UNIVERSITY OF TARTU

Faculty of Science and Technology

Institute of Bioengineering

Usman Ali Khan

Graphical programming interface for ROBOTONT, an open-source educational robot

Bachelor's Thesis

Curriculum Science & Technology

Supervisor:

Assoc. Prof., PhD Karl Kruusamäe

Tartu 2025

Graphical programming interface for ROBOTONT, an open-source educational robot

Abstract

This thesis demonstrates the development of a Blocky-based graphical programming interface as a visual programming language. It enables the control of Robotont, a research and educational mobile robot, through ROS (Robot Operating System). This research approach shows how Google Blockly can lower the complexity of robotics programming for non-expert programmers. The system developed through a design-based approach focuses on simplicity while reducing the technical barriers (ROS implementation and complexity). The key results of this study are a Blockly web app that can create custom blocks for Robotont control and offers a complete solution for ROS communication and robot simulation. This work, due to its accessibility, has the potential to be a framework for robotics education and training, and emphasizes the importance of human-robot interaction.

Keywords

Graphical programming interface, Visual programming language (VPL), Robot operating system (ROS), Web application, Google Blockly, Educational robotics

CERCS: T120 Systems engineering, computer technology; T125 Automation, robotics, control engineering

Institute name: Institute of Technology

Research group: Intelligent Materials and Systems Laboratory

Graafiline programmeerimisliides avatud lähtekoodiga õpperobotile ROBOTONT

Lühikokkuvõte

See lõputöö demonstreerib Blocky-põhise graafilise programmeerimisliidese väljatöötamist visuaalse programmeerimiskeelena. See võimaldab ROS-i (Robot Operating System) kaudu juhtida mobiilset haridusrobotit Robotont. See lõputöö rõhutab, kuidas Google Blockly võib mitteekspertide programmeerijate jaoks robootika programmeerimise keerukust potentsiaalselt vähendada. Disainipõhise lähenemisviisi kaudu välja töötatud süsteem keskendub lihtsusele, vähendades samal ajal tehnilisi kaubandustõkkeid (ROS-i rakendamine ja keerukus). Selle uuringu peamised tulemused on Blockly veebirakendus, mis võimaldab luua kohandatud plokke Robotonti juhtimiseks ja pakub täielikku lahendust ROS-i suhtluseks ja robotite simuleerimiseks. See töö võib oma juurdepääsetavuse tõttu olla robootikahariduse ja -koolituse raamistik ning rõhutada inimeste robotite suhtlemise tähtsust.

Võtmesõnad:

Graafiline programmeerimisliides, visuaalne programmeerimiskeel (VPL), robotite operatsioonisüsteem (ROS), veebirakendus, Google Blockly, haridusrobootika

CERCS: T120 Süsteemitehnoloogia, arvutitehnoloogia; T125 Automatiseerimine, robootika, juhtimistehnika

TABLE OF CONTENTS

TERMS, ABBREVIATIONS, AND NOTATIONS

CSS - Cascading Style Sheets

HTML - Hypertext Markup Language

ROS - Robot Operating System

VPL - Visual Programming Language

XML - Extensible Markup Language

1 INTRODUCTION

Just until a few years ago, programming and other computational skills were considered to belong to a niche group of highly skilled professionals, and the general public kept a safe distance from them. Since then, a lot has changed. Not only on an individual level, but also on a global level, making digital literacy a required skill for everyone. Computational thinking (Papert, 1980) and programming are now foundational skills in modern education to prepare the new generation for the ever-evolving technology-centric society that we are part of (Hazzan *et al.,* 2020; Nouri *et al.,* 2020; Wing, 2006), and also making them a necessary part of the early education curriculum. The main issue hindering the implementation of coding in the early school system is its syntax-heavy nature and young kids not being able to learn the basics of programming syntax and commands, as well as abstract thinking and generalizing concepts (Demir, 2022; Mladenović *et al.,* 2021).

Tools like Visual Programming Languages (VPLs), such as Scratch and Google Blockly, have revolutionised educational robotics by offering intuitive, block-based interfaces and making coding more accessible (Brennan & Resnick, 2012; Sáez-López *et al.,* 2016). For instance, platforms like LEGO Mindstorms enable learners to design, build, and program robots through tangible, hands-on projects (Alimisis, 2013; Karaca & Yayan, 2020). Despite their unmatchable benefits, VPLs have some limitations too, one of them being confined to highly simplified environments. This becomes an issue when the learners have to face complex projects within their professional domains, which require more advanced problem-solving skills (Quigley *et al.,* 2009). This raises a critical question: *what can be done to enable non-experts to transition from simple block-based programming to tackling more complex and advanced robots, like Robotont — a framework relying on ROS and requiring expertise in Linux, command-line interfaces, and ROS build systems like colcon?*

This question might sound straightforward, but it presents a tough choice. On the one hand, ROS, despite being a powerful tool, suffers from technical barriers. New learners must master skills like package creation, node communication, and colcon building, which demand significant time and mentorship (Quigley *et al.,* 2009). On the other hand, current VPLs are short-handed when it comes to their operability with ROS. This means, when learners relying on VPLs have to transition to industrial-level robotics, they are forced to leave visual programming behind and assimilate into this new, more complex environment (Mladenovic *et al.,* 2021). This gap between robotics taught with VPLs and advanced robotics creates a disconnect and excludes non-experts from contributing to real-world applications.

This thesis addresses this same challenge and aims to implement a custom Blockly-based web application that bridges simple educational and complex professional robotics workflows. The system translates block-based logic into ROS-compatible Python code, implements a code generation pipeline that uses colcon to compile ROS executable packages, and integrates a simulation-driven workflow where users validate programs in RViz. What it does is help non-expert users to control Robotont by focusing on algorithm and robot design without having to worry about syntax or system configurations.

Results show that integrating VPLs with professional tools like ROS is not only possible but also pedagogically transformative. The system successfully demonstrates that users with no prior ROS experience can program Robotont within minutes while maintaining compatibility with industrial standards. Therefore, positioning Blockly as a gateway to advanced robotics. By merging the accessibility of VPLs with the power of ROS, this research contributes to a broader vision: democratizing robotics education so that anyone, regardless of technical background, can participate in shaping the future of human-robot interaction.

2 LITERATURE REVIEW

2.1 Educational robotics

Due to its practical benefits, the use of robotics and programming in early education has gained popularity in recent times. Training targeting schoolchildren from 7 to 9 grades enables learners to make themselves proficient with the knowledge and abilities to master the essential technical and technological skills. The use of robotics in the educational process leads to a rapid development of the entire set of cognitive processes (perception, presentation, imagination, thinking, memory, speech) in learners (Ospennikova, *et al.,* 2005).

