Jiang, B. et al. (Eds.) (2025). Proceedings of the 33rd International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Educational Robotics through Web Applications: From Visual Programming to Simulation-Driven Learning

Ivan TERZIC[a*], Ivica BOTICKI[a], Matija LOVREKOVIC[a] & Lara TOPALOVIC[a]
[a]*Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia*
*ivan.terzic@fer.hr

Abstract: This paper explores the role of educational web applications in supporting robotics-based learning, with a primary focus on how such tools can enhance early STEM education. Based on two implementations developed within the MetaRoboLearn project, the paper compares and analyzes two different approaches: one utilizing block-based programming through the Blockly library, and the other using Python-based code editing with an integrated 3D simulator. While both applications control physical and simulated robots, their target audiences and pedagogical strategies differ significantly. The block-based system is tailored for primary school students and emphasizes simplicity, modularity, and abstraction of syntax. The Python-based system, on the other hand, introduces students to real code structures and debugging practices in a controlled, interactive environment. Both systems include simulation environments that allow for experimentation without direct access to hardware, increasing accessibility and safety. The paper argues that such hybrid learning environments – combining visual programming, real-time simulation, and physical interaction – provide a scalable and engaging framework for developing algorithmic thinking, problem-solving skills, and interest in computer science among young learners.

Keywords: educational robotics; visual programming; Python programming; web-based simulation; ROS2; Blockly; sim-to-real learning; hybrid robotics platforms

1. Introduction

Robotics has emerged as a powerful medium for engaging students in STEM disciplines by blending mechanical systems, coding, and real-world logic into tangible learning experiences. However, widespread adoption of robotics in education is often hindered by hardware costs, classroom complexity, and a steep learning curve for programming. To address these challenges, web-based educational applications have become increasingly popular, offering accessible, flexible, and scalable learning environments.

This paper presents two complementary web applications developed within the MetaRoboLearn project. Both systems enable students to program and control robots through their browsers, with support for virtual and physical execution. The first application features a visual block-based interface suitable for younger learners, while the second provides a text-based Python coding environment aimed at more advanced users. Both platforms integrate with a 3D web simulator and can also connect to a ROS2-enabled physical robot, allowing users to transition seamlessly between simulation and real-world testing.

The designed technology ecosystem is primarily intended for use in primary and secondary STEM education, where students have diverse levels of prior programming experience. In early primary school settings, the block-based Blockly environment enables learners to construct robot programs without worrying about syntax, supporting inclusive participation and early exposure to algorithmic thinking. In lower and upper secondary education, the Python-based application provides a bridge toward more advanced computational practices, enabling students to work with structured code, debugging, and problem decomposition. Both tools can be flexibly integrated into classroom lessons, extracurricular robotics clubs, or teacher-led workshops, and support collaborative learning where groups of mixed-ability students solve similar tasks using different interfaces.

The following sections detail the pedagogical context, system architecture, individual applications, and their educational implications. By analyzing these tools side-by-side, the paper explores how hybrid web environments can support progressive learning in robotics, from visual abstraction to structured coding, within a unified technical ecosystem.

2. Related work

Educational robotics (ER) is increasingly recognized as an effective pedagogical approach that supports learning through hands-on engagement with physical and virtual robots. It promotes the development of computational thinking, problem-solving, and collaborative skills, particularly within STEM education (Mikropoulos & Bellou, 2013; Zhong & Xia, 2020). Numerous empirical studies have shown that ER can enhance student achievement, motivation, and confidence, especially when integrated into game-based or collaborative activities (Chen et al., 2023; Ouyang & Xu, 2024). A key distinction in ER lies between physical and virtual robotic systems. Physical platforms, such as LEGO Mindstorms or Arduino kits, offer direct interaction with hardware and, allowing students to experience real-world constraints and feedback (García-Tudela & Marín-Marín, 2023; González-García et al., 2020). These systems often enable both basic command execution and more advanced programmable control using custom code or libraries. However, they also present barriers such as higher cost, setup complexity, and limited scalability in classroom environments.

