Blockly Goes to Work: Block-based Programming for Industrial Robots

David Weintrop
College of Education
College of Information Studies
University of Maryland
weintrop@umd.edu

David C. Shepherd
ABB Corporate Research
david.shepherd@us.abb.com

Patrick Francis
ABB Corporate Research
patrick.francis@us.abb.com

Diana Franklin
UChicago STEM Education
Department of Computer Science
University of Chicago
dmfranklin@uchicago.edu

***Abstract*— The block-based approach to programming is an effective way to engage young learners in programming and the powerful ideas of computing. In this paper, we explore the potential of using this same approach in a very different programming context: industrial robotics. Using a customized language built with the Blockly library, we created a block-based interface for programming a one-armed industrial robot. This paper presents a block-based robot programming language called Robot Blockly, focusing on how the various affordances of block-based programming were utilized to make the challenge of robot programming more accessible. We also present results from a small-scale study showing adults with no prior programming experience successfully programming a virtual robot to accomplish a pick and place task. The contribution of this work is in showing the potential for block-based programming beyond young learners and classrooms.**

Keywords—block-based programming, robot programming, graphical programming

I. Introduction

Technology is changing our world. This can be seen across diverse domains and sectors, including industrial manufacturing, where manual labor jobs once filled by large numbers of individuals are steadily being automated with machines. In the United States, despite record levels of manufacturing output, manufacturing jobs are stagnant [1]. While there are many factors that contribute to this trend, one component is the improvement of technological infrastructure and the introduction of automated elements to the manufacturing process [2]. Whereas American manufacturing jobs have declined, particularly in less technologically-intensive sectors, other countries have seen less job loss by restructuring their manufacturing base to emphasize more technologically intensive sectors [3]. Similarly, others have argued that the emergence of technology and automation does not replace jobs, instead; it changes them and the skills needed by members of the workforce [4].

In this new, computationally-driven manufacturing economy, programming is becoming a valuable skill for workers to be able to contribute and succeed. However, programming takes years to master, especially given the fact that the current programming languages used in industrial settings are designed by engineers, for engineers. This means writing the programs necessary to enable the shift toward automation requires years of training, often resulting in the need to hire expensive specialists in order to program even the most basic routines. A side effect of this reality is that robotics projects requiring programming often stall or are not even considered due to the expense associated with hiring developers to implement the desired automation routines. However, advances in the design of programming environments for novices may present an alternative path forward for robot programming.

Alongside the growth in computationally-intensive manufacturing jobs, there has been substantial progress made in the design of tools and technologies to make programming more accessible and intuitive. In particular, the emergence of the block-based programming paradigm has resulted in dozens of programming tools that have introduced millions of young learners to the powerful concepts of computing [5]. This includes robotics construction kits and toys like Lego Mindstorms, Dash and Dot, and Ozobots.

While early work in end-user programming and visual programming languages interfaces was driven by the goal of making computers and programming accessible to professionals [6], the last twenty years of design innovation to make programming more accessible has largely focused on younger learners [7], [8]. The successes of these child-oriented innovations suggest that lessons learned from this work might also be effective for industrial robot programming. In this work, we seek to bring these two lines of work back together, investigating if and how block-based programming can be used in service of professional ends, specifically, the task of industrial robotics programming. This paper introduces Robot Blockly, a block-based programming interface for ABB's Roberta, a single-armed industrial Robot (Fig. 1) and present results from a user study showing novices with no prior programming experience successfully writing programs to accomplish basic robotics tasks.

