Evaluating CoBlox: A Comparative Study of Robotics Programming Environments for Adult Novices

David Weintrop[1], Afsoon Afzal[2], Jean Salac[3], Patrick Francis[4], Boyang Li[4], David C. Shepherd[4], Diana Franklin[3]

[1]University of Maryland, College Park, Maryland, United States
[2]Carnegie Mellon University, Pittsburgh, Pennsylvania, United States
[3]University of Chicago, Chicago, Illinois, United States
[4]ABB Corporate Research, Raleigh, North Carolina, United States

weintrop@umd.edu, afsoona@cs.cmu.edu, {salac, dmfranklin}@uchicago.edu,
{patrick.francis, boyang.li, david.shepherd}@us.abb.com

ABSTRACT
A new wave of collaborative robots designed to work alongside humans is bringing the automation historically seen in large-scale industrial settings to new, diverse contexts. However, the ability to program these machines often requires years of training, making them inaccessible or impractical for many. This paper rethinks what robot programming interfaces could be in order to make them accessible and intuitive for adult novice programmers. We created a block-based interface for programming a one-armed industrial robot and conducted a study with 67 adult novices comparing it to two programming approaches in widespread use in industry. The results show participants using the block-based interface successfully implemented robot programs faster with no loss in accuracy while reporting higher scores for usability, learnability, and overall satisfaction. The contribution of this work is showing the potential for using block-based programming to make powerful technologies accessible to a wider audience.

Author Keywords
Block-based programming; Industrial robotics interfaces

ACM Classification Keywords
D.2.3 Coding Tools and Techniques; H.5.2 User Interfaces

INTRODUCTION
In recent years robots have become safer and more flexible, resulting in a greater presence in our world. This is especially true in the workplace, where robots are being used in a growing number of roles. While the larger narrative around the introduction of robots into the workplace often frames these technologies as replacements for workers, scholarship is finding that automation does not necessarily replace workers, but it does change the nature of the work [9].

Collaborative robots, which are intended to work safely alongside humans, exemplify this trend [12,22,27]. Collaborative robots take advantage of "the interplay between machine and human comparative advantage [that] allows computers to substitute for workers in performing routine, codifiable tasks while amplifying the comparative advantage of workers in supplying problem-solving skills, adaptability, and creativity" [9]. In order to support new challenges that emerge from being placed in smaller factories and given a wider variety of tasks, these new robots must be safe, efficient and, support quick reprogramming.

While the design of the machines themselves has resulted in more powerful and flexible robots with a greater set of capabilities, relatively little attention has been given to the accompanying programming tools to make them more accessible or intuitive. Programming languages used in industrial settings, many derived from Pascal and BASIC and created in the early 1990s, have historically been designed by engineers, for engineers. As such, writing the programs necessary to introduce robots into the workplace is time-consuming and often requires years of training, meaning many small and medium-sized enterprises are not able to benefit from robotic automation [37,38].

Fortunately, advances in the design of programming environments for novices may provide some guidance on ways to redesign these robot programming interfaces. While early work in end-user programming focused on making computers and programming accessible to professionals [6], the last twenty years has produced major advances in designing introductory programming environments for younger learners [14,26]. In particular, the emergence of the block-based programming paradigm has introduced millions of young learners to the powerful concepts of computing through Scratch, Lego Mindstorms, and other toys [4]. This paper presents the results of an investigation into if and how block-based programming, designed for young learners, can be used to make the task of programming industrial robots accessible to adult novices.

CHI 2018, April 21–26, 2018, Montreal, QC, Canada

ACM 978-1-4503-5620-6/18/04…$15.00
https://doi.org/10.1145/3173574.3173940

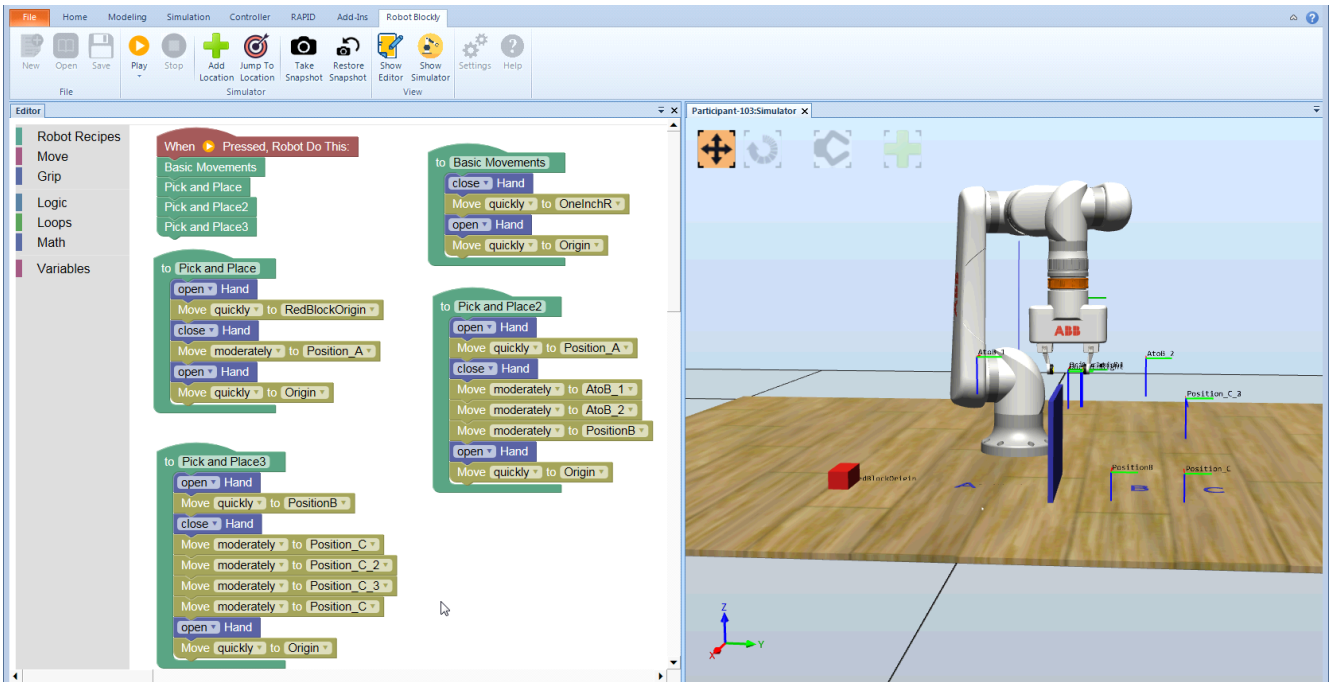


Figure 1. The CoBlox programming environment. The left side of the environment contains the block-based robot programming interface for Roberta, shown on the right.

