Computer Science Outreach with End-User Robot-Programming Tools

Vivek Paramasivam, Justin Huang, Sarah Elliott and Maya Cakmak
Computer Science & Engineering, University of Washington
185 Stevens Way, Seattle, WA 98195
{paramv,sksellio,jstn,mcakmak}@cs.washington.edu

ABSTRACT

Robots are becoming popular in Computer Science outreach to K-12 students. Easy-to-program toy robots already exist as commercial educational products. These toys take advantage of the increased interest and engagement resulting from the ability to write code that makes a robot physically move. However, toy robots do not demonstrate the potential of robots to carry out useful everyday tasks. On the other hand, functional robots are often difficult to program even for professional software developers or roboticists. In this work, we apply end-user programming tools for functional robots to the Computer Science outreach context. This experience report describes two offerings of a week-long introductory workshop in which students with various disabilities learned to program a Clearpath Turtlebot, capable of delivering items, interacting with people via touchscreen, and autonomously navigating its environment. We found that the robot and the end-user programming tool that we developed in previous work were successful in provoking interest in Computer Science among both groups of students and in establishing confidence among students that programming is both accessible and interesting. We present key observations from the workshops, lessons learned, and suggestions for readers interested in employing a similar approach.

CCS Concepts

•**Social and professional topics** → **K-12 education;**

Keywords

Robotics; Accessibility; End-User Programming; Outreach

1. INTRODUCTION

Robots are becoming popular tools for Computer Science (CS) outreach efforts that introduce programing to K-12 students [2, 6, 13, 14, 15]. The goal of these outreach activities is to demonstrate the allure of computing and engineering disciplines, taking advantage of the immediate engagement that robots offer. As with many CS outreach activities, it is important that the interface used for programming has a low barrier to entry. The simple toy robots used in previous outreach efforts have highly simplified, easy-to-learn interfaces, often based on visual programming.[1]

The necessity of a simple interface tends to preclude the use of more advanced functional robots in K-12 outreach curricula, since these kinds of robots require a significant amount of background knowledge and skill to program. For example, programming a Clearpath Turtlebot (Fig. 1) requires relatively advanced knowledge of Python or C++

Figure 1: Chester, a waiter robot programmed by students at the 2015 Robot Programming Workshop, (a) escorts guests, (b) takes orders, and (c) delivers orders at "Cyber Cafe". Robots programmed by individual students at the 2016 workshop include (d) a cleaning robot (FRED), (e) a medication reminder/delivery robot (Ultravax), (f) a language tutor robot that names objects in the room (Lærer), (g) a math tutor robot (Khan), (h) a pet robot that talks about sports (Sport), and (i) a dog walking robot (Walker).

[1]Examples of educational toy robots include the Tymio (https://www.thymio.org/) and Dash&Dot (https://www.makewonder.com/).

SIGCSE '17, March 08 - 11, 2017, Seattle, WA, USA

ACM 978-1-4503-4698-6/17/03…$15.00

DOI: http://dx.doi.org/10.1145/3017680.3017796

languages and familiarity with ROS (Robot Operating System).[2] However, in contrast to toy robots, the Turtlebot can deliver items, interact with people on a touchscreen, and autonomously navigate its environment. Involving such robots in CS outreach, especially when targeting students with disabilities, can better convey how programming skills can enable them to tackle everyday problems.

In this work we propose to use end-user programming (EUP) tools [9], to enable CS outreach with human-scale, functional robots. EUP is an active research area that aims to enable users with limited technical knowledge to create software that accomplishes desired tasks. This experience report describes two offerings of a week-long introductory workshop in which high-school students with various disabilities learned to program a Clearpath Turtlebot using EUP tools for programming robots developed in our prior work [5] We found that the robot and the EUP tool were effective in allowing novice programmers to bring their ideas to life, garnering their interest in CS, and establishing confidence about their ability to pursue CS. We present key observations from the workshops, lessons learned, and suggestions for readers interested in employing a similar approach.