978-1-5386-2480-7/17/$31.00 ©2017 IEEE

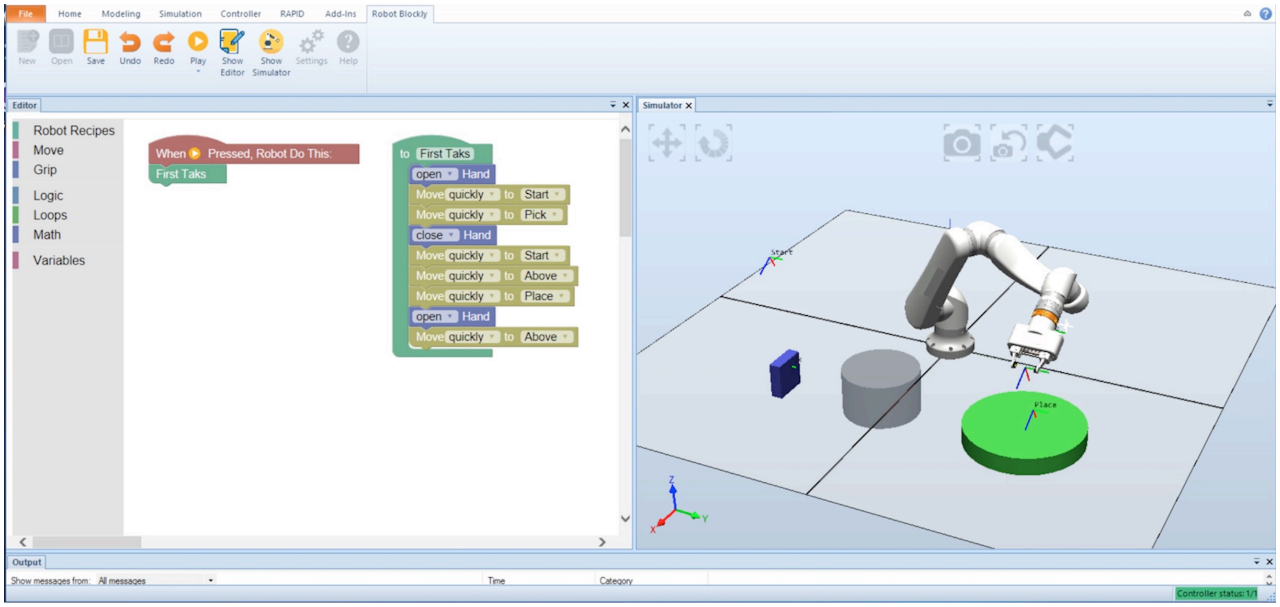


Fig. 1. The Robot Blockly programming environment. The left side of the environment contains the block-based robot programming interface for Roberta, shown on the right.

We begin this paper by reviewing prior work that informed the design of Robot Blockly. We then introduce Robot Blockly, describing features of the environment and how it takes advantage of the affordances of block-based programming in order to make the task of programming an industrial robot more accessible. Next, we present the user study including the study design, participant information, and findings. The paper concludes with a discussion of the results, the larger potential of block-based programming beyond younger learners, and next steps for this line of work.

II. Prior Work

In this section, we review the three main literatures that informed this work: end-user programming for robots, graphical approaches to robot programming, and block-based programming.

A. End-User Programming for Robotics

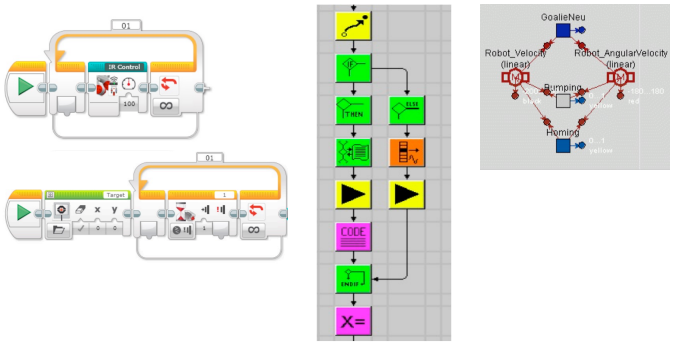
End-user programming is defined as programming to achieve the result of a program primarily for personal, rather public use" [9]. In the case of robot programming, this means the author is writing a routine for a specific, immediate task, as opposed to creating a general-purpose program or a template script that others will later modify. Our focus on end-user robotics programming languages is due to our interest in making the power of industrial robots accessible to a wider audience of potential users, including those employed in the industrial sector as well as entrepreneurs and small business owners.

For as long as there have been robots, there have been robot programming languages. Lozano-Pérez [10], in an early paper on the landscape of robot programming tools, broke the space down into three over-arching categories: guiding systems, robot-level programming systems, and task-level programming system. To date, guiding systems, notably, the programming-by-demonstration approach [11], has been the predominant end-user programming strategy used in robotics. This type of programming involves physically moving the robot into a desired position and then recording it. Programming-by-demonstration is often used in conjunction with text-based robotics programming. A more recent categorization of robot programming focuses more closely on characteristics of the programming task, breaking the robot programming landscape into automatic programming tools (e.g. programming-by-demonstration, learning systems, etc.), manual programming tools (e.g. text-based programming languages, flowchart systems, etc.), and software architectures (tools and libraries used to support the robot programming tasks) [12]. The manual programming category is divided into text-based and graphical programming systems, with the graphical subgroups being further decomposed into graph systems, flowchart systems, and diagrammatic systems. Within this taxonomy, the work we present in this paper falls under the graphical programming system categorization, but in a new sub-genre, the authors did not include: Block-based programming. Given this positioning, we present a more detailed review of graphical robot programming systems in the next section.

B. Graphical Robot Programming

Graphical programming replaces 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 [13]. A number of graphical programming tools have been created to support robot programming. The most well-known of which is the Lego Mindstorms tool (Fig. 2a), which uses visual blocks to represent basic robot actions which the user can organize to produce desired outcomes [14]. This approach shares features with the block-based approach we

use with Robot Blockly that is the focus of this paper. The major difference between Robot Blockly and Lego Mindstorms is the role that text plays and the closeness of mapping between the graphical programming interface and text-based alternatives. A second graphical approach to robot programming can be seen with MORPHA (Fig. 2b), which uses an icon-based approach and flowchart-like layout to let users define instructions for their robot [15]. MORPHA was intended to be used in industry but never achieved widespread adoption, in part due to the challenge of interpreting its symbols. A third example of a graphical programming tool can be seen with the DD-Designer (Fig. 2c), which takes a behavior-based approach and uses a data processor hypergraph layout to give the author control over the robot. In presenting these different graphical approaches to robot programming we are trying to highlight different strategies taken to date that can be contrasted with the block-based approach we user for Robot Blockly.