This paper introduces CoBlox, a block-based programming interface for Roberta, a single-armed industrial robot (Figure 1) and presents results of a comparative study showing how the CoBlox interface outperforms two of the most widely used robotics programming approaches with respect to speed of authoring programs with no loss of accuracy and fostering more positive attitudes and higher levels of satisfaction for adult novice programmers.

The contribution of this work is that it shows how the affordances of block-based programming can be used to make a complex task, like industrial robot programming, more accessible to adult novices. In doing so, we provide an empirical basis for the use of block-based programming as an effective programming interface for the growing set of applications and contexts where programming by non-experts might occur. As programming becomes more mainstream for non-technical employees, there is a growing audience of designers that may benefit from this work. Additionally, this work shows that drawing inspiration from learning environments designed for young novices can effectively inspire tools intended for wider audiences.

RELATED WORK

The work presented in this paper brings together design innovations from research into making programming accessible to young learners with the large body of work investigating different approaches to programming robotic systems. In this section, we review relevant prior work from these two literatures, focusing specifically on end-user robotics programming and block-based programming, positioning our work at the intersection of the two.

End-User Robotics Programming

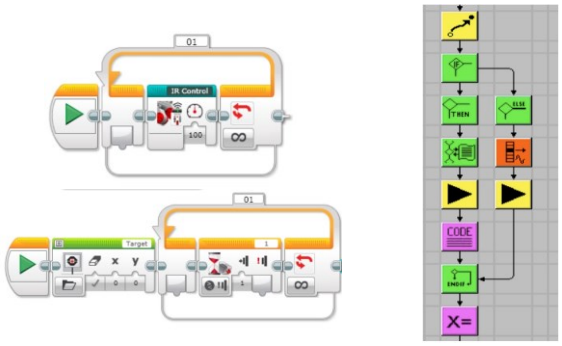
End-user programming is defined as "programming to achieve the result of a program primarily for personal, rather public use" [9]. In the case of robot programming, this means the author is writing a routine for a specific, immediate task, as opposed to creating a general-purpose program or a template script that others will later modify. This review focuses on end-user robotics programming languages due to our goal of making the power of industrial robots accessible to a wider audience of potential users.

In their survey of robot programming systems, [6] break down end-user robot programming into two main categories: manual programming systems and automatic programming approaches.

Manual Programming Systems

Manual programming systems are defined as robot programming interfaces where the user has direct control over individual programming instructions. These interfaces can present users with a text-based interface for controlling robots, which has historically been the predominant approach for robot programming, or use graphical representations to give a user control over the robot.

Almost all major industrial robots can be controlled via a proprietary, text-based programming language [6]. These languages often draw inspiration from early programming languages like BASIC and Pascal. Examples of these systems include ABB's RAPID and KUKA's KRL, which provide core functionality along with libraries that cover an increasing array of common robotics tasks. In response to this segmentation, there are efforts to create generalized robot programming languages [19] as well as extensions for



(a) (b)

Figure 2. Two examples of graphical robot programming tools: (a) Lego Mindstorms and (b) MORPHA.

general-purpose languages like C++ [23] and Python [9] to make the language more suitable for general robot programming tasks.

A second form of manual robot programming systems adopt a visual programming approach and incorporate graphics and icons into the programming interface. These environments replace text-based instructions with icons, diagrams, or some other graphical representation that can be rendered in two dimensions which can then be manipulated by the user to define instructions for the robot to follow [35]. A number of graphical programming tools have been created to support robot programming. The most well-known of which is the Lego Mindstorms tool (Figure 2a), which uses visual blocks to represent basic robot actions which the user can organize to produce desired outcomes [30]. A second example of this approach is MORPHA (Figure 2b), which used an icon-based approach and flowchart-like layout to let users define instructions for their robot [8]. MORPHA was intended to be used in industry but never achieved widespread adoption, in part due to the challenge of creating a meaningful icon for every possible command.

Another graphical approach to robot programming represents programs as trees of hierarchical tasks [20]. With tree-based representations, the task of creating a robot routine is broken down into a series of steps, with each step potentially have sub-steps, resulting in a hierarchical organization and presentation of the program. Figure 3 shows Universal Robots' Polyscope interface, which includes a tree-based program on the left-side of the screen. Programming in tree-based environments is accomplished through menu-based navigation, where new commands are introduced by clicking buttons and defining inputs, as can be seen in Figure 3, which shows how a new waypoint can be added to a program. These interfaces often employ "wizards" to walk the user through creating common sequences. This approach has been well-accepted in practice.

Automatic Programming Systems

Automatic end-user robot programming systems give the user the ability to program a robot, but unlike manual programming systems, these environments hide the programming language from the users. Examples of

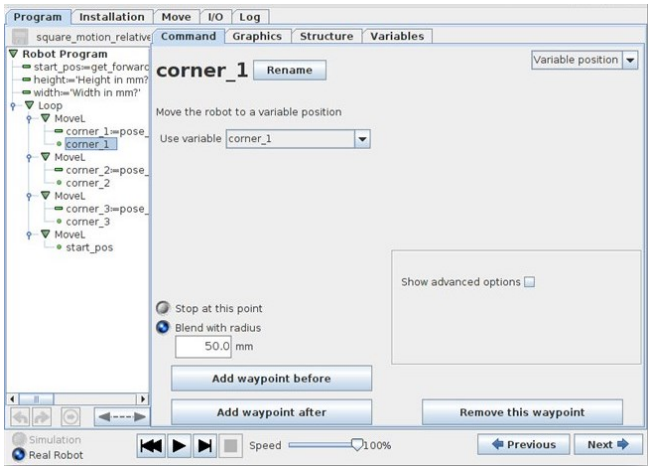


Figure 3. Universal Robot's tree-based programming tool.

automatic programming approaches include learning systems [1], gesture-following robots that imitate human actions [10], and the widely-used programming-by-demonstration approach [7].

Programming-by-demonstration takes the form of physically moving the robot into the desired position and then recording its state. By sequentially positioning the robot in different states, the operator can define a robot routine. This form of input is made possible through the combination of force-free control (meaning the robot can be moved as if it was in a zero-gravity environment) and a hand-held device, called a teach pendant [28]. All of the Big Four robotics manufacturers (ABB, Kuka, Fanuc, and Yasukawa), which account for over 60% of the world's industrial robots, provide a teach pendant (examples of which are shown below in Figure 4). This makes the teach pendant approach the primary method of end-user programming in the field.

Figure 4. Four teach pendants used for robot programming.

