

# Block-based Programming for Two-Armed Robots: A Comparative Study

1  
2  
3  
4  
5 Felipe Fronchetti\*  
fronchettl@vcu.edu  
6 Virginia Commonwealth University  
7 Richmond, Virginia, USA  
8

Nico Ritschel\*  
ritschel@cs.ubc.ca  
University of British Columbia  
Vancouver, British Columbia, Canada  
9

Logan Schorr\*  
schorrl@vcu.edu  
Virginia Commonwealth University  
Richmond, Virginia, USA  
10

Chandler Barfield\*  
barfieldca@vcu.edu  
11 Virginia Commonwealth University  
12 Richmond, Virginia, USA  
13

Gabriella Chan\*  
grazianige@vcu.edu  
Virginia Commonwealth University  
Richmond, Virginia, USA  
14

Rodrigo Spinola\*  
spinolaro@vcu.edu  
Virginia Commonwealth University  
Richmond, Virginia, USA  
15

Reid Holmes\*  
rholmes@cs.ubc.ca  
16 University of British Columbia  
17 Vancouver, British Columbia, Canada  
18

David Shepherd\*  
shepherdd@vcu.edu  
Louisiana State University  
Baton Rouge, Louisiana, USA  
19

## ABSTRACT

20 Programming industrial robots is difficult and expensive. Although  
21 recent work has made substantial progress in making it accessible  
22 to a wider range of users, it is often limited to simple programs  
23 and its usability remains untested in practice. In this article, we  
24 introduce Duplo, a block-based programming environment that  
25 allows end-users to program two-armed robots and solve tasks that  
26 require coordination. Duplo positions the program for each arm  
27 side-by-side, using the spatial relationship between blocks from  
28 each program to represent parallelism in a way that end-users can  
29 easily understand. This design was proposed by previous work, but  
30 not implemented or evaluated in a realistic programming setting.  
31 We performed a randomized experiment with 52 participants that  
32 evaluated Duplo on a complex programming task that contained  
33 several sub-tasks. We compared Duplo with RobotStudio Online  
34 YuMi, a commercial solution, and found that Duplo allowed par-  
35 ticipants to solve the same task faster and with greater success.  
36 By analyzing the information collected during our user study, we  
37 further identified factors that explain this performance difference,  
38 as well as remaining barriers, such as debugging issues and diffi-  
39 culties in interacting with the robot. This work represents another  
40 step towards allowing a wider audience of non-professionals to  
41 program, which might enable the broader deployment of robotics.

## KEYWORDS

two-armed, robots, end-users, block-based, programming

## ACM Reference Format:

Felipe Fronchetti, Nico Ritschel, Logan Schorr, Chandler Barfield, Gabriella Chan, Rodrigo Spinola, Reid Holmes, and David Shepherd. 2023. Block-based Programming for Two-Armed Robots: A Comparative Study. In *Proceedings of 46th International Conference on Software Engineering (ICSE 2024)*. ACM, New York, NY, USA, 12 pages. <https://doi.org/XXXXXX.XXXXXXX>

50 Permission to make digital or hard copies of all or part of this work for personal or  
51 classroom use is granted without fee provided that copies are not made or distributed  
52 for profit or commercial advantage and that copies bear this notice and the full citation  
53 on the first page. Copyrights for components of this work owned by others than ACM  
54 must be honored. Abstracting with credit is permitted. To copy otherwise, or republish,  
55 to post on servers or to redistribute to lists, requires prior specific permission and/or a  
fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

56 ICSE 2024, April 2024, Lisbon, Portugal

57 © 2023 Association for Computing Machinery.  
58 ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00  
<https://doi.org/XXXXXX.XXXXXXX>

## 1 INTRODUCTION

The introduction of collaborative robots [13]—robots that can safely work hand-in-hand with humans—allows robots to be deployed in many new environments and dramatically increases the range of tasks robots can perform [31]. Unfortunately, even though this increases the theoretical utility of industrial robots, they remain difficult and expensive to program, hindering their practical adoption [27]. For traditional robot programming environments such as *ABB RobotStudio* [2] or *ROS Development Studio* [38], being easy to learn and use is not a priority, which means that they can intimidate new users and overwhelm novices. To use them effectively, engineers must be both competent general-purpose programmers and experts in the domain of robotics [34], a combination that is both rare and expensive. This has led to the creation of a cottage industry of robotics integration companies that buy, program, and re-program robots for companies.

There have been many attempts to make robot programming simpler and more accessible to non-experts, such as *end-users* who usually do not have formal programming education [11]. Such work has focused on tasks like moving robotic arms, picking up and carrying items, or tending machines. Programs for these simple tasks can sometimes be created without any traditional programming at all, using tools often branded as *no-code* environments (e.g., ScalableArc<sup>1</sup>, FuzzyStudio<sup>2</sup>). In contrast to creating traditional programs, users leverage techniques like demonstration-based learning [8], natural language teaching [37], or object recognition to train robots [7]. However, no-code approaches are inherently limited: replaying movements precludes branching behavior, modifying existing programs becomes tedious, and more complex tasks such as coordinating multiple robots are difficult or impossible.

To bridge the gap between traditional programming and no-code approaches, a number of *low-code* tools, which make programming accessible to users with little to no training, have been developed. Figure 1 shows *RobotStudio Online YuMi (ROY)*, a commercially-available low-code programming environment that targets two-armed robots. ROY builds on top of the widely-deployed expert robotics language *RAPID* [4], but instead of using low-level, built-in commands, it provides a higher level of abstraction via commands

<sup>1</sup><https://scalablerobotics.ai/>

<sup>2</sup><https://flr.io/products/fuzzy-studio/>

like Move or OpenHand. The ROY environment does not allow direct editing of program code as text, but instead uses a set of buttons (on the right of Figure 1) that trigger a graphical workflow to add and define new commands. ROY helps end-users by raising the programming abstraction level and guiding their edits, but inherits many of the drawbacks of both traditional and no-code tools. For example, while graphical workflows help with inserting new code, they make editing existing code complicated, and the environment provides little help with understanding advanced commands like MoveSync that target two robot arms at once.

Following the increasing interest in collaborative robots and the wide range of applications they can target, we explore how block-based programming can support end-users in developing code for collaborative two-armed robots. In this paper, we refer to end-users as individuals with little to no experience in robot programming. *Block-based programming* is a framework commonly used as a foundation for low-code systems, including the robotics domain [39]. It has become prevalent in computer science education, with millions of children learning to code in environments like Scratch and Alice [14, 22, 40].

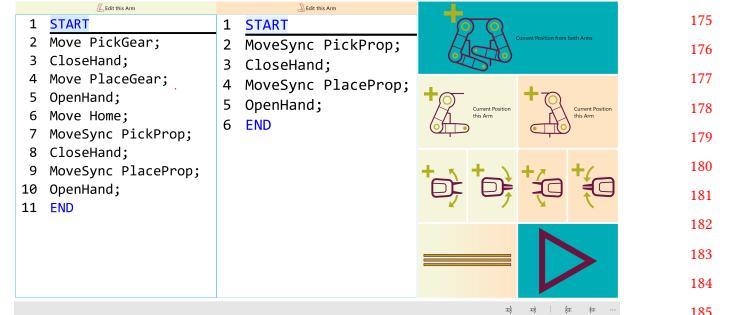
Unlike traditional programming tools, block-based programming environments do not use text, but instead visualize the structure of a program using jigsaw-like shapes that can be dragged, dropped, and connected. This visual metaphor allows users to predict how code elements interact with each other and makes it explicit whether code elements can be combined into an executable program [44]. Numerous studies have shown that block-based programming is an effective mechanism for introducing novices to programming [43].

There exist several tools that try to adapt the block-based framework to robotics, the most notable being *OpenRoberta* [18] and *CoBloX* [42]. However, these approaches do not target robots with more than one arm, largely due to the inherent complexity that arises from parallel programming. This makes them inherently limited, as common industrial tasks require two robot arms to cooperate to solve tasks, such as carrying large items or assembling machine parts [20]. Due to the lack of beginner-friendly environments focused on two-armed robots, our study proposes a practical investigation of how block-based environments can be used in the programming of two-armed robots by end-users.

To do so, this paper introduces and evaluates *Duplo*, an easy-to-use block-based programming system for *cooperative programs* that controls an industrial robot with two arms. As illustrated by Figure 2, Duplo presents users with two block-based programming canvases side-by-side and features cross-canvass blocks that are synchronized between the canvases to represent parallel commands. This design was explored by previous work through mock-up programs and a front-end prototype, but never implemented or tested with actual robots [32]. Our work implements the design in a refined form and evaluates the use of the block-based paradigm using a two-armed robot.