(a) (b) (c)

Fig. 2. Three examples of graphical robot programming tools: (a) Lego Mindstorms, (b) MORPHA, and (c) DD-Designers.

C. *Block-based Programming*

The block-based programming approach used in Robot Blockly blends affordances of the graphical approach to robot programming discussed above with characteristics of conventional text-based programming. Block-based programming 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. Writing a program in a block-based environment takes the form of dragging-and-dropping instructions together 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 having a drag-and-drop assembly mechanism that is easier and faster than typing command character-by-character with the keyboard [16]. A growing body of literature is showing 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]–[19].

Led by the popularity of block-based tools including Scratch [20] and Alice [21], there is a growing ecosystem of block-based environments that support a variety of programming activities. Alice [21], and other block-based tools like AgentCubes [22], are noteworthy in that they allow the user to program simulations in three dimensions, akin to the type of movements supported in Robot Blockly. 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 [23]–[25], mobile application development [26], [27], playing video games [28], [29], and manipulating media [30]. 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 [31], [32]. Finally, the block-based programming approach has been used in robotics kits for kids, like the aforementioned Lego Mindstorms, as well as Open Roberta, Dash and Dot, and the Finch Robot. Collectively, the variety of applications for which block-based programming has been applied, along with the growing evidence of its effectiveness, suggests there is potential for bringing this programming approach to the world of industrial robotics programming.

III. Meet Robot Blockly

Robot Blockly (Fig. 1) is a block-based programming environment designed for Roberta, a one-armed industrial robot. Using the robot programming categorizations discussed in the literature review [10], [12], Robot Blockly is a robot-level programming system and a manual programming tool. As shown in Fig. 1, the interface of Robot Blockly is broken down into two distinct panes: the Blockly pane and the Robot pane. The Blockly pane contains the block-based programming interface in which programs are defined, while the Robot Pane shows a virtual version of Roberta and is used to both position the robot during program construction and watch a program run after it is completed. The program shown in Fig. 1 is a pick and place routine that was authored by a participant in the study, running this program results in the virtual robot picking up the blue block and placing it on the green pedestal.

Writing a program with Robot Blockly requires users to move back and forth between the two panes of the interface. Users start by defining a set of steps for the robot to follow by dragging-and-dropping commands and placing them under the maroon colored start block (shown in Fig. 4a). Users define movement commands by adding the `move` block to their program. The text on the `move` block reads: `Move quickly to <somewhere>`, with `quickly` and `<somewhere>` being dropdown menus that allow the user to customize the movement of the robot arm. To tell the robot arm where to move, the user creates a *Location*, which defines and names a robot position, including its x, y, and z coordinate in the Robot

pane and the orientation of the gripper. To do this, the user selects the `Add Location` option in the dropdown showing `<somewhere>`. When this happens, the Robot pane becomes active, with arrows emerging from the robot's gripper along the x, y, and z axes (Fig. 3a). The user can then click-and-drag on the three arrows to position the robot arm. Once the robot is in position, the user clicks a check box at the top of the screen, gives a name to the *Location* (e.g. `Start`, as seen in Fig. 1). Once the *Location* is defined, control returns to the Blockly pane, and the `<somewhere>` text in the dropdown is replaced with the newly entered name. The resulting command now reads: `Move quickly to Start`. This process is similar to the programming-by-demonstration approach commonly used in robotics programming, 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.

At any point during program development, the user can click the *play* button at the top of the interface to watch a simulation of Roberta carrying out the programmed instructions. When the user clicks the play button, the Robot Blockly instructions are transpiled into the pre-existing text-based Robot programming language that Roberta is traditionally programmed with. In this way, the Robot Blockly language can be thought of as a layer of abstraction that lives on top of the native robot programming language in order to make programming more intuitive and accessible.

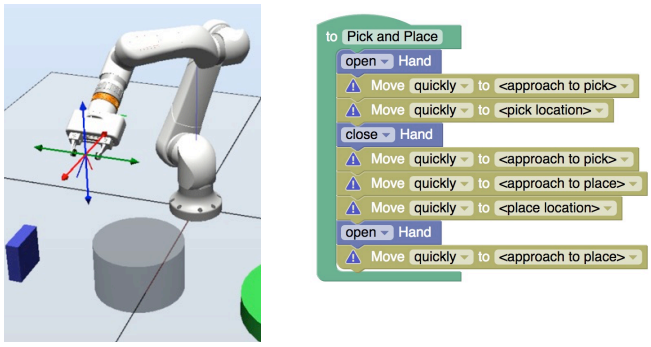


Fig. 3. (a) The Robot pane interface when users are defining a *Location*. (b) The `Pick and Place` *Robot Recipe*.