Block-based Programming

Block-based programming (visible on the left side of Figure 1) is an increasingly popular approach in the design of introductory programming environments that uses a programming-command-as-puzzle-piece metaphor to present commands to the user [4,32]. Writing a program in a block-based environment takes the form of dragging-and-dropping instructions into place on screen. Each individual command includes visual information about how and where it can be used, ensuring that incompatible instructions cannot be combined, thus preventing syntax errors in the program. Additionally, block-based programming environments include a number of features that have been identified as productive for novice programmers, including supporting

natural language commands, presenting available commands in logically ordered and easily browsed ways, and using a drag-and-drop authorship mechanism that is easier and faster than typing a command character-by-character with the keyboard [45]. A growing body of literature shows that the block-based approach to programming is an effective way to enable novices to write successful programs with little prior experience and can serve as an accessible introduction to programming [17,21,43]. The present study contributes to this body of research by studying adult novices in a professional setting instead of young learners in an educational context. Additionally, the fact that our tool is the end goal language, not a stepping stone to professional software development, represents a major change in the purpose of the language.

Led by the popularity of block-based tools including Scratch [39] and Alice [13], there is a growing ecosystem of block-based environments that support a variety of programming activities. Alice [13], and other block-based tools like AgentCubes [25], are noteworthy in that they allow the user to program three-dimensional simulations. While much of the focus of block-based tools has been on the creation of digital media (like stories, animations, and games), block-based programming environments exist for modeling and simulation tools [5,24,46], mobile application development [40,47], playing video games [15,44], and manipulating media [33]. At the same time, there are a growing number of libraries and tools designed to make it easy to create new block-based languages or embed block-based programming interfaces into existing applications [3,18]. Finally, the block-based programming approach has been used in robotics kits for kids, which we discuss below.

Educational and Entertainment Robots

The final section in our review of prior work looks at educational robots and toys, where the intersection of block-based programming and robotics has already begun. Lego Mindstorms [30] provides a LabView-based system to program Lego-based creations using sensors and motors (Figure 2a). While the interface uses an icon-based language, rather than text-based commands, it has proven to be powerful for beginners and more advanced users. There are also block-based interfaces for Mindstorms kits (and other similar robots), such as Open Roberta [50]. Beyond Mindstorms, there are also a growing number of robotics toys designed not as construction kits, but as robots to teach programming using a block-based interface. Examples of these tools include Dash and Dot [48], the Finch Robot [29], mBots [31] , Edison [34] and Ozobots[36]. While these educational robots share the larger goal of making robot programming easier for novices, they lack the capabilities, power, and the ability to support the types of complex instructions required for a collaborative robot in industry.

COBLOX DESIGN

This paper investigates ways to make robot programming more accessible, especially to adults with little or no programming experience. Our approach leverages block-based programming, a technique that has seen widespread success in educational contexts, and applies it to the challenge of robot programming by creating a custom robot language, a novice-focused editor, and a robot simulation interface. The essential design elements being investigated in this work include the use of the block-based interface integrated into a virtual robotics environment and the custom designed, domain specific language that accompanies it. Here, we provide an overview of the CoBlox design to contextualize the comparative study. The design is presented in greater detail in [42].

The CoBlox environment (shown in Figure 1) is comprised of a custom-designed block-based programming interface built with the Blockly library [18] and an embedded virtual robot simulator, which we discuss in the following walkthrough. Users write programs in CoBlox by dragging-and-dropping pre-defined robot commands and snapping them together to define sequences of instructions for the robot to follow. Users can define movement commands by adding the `move` block to their program. The text on the `move` block reads: `Move quickly to <somewhere>`. The `quickly` statement is a dropdown that specifies the speed of the movement (the other choices are `slowly` and `moderately)`. `The <somewhere>` portion of the `move` command specifies the `Location` the robot will move to and includes a list of all previously defined `Locations` along with an option to define a new `Location`. A `Location` is a programming construct we developed that is used to define a robot's position, which includes its x, y, and z coordinates and the orientation of the tool attached to the end of the robot arm, in this case, a gripper. To define a new position for the robot, the user selects the `Add Location` option in the `<somewhere>` dropdown. When this happens, the user is prompted to use the virtual robot interface to click-and-drag the robot arm into place. Once the robot is in position, the user clicks a check box at the top of the screen, and gives a name to the `Location` (e.g. `RedBlockOrigin`, as seen in the `Pick and Place` recipe in Figure 1). Once the `Location` is defined, the `<somewhere>` text in the dropdown is replaced with the newly entered name. This process is similar to the programming-by-demonstration approach commonly used in robotics programming [7], just replacing the physical robot with a virtual one and introducing the programming construct of a `Location` that can be reused throughout the block-based program. With this feature, we highlight the drag-and-drop programming mechanisms of block-based programming, the ability to blend input features within a programming command (adding dropdowns and buttons inside a programming command), and the dynamic interface (for shifting between the programming and robot interfaces), as ways to make the task of programming more accessible.

Another innovation of the CoBlox interface that uses the affordances of the block-based modality is the introduction of *Robot Recipes*. Robot Recipes are predefined functions

that serve as templates for commonly carried out actions. In the study presented below, the environment includes a single Robot Recipe called `Pick and Place`. The `Pick and Place` recipe defines the sequence of steps a robot follows to pick up an object in one location and place it somewhere else, a very common task for industrial robots. Robot Recipes are comprised of blocks available to the user, with suggested default arguments provided to help make the template easier to follow. For example, in the `Pick and Place` recipe, the first Move command reads `Move quickly to <approach to pick>`, which is meant to let the user know that the first `Location` to be defined is where to put the robot arm ahead of its approach to the pickup position. The goal of Robot Recipes is to further scaffold adult novice users by providing easy-to-follow templates to carry out common robot programming tasks. Additional features of CoBlox, including results from a small-scale user study, can be found in [42].

EXPERIMENTAL DESIGN

To evaluate our block-based technique for industrial robot programming we conducted a user study comparing CoBlox to two widely-used professional tools. This section presents the study design, including the procedure followed, data collected, and analytic techniques used.

The robot programming environments

The independent variable in this user study is the programming environment in which the participant was asked to work. We compared our programming environment against two of the mostly widely-used industrial robot programming approaches. This ensures a benchmark against the leading approaches, enabling prospective adopters to assess its practical impact. We reviewed both research literature and products currently on the market to select the environments for comparison.

After our review of robot programming environments, we chose two comparison environments: ABB's Flex Pendant and Universal Robot's Polyscope. For all three environments we had participants use an "offline" robot programming model [49] which includes a robot simulation where the virtual version of the robot can be manipulated as part of the programming interface. For Polyscope, we added the virtual robot by using RoboDK, a third-party simulator recommended by Universal Robots. The offline approach offers a number of advantages over the alternative which requires a physical robot to be available, including cost, ease of development and modification of programs, and development can be accomplished while the robot is in use [37]. At any point during program development, the user can click the "Play" button and watch a simulation of the robot carrying out the programmed instructions. Tutorial videos showing how to program a robot in all three environments are available in the online supplemental materials.