Other studies have also shown positive effects of integrating robotics in children's training and learning (Karna-Lin, *et al.,* 2006) by improving their abilities in robot design (Soares, *et al.,* 2011) and enhancing their confidence in problem-solving, decision-making, teamwork, physics, and mathematics (Thomaz, *et al.,* 2009).

Learning programming as 21^{st} century knowledge: It is becoming evident that only theoretical knowledge is not enough for 21st-century graduates. As university graduates are often found to be incapable of performing their professional work-related duties due to the lack of abilities required to create something new, to be innovators and not only users (Bialik, *et al.,* 2015). The 4Cs (Creativity and innovation, Critical thinking and problem-solving, Communication, and Collaboration) are the requirements in creating "globally competitive learners" (Zain, *et al.,* 2016). Last century knowledge, which was based on "Three Rs" (reading, writing, and arithmetic), is not enough in the current technologically advanced society or probably in the next two decades when today's students finish school (Alismail, *et al.,* 2015). Large groups of educators have concluded that to incorporate the 4Cs, they must engage learners in more problem and project-based learning in classrooms (Lamb, *et al.,* 2017).

2.2 Advantages of learning programming

What is programming? Definition from Khan Academy says: "Programming is the process of creating a set of instructions that tell a computer how to perform a task" (Khan Academy, 2019). There are at least five main steps to develop in the programming process: defining and analyzing the problem, planning the solution, coding the program, testing, and evaluating the program. Writing the program is quite a complex process; hence, learning programming is not easy (Tiky, 2016). In the recent past, programming was considered to be mastered by some

special, talented people with high cognitive abilities, but now it has become a necessary component of teaching curricula, even in primary schools (Sterling, 2016).

Various studies have been conducted to observe if the learners improve their skills listed in the 4Cs. Programming allows learners to use imagination, knowledge, background, and experience to come up with a solution, which improves creativity and also provides them with the opportunity to implement their ideas (Ryan, 2019). Nelson (2018) concluded that learners acquire decision-making and problem-solving skills that can help them solve real-life problems, because during program writing, one always needs to go back-and-forth to fix incorrect results. This approach of repeating attempts until you succeed can be very useful for their future life.

To create a program, learners need to write an algorithm that is a set of sequentially formulated instructions that must be precisely executed to achieve the results accurately, which increases their planning skills (Saeli, *et al.,* 2011). Most programming projects are developed by a group of individuals, complex projects may require many different skills, where each part can be designed by a professional to deliver the final product, enhancing collaboration skills (Nelson, 2018). During collaborative programming activities in classrooms, learners share their ideas, communicate with other program users or teachers for potential changes to make their programs more user-friendly, which helps to elevate communication skills (Ryan, 2019).

2.3 Visual programming languages (VPLs)

Graphical programming interfaces are simple easy-to-use tools that enable users to control their program elements using visual interfaces. These visual interfaces are generally in the form of a visual programming language (VPL). VPLs enable users to create a program and control the program components graphically rather than defining them textually. It is a new trend for educational robotics environments (Jost, *et al.,* 2014). The main feature that makes a successful visual programming environment is the use of visual blocks/images or diagrams. This setup makes VPLs a great way to introduce new learners to coding, since it eliminates the need to type the text manually, minimizing the possibility of syntax errors, which happens to be one of the main challenges faced by new learners (Karaca & Yayan, 2020).

Different research lines have been introduced to help learners develop algorithms without prior knowledge of programming tools and concepts. Game-like structures are often used to integrate programming into the education system due to their simpler design and higher appeal for the younger generation. *Scratch* (Kaučič & Asič, 2011; Scratch, 2007), developed

especially for children aged between 8 and 16, is intended to aid children's cognitive and social development by providing them with opportunities to design and share their interactive activities, like games, stories, animations, and music. It is beneficial for enhancing creative thinking, systematic reasoning, and teamwork, a view of the Scratch developer console (see Figure 1).

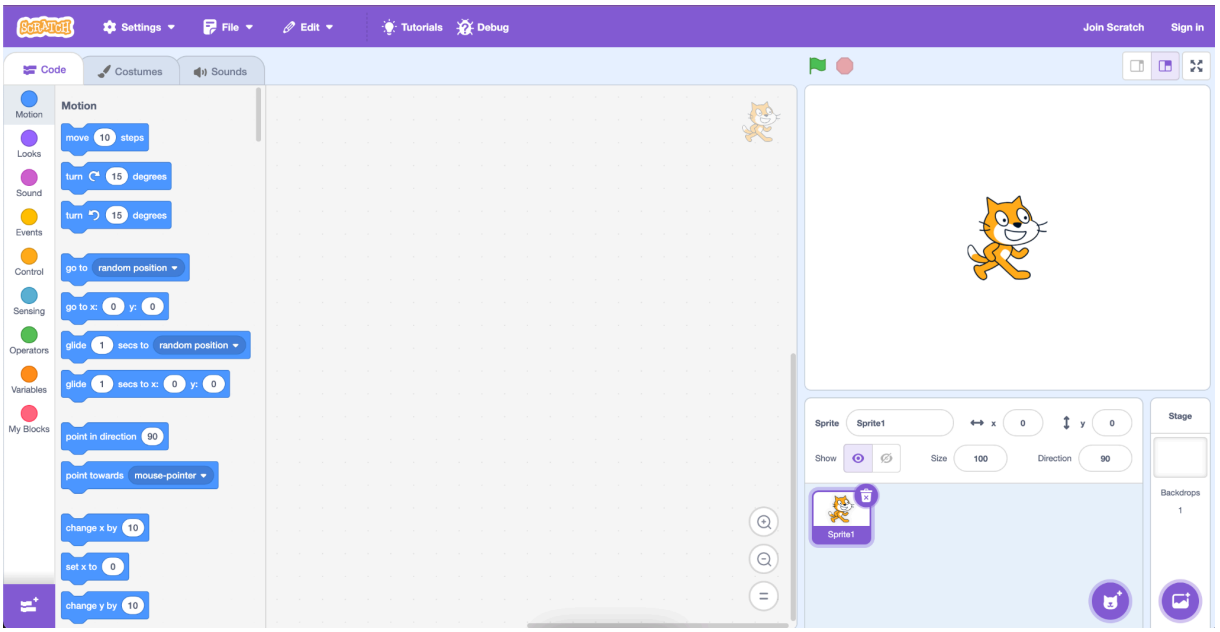

Figure 1: Screenshot from Scratch

Likewise, NASA has developed a project named *Rover* ("NASA Rover Game", 2018). In this educational game, users are instructed to write algorithms to make the rover move and to reach a specific point while dodging the obstacles and controlling the fuel. *Hammer* (Mateo, *et al.,* 2014) is another project worth mentioning. Hammer is an Android-based application developed especially for those with little to no programming knowledge. Using no specific programming language, the aim is to build a new robot program or to modify an existing one using visual blocks (see Figure 2).