A. The Robot Blockly Language

The block-based language used in Robot Blockly is designed specifically for the task of programming Roberta and takes advantage of a number of affordances made possible by the block-based modality. The goal of the language is to abstract away unnecessary detail from the user and present an easily-understood set of instructions to programming novices. Along with conventional programming constructs (like conditional logic, looping commands, and variables), the Robot Blockly language includes three custom block types. The first unique command controls the robot's gripper. The text on the block reads `open hand`, where `open` is a dropdown menu with two options: `open` and `close`. The second custom block in Robot Blockly is the `move` block discussed in this last section. The `move` command reads `Move quickly to <somewhere>`, where `quickly` is a dropdown containing the options: `quickly`, `moderately`, and `slowly`, which control the speed of the robot movements. The second dropdown has the default value of `<somewhere>` and includes all the *Locations* that have been defined in the program along with an option to define a new *Location*.

The final custom command in Robot Blockly's language is the inclusion 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`, shown in Fig. 3b. The `Pick and Place` recipe defines the sequence of steps a robot follows in order to pick up an object in one location and place it somewhere else; which is a very common task for industrial robots to carry out. *Robot Recipes* are comprised of blocks available to the user, with suggestive 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 the first *Location* to be defined is where you want to put the robot arm ahead of its approach to the pickup position.

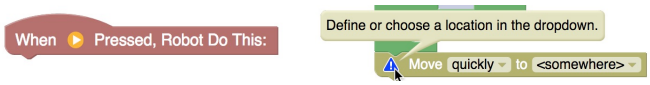
B. Block-based Affordances in Robot Blockly

The design of Robot Blockly takes advantage of a number of the affordances that come with block-based programming. These affordances include many of the features common to block-based environments, such as the visual cues on blocks denoting how they can be used and the ease of discovering new commands due to the organization and presentation of commands in the block drawers on the left-hand side of the programming canvas. Additionally, Robot Blockly retains the "tinkerability" of block-based programming environments, meaning it is easy to try things out and make small, incremental changes while developing a program. Beyond these features common across all block-based environments, the design of Robot Blockly further leverages four block-based programming features which we will discuss in greater detail below: (1) its use of natural language expressions in programming commands, (2) the ability to include images alongside text in commands, (3) the dynamic rendering (and re-rendering) of commands, and (4) the logical organization of scripts on the blocks canvas.

The first affordance of block-based programming utilized by Robot Blockly is the ability for the labels on the commands to use natural language expressions and images to convey meaning and greatly simplify both the comprehension of existing programs and the composing of new programs. For example, the *Location* construct in Robot Blockly allows users to provide a simple label (like `Start`) to define the exact position and orientation in a program for Roberta. A single *Location* in a Robot Blockly program replaces a set of 17 numbers that would otherwise need to be typed in to specify the exact position of each component of the robot arm. A second example that further demonstrates the power of natural language in Robot Block is the `Move` command that has been

referenced a number of times in this manuscript. A call to `Move`, which in Robot Blockly might read: `Move quickly to start` would take the following form in the conventional Roberta programming language: `MoveJ rb_Location1, v1000, fine, tGripper, \WObj:=blocklyWobj_1`, where `Movej` defines the type of movement, `rb_Locaiton1` is the 17-argument position mentione above, `v1000` is the speed of the movement, `fine` defines the desired level of accuracy, `tGripper` defines the tool attached to the end of the robot, and finally, the `/WObj` expression further defines characteristics of the environment. Additionally, there are other move commands, such as `MoveL`, that users need to distinguish between, further complicating the creation of relatively simple programs. While the Robot Blockly version of this command loses some of the detailed control that the conventional text version has (e.g. the user cannot change the level of accuracy for the movement), doing so makes the commands clearer and hides details that are not necessary for a majority of uses, especially routine behaviors.

Another example of how Robot Blockly takes advantage of the ability of the block-based modality is its blending of text and images within blocks, as can be seen in the start block (Fig. 4a). In this case, it embeds the image of the play button of the runtime environment into its label to help users link the instructions added under the block with the button that needs to be pushed to begin execution of the program.



(a) (b)

Fig. 4. (a) Robot Blockly's start block that blends text and images. (b) The warning icon and accompanying message for not-yet-configured `Move` commands.

A third way that Robot Blockly takes advantage the block-based programming approach is in its use of the dynamic rendering capabilities of blocks to present additional information. When a `Move` block is added to a program, it starts with a default *Location* argument of `<somewhere>`, which serves as a placeholder until the user defines it. With Blockly's dynamic rendering feature, we can add a warning icon with a message letting users know that the block needs to be configured before it is used, shown in Fig. 4b. Further, after a *Location* is defined, the block automatically updates to use the new *Location*.