ABB's Flex Pendant

Because nearly all of the hundreds of thousands of deployed industrial robots are attached to a teach pendant (like those shown in Figure 4), we selected this type of approach as one of the comparative programming environments. After verifying that ABB's Flex Pendant (Figure 5) was comparable to other widely-used teach pendants in terms of quality, capabilities, and use in industry, we chose it as an exemplary teach pendant programming environment for this study. The Flex Pendant allows users to open or create programs, edit them line by line, and input positions by moving the robot via the attached joystick. As shown in Figure 5, the programs authored in the Flex Pendant are represented as text-based programs in the RAPID programming language. Adding commands to a program via the Flex Pendant takes the form of navigating menus and selecting commands and parameters from lists. A tutorial video showing this process is included in the online supplemental materials for this paper. Note that a virtual version of the pendant was used for the study, the reasons for which are discussed as part of the study procedure below.

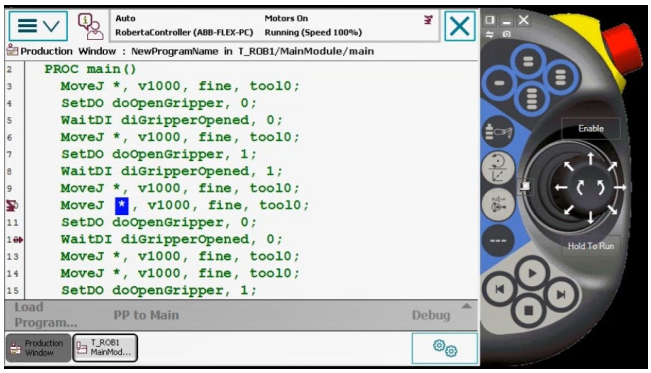


Figure 5. The virtual version of ABB's Flex Pendant programming interface.

Universal Robots' Polyscope

The second robot programming environment chosen for this study is Universal Robot's Polyscope tool, shown in Figure 3. This tool was chosen because it represents the most successful—in terms of robots sold—approach to end-user robot programming. Additionally, it uses a tree-based, dialog driven strategy, thus providing another end-user programming paradigm to compare to CoBlox. Authoring a program in Polyscope takes the form of navigating through screens and menus, defining the specifics of each step by inputting values into text fields and clicking buttons associated with the desired behavior. This menu-based programming approach is distinct from conventional programming in that the resulting program is not represented in text, but instead as a series of nodes in a hierarchical tree.

Participants

The goal of this study is to create a robot programming interface that is accessible and usable by adults with little or no prior robotics programming experience. As such, we sought to recruit a diverse set of professionals to match this profile. Participants were recruited from an office of a multinational engineering conglomerate located in the eastern United States. Only employees outside of the

company's Robotics Division for whom computer programming is not one of their core competencies and who do not do it in their jobs were invited to participate. This population matches our target users in that they do not program computers or work directly with industrial robots as part of their daily job requirements. The study was approved by the institutional review board at the primary research center with permission of the industry partner.

We recruited participants by inviting them via inter-office email or face-to-face contact, offering a complimentary lunch as an incentive. Approximately 80% of contacted employees accepted our appointment request, leading to 110 participants. To assign potential participants to environments in a uniform manner, we first divided them into three groups: research interns (17), researchers (44), and non-researchers (49). Participants from each group were then sorted alphabetically and assigned to the treatments round robin.

Of the 110 scheduled users, 89 participated, and 67 were included in the final results. Seventeen participants were disqualified for procedural violations (e.g., an emergency meeting pulling the participant away from the task) and another 5 participants were removed because they had taken more than five programming courses in their lifetime. Of the 67 participants, 59 were male and 8 were female. The average age of the participants was 35.3 years (*SD* 9.1), they had an average professional experience of 9.7 years (*SD* 8.4) and had taken an average of 1.5 programming courses (SD 1.4) with over half of the participants (37 out of 67) having one or fewer programming courses in their lives. Participants were from a variety of work areas, including development, sales, testing and quality assurance, and various forms of engineering.

User Study Procedure

For each participant in the study, we followed the same procedure, shown below in Figure 6. During setup, we connected their work machine to our remote virtual machine that had the robotics software pre-loaded and gave them a reference sheet for their programming environment. The procedure was conducted at participants' desks to minimize the disruption participation would cause to their workday. Each participant was then asked to (i) fill out a brief demographic survey, (ii) watch an approximately 10 min training video, and (iii) complete as many of the specified sub-tasks as possible during the allotted time of 60 minutes. While it would have been preferable to have no time limit, conducting studies with busy employees necessitated the chosen approach, and we felt that access to this representative population was worth this shortcoming. The session ended with a post survey, which asked participants about their experiences working on the programming activity. The researcher left during the programming portion of the procedure because we wanted participants to rely on the training materials as much as possible. The participant was provided an instant messaging account to contact the researcher if they had questions. The researcher reappeared at the participant's desk after the allotted time to ensure the user had encountered no major problems.

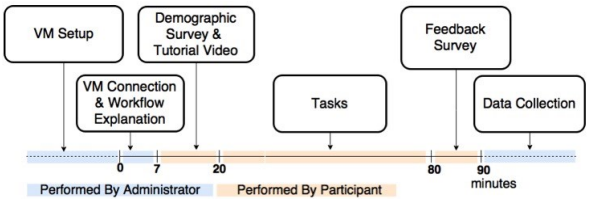


Figure 6. The timeline for the study procedure.

Training

Prior to attempting the assigned task users were trained on their programming environment. To provide the most realistic training, we mimicked both ABB and Universal Robots' practice of providing training videos. We created an approximately ten-minute video for each environment. Each video discussed the basic layout of the tool, how to navigate the 3D simulation environment, how to add common commands, how to use task templates, how to run programs, and how to jog the robot. Each video's content covered the same materials, with the details varying according to each environment. In addition to the video-based training, we also included a single, two-sided 8.5 by 11-inch reference sheet that users could refer to throughout the task, which contained screenshots and explanations of the commands used during the video. All of these materials are available in the online supplemental materials that accompany this paper.

Tasks

Participants were asked to complete a series of tasks inspired by real-world robotics tasks—specifically pick and place—and designed to be challenging to finish in an hour. The four tasks were designed to be cumulative and of increasing difficulty. Each task was logged and evaluated separately.

The first task was to open and close the robot gripper, and to move the robot arm approximately one inch to the right and then to the left. This was intended to serve as an initial, orienting task to get users familiar with the robot and its movements. The second task was a basic pick and place routine with no outside constraints, asking the users to move the red block shown in Figure 1 from its starting point to point A. Task 3 was a pick and place routine, but with a small wall to be avoided in between the beginning and ending locations (moving the block from point A to point B in Figure 1). The fourth and final task was a pick and place with reorientation, picking up the block from point B and placing it on its side at point C. These four tasks represent realistic programming activities for an industrial robot, as positioning the robot arm and moving and reorienting objects are core to the functioning and use of these types of robots. The exact wording of the task instructions can be found in the supplemental materials.

