# ROSBLOCKS: A Visual Programming Interface for ROS2 Robots

Fernando Costa Nogueira[1][a], Dieisson Martinelli[1,2][b], Lucas Alexandre Zick[1,2][c], André Schneider de Oliveira[2][d] and Vivian Cremer Kalempa[1][e]

[1]*Department of Information Systems, Universidade do Estado de Santa Catarina (UDESC), São Bento do Sul, Brazil*

[2]*Graduate Program in Electrical and Computer Engineering, Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba, Brazil*

Keywords: ROS2, Blockly, Visual Programming, Educational Robotics.

Abstract: This work presents the development of a visual programming interface for robots compatible with ROS2, called ROSBLOCKS, using a modern architecture based on React, Blockly, Node.js, and Electron. The proposal aims to make robot programming more accessible, especially in educational contexts, by allowing users to create complex robotic behaviors through visual blocks, without the need for prior knowledge in programming languages such as Python or C++. The system is cross-platform and flexible, working with both simulated and physical robots that use ROS2, and allows for automatic code generation and execution from the visual assembly. Additionally, the system was designed to facilitate integration with different types of ROS2 topics, services, and actions. The system was tested in the classroom with undergraduate students who already have practical experience with ROS, enabling an assessment of its applicability in real teaching scenarios and allowing the observation of gains in productivity, engagement, and clarity in the construction of robotic behaviors. The complete source code and all validation materials from this study are openly available on GitHub at https://github.com/ferssor/rosblocks.



## 1 INTRODUCTION

The use of robots has significantly expanded across various domains, such as education (Favaretto et al., 2024), industry (Lettera et al., 2025), agriculture (Maldonado-Mendez et al., 2025), and scientific research (Martinelli et al., 2020; Zick et al., 2024; Kalempa et al., 2023). In this context, the Robot Operating System (ROS) has established itself as one of the main platforms for the development of robotic applications, offering a modular, scalable infrastructure that is widely adopted by both the academic and industrial communities (Patkar et al., 2023). With the emergence of ROS2, substantial improvements have been incorporated, such as support for distributed systems, security, real-time capabilities, and a standard middleware, making it even more attractive for robust and collaborative projects.

Despite these advantages, the learning curve of ROS2 remains a challenge, especially for beginners or undergraduate students in the early stages of their academic journey. The need to understand concepts such as nodes, topics, services, and actions, combined with the use of programming languages like Python or C++, can represent a significant entry barrier. Tools that simplify this complexity and make robot programming more accessible are, therefore, of great importance.

Some solutions have sought to address this need through visual programming interfaces. TurtleBot3 Blockly (Industries, 2017) allows programming the TurtleBot3 using visual blocks integrated with ROS, but it is strongly coupled to that specific robot and version of ROS. The Robot Ignite Academy (The Construct, 2025), developed by The Construct, offers an online environment with simulations and block support, but requires a cloud connection and is generally geared toward controlled exercises. Another approach is the use of tools like Niryo Studio (Niryo, 2025), designed for Niryo's robotic arms, which also uses Blockly blocks but is specific to the brand's own

[a] https://orcid.org/0009-0009-4751-0546
[b] https://orcid.org/0000-0001-7589-1942
[c] https://orcid.org/0009-0001-8645-9781
[d] https://orcid.org/0000-0002-8295-366X
[e] https://orcid.org/0000-0001-9733-7352



Nogueira, F. C., Martinelli, D., Zick, L. A., Schneider de Oliveira, A. and Kalempa, V. C.
ROSBLOCKS: A Visual Programming Interface for ROS2 Robots.
DOI: 10.5220/0013714000003982

In *Proceedings of the 22nd International Conference on Informatics in Control, Automation and Robotics (ICINCO 2025) - Volume 2*, pages 267-274
ISBN: 978-989-758-770-2; ISSN: 2184-2809

robots.