The final way that Robot Blockly leverages features of the block-based approach to programming is in its ability to automatically organize programs into organized columns, as can be seen in Fig. 1. The left column of Robot Blockly will always be the main function of the program, with the second column comprised of the other blocks used in the program (most often being *Robot Recipes*, but also sometimes includes blocks put off to the side). The outcome of this features is the users can easily see all of their commands at the same time with calls to *Robot Recipes* being positioned alongside the recipe definitions.

IV. Methods and Participants

Having introduced Robot Blockly and discussed some of the ways that the environment utilizes features of the block-based approach to programming, we now present the methods and study design used for our initial evaluation of the environment. This paper presents results from a small-scale user study where participants were asked to write two basic programs in the Robot Blockly environment in a one-on-one interview setting. The interviews began with a short (7 minute) video that introduced the Robot Blockly environment, demonstrating how to write programs, manipulate the virtual robot, and run programs. After the video, participants were given a double-sided reference sheet that summarized the information presented in the video, which was intended to be used throughout the programming tasks to help remind participants of features of the tool. After this introductory portion, participants were put in front of a laptop running the Robot Blockly environment (in the same configuration depicted in Fig. 1) and given their first programming task: to write a pick and place routine to pick up the blue block and place it on the green pedestal. After completing this first programming task, the environment was reset (meaning the blue block was put back to its starting position and the program was deleted) and participants were given the second programming task: pick up the blue block, "dunk" it into the silver container, and then place it on the green pedestal. The idea behind the design of these tasks was to first ask participants to author a conventional pick and place routine, either with the `Pick and Place` *Robot Recipe* or on their own, and then attempt to write a modified pick and place routine, where additional steps are required. At the conclusion of the programming portion of the interview, participants were asked a series of questions about their experience working with Robot Blockly. The programming and post interview portions of the interview lasted an average of 33 minutes and 15 seconds combined and were recorded using software that captured both the on-screen actions along with audio and video using the laptop's camera and microphone.

The data presented below are from 5 interviews conducted with adults affiliated with an education research center in the American Midwest. They were recruited through an introductory email with the primary qualification for inclusion in the study being that they have no prior programming experience. Below we present preliminary findings, documenting both successes and challenges identified through analyzing the collected data.

V. Findings

Given the small scale of the study, we briefly provide summative data across the five sessions, then focus the majority of our analysis on qualitative findings from the interviews. The qualitative analysis first looks at successes of Robot Blockly, then discusses challenges the participants encountered.

All five participants were able to write a successful program to carry out a pick and place routine (although some technical issues prevent all successful programs from being

observed). One of these programs is shown in Fig. 1, with two other programs that did not use *Robot Recipes* presented in Fig.5. It is important to note that none of these participants would have been able to complete a program using the existing text-based programming language currently required to control Roberta. Of the five participants, only one successfully implemented the second task, with two other participants starting the task but unable to complete it due to technical issues with the software. Both of these participants were able to verbally explain the programs they intended on writing. Two of the five participants chose to use *Robot Recipes* in their projects. When one of the participants who chose not to use a *Robot Recipe* was asked why she chose to implement the algorithm from scratch, she laughed, then said: "*I completely forgot about it, it's also fun to just try it out on your own.*" At the conclusion of the interview, participants were asked if they could see uses for this type of industrial robot in their homes or in their professional lives. All five participants gave meaningful responses, including tasks such as folding clothes and compiling materials into folders in preparation for teacher workshops.

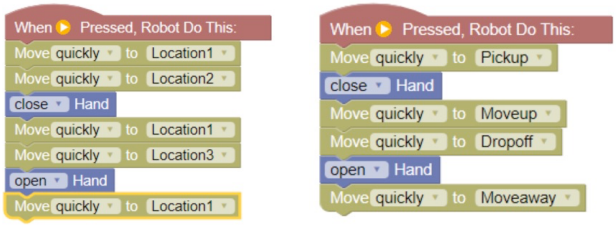


Fig. 5. Two successful pick and place programs written by participants.

A. *Successful Aspects of Robot Blockly*

This small user study of the Robot Blockly environment revealed a number of successful aspects of the design, as well as some challenges that still remain. The successes included (1) writing programs in a block-based environment, (2) using *Robot Recipes* to help structure and complete the program, (3) using *Locations* effectively to write a robot routine, and (4) making it so that the programming was not seen as the challenging part of controlling a virtual robot. We discuss each of these four successes below, using data from the interviews to illustrate each point.

The first success to note was that, as mentioned above, all participants made progress towards writing a successful robot program. As one participant reflected: "*Once I got the steps going, it seemed pretty easy. You just need to remember to tell it to do everything, like things we take for granted, like grabbing stuff, you got to make sure to tell it to grab, and then pick it up and make sure to tell it to release.*" What is interesting about this quote is how this brief robot programming task seems to have effectively conveyed one of the central ideas about programming: the need to be explicit.