Pilot Study

To ensure that all unforeseen issues were fixed prior to data collection, we piloted our study with 12 participants. As a

result, we identified two task descriptions that users found confusing, which we addressed by revising the text; a bug related to the robotic gripper that caused some programs to hang, which we patched; and a missing reset feature in the Universal Robots environment, which we added. These improvements led to a much lower procedural failure rate during the main data collection period.

Data Collection and Analysis

Survey and Interviews

Survey responses were collected from all 67 participants. To analyze the textual data of the survey responses, we used a Grounded Theory approach to determine higher level themes [41]. To establish a common set of codes and themes, two researchers applied open axial coding to the same subset of survey responses and then established a common understanding and defined a structure for the most commonly mentioned concepts. To validate the analysis of the survey results, two additional researchers extracted their main findings from a subset of the responses independently.

Self-Reported Progress

We collected progress logs from every participant during their session, which included the completion state and time-on-task for each task. This data allows us to compare user progress in terms of time and completion rate.

Programs Authored

For each participant, we collected the program they authored as well as a video of each participant's program running. We used these videos to verify that self-reported progress logs were accurate and to evaluate the correctness of the program with respect to the stated objective of each task. For each of the four tasks, a 10-point grading rubric was created to evaluate the correctness of the program. To ensure a consistent quality grade, two researchers scored each video individually, and all disagreements in scores were resolved through discussion. The grading rubric can be found in the online supplemental materials.

RESULTS

The findings section is broken up into two sections. First, we report on an analysis of the programming portion of the study, reporting differences in completeness, correctness, time on task, and findings from a qualitative analysis into types of errors made by each condition. The second portion of this section presents results from an analysis of the post survey, looking at differences in reported usefulness, ease-of-use, and satisfaction, as well as report on patterns in responses to open-ended prompts of users' experiences during the programming task.

Results from the Programming task

Data collected during each participant's session included time-on-task for each of the four tasks and videos of their final programs, which were evaluated for correctness. This section presents the result of these analyses.

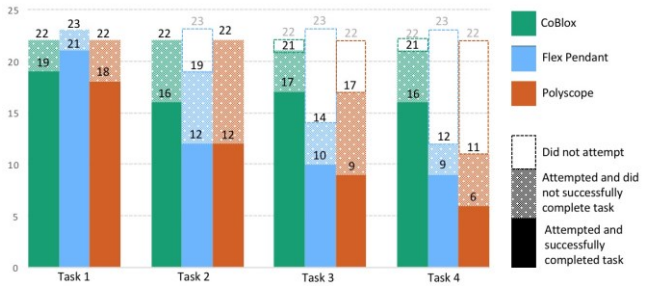


Figure 7. Number of participants that attempted and completed each task, grouped by condition.

Progress and Time on Task

The programming activity was broken down into four cumulative tasks, with each task building off the functionality authored in the previous task and increasing in complexity. As participants progressed through the programming activity, they logged when they completed each task. We use these self-reported logs to track how many tasks participants completed, and the time required to complete each task. The number of participants who attempted each task is shown above the lightly shaded portion of each column in Figure 7. The solid portion of each column reflects the number of participants who scored 6 or higher on each task. The average time on task *of those who attempted the task* is shown in Table 1.

	Time on Task (in seconds)			
Condition	**Task 1**	**Task 2**	**Task 3**	**Task 4**
CoBlox	438.36	843.64	481.43	621.29
Flex Pendant	1679.08	1003.32	506.93	605.00
Polyscope	940.73	1398.59	801.76	653.09

Table 1. Time-on-task in seconds for each condition, including only participants that attempted each task.

All 67 participants attempted the first programming task with many completing it. Comparing the average time-on-task across the three conditions for the first task, we see a significant difference ($F(2, 65) = 15.97$, $p < .001$). A Tukey Post Hoc HSD shows there to be significant differences between all three conditions (CoBlox and Flex Pendant $p < .001$; CoBlox and Polyscope $p < .05$; Flex Pendant and Polyscope $p < .01$), with CoBlox users completing the first task the fastest and Flex Pendant users taking the longest.

The second task, attempted by almost all participants, again shows a significant difference between the conditions ($F(2, 61) = 3.50$, $p < .05$), with the Polyscope participants significantly slower than the CoBlox participants ($p < .05$).

Task 3 represents a turning point. Tasks were cumulative, so participants who did not complete one task did not attempt any later tasks. All but one CoBlox participant attempted Tasks 3 and 4, but only 12 of the 23 Flex Pendant participants

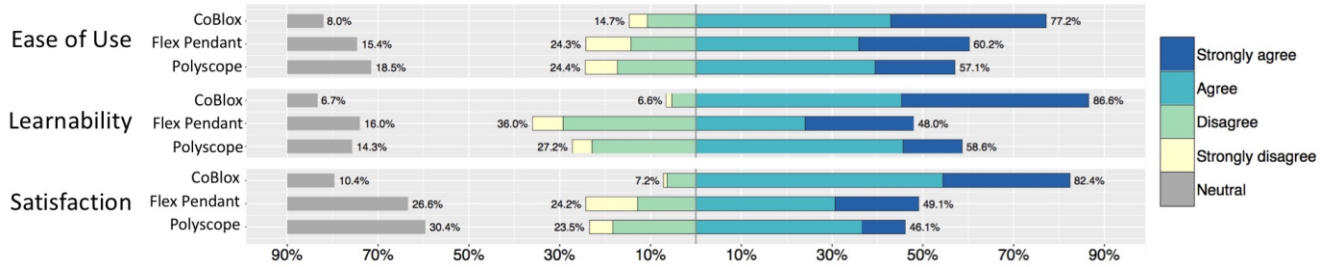


Figure 8. Composite scores for three attitudinal dimensions for the three conditions based on responses to the post survey. The differences between the three conditions are statistically significant for all three categories.

and 11 of the 22 Polyscope participants attempted Task 4. This steep drop-off in participation corresponds with a shifting trend in time to completion. As the number of participants attempting tasks in Polyscope and Flex Pendant dwindles, the time on task of those who did attempt the latter tasks becomes comparable to CoBlox. This is true for both Tasks 3 and 4. However, there is a significant difference between the number of people who successfully complete Tasks 3 and 4 in the CoBlox condition compared to the others, as shown in Figure 7. *In summary,* ***users complete more tasks, more quickly, using CoBlox****.*

Correctness

As Figure 7 shows, the number of participants who successfully completed a task is not identical to the number of participants who attempted the next task. Therefore, we separately discuss the correctness of programs. When *only including participants who attempted each task*, there was no statistically-significant difference by condition in scores on Tasks 1, 2, or 4. On Task 3, there is a significant difference between the three conditions ($F(2, 50) = 3.23$, $p < .05$). A Tukey HSD Post Hoc calculation shows the CoBlox condition to be significantly different than the Flex Pendant condition ($p < .05$). *In summary,* ***users achieve the same level of quality when using CoBlox, even though they complete more tasks more quickly.***

Patterns in Errors

An analysis of participants' final projects reveals two major types of errors across the three conditions: missing code snippets and incorrect location specification.