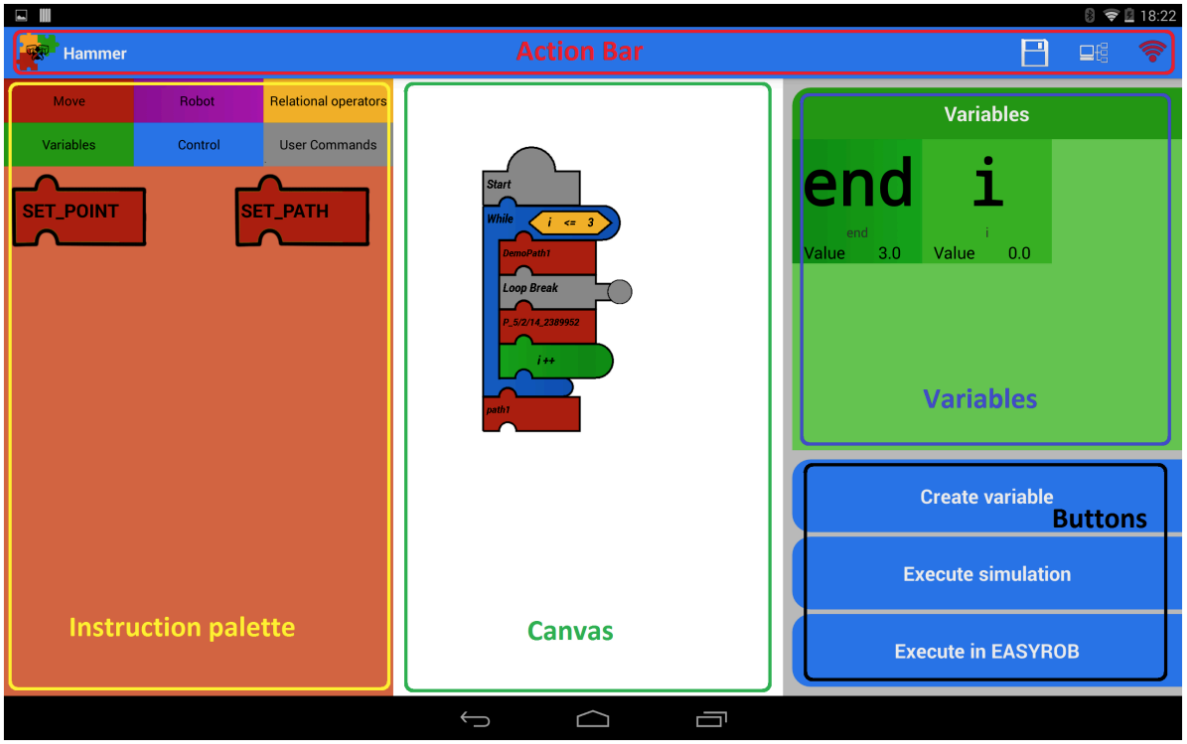

Figure 2: Hammer scratch view

The key to success for these programs is their simplicity, and these programs are being introduced to children as early as their primary education days. For instance, in Japan, an educational system called *P-CUBE* has been implemented (Kakehashi, *et al.,* 2013; Kakehashi, *et al.,* 2014). Using functional cubes, children build algorithms that move a robot. To make this possible, RFID tags are used in the construction of the cube, and IR sensors are added on the mobile robot. *Algorithmic Bricks* (Kwon, *et al.,* 2012) is a simplistic system designed for young children. With the purpose of teaching programming logic and processes, the system consists of a robot and visual objects, and the aim is to make the robot move by building algorithms with the visual blocks.

LSOFT, developed by Lego, is a visual programming interface. Using the drag and drop technique, this tool is built to program a robot called the Mindstorm NXT robot, which makes the programming easier and even suitable for people with limited mobility (Kim & Jeon, 2008). Other works worth mentioning are the ones developed by the University of Deusto Bilbao in Spain (Iturrate *et al.,* 2013) and Lego (Kim & Jeon, 2008). Iturrate, *et al.,* (2013) have created a maze scenario game, in which the user has to gather objects with the robot, programmed by the created blocks of VPL Google Blockly, a view of the Blockly developer console (Google Blockly, 2012) (see Figure 3).

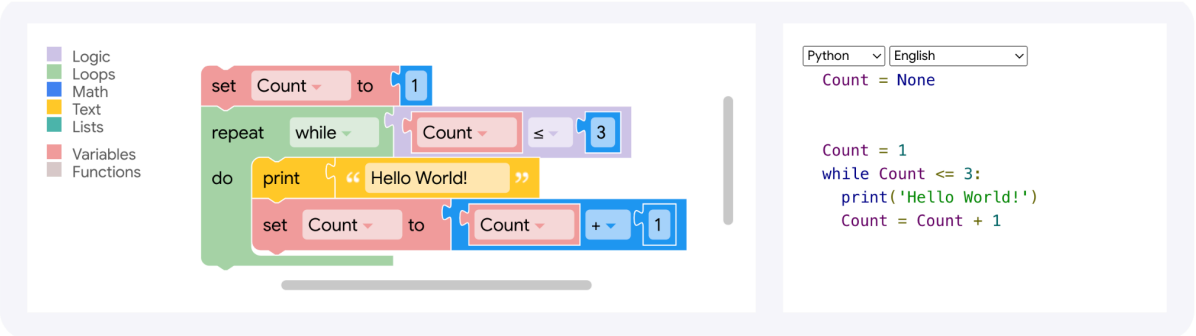

Figure 3: Google Blockly developer console

In a recent study (Karaca & Yayan, 2020), *Robot Operating System* (ROS) compatible system was developed for Evarobot ("Evarobot Tanıtım Sayfası", 2018), which is a mobile robot intended for educational and research and development purposes (see Figure 4).