2. RELATED WORK

Robots in Computer Science Education. The effectiveness of robots in CS education is well documented. Robots with easy-to-use programming interfaces, such as LEGO Mindstorms[3], have been shown to be effective in developing programming skills as well as in fostering interest in CS [8, 13] and in neighboring STEM fields such as mathematics and physical sciences [10]. Middle and high school students mentored by college students in a robotics-based introductory CS course at Brooklyn College were observed to switch career paths to pursue CS [13]. Virtual robots have also been employed as part of Hour of Code[4] exercises.

One study demonstrated the effectiveness of robotics in promoting gender diversity in CS education and technology literacy programs through a "strong social narrative" [3]. Their study provided a customizable robot that expressed emotions designed by its creator using light, sound, and movement. Similarly, our work builds on the idea that personalization and individual creativity can help engage the interest of students in programming tasks.

End-user Programming. End-User Programming (EUP) is an active research area in human-computer interaction that aims to enable everyday people, who are not professional software developers, to create custom programs that meet their particular needs [7, 9]. The most popular examples of EUP are spreadsheets [12] and webpage development [16]. In their 2006 book, *End User Development*, Leiberman et al. stress that EUP systems must be "easy to understand, to learn, to use, and to teach" [9]. These values are very relevant to CS outreach tools as well.

One end-user programming technique that has been adopted in robotics is *visual programming* [1, 11], in which users create and modify programs by manipulating visual representations of program components. These systems generally involve simple toy robots [2, 6, 14, 15] that allow programming at a low-level where individual sensory inputs can be tied directly to actuators. In our previous work, we developed visual programming tools for functional mobile delivery robots [5] and manipulators [4].

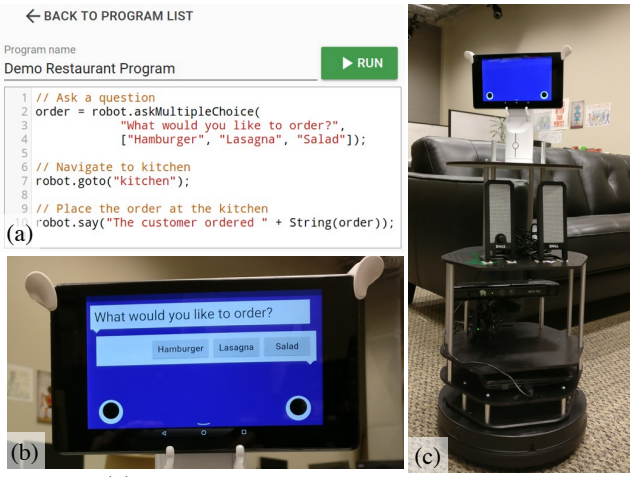

Figure 2: (a) The browser-based CodeIt interface showing a sample program, (b) the robot's screen during the operation of the sample program, (c) the Turtlebot.

3. ACCESSIBLE ROBOT PROGRAMMING

Next, we describe the robot platform and EUP tool used in our robot programming workshops.

3.1 Robot Platform

The Clearpath Turtlebot[5] is a cylindrical robot that uses a differential drive for motion and a Microsoft Kinect to perceive its environment. The Turtlebot is modular and easy to reconfigure with different height platforms and other components. We allowed students to specify minor hardware modifications to the robot, such as adding a basket, box, brush, or other tool onto the robot. Our Turtlebots held tablets which served to display information and receive input in the form of button presses. In addition, they are capable of speech input and output.

Given a map of its environment, the Turtlebot can localize itself and navigate between any two points on the map while avoiding obstacles using a non-deterministic path planning algorithm. Locations on a map can be given names for easy referencing. Prior to the workshop, we used the Turtlebot's navigation tools to create a map of the room and define names for useful locations, such as "Door" and "Table."