Missing pieces of code errors were identified in programs for 18 (81.8%) of the 22 CoBlox users, 14 (60.9%) of the 23 Flex users, and 16 (72.7%) of the 22 Polyscope users. The two most common errors resulting from missing code were failing to avoid obstacles (i.e. the wall or the floor) and skipping steps defined in the programming tasks. In CoBlox, 10 (55.6%) of the 18 programs with missing code resulted in collisions with the floor due to not lifting up the arm vertically before moving it horizontally. In Flex Pendant, 7 (50%) of the 14 programs with missing code resulted in their robot arm skipping steps specified in each of the programming tasks. In Polyscope, 11 (68.8%) and 12 (75%) of the 16 programs with missing code resulted in the robot arm colliding with the wall and the floor, respectively. 13 (81.3%) of the 16 programs resulted in the robot arm skipping steps specified in the tasks.

Incorrect location specification, meaning the participant moved the arm to the wrong for the task (e.g. moving the arm through the floor), accounted for errors in 12 (54.5%) of the 22 CoBlox users, 10 (43.5%) of the 23 Flex users, and 16 (72.7%) of the 23 Polyscope users. The mechanism for location specification was outside the scope of the CoBlox design. *In summary,* ***CoBlox users made the same types of mistakes as users of the other two environments.***

We also want to highlight two errors unique to Flex participants: failure to compile (1 of 23 participants) and incorrect parameters (1 of 23 participants). To control the gripper, Flex Pendant requires multiple *properly ordered*, *consecutive* instructions in order to compile. Both CoBlox and Polyscope provide this functionality through a single dropdown command. Further, in the Flex Pendant interface participants had to select parameters from a number of different menus and interfaces, making it more likely to select incorrect parameters. While these errors were infrequent, we mention them to highlight the fact that such mistakes are not possible in the Polyscope or CoBlox due to the composition constraints designed into the system.

Results from the Post Survey

In this section, we present an analysis of participants' experiences using the different tools by analyzing responses given on the survey at the conclusion of the study protocol. The survey largely consisted of 5-point Likert scale questions. In this analysis, we group similar questions together to create composite measures for perceived ease-of-use, learnability, and satisfaction. The results of this analysis, grouped by condition, are shown in Figure 8. The survey also included free response questions, which are also included in the analysis. The survey, including how the questions were grouped, can be found in the online supplemental materials.

Ease-of-Use

Our measure for ease-of-use for each of the tools was calculated by combining responses to prompts asking about how easy the tool was to use overall, how easy it was to do specific things (like add commands or fix errors), and how user-friendly the interface was. For this section, we combined 7 prompts related to ease-of-use that were found to correlate at an acceptable level across all conditions (Cronbach's $\alpha = .86$). Comparing the aggregate ease-of-use

scores between the three conditions, we find the scores to be significantly different from each other $F(2, 65) = 3.45$, $p < .05$. A Tukey post hoc HSD calculation shows the difference to be between CoBlox and Polyscope ($p < .05$).

These results show users found the CoBlox interface to be easier to use than the other two interfaces, a finding that is further supported by the responses given to the short answer questions from the post survey. When asked what they liked about the programming environment they used, 76% of CoBlox users give a response that identified its ease of use, saying things like "*Very easy to use. I made it a point not to use the reference sheet*" and "*User friendly interface and environment*". Half of the Polyscope participants and 20% of the Flex Pendant users attended to the ease-of-use of the interface as something they liked about the environment.

A third data point that further illustrates this difference in ease of use is the number of participants in each condition who needed to ask a researcher for assistance during the study. Seven of the 22 CoBlox participants asked for help during the study, compared to 12 of the 23 Flex Pendant participants, and 14 of the 22 Polyscope participants. Again, this data reinforces that the CoBlox interface is the easiest to use of the three. *In summary,* ***users perceived CoBlox as easier to use than the other environments.***

Learnability

The composite learnability score was calculated by combining responses to seven survey prompts related to how easy the environment was to learn. Examples of these prompts include: "Overall, it was easy to learn to use", "I learned to use the whole environment quickly", and "I could use this environment without the reference sheet". The seven questions in this section correlate with each other at a level beyond the .80 level conventionally used (Cronbach's $\alpha = .86$). Running an analysis of variance calculation shows the composite learnability scores to be significantly different from each other across the three conditions $F(2, 65) = 4.93$, $p = .01$. A Tukey post hoc HSD calculation shows the differences to be significant between CoBlox and Polyscope ($p < .05$) and CoBlox and Flex Pendant ($p = .01$), with no difference being found between Flex Pendant and Polyscope.

On the free response portion of the survey, a small number of participants gave feedback related to how easy or difficult it was to learn the environment. For example, one CoBlox participant wrote: "*It was easy to learn and saw the output change immediately from my code. That makes change easy to understand.*" This type of response is different from the way participants in the Flex Pendant condition spoke about how easy it was to learn to program in that interface, giving responses such as: "*The program is not intuitive. It will require more than a tutorial to learn, and must have training.*" *In summary,* ***users perceived CoBlox as easier to learn than the other environments.***

Satisfaction

For the composite satisfaction measure, four prompts were combined, including "I am satisfied with this programming environment" and "I would recommend this tool to someone new to robot programming". These questions all correlated with each other (Cronbach's $\alpha = .86$), suggesting they are measuring the same underlying perception of the interface used. There was a significant difference in user satisfaction across the three conditions $F(2, 65) = 5.27$, $p < .01$. A post hoc comparison using a Tukey HSD shows there to be a significant difference between CoBlox and Flex Pendant ($p = .01$) and CoBlox and Polyscope ($p < .05$), with CoBlox receiving higher satisfaction scores. *In summary,* ***users were more satisfied with CoBlox than the other environments.***

Criteria	CoBlox	Flex Pendant	Polyscope
Faster Task Completion	✓		
More Correct			
Easier to Use	✓		
Easier to Learn	✓		
Higher Satisfaction	✓		

Table 2. Summary of the comparative findings

DISCUSSION

Improving the design of Robot Programming Interfaces

The first contribution of this work is showing how the block-based CoBlox design performed relative to two of the most widely-used industrial robot programming approaches. Our results, summarized in Table 2, show adult novices using CoBlox were able to successfully complete more programming tasks more quickly than those using conventional interfaces, without sacrificing accuracy. Further, CoBlox had significantly higher scores for ease-of-use, ease-of-learning, and levels of satisfaction relative to the other environments. Collectively, these findings show CoBlox, and the block-based approach to robotics programming more broadly has great potential for making industrial robotics programming more accessible to adults with little to no formal programming training.