The solution presented in this work stands out for its generality and portability: it is a cross-platform system that allows the creation of visual programs for any robot using ROS2, without depending on specific robots or external infrastructures. The system can be run locally on conventional computers and offers a direct bridge to ROS2 topics, services, and actions, enabling full control of real or simulated robots through visual blocks.

This work presents the development of a block-based visual programming interface designed to facilitate the use of ROS2 through an intuitive graphical experience. The system was built using modern web and desktop development technologies React(React, 2025), Blockly(Google, 2025), Node.js(Node.js, 2025), and Electron(OpenJS Foundation, 2025) and enables the creation of robot programs through a drag-and-drop block logic, abstracting implementation details and syntax.

The proposed solution is cross-platform, extensible, and compatible with any robot using ROS2, whether in a simulated or real environment. The system was validated in the classroom with undergraduate students who already had prior experience with ROS, allowing the observation of gains in clarity, productivity, and engagement in building robotic behaviors.

This paper is organized as follows: Section 2 presents related work and existing visual programming solutions in robotics; Section 3 describes the system architecture; Section 4 presents use cases and tests in an educational environment; Section 5 discusses the results and limitations; and Section 6 provides the conclusions and future directions for the project.

## 2 RELATED WORKS

Several initiatives aim to simplify the development of robotic applications through visual programming interfaces, especially in educational contexts. These approaches are inspired by the success of tools such as Scratch (Resnick et al., 2009), which popularized the use of visual blocks to introduce programming logic in an accessible way. In robotics, this paradigm has been applied at different levels of complexity and integration with middlewares such as ROS.

One of the most well-known solutions is TurtleBot3 Blockly, developed by ROBOTIS, which allows controlling the TurtleBot3 robot through a Blockly interface integrated with ROS. While effective for introducing robotics, this tool is strongly coupled to the TurtleBot3 robot and to the ROS version, limiting its applicability in more general contexts or with different robot platforms.

Its main merit lies in its educational approach, which makes the learning process more accessible, especially for high school students or early undergraduates. Through pre-configured blocks, users can perform tasks such as moving the robot, reading sensors, and executing conditional behaviors without writing code in Python or C++.

In contrast, the system proposed in this work was designed to operate with any robot that uses ROS2, regardless of model or manufacturer, and allows modular creation of new Blockly blocks directly connected to the specific resources of each robot. This provides greater flexibility, reusability, and applicability in both educational and industrial contexts.

The Robot Ignite Academy, developed by the company The Construct, offers an online environment for learning ROS with real-time simulations. Although the platform provides visual blocks in some introductory courses, their use is restricted to the cloud environment, requiring a constant connection and a subscription structure. Furthermore, the code generation is more focused on demonstration than on direct applications with local physical robots.

The system offers a complete learning experience, allowing users to switch between code, simulation, and conceptual explanations within the same web environment. In addition, it supports both ROS and ROS2 in several courses, making it an excellent teaching tool for beginners and intermediate users.

The system proposed in this paper stands out for its local execution and independence from the cloud, offering a Blockly interface that can be used directly with real robots controlled via ROS2. Furthermore, the tool is designed to be freely extensible, allowing the creation of custom blocks that connect to any ROS2 node, which expands its applicability for both teaching and the development of real-world applications.

These characteristics make the tool especially attractive for use in technical and university level educational environments, promoting autonomy in the construction of robotic behaviors and incremental learning of ROS2.

## 3 SYSTEM ARCHITECTURE

The system developed in this work was designed with the goal of providing an accessible, modular visual programming interface compatible with any robot using ROS2. To achieve this, an architecture based on modern frontend and backend technologies was

adopted, with a strong emphasis on extensibility and portability.

The system architecture is composed of three main layers, each responsible for a specific role in the application's operational flow: (i) the visual interface layer (frontend), which provides the user with an intuitive block-based programming experience; (ii) the middleware layer (backend), which interprets the blocks and manages communication between the interface and the robots; and (iii) the robotic execution layer, composed of the ROS2 environment, which executes commands and controls the robots, whether simulated or real.