To address these challenges, virtual environments have gained prominence. Platforms like Gazebo, Open Roberta Lab, and Unity-based simulators offer simulated robot control without the need for physical hardware (Tselegkaridis & Sapounidis, 2021). Although these environments lack the tactile experience of hardware interaction, they reduce logistical constraints and allow for repeated experimentation and safe failure – key principles of exploratory learning (Su et al., 2021). Within these educational contexts, block-based programming has emerged as a powerful method for lowering the entry barrier to coding. Tools like Scratch and Blockly use visual metaphors, such as snapping together blocks, to teach programming logic without requiring knowledge of syntax (Maloney et al., 2010). Block-based interfaces have been shown to improve students' algorithmic thinking and provide a more inclusive environment, especially for younger learners or those with limited prior exposure to technology (Tarrés-Puertas et al., 2023). ROS2 has become a widely adopted framework in educational robotics due to its modularity, real-time communication, and support for sim-to-real transfer. Platforms like NVIDIA Isaac Sim allow students to test ROS2 nodes in realistic virtual environments and then deploy the same code on physical robots powered by hardware such as the Jetson Orin Nano. This architecture ensures consistency across platforms and supports advanced AI features, including object detection and voice control, making it suitable for scalable, modern educational use (Bonci et al., 2023; Salimpour et al., 2025).

The applications analyzed in this paper reflect these pedagogical principles. One system uses a Blockly-based interface to allow students to generate code from drag-and-drop elements, which can then be executed on both a simulator and a physical robot. The second application introduces textual Python coding through a web-based editor, supported by syntax highlighting. By combining simulation, physical interaction, and both block-based and text-based programming paradigms, these tools provide a flexible and scalable framework for robotics education that aligns with the evolving technological and pedagogical landscape.

3. System Architecture and Robot Platform

The educational platform centers on a mobile robot designed for both simulation and real-world use, with ROS2 serving as the middleware for modular, real-time communication between software and hardware components. At the hardware level, the robot includes motors, different sensors and an RGB camera. The processing unit is the NVIDIA Jetson Orin Nano. Closed-loop motor control is achieved using a PID controller, and sensor data is published via ROS2 topics for consumption by other nodes or external applications. The robot exposes lower-level velocity control interfaces through ROS2 services and topics. Higher-level commands, such as “move forward for 2 seconds” are programmatically abstracted for educational purposes. This abstraction enables the integration of external user-facing applications, such as block-based or Python-based programming environments, where students can send structured commands to the robot without interacting directly with its hardware APIs.

A key strength of the platform is its dual simulation capability. While tools like NVIDIA Isaac Sim have been explored for high-fidelity robotic simulation, including AI-in-the-loop workflows and ROS2 compatibility, the primary focus of this paper is on web-based simulation environments developed specifically for educational use. These custom web simulators provide an accessible 3D interface directly in the browser without requiring installation or GPU acceleration. They replicate the robot's movement logic and are integrated with the educational applications described in subsequent sections and are, as opposed to the NVIDIA Isaac Sim, much less resource-intensive to run. Isaac Sim is used to replicate physical robots, while the web application simulator is designed exclusively for validating the logic of robot movements. Communication between the web simulator and the backend is abstracted to match the ROS2 data flow, enabling code reuse and conceptual consistency. Students can write and test code in the simulators and then seamlessly transfer it to a physical robot if available. The system architecture is shown in Figure 1. By combining ROS2, modular hardware, and web-native simulation, the system provides a scalable and approachable entry point into robotics education – bridging the gap between simulation, code, and real-world execution.

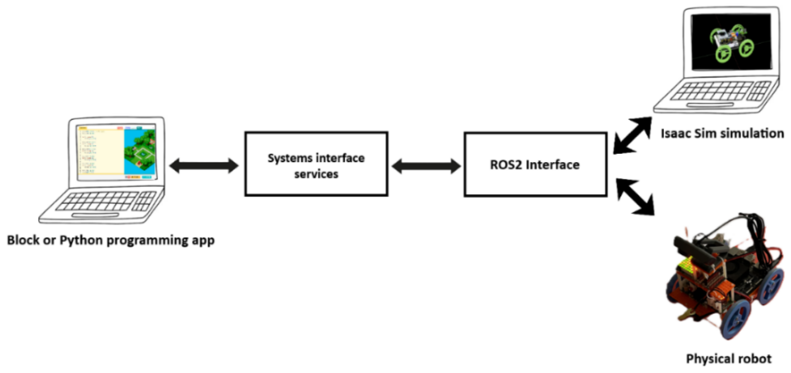


Figure 1. System architecture

4. Visual Programming with Blockly

The first developed application is a browser-based programming environment that enables users to control a robot through a visual block-based interface. It is built using Google’s Blockly library and tailored specifically for robotics control via a set of custom logic blocks. These blocks abstract common robotic commands such as movement, turning, waiting, and sensor interaction, providing a simple and accessible user experience for executing structured tasks. The interface consists of two main components: a block workspace (users construct programs by arranging graphical blocks into logical sequences) and a simplified simulator – designed as a quick way to check the logic of a created program without the need to connect to a ROS2 interface.