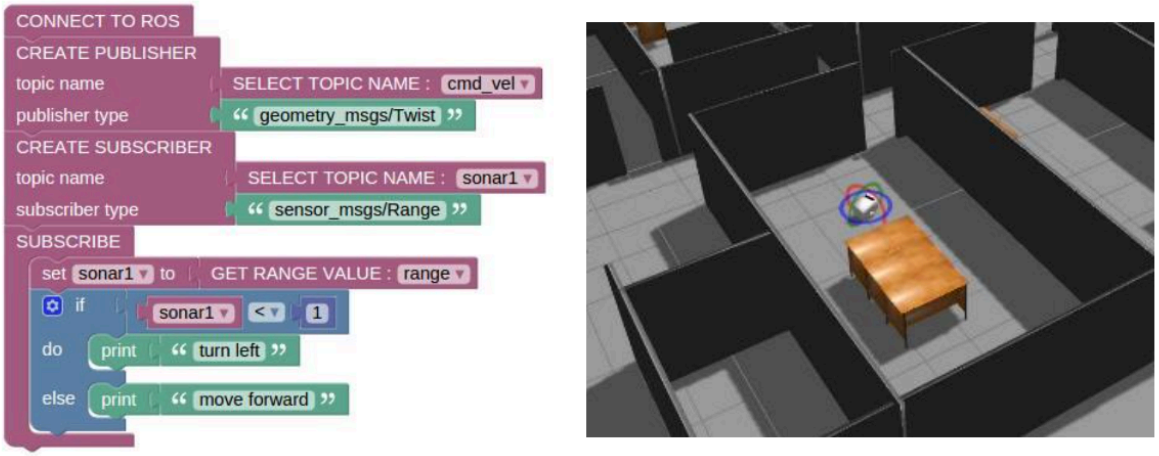

Figure 4: Evarobot Wander Code

Karaca & Yayan (2020) described the system architecture and showed the block definitions in their paper, however, the unavailability of the public code repository and the lack of system setup instructions limit reproducibility. Evarobot's system depends on the ROSBridge to communicate between Blockly and ROS backend, which can introduce latency, connectivity delays, and security risks. Evarobot's system lacks offline support and does not allow users to download the code and reuse it as a standalone ROS package node. Evarobot's system is especially designed for its specific hardware and simulation tools, making it incompatible with Robotont. Evarobot's solution uses ROS 1, which is deprecated now, and the ROS team might not offer support for ROS 1 soon.

3 AIMS OF THE THESIS

3.1 Integrating a graphical programming interface to control a ROS-enabled robot

Developing a web-based interface that runs Google Blockly as a visual language programming platform and translates Blockly blocks into a ROS-compatible package, which can be used to control a ROS-supported robot.

3.2 System requirements

- ROS2 Jazzy Jalisco
- Ubuntu 24.04 (Noble Numbat)

4 EXPERIMENTAL PART

4.1 Software tools and requirements

ROS: ROS is an open-source software operating system for writing robotics programs. It is a distributed framework of processes (aka *Nodes*) that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into *Packages* and *Stacks*, which can be easily shared and distributed. The nodes can publish or subscribe to a topic (Documentation - ROS Wiki, 2024).

Blockly by Google: Blockly is an open-source JavaScript library for creating visual programming interfaces (Google Blockly, 2012). Blockly blocks can be translated into executable ROS-node code. The generated code makes use of topics/nodes for communication and can be deployed as a ROS node onto a robot to control its behavior, for instance, doing some basic movements (Karaca & Yayan, 2020).

Robotont - a mobile robot platform: Robotont is an open-source hardware and software platform for advanced robotics education and research. It is an omnidirectional mobile robot with ROS (Raudmäe, *et al.,* 2023). Robotont has been chosen to carry out the testing for the code generated by Blockly blocks. To test the code, we first need to know how to build this code into a ROS node on Robotont.

Tools for simulation: RViz simulator is used for testing the results without needing hardware, by adding some external ROS packages to make it compatible with Robotont (Kam, *et al.,* 2015).

Web application stack: HTML, CSS, JavaScript, Node.js (for serving web pages).

Programming languages: Blockly blocks will be translated using Python as the core language.

4.2 System design and workflow

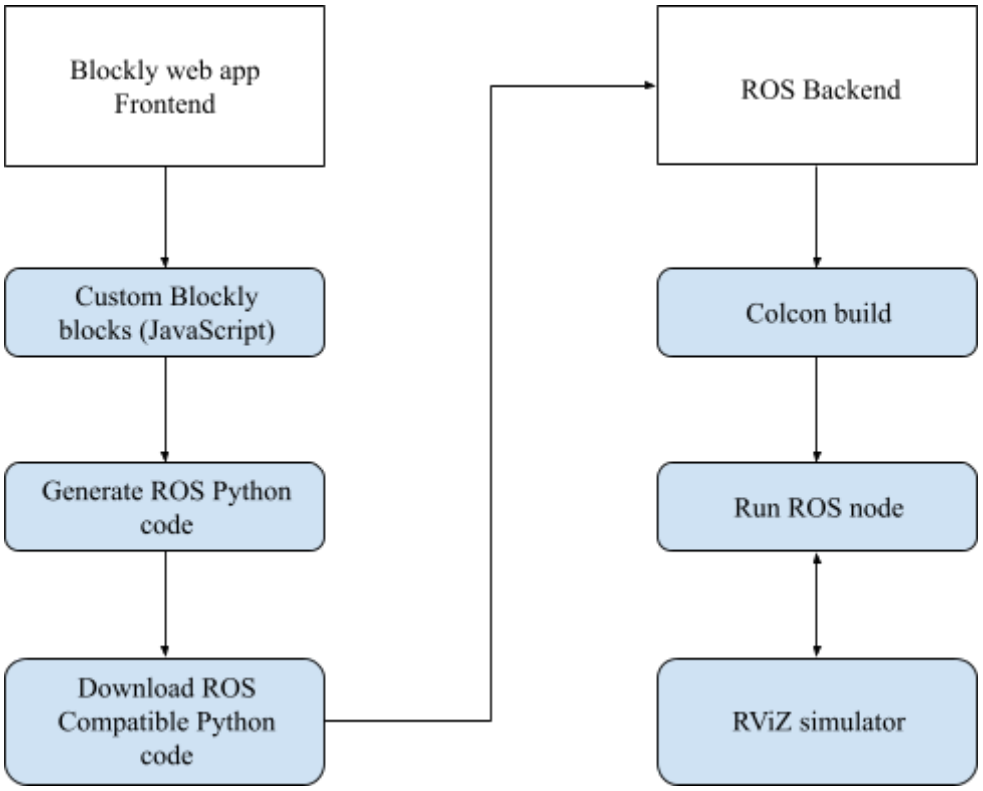

Figure 5: System design and workflow

The system workflow follows the above-mentioned sequence of steps (see Figure 5). Blockly web app runs a Visual programming language platform to write the program (drag/drop blocks). Blocks are customized to handle ROS Python code generation and functionality. The generated code is then downloaded as a ROS executable Python node. This downloaded ROS node is later compiled into a ROS executable package using a standalone ROS backend with `colcon` build tools setup. ROS backend runs the setup nodes and launches `RViz` simulator to test your results.