In our evaluation of Duplo, we compare it to RobotStudio Online YuMi [1]. ROY is the only existing robotics tool we are aware of that targets end-users and supports the programming of two-armed robots. In addition, ROY provides a similar set of features as Duplo and targets the same two-armed robot model, with equivalent robot commands being available in both languages. This makes ROY an ideal candidate for evaluating the impact that the block-based editing paradigm and synchronized blocks have on end-users programming two-armed robots.

To compare the two programming environments, we performed a randomized controlled experiment with 52 end-user participants. Our participants, primarily university students with little to no



**Figure 1: RobotStudio Online YuMi programming interface for two-armed robots: Two canvases show RAPID code for each arm side-by-side, with buttons allowing the quick insertion of commands.**



**Figure 2: Interface of the Duplo programming language.** Users can drag blocks from a toolbox on the left onto the programming canvases and attach them to existing blocks to create a program. Blocks that affect both arms are duplicated across canvases and vertically aligned.

robot programming experience, were randomly assigned to one of the two programming systems and given a brief introduction on how to use it. They were then presented with a programming task that consisted of multiple stages in which participants had to coordinate two robot arms to jointly carry and assemble a series of items. We recorded whether participants completed each stage of the programming task, how long they took to do so, the number and kind of programming mistakes they made, and how many attempts they needed to test their programs. We found that participants who used the Duplo block-based environment made fewer programming mistakes, were able to solve the given tasks with greater success, and required less time on average.

## 2 BACKGROUND

In this Section, we provide a brief overview of the two programming environments we compare in our experiment: the existing graphical-based system RobotStudio Online YuMi that is available commercially [1], and the block-based programming tool Duplo we introduce in this work.

175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232

233

## 2.1 RobotStudio Online YuMi

234

RobotStudio Online YuMi<sup>3</sup> (ROY) is a programming interface created by the robot manufacturer ABB to control their two-armed collaborative robot YuMi. It is designed to demonstrate the range of tasks that two robot arms can solve when they cooperate. ROY is developed to be beginner-friendly and accessible to end-users who would be overwhelmed by traditional tools like ABB's RobotStudio [2]. To the best of our knowledge, it is the only programming tool available in the market that is both beginner-friendly and supports the programming of a two-armed industrial robot.

243

As Figure 1 illustrates, ROY's main interface contains two canvases on the left where the RAPID code is displayed and a panel on the right with graphical buttons used to implement new robot instructions. The two canvases on the left contain the program code for each robotic arm that is targeted by the environment, and are used to visualize instructions created by the user. When the user presses a button on the right panel, the RAPID instruction assigned to that button is displayed on the canvas respective to that robot arm. The instructions displayed on the canvases are similar to the ones from professional tools of the same manufacturer, although boilerplate code such as function headers and variable definitions are hidden. As users compile and run their code (using the button on the bottom right), the two programs are combined with their respective boilerplate code and deployed onto their respective robot arms. The entire project can also be saved into a single file and restored later to continue editing the code in ROY.

257

The programming process in ROY is purely based on button interactions on the interface. The first button on the main interface, "Current position from both arms", generates code that simultaneously moves both arms to their current position. The second row has two buttons, "Current Position this arm" for each arm, that generate code to move only the corresponding arm. For these buttons, the target location is automatically set to a variable that contains the current location of the connected physical robot at the moment the button is clicked. This way of capturing locations by directly manipulating a physical robot is called *lead-through programming* and is commonly used in end-user robotics systems [8, 39]. The third row has four buttons, "Open Gripper" and "Close Gripper" for each arm, that generate code to open or close the grippers in its respective canvas. Lastly, the button on left side of the bottom row adds a command to both canvases that makes the program being executed in each arm wait for each other, for example when one arm is supposed to remain idle while the other one conducts work.

273

All of the described buttons can only be used to insert new code at the currently selected line number. However, users can also edit the program's source code of each arm using the "Edit this arm" button on top of each canvas. This button switches into a similar interface that only shows a specific arm's instructions. To edit existing commands, users can manually delete code and replace it with new instructions using buttons on the "Edit this arm" interface. Existing variables can also be overridden with a new location, allowing users to fine-tune their existing programs without having to edit the code manually. By providing a programming environment where beginners can implement programs for two-armed robots using simple button interactions, ABB defines ROY as a "*fast introduction to robot programming*" [1].

285

286

287

288

289

<sup>3</sup><https://www.youtube.com/watch?v=jEbaaqNPh9c>

290

## 2.2 Duplo: Block-based Cooperative Programming

291

292

Duplo is a block-based programming tool that supports cooperative two-armed robot programs. Duplo uses similar commands as existing beginner-friendly block-based programming languages for robots [45, 17, 42], which feature high-level commands such as "*Move arm <speed> to <position>*" and "*Open gripper*". However, unlike previous block-based languages, which focused on a single robot arm executing a single program, Duplo targets two robot arms at once and allows users to write programs that are executed simultaneously. It further integrates lead-through programming to define locations, similar to how positions are declared in ROY.

Figure 2 shows Duplo's user interface. Similar to ROY, it features two programming canvases side-by-side, each containing the program for one robot arm. On the left side of the environment, a sidebar provides a number of drawers with available programming blocks. The blocks are grouped thematically into movement commands, gripper commands, and synchronization commands. Blocks can be dragged from the drawers onto one of the two canvases and attached to existing blocks, as illustrated by their jigsaw shape.

A unique feature of Duplo compared to other block-based languages is the availability of blocks that target both robot arms at once, and that therefore exist in both of the programming canvases. The "*Wait for each other*" block synchronizes the state of both arms before proceeding to the next instruction, and the "*Move arm <speed> to <position> and follow on the other side*" block is used to perform a simultaneous movement of both arms at once. When a programmer drags one of these blocks onto one of the programming canvases, it is automatically inserted into the other canvas as well. The blocks can be edited and moved to a different point in the program by dragging either of the two representations, with the other representation following along accordingly. The environment ensures that complementary blocks are always vertically aligned, making it easy to identify how they correspond to each other. This allows users to visually track the timing of the two programs as they edit them, and signals them to potentially add more synchronization blocks to ensure the correct sequence of commands.

The design of Duplo is based on the findings of previous work [32], which evaluated different design alternatives for coordinating two-armed robots. The Duplo environment follows the design approach that was deemed best by that work, using explicit synchronization blocks and vertical alignment to represent concurrent robot behavior. However, unlike previous work, which used mock-ups and front-end prototypes to compare design alternatives, Duplo features a fully-functional implementation that targets a real two-armed robot.

This implementation allows us to compare Duplo to ROY, which features a similar complexity of programming features and targets the same robot model. Even though both languages adopt distinct programming styles (*block-based* vs. *graphical-based*), the similarities between Duplo and ROY make them suitable for comparison. Both are among the few tools available for two-armed robot programming. They are specifically designed to be user-friendly for individuals without experience in robot programming and offer comparable movement and synchronization commands. The industrial robot and the *lead-through capability* used in both languages are also the same. In other words, equivalent solutions can be implemented in both languages. More technical details on how we

349 implemented Duplo and the data collected from our case study are  
 350 available in our replication package<sup>4</sup>.  
 351

### 352 3 RELATED WORK

353 Our work on Duplo is inspired by previous attempts to make robot  
 354 programming accessible to end-users. We have already explained  
 355 the most directly related tool, RobotStudio Online YuMi, in Sec-  
 356 tion 2.1. In this section, we situate Duplo and ROY in the context of  
 357 other relevant work. We focus on end-user approaches that either  
 358 target the domain of robotics or that use block-based program-  
 359 ming.  
 360

#### 361 End-user Robotics Tools

362 There exists a large corpus of previous work that has tried to make  
 363 robot programming more accessible to end-users. Most studies  
 364 focus on implementing new programming systems to support end-  
 365 users, and we will discuss them in this section [6, 9]. We use the  
 366 categories of Biggs and MacDonald’s survey of programming sys-  
 367 tems from 2003 to categorize them, as their categories still apply  
 368 to today’s systems [11]. This survey categorized robot program-  
 369 ming systems based on their approach: *manual* or *automatic*. Man-  
 370 ual tools use custom domain-specific programming languages like  
 371 *RAPID* [4] to support robot programming. Automatic tools strive  
 372 to eliminate the need for traditional programming by using alter-  
 373 native modalities, such as *lead-through programming* [27] or  
 374 *demonstration-based learning* [8].  
 375