We also see the visual block-based programming approach supporting the programmers in the same way as has been reported in the literature. As one participant said during her session when trying to add a *Robot Recipe* directly to her program: "*I wanted to put that under there but it is not going to snap, I can see it is not going to snap.*" In this case, the visual rendering of the *Robot Recipe* conveyed to her the information that it could not be added to the program directly, but instead, must be called with another block. Here, we see a programming novice attend to both the shape and sound associated with the blocks to explain how she knew what was possible with the given command.

Fig. 1 shows a final program written by one of the participants in the study. As can be seen on the left-hand side of the environment, the participant used a *Robot Recipe*, which she called `First Task`, to carry out the pick and place routine. In creating the program, the participant dragged out the `Pick and Place` block from the *Robot Recipes* drawer, then systematically went through the `Move` commands from top to bottom, defining and renaming each *Location* in the program. When asked about the role of the recipe at the conclusion of the interview, this participant said "*I found the recipe really helpful just as a template to start thinking about it. It helped me, sort of solidify what the commands did and what I could edit about them.*"

One of the more successful design aspects of Robot Blockly based on the user study was the construct of *Locations*. Four of the five participants renamed the *Locations* with meaningful labels like `Pickup`, `Dropoff`, (as seen in Fig 5) and the four *Location* names used in Fig. 1: `Start`, `Pick`, `Above`, and `Place`. Further, two of the three participants who worked on the second programming task reused *Locations* they had defined for the first task. We view these as promising findings suggesting the notion of *Locations*, in conjunction with the natural language expressiveness of block-based programming, as being a successful design feature of Robot Blockly.

A final aspect of the robot programming task that we observed in this study can be viewed as both a success and an outstanding issue. Across all five interviews, the biggest challenge for the participants was positioning the virtual robot in the correct place. This fact is reflected in the breakdown of time during the interview, with participants spending an average of 4 minutes and 19 seconds working in the Blockly pane compared to an average of 9 minutes and 25 seconds in the Robot pane. Participants comments during the post interview further support the interpretation of the robot positioning being a major challenge. As one participant put it: "*it's frustrating to see if you're really over [the block]. That was the hardest part for me.*" We view this type of response as successful in that the programming component of the activity was not seen as the hard part. In this way, the block-based design was successful in making the act of defining the sequence of instructions for the robot accessible. As another participant said: "*So the writing the program part, I didn't find that as hard, but moving things around on the screen was harder.*" However, we don't view this feedback as a complete success as positioning the robot arm is an essential part of this type of programming task, so while block-based programming helped with part of the challenge, there is still design work to do to further lower the threshold to entry.

B. Remaining Challenges in Robot Blockly

While some aspects of the Robot Blockly were successful in supporting our novice programmers, the user study also revealed some remaining challenges. In this section, we discuss four challenges we identified in our analysis of the Robot Blockly interviews: 1) instances where participants struggled with the block-based programming interface, 2) conceptual issues related to initialization, 3) positioning the robot arm, and 4) making clear the relationship between the Robot pane and the Blockly pane.

First, while there were many successes related to adult novices assembling instructions in the block-based interface, it is important to note that the block-based authoring interaction was not without its issues. For example, one participant really struggled when trying to modify the `Pick and Place` *Robot Recipe* to include extra steps for the dunk portion of the second programming challenge. She started by trying to modify the position of an already defined *Location*. After redefining it, she went back to the recipe and clicked the dropdown, got a confused look on her face saying "*I want to add a command,*" and then explained how she had hoped redefining the *Location* would introduce the extra step of the recipe. Only after the interviewer intervened and showed her how she can insert blocks inside the recipe by dragging-and-dropping them into place was it clear how to proceed, with her saying "*oh, you can drag that one down, ok and then stick it in there, ok.*" This utterance suggests it was not clear that drag-and-drop composition approach could be used to insert new commands. The take away from this episode is a recognition that there is still work to do on the design of both the interface and the instructional materials to help novices understand the block-based programming approach.

A second challenge matches a finding from prior work on teaching younger learners to program and relates to the notion of initialization [33]. The `Pick and Place` *Robot Recipe* starts by telling the robot to open its hand before moving the arm into place. At the start of our interview protocol, the robot starts with an open hand. As a result, none of the participants who wrote their own sequence of steps (i.e. did not use a *Robot Recipe*) included the `open hand` command at the start of their program. As a result, there were times when users reran their programs but the robot's hand was closed, causing their programs to not work because the robot hand was closed and could not pick up the block. Further, in the case of one participant, she dragged the `Pick and Place` *Robot Recipe* onto the canvas with the intention of using it, but upon seeing it start with the `open hand` command, she deleted the recipe and started writing her own routine that began with movement commands. This outcome of novices not considering program initialization has been documented in the literature as a challenge faced by novice programmers [33] and highlights another remaining design challenge for Robot Blockly.