4.3 ROS integration workflow

To use ROS functionality effectively, the system design uses ROS2 Jazzy Jalisco, which provides a bridge between Blockly web application and Robotont hardware or simulator. To translate Blockly custom blocks into executable commands, for instance, controlling the velocity by utilizing the topic `/cmd_vel`. ROS publishers/subscribers for data exchange and packages designed to work with Robotont hardware or simulator are the essential elements of the system. This integration ensures the seamless execution of user-defined logic in both real-world hardware and RViz simulator.

Communication model and design of ROS nodes: The system uses a modular approach to allow consistent interaction with Robotont. Every node is responsible for controlling a function, like velocity control, sensor data, or simulation interface. Velocity control node uses/cmd_vel topic with Twist message type to manage the linear and angular movement. Topics like /scan and /battery_status track robot health or environmental metrics. Visualization in real-time is made possible with the RViz simulator to observe the actual robot state. ROS topics are organized sequentially to mitigate latency and to ensure scalability. The /cmd_vel topic, for example, uses a publish-subscribe model in which Robotont's driver node subscribes to Blockly-generated node, which acts as the publisher.

Code generation: Customised Blockly-generated blocks are automatically formatted into Python ROS node templates with predefined scaffolding. Blocks are mapped to ROS messages fields (linear/angular velocity), and to wrap the logic into ROS node structure, for example, using rclpy.init(), create_publisher(). Customised blocks also offer run-time compatibility with the robot environment by using time-based executions, see Appendix A for an example template definition for ROS node initialization, publisher creation, and timer execution. The complete solution of a ROS workspace with documentation is available on GitHub https://github.com/usman125/robotont-blockly-app.

4.4 ROS-Blockly integration workflow

Blockly web application serves as a bridge between program design and ROS code execution. Blockly workspace comes loaded with the basic programming functional blocks like simple if/else logic blocks, basic loop blocks, and basic mathematical functions. Unlike vanilla Blockly setup, this system leverages Blockly customization and extension of the base functionality to add ROS compatibility by generating ROS-specific blocks, which are translated into ROS executable nodes using JavaScript functions (see Figure 6). Custom Blockly blocks map the visual logic to ROS topics and message types. For instance, (see Figure 7), a robot motion block generates a Twist message depending on the option selected from the dropdown field to control robot's velocity and movement to draw a circle or a rectangle, it also takes a number input to draw the shape more than once; JavaScript definition of this block is shown in Appendix C. Another block definition (see Figure 8) that adds a ROS node boilerplate (e.g., Node class inheritance, rclpy.init()), see Appendix D for JavaScript definition and Python code generated from this block. Block definitions are injected into Blockly workspace using XML; see Appendix B for an example HTML file with Blockly integration. *Table 1* lists all the available custom blocks, their functions, and usage.

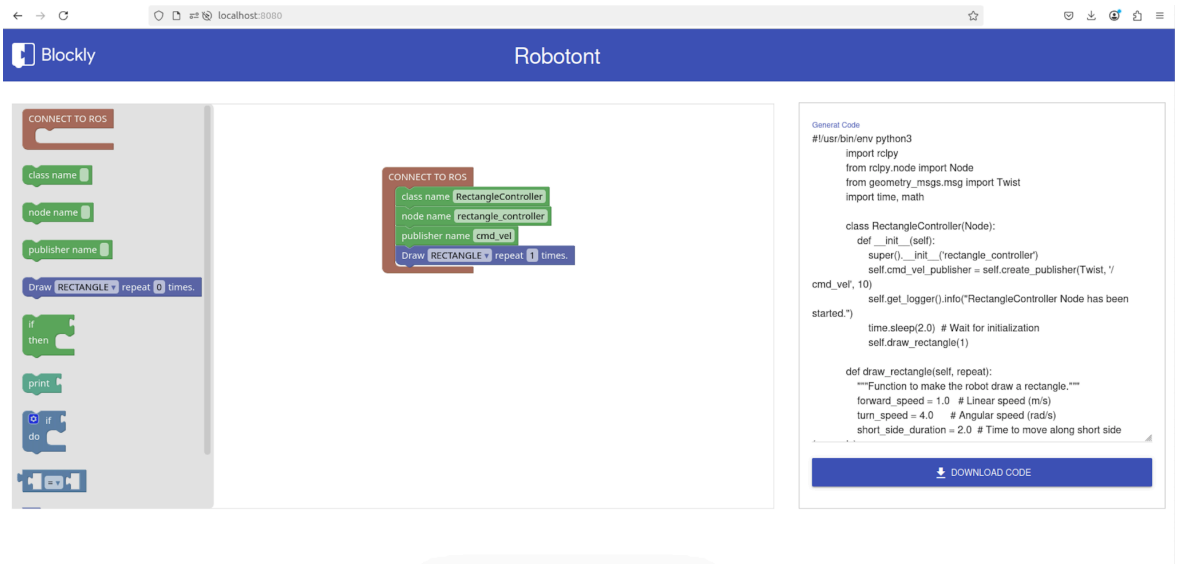

Figure 6: A screenshot of the implemented Blockly workspace using HTML/CSS, JavaScript

Figure 7: A custom block for basic robot movement

Figure 8: Example of a block to add ROS node boilerplate

Block Name	**Function and usage**
Connect to ROS	Generate ROS node Scaffolding code
Draw	Generate the Twist messages to control the robot's velocity and movement
Class name	Overwrite the default class name
Node name	Overwrite the default node name
Publisher name	Overwrite the default publisher name

Table 1. lists the available custom blocks

Blockly's workspace takes advantage of the native functions, especially python.pythonGenerator.forBlock["block_name"], which scaffolds a block's visuals with the ROS-compatible code. Each block needs to have its pythonGenerator function call defined, an example code is given below:

```
python.pythonGenerator.forBlock['ros_connection'] = function(block) {
  return 'import rclpy\nfrom rclpy.node import Node\n';
};
```

Blockly.Python.workspaceToCode(workspace) returns the code generated from the blocks arranged on the workspace into a single Python executable code block. Whenever Blockly workspace changes (a block is dragged in/out of the workspace), the following function is called to keep the code generation in sync:

```
workspace.addChangeListener(() => {
      const code = Blockly.Python.workspaceToCode(workspace);
      . . .
});
```

The generated code is scaffolded using rclpy.init(), create_publisher(), publish(), ensuring compatibility with Robotont's execution, and was downloaded as a Python executable file using JavaScript core functionality. The approach used in this workflow eliminates the connectivity issues and lags that can arise due to the dependencies on web middlewares like RosBridge, and simplifies hardware integration. This reduces the technical barrier by abstracting the complex ROS logic and focusing on algorithm building. A complete solution to a web application running Blockly with the above-defined blocks, with documentation and examples, is available on GitHub https://github.com/usman125/robotont-blockly-client.