376 Both manual and automatic tools have an extensive range of  
 377 target audiences in end-user robot programming, and their ap-  
 378 proaches to making robot programming easier are diverse. The  
 379 manual systems *Open Roberta* [18] and *BEESM* [35], for example,  
 380 implement block-based languages to support end-users with no  
 381 programming experience in robotics. Other manual systems, such  
 382 as the *RC+ Express* [16] and *Polyscope* [29], propose more complex  
 383 programming environments and target users with prior experience  
 384 in robot programming. In automatic systems, the alternatives are  
 385 also diverse. To replace traditional programming languages, studies  
 386 have explored other technologies to support the programming of  
 387 robots. Mixed and virtual reality environments have been created as  
 388 an alternative to more conventional approaches [26, 21, 46, 12]. Pro-  
 389 cedural languages have also been replaced with other programming  
 390 tools based on familiar robotics concepts, such as *path planning* [30,  
 391 28].  
 392

393 The primary influence for our work was *CoBlox*, a beginner-  
 394 friendly programming environment for collaborative robot arms [39].  
 395 CoBlox combines manual and automated robot programming tech-  
 396 niques: Users can use a block-based programming environment to  
 397 define the overall structure of their programs and then use lead-  
 398 through programming to define the target position for each arm  
 399 movement. An evaluation by the authors of CoBlox found that  
 400 novice programmers can learn how to use CoBlox faster and solve  
 401 simple programming tasks more effectively than with other com-  
 402 mercially available tools [42]. CoBlox is easy to learn but also limits  
 403 the complexity of the programs it supports. All programs are linear  
 404 sequences of commands that move a single robot arm. CoBlox can-  
 405 not be used to program any tasks where multiple robot arms need  
 406 to interact.  
 407

408 Both manual and automated systems typically have to compro-  
 409 mise between beginner-friendliness and the maximum task com-  
 410 plexity they can support. CoBlox, the main inspiration for Duplo,  
 411 combines the qualities of both types of systems but is limited to  
 412

<sup>4</sup><https://zenodo.org/record/8133187>

413 one-armed robots. The study presented in this work aims to ex-  
 414 pand the capabilities of CoBlox to two-armed robots to evaluate  
 415 how end-users perceive block-based programming systems on real  
 416 two-armed robots.  
 417

### 418 Block-based Programming

419 Block-based programming is a visual alternative for text-based lan-  
 420 guages [24]. In this visual approach, programmers combine visual  
 421 jigsaw-styled blocks representing programming functionalities to  
 422 implement their solutions. These blocks are usually organized and  
 423 colored by type (e.g., operators, data, control) and can be dragged  
 424 and dropped into a canvas where the program is visually repre-  
 425 sented. Block-based languages are widely used in computer science  
 426 education and other fields related to end-user programming. In  
 427 robotics, block-based programming is also used as one of the com-  
 428 mon end-user programming alternatives to traditional languages.  
 429 Blocks in the robotics domain usually represent robot instructions  
 430 translated to the robot controller by a back-end application.  
 431

432 Among the block-based solutions that inspired our work are  
 433 educational tools such as *Scratch* [22] and *Snap!* [45], as well as the  
 434 aforementioned robotic solution CoBlox [39]. Regarding two-armed  
 435 robot programming, we focused our inspiration on a previous work  
 436 that explored different layouts for block-based languages in this  
 437 context [32]. Their best-performing design solution was adapted  
 438 to our application and used as the primary influence for Duplo.  
 439 Although many studies already exist on block-based programming,  
 440 none has yet explored the programming of two-armed robots on  
 441 physical robots in practice. We believe that our study may provide  
 442 insight into end-user programming with block-based languages on  
 443 real-world robots.  
 444

## 445 4 METHOD

446 In this section, we describe the controlled experiment we conducted  
 447 to compare Duplo and RobotStudio Online YuMi using a two-armed  
 448 collaborative robot. The goal of our experiment was to evaluate how  
 449 end-users solve a complex task using a two-armed robot using either  
 450 ROY or Duplo. Figure 3 presents an overview of the experimental  
 451 procedure and the individual steps that were split across three  
 452 experimental phases. The entire methodology was refined through  
 453 a pilot study using feedback from 31 individuals and approved in  
 454 advance by an institutional review board.  
 455

### 456 4.1 Recruitment

457 We advertised our experiment to undergraduate students enrolled  
 458 at a single university in the United States. To ensure a diverse  
 459 range of participant backgrounds, we sent advertisement emails  
 460 to different departments and distributed flyers to students around  
 461 the campus. We advertised that the experiment involved robot  
 462 programming, but emphasized that it was focused on end-users  
 463 without any previous programming experience. A 50 USD gift card  
 464 for a local bookstore was offered as an incentive to all participants.  
 465

### 466 4.2 Experimental Setup

467 Each participant was provided with one of the two programming  
 468 environments and a robot to execute their code and record positions  
 469 via lead-through programming. The robot used in the experiment  
 470 was an ABB YuMi (IRB 14000) [3], a collaborative robot with lead-  
 471 through capabilities. Figure 4 shows the physical layout of the  
 472 experiment as it was presented to participants. A laptop running  
 473 the assigned programming environment was placed on a table adja-  
 474 cent to the workstation where the two-armed robot was mounted.  
 475

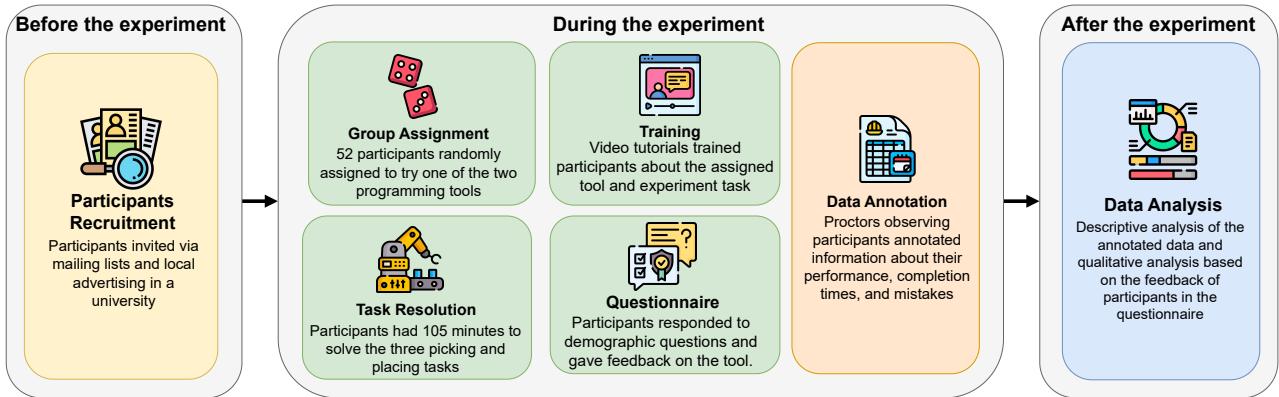


Figure 3: Experimental design and procedure divided into three phases: before, during, and after the experiment.



Figure 4: Experimental setup: Participants used a touch-screen-enabled laptop (left) to program a two-armed collaborative robot (right).

Participants were able to program using either the laptop's touch-screen monitor or the built-in keyboard and touch-pad. In addition to the robot, the workstation contained 3D printed objects relevant to the task participants were asked to solve.

### 4.3 Experimental Procedure

Participants were randomly assigned to one of two groups after they consented to join the experiment. One group was assigned the Duplo environment, and the other group used the ROY environment. Although participants were divided into two groups, each participant was scheduled for an individual session to try the experimental procedure. The only individual rather than the participant in a session was a proctor, who was instructed not to provide extra information to the participants.

Other than the assigned programming environment and respective training, both groups were provided with the same setup and task descriptions. The experiment was limited to a maximum of 120 minutes. During the first 15 minutes of the experiment, participants were introduced to their assigned programming environment and trained on how to use it. Next, participants received the task they were to solve along with a clear understanding of what would constitute successfully accomplishing the task. During the following 105 minutes, participants were allowed to solve the given task at their own pace.

4.3.1 *Training Procedure.* The training for both environments was designed to be as similar as possible consisting of two brief videos. The first video explained how to use the assigned programming environment. Although the two environments required different instructions and consequently different videos, we made them as similar as possible in both length and content. Each video was approximately 7 minutes long and covered all the basic programming features of the respective environment. Neither video referenced the concrete task that participants were expected to solve.