The third issue we identified, which was mentioned previously, relates to correctly positioning the robot arm in the three-dimension workspace. Both the training video and the Robot Blockly reference sheet included instructions on how to navigate the three-dimensional space (including panning and rotating the perspective as well as moving the robot arm and hand), but these resources ended up not being sufficient. Despite this instruction, all five participants spent substantial time adjusting, and readjusting the robot arm. As one participant said: "*I had a little trouble navigating the screen and dragging the robot arm around,*" with another participant echoing these sentiments: "*That was the hardest part, just like, moving the arm, just because I'm not familiar with it*."

A final challenge, related to the first, is making clearer the relationship between the block-based programming interface and the virtual Robot pane. In our design, when the user chooses the `Add Location` option in the dropdown the Robot Pane becomes active (as shown in Fig. 3a). A number of participants (3 of the 5) moved the robot arm into place, then went back to editing the program without finalizing the *Location* definition (by clicking the done checkbox). Unfortunately, in the prototype version of Robot Blockly used in the interviews, returning to the program before finalizing the *Location* definition often lead to the program falling into an invalid state where the program could no longer be run. This happened a number of times and prevented some participants from having time to work on the second programming task. This issue has already been resolved in the most recent version of Robot Blockly.

VI. Discussion

Having presented the design of Robot Blockly and some preliminary results from a small-scale user study, we now conclude with a brief discussion of the findings and next steps for this line of work. The first take away from this study is the promising results from bringing design innovations from the creation of programming environments for young learners into new novice programming contexts for older learners. This strategy may become increasingly adopted as more professionals, technologies, and activities incorporate programming in some capacity. One emerging finding from this study is that when presenting a virtual robot programming task with a block-based interface, the largest hurdle was not with the programming part of the task, but instead, was tied to controlling the three-dimensional virtual robot on a two-dimensional screen.

A second discussion point from this work is considering how to design programming environments for adult novices. Shifting the framing of introductory programming tools from being designed for young learners towards being design for programming novices of all ages may serve as a productive first step for thinking about how to bring advances from educational contexts to the wider set of potential users. That being said, there are also clear differences between adult novice programmers and younger learners. Understanding the set of differences between these two sets of novice learners is one line of research that is worth pursuing to better understand how to design for adult novices. Such work has begun for conventional text-based languages [34] but has not yet been investigated for block-based introductory tools.

The work presented above serves as an initial investigation into the potential of block-based programming for industrial

robots. As such, there are a number of future directions planned for this work, both methodologically and in terms of the design of Robot Blockly. On the methodological front, we are in the process of designing a comparative study to better understand how Robot Blockly performs relative to other types of end-user robot programming tools. In terms of the design of Robot Blockly, we are investigating better ways to integrate the Blockly and Robot panes. We are also working on building out a suite of *Robot Recipes* to include other commonly performed tasks beyond pick and place. Finally, we are beginning to develop educational materials to accompany Robot Blockly and thinking about how Robot Blockly might fit into existing vocational education contexts.

VII. Conclusion

The ability to program is becoming increasingly useful in our digital world. In response to this trend, a growing number of introductory programming approaches have been developed to make the task of programming more accessible and more intuitive. Much of the design effort to date has focused on younger learners in hopes of preparing them for future computer science instruction. In this work, we seek to bring those design innovations to the current landscape of industrial robotics. In creating an accessible, block-based programming interface for industrial robots, we hope to make the task of programming such machines accessible to individuals who otherwise would not be able to use them in their work. Likewise, in recognition of the changing nature of the workforce, especially in the manufacturing sector, tools like Robot Blockly may be able to better prepare workers for the increasingly technological nature of manufacturing and industrial jobs. While there is still much work to be done, the findings from this study suggest that block-based programming may have a home beyond the classrooms and computer clubhouse where it first grew up.

References

[1] M. J. Hicks and S. Devaraj, "The myth and the reality of manufacturing in America," CBER Ball State University, 2015.

[2] D. Dorn, G. H. Hanson, and others, "Untangling trade and technology: Evidence from local labour markets," *Econ. J.*, vol. 125, no. 584, pp. 621–46, 2015.

[3] R. D. Atkinson, L. A. Stewart, S. Andes, and S. Ezell, "Worse than the Great Depression: What experts are missing about American manufacturing decline," *Wash. DC Inf. Technol. Innov. Found.*, pp. 327–339, 2012.

[4] H. David, "Why are there still so many jobs? The history and future of workplace automation," *J. Econ. Perspect.*, vol. 29, no. 3, pp. 3–30, 2015.

[5] D. Bau, J. Gray, C. Kelleher, J. Sheldon, and F. Turbak, "Learnable programming: blocks and beyond," *Commun. ACM*, vol. 60, no. 6, pp. 72–80, May 2017.

[6] A. F. Blackwell, K. N. Whitley, J. Good, and M. Petre, "Cognitive factors in programming with diagrams," *Artif. Intell. Rev.*, vol. 15, no. 1–2, pp. 95–114, 2001.