Challenges not solved through CoBlox

CoBlox was very successful at increasing the speed and ease with which users entered instructions. However, we learn as much from what was *not* solved as what was.

First, logical errors, in which users missed steps (such as not lifting the robot arm before moving it horizontally), were prevalent in all three conditions. While being able to see readable instructions in an intuitive format *could* have helped users see when there were missing steps, it didn't. Therefore, more research is needed to provide strategies for helping users with such common mistakes, specifically thinking about how the robotics programming context can perpetuate these errors while also potentially provide design opportunities to help users not make these mistakes.

Second, the prevalence of incorrect location specification as a cause of error suggests that participants also had difficulty with the interface provided to manipulate the robot arm. This finding suggests that more effort must go into interface design. When asked about frustrations related to the programming interface, many participants (37 out of the 67) mentioned some aspects of manipulating the virtual robot. For example, one participant stated: "*the positional system is bad, you could miss the point unless you put in exactly the location. There should be more convenient 3D positioning system*." This finding replicates what we found in our small-scale pilot study [42]. This leads to potential directions for future work. One of the emerging findings that will be shaping our next iteration of this work is thinking more carefully about how to better integrate the robot positions as part of the programming task, and redesign the virtual robot space in hopes of making it more intuitive and accessible.

Bringing Innovations for Kids to Adult Environments

One of the major contributions of this work is showing the potential for taking design innovations targeted at one group of novices (in this case, young learners) and employing them for another group of novices (adults). One hypothesis we brought to this work was that the block-based approach to programming that has been successful for young learners could help adult novices. The ease of assembling programs with the drag-and-drop interface, the removal of syntax errors, the ability to define domain-specific semantics, and the affordances of the graphical presentation all contributed to making robot programming easier for adult novices.

In fact, an argument against the use of block-based languages in professional settings is the perception that block-based tools are "not real programming" and less powerful than conventional text-based tools [45]. However, growing applications of block-based programming, such as distributed computing [11], parallel computing [16], and data sciences [2], show this to not be the case. Further, block-based programming may be particularly well suited to the context of collaborative robots, which represents an in-between use of programming – someone who perhaps will never become an expert programmer but needs to program occasionally, maybe no more than one small program each day (to be run by the robot for the next day).

Adult Novices and Collaborative Robots

One of the goals of this work is to create a robot programming interface that is powerful enough to be of use in professional and industrial settings, while also being intuitive enough for adult novices. Creating such an interface is important giving the shifting nature of manufacturing and industrial jobs in the 21st century. Increasingly, positions that were once labeled "manual labor" are changing. Collaborative robots are one emerging form of the new 21st-century blue collar position that tasks workers with working alongside, and interacting with, robots. Accessible programming interfaces are essential for helping workers make this transition to working alongside autonomous machines. Further, given the focus on speed and accuracy, CoBlox may have a home in the workplace of tomorrow.

Limitations

While we tried to make the conditions as similar as possible there were some differences that introduce limitations to the study. For example, the CoBlox and Flex Pendant conditions used the same virtual robot interface, but, due to technical reasons, the Polyscope condition used a different virtual robot interface, thus introducing a difference that may influence our findings. While there was some evidence that these differences may have contributed to differing experiences for users as seen in a small number of comments, it does not seem substantial enough to explain the significant differences between conditions reported in this work given that the there was no difference in capabilities between the interfaces.

A second limitation relates to the diversity of the tasks that participants were asked to do and how it speaks to the larger universe of activities that robots can perform. The robots used in this study all had open/close grippers equipped, and the participants were asked to program a "pick and place" routine. This is a relatively narrow set of functionality that is serving as representative for all industrial robotics. While we intend on introducing additional Robot Recipes and broadening the scope of activities the language has been designed for, there is still work to be done to verify that the positive outcomes from this work remain when tasks become larger, more complicated, and more diverse.

A final limitation relates to our recruitment of participants, all of which came from the same company. This results in a lack of geographic diversity and a shared background that could potential affect our results. We view this as a relatively minor, but noteworthy, concern and something we seek to address in future iterations of this work.

CONCLUSION

The goal of this work was to explore ways of making industrial robot programming more accessible to people with little or no prior programming experience. Drawing on successful design strategies used to introduce young learners to the practice of programming, we created CoBlox and showed how it outperforms the most wide-spread robotics programming approaches used today. The analysis shows the CoBlox helped adult novices program more tasks successfully by decreasing time on task while maintaining quality. In addition, the participants found it easier to use and enjoyed it more. Collectively, with this work, we advance our understanding of ways to make robot programming more accessible to a wider range of users. We view robotics as merely a single example of a field in which a block-based interface can be used. This study shows the block-based approach making a robotics programming task easier for novice adults, providing an empirical basis for future work concerned with making programming accessible to all. In doing so, we contribute to the larger goal of giving people access to and control over the technologies around us.