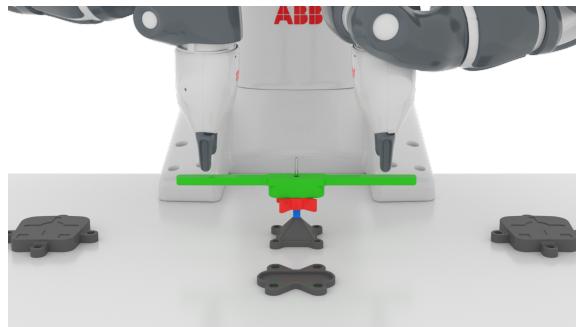
The second video was three minutes long and introduced the task to the participants. This video introduced them to the 3D-printed objects they were to assemble. It also showed them the desired, fully assembled state of the components after all steps of the task were completed. The video did not show the assembling process, as we expected participants to implement their own solutions. It did however suggest the order in which participants could tackle the assembly steps, effectively breaking the task down into three smaller sub-tasks that allowed participants to keep track of their progress.

After watching both training videos, the proctor supervising the experiment gave participants a brief in-person introduction to the robot workspace. The proctor demonstrated how to execute programs and how to move the robot arm in lead-through mode to capture its current position. Participants were also given a "cheat sheet" with some reminders and tips, including how to open and close the robot's grippers while recording positions. The training videos are also available in our replication package<sup>5</sup>.

4.3.2 *Task Procedure.* The experiment task involved writing a program that could perform a series of steps that involved picking and placing 3D-printed objects and executing this program on the physical robot to assemble the final object. Pick-and-place tasks are commonly used in robot experiments because they often occur in practice and typically lead to challenges that participants encounter and have to overcome [10]. In our experiment, participants were asked to assemble three components using the two-armed robot.

First, participants had to pick up a "spacer", a small cuboid-shaped plastic component, and place it onto a narrow metal shaft in the center of the robot workstation. This part of the task only involved one robot arm and was therefore suitable as a warm-up step. However, only one of the two robot arms could reach the

<sup>5</sup><https://zenodo.org/record/8133187>



**Figure 5: Potential solution:** The spacer at the bottom (in blue), the gear in the middle (in red) and the propeller at the top (in green), all placed into the small metal shaft. There is no specific order of how the objects had to be placed.

item, so this step also served as a check whether participants could identify and program the correct robot arm for the sub-task.

Second, participants were instructed to pick up a “gear”, a larger and differently shaped component, and move it on top of the previously placed spacer. To solve this programming step, participants had to use the robot’s second arm as only that arm was able to reach the item. While this sub-task seems similar to the previous one (requiring only one robot arm), it did involve coordination because the two tasks had to be executed in the right order. This and the previous sub-task could be parallelized by picking up the two items simultaneously and then placing them one after another, but we did not instruct participants to solve them this way. However, a naive implementation is likely to cause a concurrent execution of both tasks in both programming environments. If participants decided to retain the parallelism, they had to ensure the two arms do not collide, which can occur if they try to place objects simultaneously or do not move out of each others’ way.

For the last sub-task, we asked participants to pick-and-place a “propeller”, a wide plastic object that could not be carried and assembled accurately using a single robot arm. Instead, participants had to carry the item with both robot arms and use synchronized movements to move it into the correct position on top of the previously placed items. Unlike in prior sub-tasks, this required the two arms to work in tandem. This also required synchronization to ensure it was only executed after both previous sub-tasks were completed.

In Figure 5 and in this video example<sup>6</sup>, we show a potential solution for the assembly task, although in the experiment, the specific sequence in which objects had to be placed was not specified or enforced. For example, while Figure 5 shows the order we used above (spacer first, gear second, and propeller third), participants were allowed to start by placing the propeller and then placing the gear or spacer. This means that participants could move on to a different sub-task if they felt like they were stuck. The task description video did, however, show the assembly in the order as shown in Figure 5, as we expected this to be the order with the most appropriate difficulty curve for participants.

Participants were free to spend the 105 minutes provided for the experiment as they wished. In particular, we did not direct them to move on to a new sub-task if they spent longer than anticipated on a

Type	Variables	
Integer	# Program Executions, # Robot Positions Created, # < Blocks, Lines > Implemented, # Objects Dropped # Robot Collisions with < Environment, Robot >	639 640 641 642 643 644
Datetime	Participant started the experiment, Participant < picked / placed > the < spacer / gear / propeller >	645 646 647 648 649 650 651 652 653 654 655 656 657 658

**Table 1: Variables annotated by the proctor during the experiment. Each variable corresponds to a different column in the spreadsheet. Similar or equivalent variables are grouped using < symbols > in the table.**

single step. The proctor would only intervene if participants explicitly requested them to repeat previously given instructions or the task description, or if there were technical issues. After participants indicated that had finished the task, the proctor would ask them to run their program one final time to verify the solution. If the program did not solve the task and there was time left, participants could resume programming and try to fix their mistakes.

**4.3.3 Post-Experiment Questionnaire.** After completing the experiment, participants were asked to answer a post-experiment questionnaire. The questionnaire was composed of eight questions. The first five questions asked participants about their demographic information; this included participant age, area of study, overall programming experience, experience with robot programming, and whether they had ever used a block-based programming language before. The last three questions were open-ended and asked what participants found easy, what they found difficult, and if they had any other feedback about the experiment or the environment they used.

#### 4.4 Data Collection and Analysis

A proctor supervised the experiment at all times and collected data in a spreadsheet for later analysis. Using a digital clock, the proctor recorded the duration the participant took to complete each sub-task in the experiment, from picking up their first object to placing the last one. They also counted occurrences of particular events in the experiment, including the number of times a participant executed their program solution, the number of objects accidentally dropped during executions, and the number of times the robot collided with itself or the surrounding workspace. Once a participant had successfully finished the task, the proctor inspected the code and collected information about the participant’s solution, including the number of used lines or blocks of code used and the number of robot positions the participant defined in their solution. All the collected variables are defined in Table 1.

We performed a range of analyses on the data collected by the proctors and provided in the post-experiment questionnaire. While most experimental data was quantitative, the written responses to the questionnaire required qualitative analysis that was performed by three researchers who used open card sorting [36] to organize and categorize responses. The researchers were instructed to create codes for participant comments that described features or attributes they found easy and difficult to use. At first, each researcher performed the analysis individually and met to compare their results to arrive at a final, common set of codes. Constant comparison was employed to guarantee consistency in the codes [15]. Finally, two additional researchers inspected the final set of codes to ensure they were easy-to-understand.

<sup>6</sup><https://zenodo.org/record/7781859>

697

## 5 RESULTS

698

In this section, we present the results of our experiment. We begin with an introduction of the participants' demographics and then analyze the performance data we collected throughout the experiment. Finally, we present the feedback participants provided through the post-experiment questionnaire.

703

704

705

### 5.1 Demographics

707

A total of 52 participants joined and completed the experiment. Participants were randomly assigned into one of two groups of 26 participants, with one group using Duplo and the other using ROY for the experimental task. The participants indicated that they pursued 31 distinct majors. Some were from computing-related domains such as Electrical Engineering (4 participants) and Computer Science (4 participants), but the vast majority was from other areas of study, such as Biology (6 participants), Cinema (3 participants), and Nursing (2 participants). The average participant age was 22 years (min: 17, max: 50, sd.: 6.61).

716

When asked about their programming experience, 10 participants using Duplo (38%) and 11 participants using ROY (42%) had no prior programming experience. The remaining participants declared some level of programming experience, with 9 participants of each group (34%) indicating one or more years of experience in programming. For Duplo, 12 participants (46%) indicated having at least some experience with block-based languages, while only 5 participants testing ROY (19%) indicated the same. Only 2 participants testing Duplo (8%) and 1 participant testing ROY (4%) indicated at least some robot programming experience.

725

726

727

### 5.2 Participant Performance

728

During the experiment, the proctor recorded the times and outcomes at which participants completed the sub-tasks and the overall task. Figure 6 shows the success rates for each sub-task in the experiment, split by their assigned programming environment. Note that, as explained in Section 4.3.2, sub-tasks were not strictly sequential, so participants might have completed later sub-tasks despite missing previous ones.

735

For Duplo, 21 out of 26 participants (80%) completed all sub-tasks successfully. All participants in the Duplo group successfully completed the first sub-task placing the spacer on the rod. One participant failed placing the gear for the second sub-task. 4 participants could not pick up or place the propeller in the third sub-task.

740

For the ROY alternative, only 12 out of 26 participants completed all six sub-tasks (46%). 3 participants failed to pick up the spacer during the first sub-task, and 2 more were unable to place it correctly. For the second sub-task, 4 participants did not succeed at picking up the gear and 3 more failed to place it. For the third sub-task, 11 participants failed to pick up the propeller and 3 more were unable to place it.