Each user action is compiled into Python instructions that follow a predefined command format. These include high-level commands such as move_forward(time, speed) or turn(time, speed). The communication between the Blockly frontend and the server is handled over HTTP and WebSocket protocols, ensuring responsiveness during execution and real-time feedback. The robot platform supports synchronous and asynchronous commands, allowing for both simple and complex behaviors to be constructed.

The code is validated on the server side and executed. The server interfaces with two possible execution environments: a 3D simulator, integrated in the app, for immediate feedback or a ROS2 interface, which represents either the Isaac Sim or a physical robot for real-world testing and deployment. This architecture allows for seamless switching between virtual and physical modes, without requiring the user to change their code or configuration. When connected to a real robot, the application also provides a tab for previewing the live stream from the robot's camera. The application interface is shown in Figure 2.

This application is designed to be modular, extensible, and platform independent. New blocks can be added via a JSON-based definition system, allowing for easy customization. All communication is encapsulated and abstracted to match ROS2 message structures, making the same codebase executable on both simulated and real hardware.

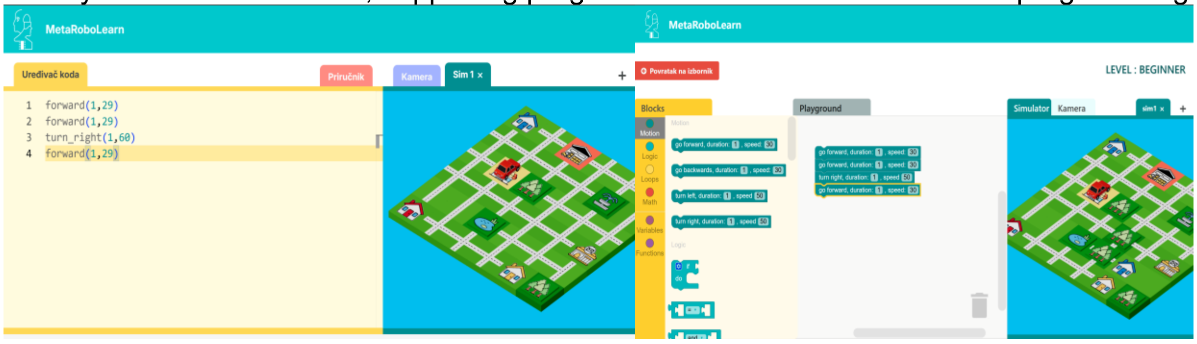
5. Python-Based Robot Programming

The second application is a browser-based Python programming environment that allows users to write and execute custom code to control a mobile robot. It features a lightweight web interface providing essential programming features such as syntax highlighting, indentation support, and error hints. The environment exposes a simplified API for robotic control, allowing users to issue commands like move_forward(time, speed) or turn(time, speed) directly in Python, integrated as functions in regular Python code.

The user interface consists of a text editor and a built-in 3D simulator panel. Unlike the Blockly application, this interface requires users to understand code structure and logic, enabling more flexible and precise program design. Upon execution, the written Python script is sent to the backend via HTTP, where it is validated and parsed. Each command is processed through a sandboxed Python runtime and then translated into executable instructions.

Other aspects of the application, such as server-side validation and possible execution environments are identical to the Blockly app, ensuring scalability and platform independence. The API is modular, and new functions can be added with minimal effort. Communication patterns mirror those used in real ROS2 systems, allowing learners to transition toward native ROS2 development environments without needing to relearn foundational concepts.

Figure 2 shows an equivalent robot program implemented in both applications. While Blockly lowers the entry barrier by abstracting syntax, Python allows learners to engage directly with structured code, supporting progression from visual to text-based programming.



```
forward(1,29)
forward(1,29)
turn_right(1,60)
forward(1,29)
```

go forward, duration: 1 , speed: 30
go forward, duration: 1 , speed: 30
turn right, duration: 1 , speed: 50
go forward, duration: 1 , speed: 30

Figure 2. Equivalent robot program implemented in Python (left) and Blockly (right).