The flowchart presented in Figure 1 illustrates the communication flow between the editor and the ROS2 API (Application Programming Interface). When the user interacts with the graphical interface, a request is sent to the middleware, which in turn translates this request into a command compatible with the ROS2 command-line interface (CLI). The execution occurs in the backend, and the response, containing the requested data or the result of the operation, is then returned to the user.

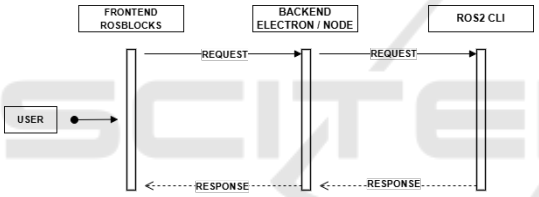


Figure 1: Request and response flow.

One of the main differentiators of the system is the functionality that allows creating and managing ROS workspaces and packages directly through the visual interface. Through this integration, the user can generate new ROS2 packages with customized dependencies, create internal nodes, and encapsulate the code generated by the blocks into scripts compatible with the ROS2 ecosystem.

This integration strengthens the bridge between visual programming and traditional textual development, offering a continuous and coherent experience. Additionally, it promotes good project organization practices by introducing from the outset the modular structure characteristic of ROS, with its concepts of packages and nodes.

## 3.1 Frontend

The visual interface layer is responsible for providing the user with an accessible, interactive, and intuitive experience for building robotic programs. This interface was developed using React(React, 2025), a widely adopted JavaScript library for modern interface development, in conjunction with Blockly(Google, 2025), a Google library for block-based visual programming.

The frontend layer was implemented with a focus on modularity, responsiveness, and seamless integration with the backend and the ROS2 system. It uses React(React, 2025) to build the user interface, leveraging the paradigm of reusable components and efficient Document Object Model (DOM)(Mozilla Developer Network, 2025) manipulation through the virtual DOM. The styling and visual organization follow modern responsive design standards, with automatic adaptation to different screen resolutions.

Visual programming is made possible through the Blockly(Google, 2025) library, embedded as a custom React component. Each Blockly block is defined as an Extensible Markup Language (XML) template extended with dynamic toolboxes, where the blocks are directly mapped to Python(Python Software Foundation, 2025) source code templates. This code generation is handled by a code generator adapted from Blockly.Python, extended to support ROS2-specific semantic structures such as rclpy, node, topic publishing and subscribing, and timer usage.

The system allows the code generated from block manipulation to be displayed in real time in an embedded text editor, enabling switching between visual and textual programming. Additionally, the interface allows the user to create, open, and manage ROS2 workspaces, with support for multiple packages, files, and nodes, ensuring that the generated structure complies with the official ROS2 directory convention.

Figure 2 presents an example of creating a functional publisher node using the ROSBLOCKS editor.

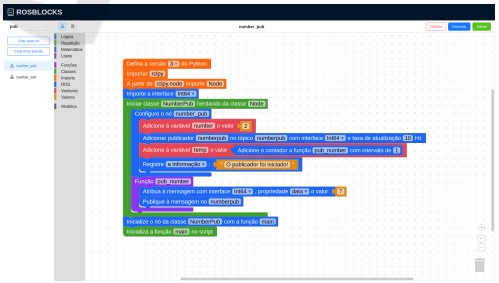


Figure 2: ROSBLOCKS editor with a functional publisher node.

The platform provides visual blocks that enable the construction of specific robotic behaviors in an intuitive way. After building the node, its execution can be initiated directly from the editor, facilitating quick validation of its functionality. The entire process of script generation and inclusion of the required packages is performed automatically, reducing complexity

for the user and accelerating the development cycle.

However, the entire orchestration between the interface, data persistence, flow control, and ROS2 command execution depends on a robust intermediate layer. This responsibility lies with the backend, whose role is to mediate interactions between the user, the local file system, and the ROS2 execution layer.