REFERENCES

1. 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.
2. A. Cory Bart, J. Tibau, D. Kafura, C. A. Shaffer, and E. Tilevich. 2017. Design and Evaluation of a Block-based Environment with a Data Science Context. *IEEE Transactions on Emerging Topics in Computing* PP, 99: 1–1.
3. David Bau. 2015. Droplet, a blocks-based editor for text code. *Journal of Computing Sciences in Colleges* 30, 6: 138–144.
4. David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak. 2017. Learnable programming: blocks and beyond. *Communications of the ACM* 60, 6: 72–80. https://doi.org/10.1145/3015455
5. A Begel and E Klopfer. 2007. Starlogo TNG: An introduction to game development. *Journal of E-Learning*.
6. Geoffrey Biggs and Bruce MacDonald. 2003. A survey of robot programming systems. In *Proceedings of the Australasian conference on robotics and automation*, 1–10.
7. Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. 2008. Robot programming by demonstration. In *Springer handbook of robotics*. Springer, 1371–1394.
8. Rainer Bischoff, Arif Kazi, and Markus Seyfarth. 2002. The MORPHA style guide for icon-based programming. In *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, 482–487.
9. Douglas Blank, Deepak Kumar, Lisa Meeden, and Holly Yanco. 2006. The Pyro toolkit for AI and robotics. *AI magazine* 27, 1: 39.
10. Cynthia Breazeal and Brian Scassellati. 2002. Robots that imitate humans. *Trends in cognitive sciences* 6, 11: 481–487.
11. Brian Broll, Akos Lédeczi, Peter Volgyesi, Janos Sallai, Miklos Maroti, Alexia Carrillo, Stephanie L. Weeden-Wright, Chris Vanags, Joshua D. Swartz, and Melvin Lu. 2017. A Visual Programming Environment for Learning Distributed Programming. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '17), 81–86. https://doi.org/10.1145/3017680.3017741
12. J. Edward Colgate, J. Edward, Michael A. Peshkin, and Witaya Wannasuphoprasit. 1996. *Cobots: Robots For Collaboration With Human Operators*.
13. S. Cooper, W. Dann, and R. Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges* 15, 5: 107–116.
14. C Duncan, T Bell, and S Tanimoto. 2014. Should Your 8-year-old Learn Coding? In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (WiPSCE '14), 60–69.
15. Sarah Esper, Stephen R. Foster, and William G. Griswold. 2013. CodeSpells: embodying the metaphor of wizardry for programming. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, 249–254.
16. Annette Feng, Eli Tilevich, and Wu-chun Feng. 2015. Block-based programming abstractions for explicit parallel computing. In *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE*, 71–75.
17. D Franklin, G Skifstad, R Rolock, I Mehrotra, V Ding, A Hansen, D Weintrop, and D Harlow. 2017. Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-based Curriculum. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '17), 231–236.
18. N. Fraser. 2015. Ten things we've learned from Blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, 49–50. https://doi.org/10.1109/BLOCKS.2015.7369000
19. E. Freund and B. Luedemann-Ravit. 2002. A system to automate the generation of program variants for industrial robot applications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1856–1861 vol.2.
20. Thomas A. Fuhlbrigge, Gregory Rossano, Hui Zhang, Jianjun Wang, and Zhongxue Gan. 2010. Method and apparatus for developing a metadata-infused software program for controlling a robot.
21. Shuchi Grover, Roy Pea, and Stephen Cooper. 2015. Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education* 25, 2: 199–237.
22. Martin Hägele, Walter Schaaf, and Evert Helms. 2002. Robot assistants at manual workplaces: Effective co-operation and safety aspects. In *Proceedings of the 33rd ISR (International Symposium on Robotics)*, 7–11.
23. Robert Hopler and Martin Otter. 2001. A versatile C++ toolbox for model based, real time control systems of robotic manipulators. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, 2208–2214.
24. M. S Horn, C Brady, A Hjorth, A Wagh, and U Wilensky. 2014. Frog pond: a codefirst learning environment on evolution and natural selection. In *Proceedings of the 2014 conference on Interaction design and children*, 357–360.
25. Andri Ioannidou, Alexander Repenning, and David C. Webb. 2009. AgentCubes: Incremental 3D end-user development. *Journal of Visual Languages & Computing* 20, 4: 236–251.
26. C. Kelleher and R. Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming

environments and languages for novice programmers. *ACM Computing Surveys* 37, 2: 83–137.
27. S. Kock, T. Vittor, B. Matthias, H. Jerregard, M. Källman, I. Lundberg, R. Mellander, and M. Hedelind. 2011. Robot concept for scalable, flexible assembly automation: A technology study on a harmless dual-armed robot. In *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM)*, 1–5. https://doi.org/10.1109/ISAM.2011.5942358
28. Daisuke Kushida, Masatoshi Nakamura, Satoru Goto, and Nobuhiro Kyura. 2001. Human direct teaching of industrial articulated robot arms based on force-free control. *Artificial Life and Robotics* 5, 1: 26–32.
29. Tom Lauwers and Illah Nourbakhsh. 2010. Designing the finch: Creating a robot aligned to computer science concepts. In *AAAI Symposium on Educational Advances in Artificial Intelligence*.
30. Lego Systems Inc. 2008. *Lego Mindstorms NXT-G Invention System*. Retrieved from http://mindstorms.lego.com
31. Makeblock Co., Ltd. 2017. mBot. Retrieved September 18, 2017 from http://www.makeblock.com/
32. J. H Maloney, M Resnick, N Rusk, B Silverman, and E Eastmond. 2010. The Scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4: 16.
33. J Maloney, M Nagle, and J Mönig. 2017. GP: A General Purpose Blocks-Based Language. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (SIGCSE '17), 739–739.
34. Microbric Pty, Ltd. 2017. *Edison Programmable Robot*. Retrieved from https://meetedison.com/
35. Brad A. Myers. 1990. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing* 1, 1: 97–123.
36. Ozobot & Evollve, Inc. 2017. *Ozobot*. Retrieved from http://ozobot.com/
37. Zengxi Pan, Joseph Polden, Nathan Larkin, Stephen Van Duin, and John Norrish. 2012. Recent progress on programming methods for industrial robots. *Robotics and Computer-Integrated Manufacturing* 28, 2: 87–94. https://doi.org/10.1016/j.rcim.2011.08.004
38. J. N. Pires, K. Nilsson, and H. G. Petersen. 2005. Industrial robotics applications and industry-academia cooperation in Europe. *IEEE Robotics Automation Magazine* 12, 3: 5–6.
39. Mitchell Resnick, Brian Silverman, Yasmin Kafai, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, and Jay Silver. 2009. Scratch: Programming for all. *Communications of the ACM* 52, 11: 60.
40. Wolfgang Slany. 2014. Tinkering with Pocket Code, a Scratch-like programming app for your smartphone. In *Proceedings of Constructionism 2014*.
41. A. Strauss and J. Corbin. 1994. Grounded Theory Methodology: An Overview. In *Strategies of Qualitative Inquiry*. Sage Publications, Inc, Thousand Oaks, CA, 158–183.
42. D. Weintrop, D. C. Shepherd, P. Francis, and D. Franklin. 2017. Blockly goes to work: Block-based programming for industrial robots. In *2017 IEEE Blocks and Beyond Workshop*, 29–36. https://doi.org/10.1109/BLOCKS.2017.8120406
43. D Weintrop and U Wilensky. In Press. Comparing Blocks-based and Text-based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education (TOCE)*.
44. D. Weintrop and U. Wilensky. 2012. RoboBuilder: A program-to-play constructionist video game. In *Proceedings of the Constructionism 2012 Conference*.
45. D Weintrop and U. Wilensky. 2015. To Block or Not to Block, That is the Question: Students' Perceptions of Blocks-based Programming. In *Proceedings of the 14th International Conference on Interaction Design and Children* (IDC '15), 199–208. https://doi.org/10.1145/2771839.2771860
46. M. H. Wilkerson-Jerde and U. Wilensky. 2010. Restructuring Change, Interpreting Changes: The DeltaTick Modeling and Analysis Toolkit. In *Proceedings of the Constructionism 2010 Conference*.
47. David Wolber, Hal Abelson, Ellen Spertus, and Liz Looney. 2014. *App Inventor 2: Create Your Own Android Apps*. O'Reilly Media, Beijing.
48. Wonder Workshop, Inc. 2017. *Dash & Dot*. Retrieved from https://www.makewonder.com/
49. Y. F. Yong and M. C. Bonney. 1999. Off-Line Programming. In *Handbook of Industrial Robotics*, Shimon Y. Nof (ed.). John Wiley & Sons, Inc., 353–371. https://doi.org/10.1002/9780470172506.ch19
50. Open Roberta. Retrieved September 18, 2017 from https://www.open-roberta.org/en/welcome/