746

We employed the Chi-squared test of independence [47] to examine the potential association between task completion and the specific tasks assigned. We rejected the hypothesis that these variables were independent for p-values < 0.05. Our analysis yielded significant results, with p-values below the threshold of 0.05. Specifically, for participants testing Duplo, the p-value was 0.016, and for those using ROY, the p-value was 0.003. These findings indicate that the success of participants in the experiment was influenced by the task they were assigned to perform.

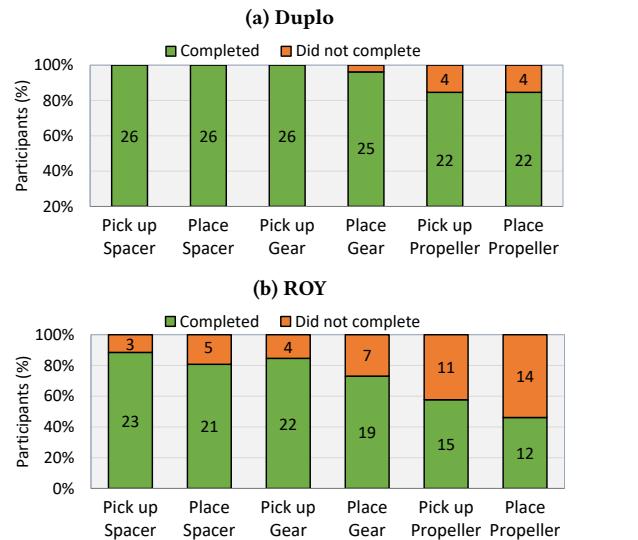


Figure 6: Percentage of participants who completed each sub-task in a programming option. The bar labels provide the exact number of participants per sub-task.

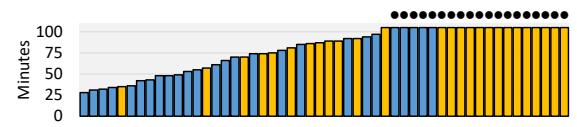


Figure 7: Participants' completion times (in minutes) in ascending order. Each bar represents one participant, and its color indicates the assigned environment (blue for Duplo, yellow for ROY). A black circle marks participants who didn't complete all sub-tasks.

### 5.3 Completion Times

During the experiment, the proctors also recorded participants' completion times for each sub-task and for overall task. Figure 7 shows the time that participants took to complete the whole experiment, with bar colors indicating each participant's assigned environment. Participants who ran out of time and did not complete all sub-tasks are marked with a black circle. As the Figure illustrates, participants who used Duplo were more successful and faster overall. This observation also holds when only considering participants that successfully completed all sub-tasks. Those who completed all the sub-tasks using Duplo took 52.7 minutes on average to finish the experiment (min: 28, max: 97, sd: 22.06), and those who used ROY took 74.6 minutes (min: 35, max: 105, sd: 18.33). Table 2 shows the average completion times for each sub-task for those participants, divided by their assigned group. Using the completion time of participants, we performed a survival analysis on the success of participants as they solved the task. Note that survival here means that participants have not yet completed a task at a given point in time. We found that there is a statistically significant difference in success between the two groups ( $df = 1, \chi^2 = 9.59, p < 0.005$ ).

755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812

	Pick up Spacer	Place Spacer	Pick up Gear	Place Gear	Pick up Prop.	Place Prop.	Total
ROY	18.70	14.05	22.68	15.26	30.67	21.25	74.58
Duplo	8.15	5.45	9.27	11.24	14.64	14.50	52.67

Table 2: Average completion time for each sub-task (in minutes). Only participants who completed the task were considered. The total average only considers participants who completed all sub-tasks.

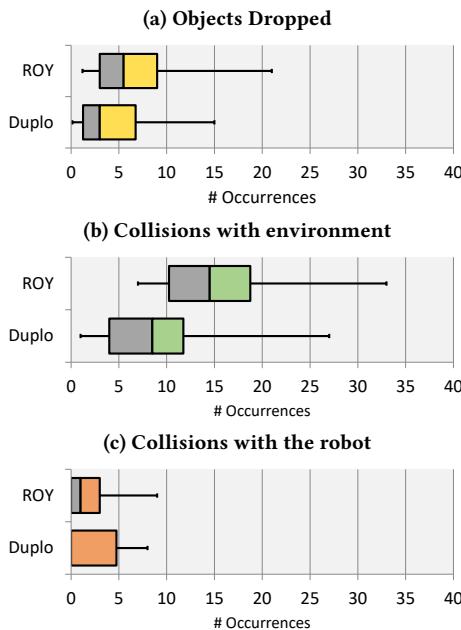


Figure 8: Box-plots of occurrence numbers of the top 3 programming obstacles.

#### 5.4 Programming Obstacles

In addition to participant performance, we also investigated the errors participants made. Three errors ended up being the most common: dropping the object held by the robot in the wrong position, collisions of the robot with its surrounding workspace, and collisions of the robot with itself. Note that unlike the other two errors, self-collisions were detected and automatically prevented by the robot's controller. This provided quicker feedback to participants and prevented damage to the robot hardware.

Figure 8 shows how often participants encountered the top three programming obstacles when using each programming method. On average, participants using Duplo dropped blocks 4.7 times during the experiment (min: 0, max: 15, sd: 4.26), while participants using ROY dropped blocks 6.9 times (min: 1, max: 21, sd: 5.22). For workspace collisions, the average number of occurrences was also higher for ROY, with an average of 15.1 workspace collisions per participant (min: 7, max: 33, sd: 6.32) compared to an average of 10.7 collisions (min: 1, max: 27, sd: 8.84) from participants using Duplo. The number of collisions prevented by the robot controller is closer for both languages: ROY users encountered an average of 2.0 prevented collisions (min: 0, max: 9, sd: 2.54), and Duplo users encountered 2.3 of them (min: 0, max: 8, sd: 3.09).

To determine if there were significant differences between the obstacles occurrences in both groups, we conducted the Mann-Whitney-Wilcoxon Test [25]. The hypothesis that the occurrences were identical for both groups was rejected for p-values < 0.05. We found a significant difference in the number of workspace collisions (p-value: 0.008). However, the number of objects dropped (p-value: 0.067), and predicted collisions (p-value: 0.684) did not reach the threshold for statistical significance. These results suggest that although two out of three obstacles were less frequent for Duplo participants, we cannot confirm a statistically significant difference in occurrence values.

#### 5.5 Program Analysis

To gain further insight into the participants' programming experience, we also analyzed their final code and the number of times they executed code while programming. Both participant groups used approximately the same number of test executions during the study: the Duplo group ran their code 37.4 times on average (min: 9, max: 85, sd: 18.59) compared to the ROY group which ran theirs 37.1 times (min: 19, max: 64, sd: 12.27). Duplo users required an average of 33 blocks to write their final solution for the entire task (min: 22, max: 44, sd: 6.42), while participants using ROY wrote an average of 45 lines of text-based code (min: 11, max: 68, sd: 36.61). Note that these numbers include incomplete programs from unsuccessful participants, but do not include empty lines in ROY. We further found that participants using ROY defined 24 positions (min: 8, max: 39, sd: 18.5) on average using lead-through programming, compared to 16 on average for Duplo (min: 8, max: 29, sd: 4.8).

#### 5.6 Feedback from Participants

We gave participants the opportunity to comment on the experiment and their assigned programming environment after they completed the task. In our analysis, we only considered those comments who were in some way related to the programming interfaces, and ignored those that provided feedback on the robot hardware, the training or the experimental task.

ROY	
Difficult to edit code.	12
Difficult to debug problems.	5
Impossible to execute specific chunks of code.	3
MoveSync command is counter-intuitive.	2
Duplo	
Arm synchronization features are confusing.	6
Difficult to reteach positions.	3
Difficult to debug problems.	3

Table 3: Programming challenges mentioned by participants in the post-experiment questionnaire.

Table 3, presents an overview of the codes we developed during our analysis that described programming challenges and their prevalence in the provided feedback. For ROY, the biggest challenge was editing code, with 12 participants mentioning this problem. Among their comments, participants highlight that it was not possible to reorganize lines of code without deleting them, that they were unable to rename robot positions, that it was not possible to edit both arms at the same time, and there was no option to undo changes. A further 5 participants also mentioned difficulties

929 debugging code, primarily because ROY does not always provide  
930 clear feedback when errors occur, which makes it difficult to locate  
931 issues. Another 3 participants complained about not being able to  
932 execute subsections of code (which could be useful for debugging),  
933 and 2 mentioned how the *MoveSync* command, used to move both  
934 arms at the same time, was confusing to them.