6. Discussion

The Blockly and Python-based applications represent two complementary approaches to programming and robotics education. Technically, both systems integrate with the same execution infrastructure, allowing the code to be tested in a 3D browser-based simulator or on a physical robot. This sim-to-real consistency encourages experimentation in a safe environment before transitioning to hardware, reinforcing the practical connection between abstract code and tangible behavior – an approach that (González-García et al., 2020) demonstrated as effective for knowledge transfer, though their work focused on single programming environments rather than integrated multi-interface systems. While both discussed applications operate on the same robot platform and share a unified backend and simulation framework, their user interfaces, target audiences, and pedagogical impact differ. Figure 3 depicts the same example being tested in real life and in the simulation and illustrates the sim-to-real consistency enabled by the shared ROS2 backend, allowing students to experiment safely in simulation before deploying code on hardware.

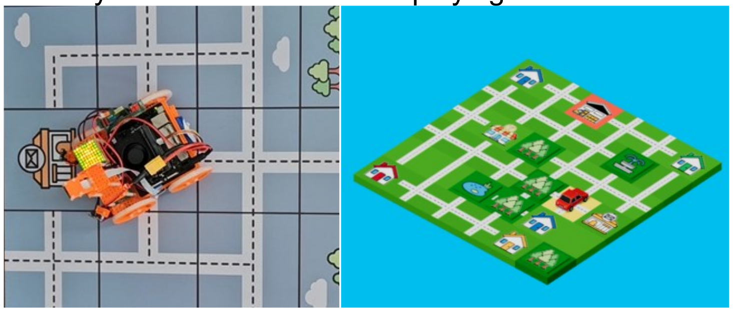


Figure 3. Execution of the same task with the physical robot (left) and in the simulation environment (right).

The Blockly-based application is designed as an entry point for novice users, particularly in primary and lower secondary education. It uses a visual, block-based interface to abstract away syntax and low-level details. Users create programs by combining predefined logical units, which are then translated into executable Python code behind the scenes. This structure makes it ideal for early learners or students without prior programming experience, building upon principles established by (Maloney et al., 2010) in Scratch, though extending beyond their single-environment approach to enable seamless progression to text-based programming.

In contrast, the Python-based application is targeted toward older or more advanced students who are ready to transition from visual programming to text-based scripting. It offers full access to Python syntax, supports conditional logic, loops, and variable handling, and introduces the structure of real-world robotics programming. The underlying command API is shared with the Blockly environment, ensuring continuity in learning.

The inclusion of simulation ensures that students can engage with robotics regardless of hardware availability, responding to accessibility barriers documented by (Tselegkaridis & Sapounidis, 2021). The 3D web-based simulator, tightly integrated into both applications, provides real-time visual feedback and reinforces the connection between code and motion. This supports iterative experimentation, a core principle of constructionist learning. The platform supports differentiated instruction, addressing calls by (Stasolla et al., 2025) for more inclusive educational robotics approaches. Educators can tailor lessons to student readiness by choosing the appropriate interface. In mixed-ability classrooms, some students may work in Blockly while others tackle the same task in Python, all within the same robotic framework – implementing the collaborative learning across skill levels that (Ouyang & Xu, 2024) identified as beneficial but noted was rarely implemented in existing platforms.

The current conceptualization of the ecosystem makes several assumptions and faces certain constraints. First, while the simulator enhances accessibility, classroom use of the physical robot still depends on hardware availability and associated costs. Second, the approach assumes stable internet connectivity and access to modern web browsers, which may limit adoption in resource-constrained educational contexts. Third, the progression model assumes that learners can transition smoothly from Blockly to Python; however, this pathway may not fit all students equally, and additional scaffolding may be required. Finally, the web simulator is intentionally lightweight and does not replicate all physical dynamics of the real robot, which may lead to differences in task execution between simulation and reality. These limitations provide directions for future refinements.