### 3.2 Backend

The application's backend functions as a middleware between the visual interface and the ROS2 environment, being responsible for interpreting commands from the frontend, performing file system operations, and forwarding instructions to the ROS execution layer. This modular architecture ensures decoupling between interface and execution, promoting system portability and scalability.

The application was developed using Node.js(Node.js, 2025), which provides an asynchronous, event-driven runtime environment ideal for file manipulation, process execution, and real-time communication with the frontend via IPC (interprocess communication). The use of the Electron(OpenJS Foundation, 2025) framework allows the system to be packaged as a cross-platform desktop application, combining web technologies (JavaScript, HTML, CSS) with native operating system access, including reading and writing files, creating directories, and executing shell commands.

The backend handles requests such as: workspace creation, ROS2 package generation, node inclusion, automatic saving of executable Python(Python Software Foundation, 2025) files, and script execution in integrated terminals. Each frontend request is mapped to a backend routine, which performs validations and then executes the corresponding ROS2 CLI commands (ros2 pkg create, ros2 run, etc.). Communication between layers is carried out using messages structured in JavaScript Object Notation (JSON), ensuring interoperability and ease of debugging.

Additionally, the backend maintains an abstraction of the ROS2 directory structure within the application, dynamically mirroring the actual state of the workspace in the editor. This ensures that any operation performed in the interface accurately reflects the state of the ROS development environment.

With this structure, the backend acts as the link between the system's visual logic and the actual ROS2 processes, but it is not directly responsible for robotic execution, a role played by the ROS2 layer.

### 3.3 ROS2 Layer

The ROS2 layer is responsible for executing the scripts generated in the editor, as well as handling middleware requests. Its main role is to serve as a bridge between the visual elements and the actual ROS2 ecosystem, converting user actions into nodes, topics, and other entities typical of the ROS architecture.

This layer interacts directly with the ROS2 command-line interface (CLI), using commands such as ros2 run, ros2 topic, ros2 service, among others, to launch nodes, monitor topics, publish messages, or retrieve system information. Script execution is handled via Node.js(Node.js, 2025), which triggers subprocesses in the locally installed ROS2 environment. As a result, the system ensures full compatibility with native ROS2 tools and allows the visual code generated to be tested, executed, and debugged within the same environment used by professionals in the field.

This approach is especially advantageous for beginners, as it abstracts the complexity associated with directly handling the command line and the ROS2 file structure. At the same time, it maintains fidelity to the original ROS architecture, allowing visually developed projects to be exported, manually edited, and used in professional environments.

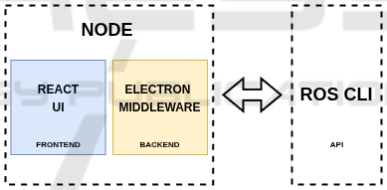


Figure 3: Editor architecture diagram.

Figure 3 illustrates the architecture of the editor, where the application functions as a repository that houses the user interface and middleware layers. The Node.js(Node.js, 2025) is responsible for executing these scripts, and the request is made by executing a command in the ROS command-line interface, which runs and provides important information about the ROS node.

## 4 VALIDATION

In order to validate the applicability and pedagogical effectiveness of the proposed system, controlled tests were conducted in an educational environment with undergraduate students from the Information Systems program at the State University of Santa Catarina – CEPLAN. The participants had prior exposure to

the ROS ecosystem, having completed practical activities with both simulated and real robots using traditional programming in Python(Python Software Foundation, 2025).

The integration of the system into the robotics course aimed to investigate its ability to:

- Reduce the complexity of programming robotic behaviors;
- Accelerate the development of tasks with ROS2 robots;
- Promote greater clarity in the logical structure of programs;
- Stimulate students' creativity and engagement through the visual interface.

The tests were conducted in a laboratory environment with workstations running the Ubuntu 24.04(Canonical, 2025) Linux operating system and the ROS2 Jazzy Jalisco distribution. Each station was equipped with the developed interface, allowing the creation, editing, and execution of nodes directly from the provided visual blocks.