935 For Duplo, participants mostly commented that using arm syn-  
936 chronization was confusing to them, including how to program  
937 both arms to execute different commands at the same time. They  
938 also mentioned issues understanding the “wait for each other” syn-  
939 chronization block. Another 3 participants also mentioned issues  
940 with the position reteaching feature as it is hidden in a drop-down  
941 menu and it was not mentioned in the training tutorial. Lastly, 3  
942 more participants mentioned problems with debugging their code,  
943 which were similar in nature to those encountered in ROY.

944 Participants also provided us with comments on what aspects  
945 of the two systems they found particularly useful or easy to use.  
946 These comments were mostly similar for both systems. For both  
947 interfaces, participants mentioned that they were simple and more  
948 intuitive than writing code manually. Also for both interfaces, par-  
949 ticipants pointed out that defining positions using lead-through  
950 programming was intuitive to them. Some participants also noted  
951 that they liked the commands for synchronized movement (the  
952 *MoveSync* command in ROY and the “follow other arm” block in  
953 Duplo, respectively). The only repeated comment we identified that  
954 was specific to one environment was that 3 participants found it  
955 helpful that they could review the definition of robot positions in  
956 Duplo.

## 956 6 DISCUSSION

957 In this section, we discuss our findings and how different factors  
958 may have influenced participants throughout the experiment.

### 960 6.1 How Do the Programming Environments 961 Affect End-user Performance?

962 Our findings in Section 5.2 indicate that the programming envi-  
963 ronment that we assigned to our participants made a substantial  
964 difference in how well they solved the given programming task. As  
965 shown in Figure 6, while less than half of the participants (46%)  
966 using ROY completed the assigned task in the given, nearly twice  
967 as many (80%) Duplo participants succeeded. Participants testing  
968 Duplo not only solved the task more effectively but did so faster.  
969 As Table 2 indicates, participants using Duplo spent less than half  
970 the time required by participants using ROY on almost all of the  
971 sub-tasks. Our performance observations match our expectations  
972 as we set out for Duplo to improve upon ROY’s usability. However,  
973 performance numbers alone cannot account for which of Duplo’s  
974 features set it apart from ROY. To do this, we refer to the feedback  
975 given by participants.

976 As indicated in Table 3, 12 of the 26 participants using ROY (46%)  
977 complained about the difficulty of editing code while using the  
978 interface. This feedback matches our own observations, as ROY’s  
979 graphical elements are only integrated with text-based editing on  
980 a superficial level. Although it is easy to insert new code in ROY,  
981 to edit an existing line of code, users have to either delete and re-  
982 write their code or access a secondary interface that allows them  
983 to re-define positions. It was also not straight-forward for users to  
984 move lines of code (for example to swap their order). In ROY, users  
985 have to manually copy and paste code similar to text-based editing,  
986 which can easily interfere with the alignment of instructions and  
987 the synchronization between the two robot arms. This introduces a

988 potential source of errors or confusion. Conversely, in Duplo users  
989 can drag and drop blocks within the canvas as desired, automatically  
990 updating the surrounding code’s alignment.

991 *Participant I (ROY/Difficult): “I couldn’t move lines of code  
992 after placing them, so I had to delete them and remake them in the  
993 correct line.”*

994 *Participant II (ROY/Difficult): “The fact that you could not move  
995 lines up and down the sequence in the code was frustrating because  
996 errors could not easily be fixed...”*

997 *Participant III (Duplo/Easy): “It was very easy to use the blocks  
998 to ask the robot to make actions, link the blocks together, and break  
999 them apart. It was also easy to move the robot’s arms.”*

1000 Another indicator which features affected our participants’ per-  
1001 formance are the second and third most frequent comment high-  
1002 lighted by ROY participants in Table 3: the difficulty to debug prob-  
1003 lems, and being unable to execute specific chunks of code. This  
1004 feature is particularly important for a use case such as robot pro-  
1005 gramming, where commands can take several seconds to execute,  
1006 making it tedious to repeatedly re-execute long sequence of code.  
1007 Although we did receive a similar comment from a Duplo partici-  
1008 pant, the block-based environment made it easier for users to test  
1009 partial programs because it allowed them to detach code from the  
1010 main program. This feature, while conceptually similar to comment-  
1011 ing out code in a text-based system like ROY, ensures that even  
1012 temporarily unused code remains valid and is not accidentally for-  
1013 gotten. ROY, being a fundamentally text-based environment, only  
1014 had regular comments available, which were also not graphically  
1015 represented in the user interface. This might be responsible for ROY  
1016 users spending more time solving the given tasks.

1017 Another debugging feature that both environments provided is  
1018 highlighting the currently executed line of code when running a  
1019 program. This feature is found in many end-user languages, but for  
1020 the specific use case of Duplo and ROY, it can become confusing for  
1021 users since there are two separate executions (for the left arm and  
1022 the right arm) that take place simultaneously. In Duplo, the vertical  
1023 alignment of blocks guarantees that synchronized blocks that get  
1024 executed, such as synchronized movements, are always in a single  
1025 line. In ROY however, this is not necessarily the case, which creates  
1026 additional mental effort for users. We speculate that this additional  
1027 mental strain might have been a factor why some ROY users found  
1028 the command for synchronized movements unintuitive.

1029 *Participant IV (ROY/Difficult): “It took me longer than it would  
1030 have otherwise taken me because I could not start in the middle  
1031 of my program...”*

1032 *Participant V (ROY/Difficult): “Setting the robot back to a  
1033 designated position required running the program from the start  
1034 and pausing before it began a new cycle.”*

1035 *Participant VI (Duplo/Difficult): “Debugging, it was extremely  
1036 difficult to control the robot outside of a program, if I wanted to  
1037 open the arms so that I could replace a piece before another cycle, I  
1038 had to start the program and stop it before it got too far.”*

1039 Our previously discussed findings align with studies where block-  
1040 based languages and other end-user robot programming tools are  
1041 evaluated [39, 41, 23]. In a study where participants had to program  
1042 a pick-and-place task using Polyscope, researchers pointed out de-  
1043 sign recommendations for new end-user programming interfaces  
1044 [5]. According to the authors, “interfaces should minimize use of  
1045 tabs and keep similar actions and commands coherently grouped to-  
1046 gether...”. They also emphasized that “...end-user robot programming

1045 *interfaces should have easy-to-use replay capabilities to visualize*  
 1046 *contextualized portions of the robot program...".* The ROY interface  
 1047 does not implement either of these features. One participant also  
 1048 highlighted the lack of options to undo commands in ROY, which  
 1049 is another recommendation proposed by the prior study.

1050  
 1051 **Summary 6.1:** By representing robot commands as puzzle  
 1052 pieces, block-based programming contributed to the ability  
 1053 of end-users to insert, edit and debug code. The freedom  
 1054 to reorganize blocks using the puzzle metaphor, and the  
 1055 alignment of instructions on vertical columns, also allowed  
 1056 Duplo participants to complete more tasks in less time. For  
 1057 future work, other common features should be included,  
 1058 such as the ability for users to undo commands.

## 6.2 What Learning Barriers Do End-users Face?

1060 Our participants only had a short amount of time to learn how to  
 1061 use either environment that they were assigned. Similar time con-  
 1062 straints are not unusual when industrial workers have to learn jobs  
 1063 on-task, and previous studies found that end-users can overcome  
 1064 learning challenges quickly as they get hands-on experience with a  
 1065 reasonably end-user friendly system [33]. Nonetheless, identifying  
 1066 and addressing potential learning barriers can substantially improve  
 1067 how quick end-users become familiar with a new system. In a study  
 1068 about learning barriers in end-user programming systems [19],  
 1069 researchers identified six categories of challenges end-users face  
 1070 while solving tasks in a programming environment: design barriers  
 1071 (*what to do?*), selection barriers (*what to use?*), use barriers (*how*  
 1072 *to use?*), coordination barriers (*how to combine different things?*),  
 1073 understanding barriers (*what is wrong?*), and information barriers  
 1074 (*how to check what is wrong?*). The difficulties highlighted by par-  
 1075 ticipants in our post-experiment questionnaire show that neither of  
 1076 the evaluated systems is free of those barriers, although end-users  
 1077 encounter them in different situations.