[7] C. Duncan, T. Bell, and S. Tanimoto, "Should Your 8-year-old Learn Coding?," in *Proc. of the 9th WiPSCE*, NY, USA, 2014, pp. 60–69.

[8] C. Kelleher and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Comput. Surv.*, vol. 37, no. 2, pp. 83–137, 2005.

[9] A. J. Ko *et al.*, "The State of the Art in End-user Software Engineering," *ACM Comput Surv*, vol. 43, no. 3, p. 21:1–21:44, Apr. 2011.

[10] T. Lozano-Pérez, "Robot programming," *Proc. IEEE*, vol. 71, no. 7, pp. 821–841, 1983.

[11] A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Springer handbook of robotics*, Springer, 2008, pp. 1371–1394.

[12] G. Biggs and B. MacDonald, "A survey of robot programming systems," in *Proceedings of the Australasian conference on robotics and automation*, 2003, pp. 1–10.

[13] B. A. Myers, "Taxonomies of visual programming and program visualization," *J. Vis. Lang. Comput.*, vol. 1, no. 1, pp. 97–123, 1990.

[14] Lego Systems Inc, *Lego Mindstorms NXT-G Invention System*. 2008.

[15] R. Bischoff, A. Kazi, and M. Seyfarth, "The MORPHA style guide for icon-based programming," in *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, 2002, pp. 482–487.

[16] D. Weintrop and U. Wilensky, "To Block or Not to Block, That is the Question: Students' Perceptions of Blocks-based Programming," in *Proceedings of the 14th IDC*, NY, USA, 2015, pp. 199–208.

[17] D. Franklin *et al.*, "Using Upper-Elementary Student Performance to Understand Conceptual Sequencing in a Blocks-based Curriculum," in *Proc.s of the 2017 ACM SIGCSE*, NY, USA, 2017, pp. 231–236.

[18] S. Grover, R. Pea, and S. Cooper, "Designing for deeper learning in a blended computer science course for middle school students," *Comput. Sci. Educ.*, vol. 25, no. 2, pp. 199–237, Apr. 2015.

[19] D. Weintrop and U. Wilensky, "Comparing Blocks-based and Text-based Programming in High School Computer Science Classrooms," *ACM Trans. Comput. Educ. TOCE*, In Press.

[20] M. Resnick *et al.*, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, p. 60, Nov. 2009.

[21] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-D tool for introductory programming concepts," *J. Comput. Sci. Coll.*, vol. 15, no. 5, pp. 107–116, 2000.

[22] A. Ioannidou, A. Repenning, and D. C. Webb, "AgentCubes: Incremental 3D end-user development," *J. Vis. Lang. Comput.*, vol. 20, no. 4, pp. 236–251, Aug. 2009.

[23] A. Begel and E. Klopfer, "Starlogo TNG: An introduction to game development," *J. E-Learn.*, 2007.

[24] M. S. Horn, C. Brady, A. Hjorth, A. Wagh, and U. Wilensky, "Frog pond: a codefirst learning environment on evolution and natural selection," in *Proceedings of IDC*, 2014, pp. 357–360.

[25] M. H. Wilkerson-Jerde and U. Wilensky, "Restructuring Change, Interpreting Changes: The DeltaTick Modeling and Analysis Toolkit," in *Proc. of the Constructionism 2010 Conference*, Paris, France, 2010.

[26] W. Slany, "Tinkering with Pocket Code, a Scratch-like programming app for your smartphone," in *Proc. of Constructionism*, Austria, 2014.

[27] D. Wolber, H. Abelson, E. Spertus, and L. Looney, *App Inventor 2: Create Your Own Android Apps*, 2 ed. Beijing: O'Reilly Media, 2014.

[28] S. Esper, S. R. Foster, and W. G. Griswold, "CodeSpells: embodying the metaphor of wizardry for programming," in *Proceedings of the 18th ACM ITiCSE*, 2013, pp. 249–254.

[29] D. Weintrop and U. Wilensky, "RoboBuilder: A program-to-play constructionist video game," in *Proceedings of the Constructionism 2012 Conference*, Athens, Greece, 2012.

[30] J. Maloney, M. Nagle, and J. Mönig, "GP: A General Purpose Blocks-Based Language," in *Proceedings of the 2017 ACM SIGCSE*, New York, NY, USA, 2017, pp. 739–739.

[31] D. Bau, "Droplet, a blocks-based editor for text code," *J. Comput. Sci. Coll.*, vol. 30, no. 6, pp. 138–144, 2015.

[32] N. Fraser, "Ten things we've learned from Blockly," in *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, 2015, pp. 49–50.

[33] D. Franklin, C. Hill, H. Dwyer, A. Hansen, A. Iveland, and D. Harlow, "Initialization in Scratch: Seeking Knowledge Transfer," in *Proceedings of the 47th ACM SIGCSE*, 2016, pp. 217–222.

[34] P. J. Guo, "Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities," in *Proceedings of CHI* 2017, pp. 7070–7083.