For educational and experimental control purposes, the Turtlesim simulator (ROS Documentation, 2025) was used due to its simplicity, lightweight nature, and compatibility with fundamental ROS2 concepts. This choice enabled a focus on programming logic while abstracting hardware complexities.

During the activities, students were challenged to complete a set of tasks with increasing complexity, including basic movement control with linear and angular velocity adjustments, reading and processing data from simulated sensors with reactive responses to environmental stimuli, and finally, the implementation of a simple proportional-integral-derivative (PID) logic for trajectory control.

All tasks were developed exclusively using the blocks provided in the interface, without the need to manually write code. The experience allowed for the verification of the tool's expressiveness and its ability to represent complex control and flow structures in an accessible way.

The system evaluation was conducted through a survey with structured Likert scale questions (Likert, 1932), complemented by direct observations during the practical activities. The results were analyzed in terms of perceived usability, clarity in program structure, and impact on student engagement.

The survey was completed by a total of 10 students immediately after the conclusion of the practical activities. The Likert scale ranged from 1 (strongly disagree) to 5 (strongly agree). Responses were statistically analyzed using measures of central tendency (mean and standard deviation), in addition to qualitative analysis of open-ended responses. Among the evaluated aspects, highlights include: ease of use of the interface, understanding of node and topic concepts, perception of productivity gains, and overall satisfaction with the experience.

# 5 RESULTS

The system evaluation was conducted with 10 undergraduate students from the Information Systems program at the State University of Santa Catarina (UDESC/CEPLAN), all enrolled in a practical robotics course that uses ROS2 as its technological foundation. The participants had previous exposure to ROS through activities involving Python(Python Software Foundation, 2025) programming and the control of both simulated and real robots.

The evaluation instrument was a structured questionnaire composed of 23 items, of which 16 were formulated using a 5-point Likert scale (1 = strongly disagree, 5 = strongly agree), and 7 were open-ended and demographic questions aimed at gathering qualitative feedback and characterizing the participants' profiles.

Of the 10 students, 100% had previously used ROS, with 80% reporting an intermediate level of familiarity and 20% a basic level. Furthermore, 60% of the students had used some form of visual programming tool before, such as Scratch(MIT Media Lab, 2025) or Blockly-based interfaces.

The responses to the Likert scale questions were grouped into four main dimensions: usability, logical clarity, understanding of ROS2 concepts, and engagement with the tool. Table 1 summarizes the means and standard deviations for each dimension.

Table 1: Means and standard deviations of the evaluated dimensions.

| Dimension | Mean | Standard Deviation |
|---|---|---|
| Usability | 4,40 | 0,81 |
| Clarity and Logical Structure | 4,65 | 0,66 |
| Understanding of ROS2 concepts | 4,70 | 0,67 |
| Engagement and motivation | 4,76 | 0,52 |

It was observed that all dimensions obtained average scores above 4, highlighting a broadly positive perception among students regarding the block-based programming tool for ROS2.

The Engagement and Motivation dimension stood out with the highest average (4.76) and the lowest standard deviation (0.52), indicating not only a high level of student involvement but also low variability in responses, suggesting consensus on the motivating nature of the tool.

The Understanding of ROS2 Concepts dimension

also showed a high average (4.70), demonstrating that the visual interface significantly contributed to the technical learning of the ROS architecture, particularly in understanding nodes, topics, and messages.

The Logical Clarity and Structure (4.65) and Usability (4.40) dimensions also received positive evaluations. However, usability showed the highest standard deviation (0.81), indicating greater variation in students' experiences with the interface—possibly related to factors such as prior familiarity with visual programming environments, first contact with the tool, or with the logic of block construction.

These quantitative results reinforce the effectiveness of the tool both as an educational resource and as a means of engagement, making it particularly relevant for introducing ROS2 in an accessible and motivating way to undergraduate students.

The open-ended responses were analyzed using thematic content analysis, revealing four main categories: ease of use, improved conceptual understanding, creativity, and suggestions for expansion, as shown in Table 2.