4.5 RViZ test simulator

To test the efficiency of the generated downloaded code from the web app on Robotont without using the hardware, RViz is used. To achieve this, following three ROS packages were downloaded in the ROS workspace:

1. `git submodule add -b jazzy-devel https://github.com/robotont/robotont_driver`
2. `git submodule add -b jazzy-devel https://github.com/robotont/robotont_msgs`
3. `git submodule add -b jazzy-devel https://github.com/robotont/robotont_description`

Then we used the following set of commands from the root of our workspace to make the Robotont simulator available (see Figure 9):

```
colcon build # build the code
source install/setup.bash # source the setup file
ros2 launch robotont_driver fake_driver_launch.py # launch a test robotont simulator
```

You can launch your downloaded node from the web app after installing it in a package, and simulate your block logic using the command `ros2 run <package-name> <node-name>`. The simulator setup was done in the code available on GitHub, shared in section 4.3.

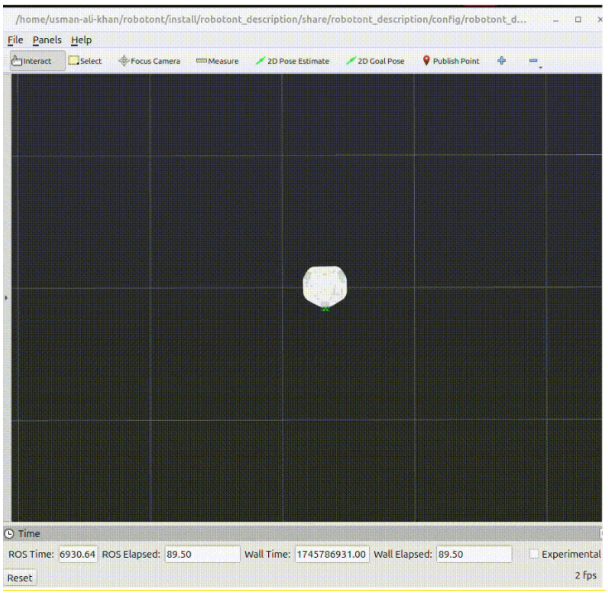

Figure 9: An overview of an RViz robotont simulator

4.6 Results and discussion

The software solution implemented aligns with the aim of the thesis listed in section 3. A web application is available with documentation on GitHub (section 4.4). The ROS project integrating Robotont simulation to validate results is available with documentation on GitHub (section 4.3). Hence, ensuring reproducibility and reducing the complexity by making the ROS syntax abstract while maintaining the functionality.

In the RViZ simulator:

- Robotont draws a near-perfect circle (see GitHub, Section 4.4), with a deviation that was negligible from the implemented radius. A little deviation from the starting point is observed.
- Robotont draws a rectangle executing sequential commands (see GitHub, Section 4.4), though minor errors occur during the rotation of the robot and a little deviation from the starting point.

This ensures that the downloaded code for both types of shapes from the web application blocks was successfully compiled using `colcon` as ROS packages and runs as standalone executable nodes. The software implementation for the thesis work is done using Ubuntu 24.04 and ROS Jazzy Jalisco; all the commands and code examples mentioned in the Appendices and sections are executed in the Ubuntu terminal and any text editor available on the operating system, respectively. This thesis work improved on a previous research conducted by Karaca & Yayan (2020) by:

- Making the code public and easily accessible ensures reproducibility.
- Enabling offline code generation and compilation.
- Eliminating connectivity and latency issues normally coupled with web-middleware like RosBridge.
- Providing cross-web compatibility.
- Ensuring ROS code maintainability by using ROS 2.

Other existing visual programming solutions also have some limitations; either they do not offer code generation or lack ROS compatibility. For example, Scratch lacks ROS compatibility, and it only offers animation code. This thesis work leverages the simplicity of Scratch and enables it for ROS robotics. Blockly abstraction provides the ease of access for non-expert programmers and focuses on logic building, eliminating syntax errors. Also, the addition of the RViZ simulator ensures the code testing before running on hardware. This can

be a great step forward in making a lightweight platform to integrate visual programming for educational and training mobile robots.

4.6.1 Limitations and challenges

The current solution is limited to basic blocks, like drawing basic geometrical shapes (predefined shapes, see section 4.4), and limited ROS functionality options. Enabling the code for testing on real-world hardware can reduce code efficiency due to real-life challenges like wheel slippage. RViZ can have more readable simulation options to observe the output.

The challenge was to overcome Blockly synchronous nature of executing the code, while ROS executed the code asynchronously, and relies on event-driven callbacks. To resolve this issue, the create_timer() function is used, which triggers a callback allowing ROS nodes to process the data continuously, however, this approach can introduce some latency in robot response. Another challenge is long-term maintenance of the Blockly-ROS integration with new ROS distributions and Blockly versions as they update rapidly, as a Blockly block for ROS 2 Jazzy Jalisco may not work in future releases of ROS 2 distributions, due to the changes in APIs or message types.

4.6.2 Future work

A better development roadmap is needed to make this solution more usable by integrating the Blockly workspace with more computationally complex ROS programmable custom blocks, e.g., reading the sensor data, using lidar to adjust, and planning the path. The web application user interface and usability can be improved by incorporating user feedback, custom blocks can be categorized based on their functionality, and embed tooltips that better guide the users about their function, implementing syntax checks, and better error logs and responses to make code debugging easier. Web application interactivity and usage could be increased by adding more custom blocks programmed to add more puzzle-like or gamified scenarios to control the robot, and maximise its potential in educational robotics. Custom blocks can support code translation into multiple programming languages to make the system more robust. For example, future releases can include C++ or Rust code translation options that can incorporate advanced users who prioritize performance, which also aligns with the ROS ecosystem, supporting multiple languages, ensuring support for diverse robotics workflows. Web application creation and Blockly customisation, ensuring ROS-compatible code generation, requires notable time and effort, which leaves limited time for implementing these advanced features.

4.6.3 Conclusion

This work showcases a Blockly platform with ROS support, enabling it to control a ROS-compatible mobile robot. The developed system allows non-expert programmers to do some basic robot operations and validate those results through the simulation workflow for real-time testing. The system’s capability to set up locally and offline makes it more robust and increases accessibility. Some real-world challenges may persist, but the system's potential to scale and the ease of use make it a viable tool. Despite promising results, improvements are still needed to make the system more accurate and user-friendly.