1078 **Understanding and information barriers.** Programming an  
 1079 industrial robot involves the understanding of both virtual (soft-  
 1080 ware) and physical (mechanical) concepts. For example, to teach a  
 1081 robot a new position, users have to use lead-through to manually  
 1082 move the robot to a new physical location, and use the program-  
 1083 ming environment to record the position. If a defect occurs in their  
 1084 code, they must determine whether the problem is in the program-  
 1085 ming logic or the physical workspace. In some cases, the logic  
 1086 behind the code may be correct, but the physical locations taught  
 1087 to the robot may still produce errors (e.g., collisions, robot singu-  
 1088 larities). In our experiment, some users struggled to identify what  
 1089 was wrong with their implementation and reported it as a difficulty  
 in the questionnaire.

1090 **Participant VII (ROY/Difficult):** "...I also did not like that it did  
 1091 not always show me which movement code line had a problem if it  
 1092 was after a movement error..."

1093 **Participant VIII (ROY/Difficult):** "...I didn't know why there  
 1094 were errors sometimes when it looked like it worked..."

1095 **Participant IX (Duplo/Difficult):** "Figuring out what I did  
 1096 wrong."

1097 We believe that lead-through programming can make defining po-  
 1098 sitions substantially easier, and participants have echoed that senti-  
 1099 ment in their feedback. However, there might be room to provide  
 1100 more guidance or training for lead-through, for example in form of  
 1101 visual aids or immediate feedback during the programming process.

1102 **Selection and use barriers.** Some participants also reported  
 1103 difficulties understanding certain features of both programming  
 1104 systems. In particular, both systems involved features to program  
 1105 two arms simultaneously, and some users commented that the  
 1106 provided synchronization commands (i.e., MoveSync in ROY, "wait  
 1107 for each other" in Duplo) were unintuitive. While we primarily  
 1108 received this feedback from ROY users, we observed that some  
 1109 participants were not aware of a feature in Duplo that allows them  
 1110 to access and review an already defined location. This feature, which  
 1111 other participants explicitly named as a useful tool to understand  
 1112 their programs, could have been represented more prominently  
 1113 in the system to make users aware of its existence, and explained  
 1114 better to convey its usefulness.

1115 **Participant (ROY/Difficult):** "...I wasted a lot of time deleting  
 1116 sync steps before I understood how to combine one-sided steps with  
 1117 sync steps... It would also be nice to tell the robot, "move to the  
 1118 locations I told you in Step 10 and then stop". It's unclear to me  
 1119 whether that's possible to do."

1120 **Participant (Duplo/Difficult):** "... The first (difficulty) was think-  
 1121 ing the "Wait for each other" blocks could be placed anywhere  
 1122 instead of only in line with each other. My solution was instead of  
 1123 placing those blocks, I slowed down the movement speed of one of  
 1124 the arms (The block did come in handy later)..."

1125 **Participant (Duplo/Difficult):** "At first, I didn't know which arm  
 1126 was the right or left. Also, I didn't know how exactly to reteach a  
 1127 position, but after a while, I got it."

1128 **Summary 6.2:** Because Duplo eliminated many of the  
 1129 programming challenges for end-users, it enabled the iden-  
 1130 tification of second-order problems. Primary among these  
 1131 was physical positioning and mapping. It became clear that  
 1132 end-users had trouble mapping between position names  
 1133 (e.g., "AboveGround") and physical positions in 3D space.

## 7 LIMITATIONS

1134 In this section, we discuss some of the limitations of our study:

1135 **Number and background of participants.** We conducted our  
 1136 study on 52 participants, which required a substantial effort to re-  
 1137 cruit, considering the time and effort that participants had to invest  
 1138 in the in-person experiment. We also aimed to recruit students from  
 1139 diverse backgrounds to represent the wide range of end-users who  
 1140 interact with robots. However, it is possible that both the limited  
 1141 number of participants and the fact that they were all students from  
 1142 the same university limit how well our participants represent the  
 1143 overall population of end-users. We hope that our findings inspire  
 1144 additional work that can be evaluated more thoroughly with real  
 1145 end-users of robotic systems.

1146 **Training and time constraints.** We had only a limited amount  
 1147 of time available to train our participants and allow them to work on  
 1148 the given task. It is likely that with more available time or resources,  
 1149 participants of both groups might have performed better overall  
 1150 when solving the task. However, as we have outlined in Section 6.2,  
 1151 we believe that restrictive time constraints are not uncommon in  
 1152 practice, and that they can be particularly useful to investigate the  
 1153 learnability and usability qualities of a system.

1154 **Task choice.** We only evaluated a single task in our study, al-  
 1155 though that task was split into several sub-tasks that required  
 1156 different forms of coordination. We believe that the combination  
 1157 sub-tasks covered a wide range of challenges and requirements that

1158  
 1159  
 1160

1161 end-users face in practice. However, future work is needed to investigate other tasks, and particularly those that require coordination  
1162 in ways that neither of the tested environments support.

1163 **Selected tools.** Comparing Duplo and ROY introduces limitations to our work as these programming environments make specific implementation decisions that may not generalize to all  
1164 programming tools. The difference in programming method (ROY is graphical-based while Duplo is block-based) provides insight into these different methods but also introduces some confounds. However, we believe that although differences exist between the two  
1165 programming systems, both are designed to be beginner-friendly and, to the best of our knowledge, represent two of the most capable  
1166 end-user tools available for two-armed industrial robots.

## 8 CONCLUSION

In this work, we have presented Duplo, a new block-based programming environment for real-world two-armed robots. This environment is the first practical evaluation of a design concept that uses blocks to visualize the flow of time and coordination between robot arms. We have evaluated the system in comparison to an existing commercial alternative that has graphical elements and targets end-users, but is inherently text-based and does not provide comparable visual support. We found that Duplo allowed participants to solve a complex two-armed pick-and-place task faster and with greater success. Based on our own observations and the responses of our participants, we have identified which differences between the two environments might have caused this effect. We have further identified barriers that remain when participants try to learn either system. We believe that this work can inform future work in how to design robot programming environments that are more user friendly and make use of the strengths of existing visual frameworks.