Table 2: Thematic Categories Extracted from Open-Ended Responses.

| Category | Example Responses |
|---|---|
| Ease of Use | "I liked how intuitive the tool is, and how easily you can follow a logical flow." |
| Conceptual Clarity | "I was able to understand more easily how nodes, publishers, and subscribers work." |
| Creativity/Exploration | "I felt more motivated to learn ROS2." |
| Improvement Suggestions | "Avoid manually repeating names in chained blocks by allowing reuse through dropdown or autocomplete." |

The most frequent mentions were concentrated in the categories Ease of Use and Conceptual Clarity, reflecting the students perception that the tool contributed to making the development process more accessible and intuitive, even for those with limited prior familiarity with ROS2 or programming in general.

In the Ease of Use category, comments highlighted the intuitive interface and the logical linearity in constructing programs through blocks, which aligns with the high satisfaction levels observed in the quantitative usability questions. Meanwhile, the Conceptual Clarity category underscores the tool's potential to make fundamental ROS2 components, such as nodes, topics, and messages—more understandable, fostering active and practical learning of these concepts.

Additionally, the Creativity/Exploration category shows that students not only understood the concepts but also felt encouraged to experiment with new ideas, indicating an environment conducive to engagement and autonomy in the learning process.

Finally, the Improvement Suggestions category gathers relevant observations about limitations perceived by users, such as the need for features that enhance the reuse of fields between blocks. These contributions are valuable for improving the tool, showing that students acted not only as users but also as critical collaborators in the pedagogical development process.

This qualitative analysis complements the quantitative data, offering a deeper insight into how the tool was perceived in terms of usability, learning, and motivation.

The collected data demonstrate that the tool achieved the proposed educational objectives: it facilitated the programming of ROS2 robots, increased the structural clarity of programs, and promoted greater student engagement. Moreover, the fact that students themselves suggested future expansions of the tool indicates a high level of ownership and interest in the technology.

# 6 CONCLUSION

This work presented the development and validation of a visual programming system for robots that use the ROS2 middleware, aiming to facilitate the learning and development of robotic behaviors in educational environments. The proposed tool combines a graphical interface built with React(React, 2025) and Blockly(Google, 2025) with a backend in Node.js(Node.js, 2025) and Electron(OpenJS Foundation, 2025), enabling the creation, editing, and execution of ROS2 nodes directly from visual blocks.

Unlike other existing solutions, which are often restricted to specific robots or limited to cloud-based environments, the system described here stands out for its flexibility, portability, and compatibility with any ROS2-based robot. Additionally, it allows the creation and organization of ROS packages directly through the interface, seamlessly integrating the visual paradigm with the modular structure native to ROS.

The validation conducted with undergraduate students demonstrated promising results: the tool was positively evaluated in terms of usability, clarity in program structuring, and positive impact on student engagement. Participants reported that the visual interface facilitated the understanding of fundamental ROS2 concepts and encouraged experimentation and creativity in the development of robotic behaviors.

Despite the positive results, some limitations were observed. The tool still relies on pre-configured blocks and lacks support for advanced features such as complex actions, integration with Artificial Intelligence (AI) or computer vision pipelines, and visual debugging mechanisms. Furthermore, the validation sample was relatively homogeneous, composed of students with prior experience in ROS, which may limit the generalizability of the results.

As a continuation perspective, the expansion of the block library is proposed to include specific sensors and actuators, integration with more advanced simulators such as Gazebo, and the application of the tool at other educational levels, such as technical education and teacher training. The addition of collaborative features and visual debugging resources also emerges as a promising path to make the system even more complete.

In summary, the developed tool represents a significant advancement toward making ROS2 more accessible and educational by facilitating the teaching and development of robotic applications through a visual, modular, and extensible approach.