SUMMARY

This thesis highlights Blockly integration as a visual programming language for ROS-compatible robots, enhancing the accessibility for non-expert programmers. The system introduces custom Blockly blocks and makes use of Blockly's native functions, e.g., workspaceToCode(), to generate a scaffold for ROS executable nodes. These blocks can be programmed to enable ROS communication through topics and message types. For instance, /cmd_vel to control the velocity of a robot supporting Twist message types. The system's offline capability eliminates the lags and connectivity issues normally associated with web middleware like RosBridge, and leverages the use of ROS build tools like `colcon` for ROS executable packages compilation. The system offers real-time code validation to test the robot without the need for hardware using the RViz simulator.

The outcome of the thesis work is a Blockly-based graphical programming interface. However, existing visual programming solutions have limitations; they might lack ROS compatibility or have platform-specific development. This work fills in the gap by utilizing Blockly abstraction and ROS functionality to develop a lightweight cross-browser web application, introduces non-expert users to interact with a robot in real time, and provides the opportunity to learn robot and algorithm designing skills.

ACKNOWLEDGEMENTS

Special thanks to my supervisor, Assoc. Prof., PhD Karl Kruusamäe, for his advice and help throughout the thesis work, and for providing valuable suggestions and information throughout my thesis work. I would also like to thank my group members and friends for providing some valuable feedback, which helped me improve my work.

Additionally, Grammarly (Grammarly: Free AI Writing Assistance, 2024) was used for correcting grammatical errors and proofreading the content. DeepSeek (Deep Think R1, 2023) assisted in improving the structure of this thesis.

REFERENCES

1. Alimisis, D. (2013). Educational robotics: Open questions and new challenges. *Themes in Science and Technology Education*, *6*(1), 63-71.
2. Alismail, H. A., & McGuire, P. (2015). 21st century standards and curriculum: Current research and practice. *Journal of Education and Practice*, *6*(6), 150-154.
3. Bialik, M., Fadel, C., Trilling, B., Nilsson, P., & Groff, J. (2015). Skills for the 21st century: What should students learn? *Center for Curriculum Redesign*, *3*(4), 29.
4. Brennan, K., & Resnick, M. (2012, April). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada* (Vol. 1, p. 25).
5. Demir, F. (2022). The effect of different usage of the educational programming language in programming education on the programming anxiety and achievement. *Education and Information Technologies*, *27*(3), 4171-4194.
6. "Evarobot Tanıtım Sayfası" (2018, November 08). Retrieved from The Evarobot website. http://www.evarobot.info/en/index.aspx
7. Google Blockly (2012). https://developers.google.com/blockly
8. Hazzan, O., Ragonis, N., Lapidot, T., Hazzan, O., Ragonis, N., & Lapidot, T. (2020). Computational thinking. *Guide to Teaching Computer Science: An Activity-Based Approach*, 57-74.
9. Iturrate, I., Martín, G., García-Zubia, J., Angulo, I., Dziabenko, O., Orduña, P., ... & Fidalgo, A. (2013, October). A mobile robot platform for open learning based on serious games and remote laboratories. In *2013 1st International Conference of the Portuguese Society for Engineering Education (CISPEE)* (pp. 1-7). IEEE. DOI:10.1109/CISPEE.2013.6701970
10. Jost, B., Ketterl, M., Budde, R., & Leimbach, T. (2014, December). Graphical programming environments for educational robots: Open roberta-yet another one?. In *2014 IEEE International Symposium on Multimedia* (pp. 381-386). IEEE.
11. Kakehashi, S., Motoyoshi, T., Koyanagi, K. I., Oshima, T., Masuta, H., & Kawakami, H. (2014, December). Improvement of P-CUBE: Algorithm education tool for visually impaired persons. In *2014 IEEE Symposium on Robotic Intelligence in Informationally Structured Space (RiiSS)* (pp. 1-6). IEEE. DOI:10.1109/RIISS.2014.7009180
12. Kakehashi, S., Motoyoshi, T., Koyanagi, K., Ohshima, T., & Kawakami, H. (2013, December). º: Block type programming tool for visual impairments. In *2013 conference on technologies and applications of artificial intelligence* (pp. 294-299). IEEE. DOI: 10.1109/TAAI.2013.65
13. Kam, H.R., Lee, SH., Park, T. *et al.* RViz: a toolkit for real domain data visualization. *Telecommun Syst* 60, 337–345 (2015). https://doi.org/10.1007/s11235-015-0034-5
14. Karaca, M., & Yayan, U. (2020). Ros based visual programming tool for mobile robot education and applications. *arXiv preprint arXiv:2011.13706*.
15. Karna-Lin, E., Pihlainen-Bednarik, K., Sutinen, E., & Virnes, M. (2006, July). Can robots teach? Preliminary Results on educational robotics in special education. In

Sixth IEEE International Conference on Advanced Learning Technologies (ICALT'06) (pp. 319-321). IEEE.
16. Kaučič, B., & Asič, T. (2011, May). Improving introductory programming with Scratch?. In *2011 Proceedings of the 34th International Convention MIPRO* (pp. 1095-1100). IEEE.
17. Khan Academy. Computer programming. Khan Academy. (2019). [Online]. Available: ttps://www.khanacademy.org/computing/
18. Kim, S. H., & Jeon, J. W. (2008, April). Using visual programming kit and Lego Mindstorms: An introduction to embedded system. In *2008 IEEE International Conference on Industrial Technology* (pp. 1-6). IEEE. DOI: 10.1109/ICIT.2008.4608362
19. Kwon, D. Y., Kim, H. S., Shim, J. K., & Lee, W. G. (2012). Algorithmic bricks: A tangible robot programming tool for elementary school students. *IEEE Transactions on Education*, *55*(4), 474-479. DOI: 10.1109/TE.2012.2190071
20. Lamb, S., Maire, Q., & Doecke, E. (2017). Key skills for the 21st century: An evidence-based review.
21. Mateo, C., Brunete, A., Gambao, E., & Hernando, M. (2014, September). Hammer: An Android based application for end-user industrial robot programming. In *2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA)* (pp. 1-6). IEEE.
22. Mladenović, M., Žanko, Ž., & Aglić Čuvić, M. (2021). The impact of using program visualization techniques on learning basic programming concepts at the K–12 level. *Computer Applications in Engineering Education*, *29*(1), 145-159.
23. "NASA Rover Game" (2018, November 08). Retrieved from https://www.nasa.gov/audience/foreducators/robotics/home/ROVER.html#.VdZTRrLtmk
24. Nouri, J., Zhang, L., Mannila, L., & Norén, E. (2020). Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. *Education Inquiry*, *11*(1), 1-17.
25. Ospennikova, E., Ershov, M., & Iljin, I. (2015). Educational robotics as an inovative educational technology. *Procedia-Social and Behavioral Sciences*, *214*, 18-26.
26. Papert, S. (1980). *Children, computers, and powerful ideas* (Vol. 10, pp. 978-3). Eugene, OR, USA: Harvester.
27. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In *ICRA workshop on open source software* (Vol. 3, No. 3.2, p. 5).
28. Raudmäe, R., Schumann, S., Vunder, V., Oidekivi, M., Nigol, M. K., Valner, R., ... & Kruusamäe, K. (2023). ROBOTONT–Open-source and ROS-supported omnidirectional mobile robot for education and research. *HardwareX*, *14*, e00436.
29. Ryan. (September 2019). Coding for kids: Reasons kids should get started, and how they can find success. iD Tech. [Online]. Available: https://www.idtech.com/blog/5-reasons-your-childshould-learn-to-code
30. S. Nelson. (March 2018). 5 Reasons why Coding is Important for Young Minds. Learning Resources Ltd. [Online]. Available: http://blog.learningresources.com/5reasonskidscoding/

31. Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. (2011). Teaching programming in secondary school: A pedagogical content knowledge perspective. *Informatics in education*, *10*(1), 73-88.
32. Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming
33. languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97*, 129-141.
34. Scratch (2007). https://scratch.mit.edu/projects/editor/?tutorial=getStarted
35. Soares, F., Ribeiro, F., Lopes, G., Leão, C. P., & Santos, S. (2011, April). K-12, university students and robots: An early start. In *2011 IEEE Global Engineering Education Conference (EDUCON)* (pp. 1133-1138). IEEE.
36. Sterling, L. (2016). Coding in the curriculum: Fad or foundational?.
37. Thomaz, S., Aglaé, A., Fernandes, C., Pitta, R., Azevedo, S., Burlamaqui, A., ... & Gonçalves, L. M. (2009, October). RoboEduc: a pedagogical tool to support educational robotics. In *2009 39th IEEE Frontiers in Education Conference* (pp. 1-6). IEEE.
38. Tiky, Y. T. (2016). Software development life cycle. *Hongkong: The Hongkong University of Science and Technology*.
39. Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.
40. Zain, I. M. (2018). The collaborative instructional design system: An innovative instructional design tool for 21st century learning. *Quarterly Review of Distance Education*, *19*(4), 11-86.

APPENDICES

Appendix A

```
# my_python_node.py
#!/usr/bin/env python3
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist

class TestController(Node):
    def __init__(self):
        super().__init__('test_controller')

        # Publisher to the /cmd_vel topic
        self.cmd_vel_publisher = self.create_publisher(Twist, '/cmd_vel', 10)

        # Publish the message
        self.timer = self.create_timer(0.5, self.publish_me)
        self.get_logger().info("Test Node has been started.")


    def publish_me(self):
        """
        Function to make the robot draw a rectangle.
        """
       ....
        # Create Twist message
        twist = Twist()
        turn.linear.x = ...
        turn.angular.z = ...

        # Publish the message
        self.cmd_vel_publisher.publish(twist)


def main(args=None):
    rclpy.init(args=args)
    node = TestController()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == "__main__":
    main()
```

A ROS Node boilerplate with initialization, publisher creation, and timer execution

Appendix B

```
<!DOCTYPE html>
<html>
 <head>
   <title>Blockly web app</title>
   <link rel="icon" href="data:," />
   <!-- Load Blockly -->
   <script src="scripts/blockly.min.js"></script>

   <!-- Blocks translation to python -->
   <script src="scripts/python_compressed.js"></script>

   <script>
     Blockly.inject("blocklyDiv", {
        toolbox: `
               <xml>
                   <block type="connect_to_ros"></block>
                   <block type="robot_print"></block>
                   <block type="robot_draw"></block>
               </xml>
           `,
        media: "https://unpkg.com/blockly/media/",
      });
   </script>

   <!-- Add styles on the page -->
   <link rel="stylesheet" href="styles/index.css" />
 </head>
 <body>
   <!-- App header container -->
   <header class="header">
     <h1 class="title">Robotont</h1>
   </header>

   <!-- Blockly Workspace Container -->
   <div id="blocklyDiv" style="height: 640px"></div>

   <!-- Textarea to view blocks python code -->
   <textarea rows="7" style="height: 640px"></textarea>

 </body>
</html>
```

An example `index.html` with Blockly integration

Appendix C

```
Blockly.Blocks["robot_draw"] = {
       init: function () {
         this.appendDummyInput()
           .appendField("Draw")
           .appendField(
             new Blockly.FieldDropdown([
               ["RECTANGLE", "RECTANGLE"],
               ["CIRCLE", "CIRCLE"],
               ["STRAIGHT LINE", "STRAIGHT_LINE"],
             ]),
             "DIRECTION"
           )
           .appendField("repeat")
           .appendField(new Blockly.FieldNumber(0, 0, 3, 1), "REPEAT_NUMBER")
           .appendField("times.");
         this.setPreviousStatement(true);
         this.setNextStatement(true);
         this.setColour(230);
         this.setTooltip("Move robot forward/backward");
       },
     };
```

Block definition for drawing a shape

Appendix D

- ROS connection block definition

```
Blockly.Blocks["ros_connection"] = {
       init: function () {
         this.appendDummyInput("DUMMY")
           .setAlign(Blockly.inputs.Align.RIGHT)
           .appendField("CONNECT TO ROS");
         this.appendStatementInput("ros_connection");
         this.setTooltip("set ros node connection");
         this.setHelpUrl("none");
         this.setColour(15);
       },
     };
```

- Python generated code

```
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
import time, math

            class RobotController(Node):
                def __init__(self):
                    super().__init__('robot_controller')
                    self.cmd_vel_publisher = self.create_publisher(Twist,
'/cmd_vel', 10)
                    self.get_logger().info("RobotController Node has been
started.")
                    time.sleep(2.0)  # Wait for initialization


            # No movement function defined
            def no_movement(self):
                self.get_logger().info("No movement pattern selected")

            def main(args=None):
                rclpy.init(args=args)
                node = RobotController()
                rclpy.spin(node)
                rclpy.shutdown()

            if __name__ == "__main__":
                main()
```

Non-exclusive licence to reproduce the thesis and make the thesis public

I, Usman Ali Khan ,
(author´s name)

1. grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

Graphical programming interface for ROBOTONT, an open-source educational robot ,
(title of thesis)

supervised by Assoc. Prof., PhD Karl Kruusamäe ;
(supervisor´s name)

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. I am aware of the fact that the author retains the rights specified in points 1 and 2;

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Usman Ali Khan
20/05/2025