H. Avanços da investigação e da prática na integração do pensamento computacional na educação matemática. **Quadrante**, v. 33, n. 2, p. 1–10, dez. 2024. Disponível em: <https://quadrante.apm.pt/article/view/39744>.

NGUYEN, T.-H.; NGO, B.-V.; NGUYEN, T.-N. Vision-based hand gesture recognition using a yolov8n model for the navigation of a smart wheelchair. **Electronics**, v. 14, n. 4, 2025. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/14/4/734>.

NUNES, T.; VIANA, C.; VIANA, L. Perspectives of robotics as a pedagogical resource applied to education 4.0: A bibliometric analysis on educational robotics. **Research, Society and Development**, v. 10, n. 4, p. 1–14, 2021.

PAPERT, S. **A máquina das crianças: repensando a escola na era da informática**. Artmed, 2008. Disponível em: <https://www.yumpu.com/pt/document/read/67463171/a-maquina-das-criancas>.

PASSAULT, G. et al. Metabot: a low-cost legged robotics platform for education. In: IEEE. **2016 International Conference on Autonomous Robot Systems and Competitions**. Bragança, Portugal, 2016. p. 1–5.

PASTERNAK, E.; FENICHEL, R.; MARSHALL, A. N. Tips for creating a block language with blockly. In: IEEE. **2017 IEEE Blocks and Beyond Workshop (BB)**. Raleigh, NC, USA, 2017. p. 21–24.

PATAKY, M.; FEELPAK, P. Remote arduino programming with blockly web interface. In: IEEE. **2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)**. London, UK, 2019. p. 23–26.

PEREIRA, J. M. et al. Robotics as a possibility for exploring laterality in early childhood education. In: IEEE. **2023 Latin American Robotics Symposium (LARS), 2023 Brazilian Symposium on Robotics (SBR), and 2023 Workshop on Robotics in Education (WRE)**. Salvador, Brazil, 2023. p. 678–682.

PINEL, G. S. F.; ARAÚJO, E. N. D.; BRANDÃO, A. S. Explorando a robótica educacional como estratégia para o desenvolvimento das habilidades socioemocionais e cognitivas. In: SOCIEDADE BRASILEIRA DE AUTOMAçãO. **Anais da Conferência Brasileira de Automação**. Rio de Janeiro, Brazil, 2025.

PINEL, G. S. F.; BRANDÃO, A. S.; FARIA, R. W. S. C. Blockly and ros: A visual programming interface for robot control. In: IEEE. **2025 Brazilian Conference on Robotics (CROS)**. Belo Horizonte, Brazil, 2025. v. 1, p. 1–5.

PIZETTA, I. H. B.; BRANDÃO, A. S.; SARCINELLI-FILHO, M. A hardware-in-loop platform for rotary-wing unmanned aerial vehicles. In: IEEE. **2014 International Conference on Unmanned Aircraft Systems (ICUAS)**. Orlando, FL, USA, 2014. p. 1146–1157.

PLAZA, P. et al. Traffic lights through multiple robotic educational tools. In: IEEE. **2018 IEEE Global Engineering Education Conference (EDUCON)**. Santa Cruz de Tenerife, Spain, 2018. p. 2015–2020.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. Los Alamitos, CA, USA: IEEE Computer Society, 2016. p. 779–788.

RESNICK, M. et al. Scratch: programming for all. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 52, n. 11, p. 60–67, nov. 2009. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1592761.1592779>.

RIBEIRO, R.; OLIVEIRA, G.; DUSI, C. Ensino de programação na educação básica: Uma abordagem progressiva de scratch, robótica educacional e python. In: **Anais do II Simpósio Brasileiro de Computação na Educação Básica**. Porto Alegre, RS, Brasil: SBC, 2025. p. 57–62. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/sbceb/article/view/34485>.

RICHARD, R. P. et al. A client-server based educational chatbot for academic institutions. In: IEEE. **2024 4th International Conference on Intelligent Technologies (CONIT)**. Bangalore, India, 2024. p. 1–5.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence, Global Edition A Modern Approach**. Pearson Deutschland, 2021. 1168 p. ISBN 9781292401133. Disponível em: <https://elibrary.pearson.de/book/99.150005/9781292401171>.

SAHAR, R. et al. Artificial intelligence in sustainable education: A bibliometric analysis and future research directions. **Education Science and Management**, 2025.

SALMA, Z. et al. Effectiveness of robot-mediated learning in fostering children's social and cognitive development. **Applied Sciences**, v. 15, n. 7, 2025. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/15/7/3567>.

SANGALI, R.; CATABRIGA, L.; BOERES, M. C. Robótica educativa para desenvolvimento de habilidades do pensamento computacional por meio de eletiva complementar. In: **Anais do XXX Workshop de Informática na Escola**. Porto Alegre, RS, Brasil: SBC, 2024. p. 437–448. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/wie/article/view/31081>.