## ACKNOWLEDGEMENTS

The project is supported by the National Council for Scientific and Technological Development (CNPq) under grant number 407984/2022-4; the Fund for Scientific and Technological Development (FNDCT); the Ministry of Science, Technology and Innovations (MCTI) of Brazil; Brazilian Federal Agency for Support and Evaluation of Graduate Education (CAPES); the Araucaria Foundation; the General Superintendence of Science, Technology and Higher Education (SETI); and NAPI Robotics.

## REFERENCES

Canonical (2025). Ubuntu: Open source operating system for enterprise and developers. Avaliable: https://ubuntu.com/. Accessed: 2025-07-30.

Favaretto, D., de Assis, V., Martinelli, D., Schneider De Oliveira, A., and Kalempa, V. (2024). Low-cost robot construction focused on educational environments. *Proceedings of the 21st International Conference on Informatics in Control, Automation and Robotics*, pages 66–72.

Google (2025). Blockly: A visual programming editor. Avaliable: https://developers.google.com/blockly?hl=en. Accessed: 2025-03-07.

Industries, D. (2017). Turtlebot3 blockly documentation. Avaliable: https://turtlebot-3-blockly-wiki.readthedocs.io/. Accessed: 2025-06-04.

Kalempa, V. C., Piardi, L., Limeira, M., and de Oliveira, A. S. (2023). Multi-robot task scheduling for consensus-based fault-resilient intelligent behavior in smart factories. *Machines*, 11(4).

Lettera, G., Costa, D., and Callegari, M. (2025). A hybrid architecture for safe human–robot industrial tasks. *Applied Sciences*, 15(3):1158.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55.

Maldonado-Mendez, C., Ruiz-Paz, S. F., Machorro-Cano, I., Marin-Hernandez, A., and Hernandez-Mendez, S. (2025). The optimization of pid controller and color filter parameters with a genetic algorithm for pineapple tracking using an ros2 and microros-based robotic head. *Computation*, 13(3):69.

Martinelli, D., Cerbaro, J., Fabro, J. A., de Oliveira, A. S., and Teixeira, M. A. S. (2020). Human-robot interface for remote control via iot communication using deep learning techniques for motion recognition. In *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*, pages 1–6. IEEE.

MIT Media Lab (2025). Scratch: Imagine, program, share. Avaliable: https://scratch.mit.edu/. Accessed: 2025-04-04.

Mozilla Developer Network (2025). Introduction to the dom. Avaliable: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction/. Accessed: 2025-07-30.

Niryo (2025). Niryostudio: Application for robotics programming and simulation. Avaliable: https://niryo.com/niryostudio/. Accessed: 2025-06-04.

Node.js (2025). Node.js: Javascript runtime environment. Avaliable: https://nodejs.org. Accessed: 2025-03-07.

OpenJS Foundation (2025). Electron: Build cross-platform desktop apps with javascript, html, and css. Avaliable: https://www.electronjs.org. Accessed: 2025-03-07.

Patkar, U. C., Mandhalkar, V., Chavan, A., Songire, S., and Kothawade, H. (2023). Robot operating system: A comprehensive analysis and evaluation. *International Journal of Intelligent Systems and Applications in Engineering*, 12(7s):516–520.

Python Software Foundation (2025). Python: Programming language and community resources. Avaliable: https://www.python.org/. Accessed: 2025-03-07.

React (2025). React: The library for web and native user interfaces. Avaliable: https://react.dev/. Accessed: 2025-03-07.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67.

ROS Documentation (2025). Using turtlesim, ros2, and rqt: Beginner tutorial. Avaliable: https://docs.ros.org/en/jazzy/Tutorials/Beginner-CLI-Tools/

Introducing-Turtlesim/Introducing-Turtlesim.html. Accessed: 2025-03-19.

The Construct (2025). Robot ignite academy: Online ros courses. Avaliable: https://www.robotigniteacademy.com/en/. Accessed: 2025-06-04.

Zick, L. A., Martinelli, D., Schneider de Oliveira, A., and Cremer Kalempa, V. (2024). Teleoperation system for multiple robots with intuitive hand recognition interface. *Scientific Reports*, 14(1):1–11.