7. Conclusions

This work presented two robot control applications with a block-based environment and a Python-based coding interface. Both systems share a unified execution backend and support deployment on a ROS2 interface (either a physical or a virtual robot simulated in Isaac Sim) or in an integrated browser-based simulator. This architecture ensures a consistent programming experience across interfaces and promotes a seamless transition from visual to text-based coding. By abstracting robot control through a simplified API and enabling real-time feedback via simulation, the platform accommodates a wide range of learners and instructional settings. The use of ROS2 as the middleware enables compatibility with modern robotics tools and supports future system expansion. Future work will focus on extending the Python API to include advanced sensor integration, expanding the Blockly command set, and improving simulation fidelity. Additional plans include integrating student progress tracking, supporting classroom orchestration tools, and exploring adaptive learning features. The platform also lays the foundation for more advanced topics, including AI and machine learning in robotics education. Through its modular design and accessibility, the system offers a scalable solution for introducing students to robotics in both virtual and physical environments.

References

Bonci, A., Gaudeni, F., Giannini, M. C., & Longhi, S. (2023). Robot Operating System 2 (ROS2)-Based Frameworks for Increasing Robot Autonomy: A Survey. *Applied Sciences*, *13*(23), Article 23. https://doi.org/10.3390/app132312796

Chen, T.-I., Lin, S.-K., & Chung, H.-C. (2023). Gamified Educational Robots Lead an Increase in Motivation and Creativity in STEM Education. *Journal of Baltic Science Education*, *22*(3), 427–438. https://eric.ed.gov/?id=EJ1382842

García-Tudela, P. A., & Marín-Marín, J.-A. (2023). Use of Arduino in Primary Education: A Systematic Review. *Education Sciences*, *13*(2), Article 2. https://doi.org/10.3390/educsci13020134

González-García, S., Rodríguez-Arce, J., Loreto-Gómez, G., & Montaño-Serrano, V. M. (2020). Teaching forward kinematics in a robotics course using simulations: Transfer to a real-world context using LEGO mindstorms™. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, *14*(3), 773–787. https://doi.org/10.1007/s12008-020-00670-z

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.*, *10*(4), 16:1-16:15. https://doi.org/10.1145/1868358.1868363

Mikropoulos, T. A., & Bellou, I. (2013). Educational Robotics as Mindtools. *Themes in Science and Technology Education*, *6*(1), 5–14. https://eric.ed.gov/?id=EJ1130925

Ouyang, F., & Xu, W. (2024). The effects of educational robotics in STEM education: A multilevel meta-analysis. *International Journal of STEM Education*, *11*(1), 7. https://doi.org/10.1186/s40594-024-00469-4

Salimpour, S., Peña-Queralta, J., Paez-Granados, D., Heikkonen, J., & Westerlund, T. (2025). *Sim-to-Real Transfer for Mobile Robots with Reinforcement Learning: From NVIDIA Isaac Sim to Gazebo and Real ROS 2 Robots* (No. arXiv:2501.02902). arXiv. https://doi.org/10.48550/arXiv.2501.02902

Stasolla, F., Curcio, E., Borgese, A., Passaro, A., Di Gioia, M., Zullo, A., & Martini, E. (2025). Educational Robotics and Game-Based Interventions for Overcoming Dyscalculia: A Pilot Study. *Computers*, *14*(5), Article 5. https://doi.org/10.3390/computers14050201

Su, L., Qiu, G., Tang, W., & Chen, M. (2021). A ROS Based Open Source Simulation Environment for Robotics Beginners. *2021 6th International Conference on Robotics and Automation Engineering (ICRAE)*, 286–291. https://doi.org/10.1109/ICRAE53653.2021.9657761

Tarrés-Puertas, M. I., Costa, V., Pedreira Alvarez, M., Lemkow-Tovias, G., Rossell, J. M., & Dorado, A. D. (2023). Child–Robot Interactions Using Educational Robots: An Ethical and Inclusive Perspective. *Sensors*, *23*(3), Article 3. https://doi.org/10.3390/s23031675

Tselegkaridis, S., & Sapounidis, T. (2021). Simulators in Educational Robotics: A Review. *Education Sciences*, *11*(1), Article 1. https://doi.org/10.3390/educsci11010011

Zhong, B., & Xia, L. (2020). A Systematic Review on Exploring the Potential of Educational Robotics in Mathematics Education. *International Journal of Science and Mathematics Education*, *18*(1), 79–101. https://doi.org/10.1007/s10763-018-09939-y