## REFERENCES

- [1] ABB Ltd. 2015. RobotStudio Online YuMi. URL: <https://apps.microsoft.com/store/detail/9NBLGGH2SQFM>.
- [2] ABB Ltd. 2023. Robotstudio suite. URL: <https://new.abb.com/products/robotics/robotstudio>.
- [3] ABB Ltd. 2023. Yumi - irb 14000 collaborative robot. URL: <https://new.abb.com/products/robotics/robots/collaborative-robots/yumi/irb-14000-yumi>.
- [4] ABB Robotics. 2014. Technical reference manual: rapid instructions, functions and data types. ABB Robotics.
- [5] Gopika Ajaykumar and Chien-Ming Huang. 2020. User needs and design opportunities in end-user robot programming. In *Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, 93–95.
- [6] Gopika Ajaykumar, Maureen Steele, and Chien-Ming Huang. 2021. A survey on end-user robot programming. *ACM Computing Surveys (CSUR)*, 54, 8, 1–36.
- [7] Peter K Allen. 2012. *Robotic object recognition using vision and touch*. Vol. 34. Springer Science & Business Media.
- [8] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57, 5, 469–483.
- [9] Barbara Rita Barricelli, Fabio Cassano, Daniela Fogli, and Antonio Piccinno. 2019. End-user development, end-user programming and end-user software engineering: a systematic mapping study. *Journal of Systems and Software*, 149, 101–137.
- [10] Johannes Baumgartl, Thomas Buchmann, Dominik Henrich, and Bernhard Westfachtel. 2013. Towards easy robot programming-using DSLs, code generators and software product lines. In *ICSOFT*. Citeseer, 548–554.
- [11] Geoffrey Biggs and Bruce MacDonald. 2003. A survey of robot programming systems. In *Proceedings of the Australasian conference on robotics and automation*, 1–3.
- [12] Andrzej Burghardt, Dariusz Szybicki, Piotr Gierlak, Krzysztof Kurc, Paulina Pietrus, and Rafal Cygan. 2020. Programming of industrial robots using virtual reality and digital twins. *Applied Sciences*, 10, 2, 486.
- [13] J Edward Colgate, J Edward, Michael A Peshkin, and Witaya Wannasuphoprasit. 1996. Cobots: robots for collaboration with human operators. In *Proceedings of the 1996 ASME International Mechanical Engineering Congress and Exposition*. ASME, 433–439.
- [14] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, 15, 5, 107–116.
- [15] Juliet M. Corbin. 2015 – 2015. *Basics of qualitative research : techniques and procedures for developing grounded theory*. eng. (Fourth edition. ed.). SAGE, Thousand Oaks, California. ISBN: 1412997461.
- [16] [n. d.] Epson rc+ express manual. [https://files.support.epson.com/far/docs/eps\\_on\\_rc\\_express\\_6-axis\\_rev.5\\_en.pdf](https://files.support.epson.com/far/docs/eps_on_rc_express_6-axis_rev.5_en.pdf). Accessed: 2023-03-01. () .
- [17] Annette Feng, Eli Tilevich, and Wu-chun Feng. 2015. Block-based programming abstractions for explicit parallel computing. In *Proceedings of the 2015 Blocks and Beyond Workshop*. IEEE, 71–75.
- [18] Beate Jost, Markus Ketterl, Reinhard Budde, and Thorsten Leimbach. 2014. Graphical programming environments for educational robots: OpenRoberta - yet another one? In *Proceedings of the 2014 International Symposium on Multimedia*. IEEE, 381–386.
- [19] Amy J Ko, Brad A Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*. IEEE, 199–206.
- [20] Soenke Kock, Timothy Vittor, Björn Matthias, Henrik Jerregard, Mats Källman, Ivan Lundberg, Roger Mellander, and Mikael Hedelind. 2011. Robot concept for scalable, flexible assembly automation: a technology study on a harmless dual-armed robot. In *Proceedings of the 2011 International Symposium on Assembly and Manufacturing (ISAM)*. IEEE, 1–5.
- [21] Konstantinos Lotsaris, Christos Gkournelos, Nikos Fousekis, Niki Kousi, and Sotiris Makris. 2021. Ar based robot programming using teaching by demonstration techniques. *Procedia CIRP*, 97, 459–463.
- [22] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10, 4, 1–15.
- [23] Christoph May-Dorn, Marin Winterer, Christian Salomon, Doris Hohnsinger, and Rudolf Ramler. 2021. Considerations for using block-based languages for industrial robot programming-a case study. In *2021 IEEE/ACM 3rd International Workshop on Robotics Software Engineering (RoSE)*. IEEE, 5–12.
- [24] Siti Nor Hafizah Mohamad, Ahmed Patel, Rodziah Latif, Qais Qassim, Liu Na, and Yiqi Tew. 2011. Block-based programming approach: challenges and benefits. In *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*. IEEE, 1–5.
- [25] Francis Sahngun Nahm. 2016. Nonparametric statistical tests for the continuous data: the basic concept and the practical use. *Korean journal of anesthesiology*, 69, 1, 8–14.
- [26] João Neves, Diogo Serrario, and J Norberto Pires. 2018. Application of mixed reality in robot manipulator programming. *Industrial Robot: An International Journal*.
- [27] Zengxi Pan, Joseph Polden, Nathan Larkin, Stephen Van Duin, and John Norrish. 2010. Recent progress on programming methods for industrial robots. In *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*. VDE, 1–8.
- [28] Marco Piccinelli, Andrea Gagliardi, Umberto Castellani, and Riccardo Muradore. 2021. Trajectory planning using mixed reality: an experimental validation. In *2021 20th International Conference on Advanced Robotics (ICAR)*. IEEE, 982–987.
- [29] [n. d.] PolyScope manual. [https://s3-eu-west-1.amazonaws.com/ur-support-site/44018/Software\\_Manual\\_en\\_Global.pdf](https://s3-eu-west-1.amazonaws.com/ur-support-site/44018/Software_Manual_en_Global.pdf). Accessed: 2023-03-01. () .
- [30] Camilo Perez Quintero, Sarah Li, Matthew KXJ Pan, Wesley P Chan, HF Machiel Van der Loos, and Elizabeth Croft. 2018. Robot programming through augmented trajectories in augmented reality. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1838–1844.
- [31] Marina S Reshetnikova, Irina A Pugacheva, Elena G Popkova, and Inna V Andronova. 2022-3-30. *The Global Industrial Robotics Market: Development Trends and Volume Forecast. Current Problems of the World Economy and International Trade*. Jai Press, Greenwich, Conn. : ISBN: 978-1-80262-090-0.
- [32] Nico Ritschel, Vladimir Kovalenko, Reid Holmes, Ron Garcia, and David C Shepherd. 2020. Comparing block-based programming models for two-armed robots. *IEEE Transactions on Software Engineering*.
- [33] Nico Ritschel et al. [n. d.] Training industrial end-user programmers with interactive tutorials. *Software, Practice and Experience*.
- [34] Gregory F Rossano, Carlos Martinez, Mikael Hedelind, Steve Murphy, and Thomas A Fuhrbrige. 2013. Easy robot programming concepts: an industrial perspective. In *2013 IEEE international conference on automation science and engineering (CASE)*. IEEE, 1119–1126.
- [35] Mazayr Seraj, Serge Autexier, and Jan Janssen. 2018. Beesm, a block-based educational programming tool for end users. In *Proceedings of the 10th Nordic Conference on Human-Computer Interaction*, 886–891.
- [36] Donna Spencer. 2009. *Card sorting: Designing usable categories*. Rosenfeld Media.
- [37] Maj Stenmark and Pierre Nugues. 2013. Natural language programming of industrial robots. In *IEEE ISR 2013*. IEEE, 1–5.
- [38] Ricardo Tellez. 2017. The ros development studio by the construct. URL: <https://www.theconstructsim.com/the-ros-development-studio-by-the-construct/>.
- [39] D. Weintrop, D.C. Shepherd, P. Francis, and D. Franklin. 2017. Blockly goes to work: block-based programming for industrial robots. In *Proc. of Blocks and Beyond Workshop (B&B)*, 29–36.

- 1277 [40] David Weintrop. 2019. Block-based programming in computer science education. *Communications of the ACM*, 62, 8, 22–25. 1335
- 1278 [41] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C 1336
- 1279 Shepherd, and Diana Franklin. 2018. Evaluating coblox: a comparative study 1337
- 1280 of robotics programming environments for adult novices. In *Proceedings of the 1338*
- 1281 2018 CHI Conference on Human Factors in Computing Systems
- 1282 [42] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C. 1339
- 1283 Shepherd, and Diana Franklin. 2018. Evaluating CoBlox: a comparative study 1340
- 1284 of robotics programming environments for adult novices. In *Proceedings of the 1341*
- 1285 2018 CHI Conference on Human Factors in Computing Systems Article 366. ACM, 1342
- 1286 366:1–366:12. 1343
- 1287 [43] David Weintrop and Uri Wilensky. 2017. Comparing block-based and text-based 1344
- 1288 programming in high school computer science classrooms. *ACM Transactions 1345*
- 1289 on Computing Education (TOCE)
- 1290 , 18, 1, 1–25. 1346
- 1291
- 1292
- 1293
- 1294
- 1295
- 1296
- 1297
- 1298
- 1299
- 1300
- 1301
- 1302
- 1303
- 1304
- 1305
- 1306
- 1307
- 1308
- 1309
- 1310
- 1311
- 1312
- 1313
- 1314
- 1315
- 1316
- 1317
- 1318
- 1319
- 1320
- 1321
- 1322
- 1323
- 1324
- 1325
- 1326
- 1327
- 1328
- 1329
- 1330
- 1331
- 1332
- 1333
- 1334
- [44] David Weintrop and Uri Wilensky. 2017. How block-based languages support 1335
- novices. *Journal of Visual Languages and Sentient Systems*, 3, 92–100. 1336
- [45] David Weintrop and Uri Wilensky. 2015. To block or not to block, that is the 1337 question: students' perceptions of blocks-based programming. In *Proceedings 1338*
- of the 14th International Conference on Interaction Design and Children. ACM, 1339
- 199–208. 1340
- [46] Enes Yigitbas, Ivan Jovanovikj, and Gregor Engels. 2021. Simplifying robot 1341
- programming using augmented reality and end-user development. In *Human- 1342*
- Computer Interaction—INTERACT 2021: 18th IFIP TC 13 International Conference, 1343
- Bari, Italy, August 30–September 3, 2021, Proceedings, Part I 18. Springer, 631– 1344
651. 1345
- [47] Minhz Fahim Zibran. 2007. Chi-squared test of independence. *Department of 1346*
- Computer Science, University of Calgary, Alberta, Canada
- , 1, 1, 1–7. 1347
- Received 1 January 2023; revised 1 January 2023; accepted 1 January 2023 1348
- 1349
- 1350
- 1351
- 1352
- 1353
- 1354
- 1355
- 1356
- 1357
- 1358
- 1359
- 1360
- 1361
- 1362
- 1363
- 1364
- 1365
- 1366
- 1367
- 1368
- 1369
- 1370
- 1371
- 1372
- 1373
- 1374
- 1375
- 1376
- 1377
- 1378
- 1379
- 1380
- 1381
- 1382
- 1383
- 1384
- 1385
- 1386
- 1387
- 1388
- 1389
- 1390
- 1391