SANTOS, D.; OLIVEIRA, M. Robótica e prototipagem para professores: uma proposta com ensino por investigação e resolução de problemas. In: **Anais Estendidos do V Simpósio Brasileiro de Educação em Computação**. Porto Alegre, RS, Brasil: SBC, 2025. p. 59–66. ISSN 3086-0741. Disponível em: <https://sol.sbc.org.br/index.php/educomp_estendido/article/view/34919>.

SANTOS, D. T.; SANTOS, S. C. d.; JAVARONI, S. L. A programação computacional e a educação matemática: aspectos da amanualidade na reorganização do pensamento. **Boletim GEPEM**, n. 79, p. 114–126, maio 2021. Disponível em: <https://periodicos.ufrrj.br/index.php/gepem/article/view/596>.

SANTOS, L. C. B. et al. Lógica de programação através da robótica: Uso do scratch e arduino para criação de robôs e projetos interativos. **LUMEN ET VIRTUS**, v. 15, n. 39, p. 2408–2421, ago. 2024. Disponível em: <https://periodicos.newsciencepubl.com/LEV/article/view/207>.

SANTOS, R. d. A.; SILVA, M. D. F. d.; ANDRADE, A. O. Robótica educacional: Uma jornada histórica. **Seminário Temático Internacional**, v. 1, n. 1, p. 1–8, abr. 2024. Disponível em: <https://anais.ghemat-brasil.com.br/index.php/STI/article/view/309>.

SCHINA, D.; ESTEVE-GONZÁLEZ, V.; USART, M. An overview of teacher training programs in educational robotics: Characteristics, best practices and recommendations. **Education and Information Technologies**, Springer, v. 26, n. 3, p. 2831–2852, 2021.

Scratch Wiki Community. **Scratch Wiki - The Free Scratch Resource**. 2025. Accessed: 2025-02-02. Disponível em: <https://en.scratch-wiki.info/>.

SOCRATOUS, C.; IOANNOU, A. Structured or unstructured educational robotics curriculum? A study of debugging in block-based programming. **Educational Technology Research and Development**, Springer, v. 69, n. 6, p. 3081–3100, 2021.

SUTHERLAND, C. J. Blockly in a box: How children explore block-based robot programming. In: IEEE. **2022 19th International Conference on Ubiquitous Robots (UR)**. Jeju, Korea, Republic of, 2022. p. 263–267.

TANG, H.; QIAN, Y.; PORTER-VOSS, S. Enhancing rural students' computer science self-efficacy in a robotics-based language arts course. **Education and Information Technologies**, Springer, v. 29, n. 18, p. 25533–25550, 2024.

TEIXEIRA, J. S. S. et al. Astrobot: A robotic platform to enhance the teaching-learning process in basic education. In: IEEE. **2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE)**. São Bernardo do Campo, Brazil, 2022. p. 1–5.

TZAFESTAS, C.; PALAIOLOGOU, N.; ALIFRAGIS, M. Virtual and remote robotic laboratory: comparative experimental evaluation. **IEEE Transactions on Education**, v. 49, n. 3, p. 360–369, 2006.

WANG, D. et al. A novel application of educational management information system based on micro frontends. **Procedia Computer Science**, v. 176, p. 1567–1576, 2020. ISSN 1877-0509. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 24th International Conference KES2020. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050920320688>.

WANG, S. et al. Artificial intelligence in education: A systematic literature review. **Expert Systems with Applications**, v. 252, p. 124167, 2024. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417424010339>.

WARDRIP, P. S.; BRAHMS, L. Taking making to school: A model for integrating making into classrooms. In: PEPPLER, K.; HALVERSON, E. R.; KAFAI, Y. B. (Ed.). **Makeology: Makerspaces as Learning Environments**. 1st. ed. New York: Routledge, 2016. cap. 7, p. 10. ISBN 9781315726519.

WING, J. M. Computational thinking. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 49, n. 3, p. 33–35, mar. 2006. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/1118178.1118215>.

XU, X. et al. Robotic kinematics teaching system with virtual reality, remote control and an on–site laboratory. **International Journal of Mechanical Engineering Education**, v. 48, n. 3, p. 197–220, 2020. Disponível em: <https://doi.org/10.1177/0306419018807376>.

YANG, W.; NG, D. T. K.; GAO, H. Robot programming versus block play in early childhood education: Effects on computational thinking, sequencing ability, and self-regulation. **British Journal of Educational Technology**, v. 53, n. 6, p. 1817–1841, 2022. Disponível em: <https://bera-journals.onlinelibrary.wiley.com/doi/abs/10.1111/bjet.13215>.

ÇETIN, M.; DEMIRCAN, H. Özlen. Empowering technology and engineering for STEM education through programming robots: a systematic literature review. **Early Child Development and Care**, Routledge, v. 190, n. 9, p. 1323–1335, 2020. Disponível em: <https://doi.org/10.1080/03004430.2018.1534844>.