3.2 CodeIt

In our workshops, we used an EUP tool called CodeIt for programming the Turtlebot. CodeIt was originally developed to enable programming of a mobile robot to perform delivery and social interaction tasks [5]. It was later extended and integrated with other tools to enable programming of mobile manipulators [4]. Both of these versions of CodeIt have a drag and drop interface based on Google's Blockly.[6] To increase accessibility for students with impaired motor skills and low vision, the versions of CodeIt

[2]http://www.ros.org/
[3]http://www.lego.com/en-us/mindstorms
[4]https://hourofcode.com/us/learn/robotics
[5]https://www.clearpathrobotics.com/turtlebot-2-open-source-robot/
[6]https://developers.google.com/blockly/

used in our workshops involved a text based interface, rather than a visual one.

The version of CodeIt used in the first run of the workshop (July 2015) used a Python interface and the second one (July 2016) used a JavaScript interface. Both interfaces presented the following functions, with minor differences in API.

- **goTo(location):** Navigates robot to the named location, avoiding obstacles.
- **say(text):** Uses voice synthesizer to say the passed-in text out loud.
- **displayMessage(message):** Displays the passed-in message on the screen.
- **askMultipleChoice(message, choices, timeout):** Displays the passed-in message and choices on the screen for up to timeout seconds. Returns the choice selected.
- **moveForward(speed), moveBack(speed):** Moves the robot forward or backward at the specified speed (m/s) for one second.
- **turnLeft(speed), turnRight(speed):** Turns the robot left or right for one second at the specified rate (rad/sec).

3.3 Accessibility

Accessibility and ease-of-use of programming tools is of vital importance to new programmers, and doubly so for students with disabilities. Implementing our interfaces as browser-based web applications allowed students to program on their own laptop computers and ensured they could use their usual accessibility tools such as screen-readers, special fonts, and zoom. This also allowed students to write programs without installing special software, independent of operating system.

During the 2015 workshop, we had the students edit code directly on Github, and notify an instructor when they wanted to have the code tested on the robot. The instructor would then pull the code onto the robot and execute it. The 2015 system was functional, but the workflow was slow. In 2016, we developed a web interface that allowed users to create, save, and execute CodeIt programs directly from a browser. This greatly expedited the development process.

4. ROBOT PROGRAMMING WORKSHOP

The robot programming workshop was offered as part of the DO-IT scholars summer program in July 2015 and 2016.[7] Both workshops took place over five days, three hours each day, for a total of fifteen hours of instruction. The 2015 workshop had five students, while the 2016 workshop had six. In 2015, all students worked together to program a single robot functionality. In 2016, each student developed their own program. Both years, students presented their programs to the entire group at the end. The DO-IT scholars summer program engages students disabilities from the Seattle area with the goal of encouraging them to go to college and facilitating their transition. Besides topical workshops like our workshop, the summer program includes activities such as self-advocacy skill building or field trips to local tech companies.

The primary goal of the robot programming workshop was to interest the students in CS as a field and to develop their confidence that programming is a fun and accessible activity.

[7]http://www.washington.edu/doit/programs/do-it-scholars/overview

Figure 3: 2016 CodeIt JavaScript Sample

```
// Portion of 2016 Sample Code
// go to table_southeast
robot.goTo("table_southeast");
// other places: door, home, trash, couch

// display message
robot.displayMessage("Hello!", "My Name is
    LoggerHead.");

// say message
robot.say("Hello, my name is Loggerhead.");

// wait for two seconds
waitForDuration(2);

// say message
robot.say("What's your favorite color?");

// Ask a question
item = robot.askMultipleChoice("What's your
    favorite color?", ["Indigo", "Jade"]);

// Display a message by adding two strings
    together
robot.displayMessage(String("You Selected ") +
    String(item));

// say message
robot.say(String("You Selected ") +
    String(item));
```

The secondary goal of the workshop was to develop introductory knowledge of CS concepts such as functions, loops, and parameters.

4.1 Curriculum

The outline of the five-day workshop was as follows.

- **Day 1:** We introduced basic concepts in robotics. Pairs of students brainstormed dozens of program ideas for what to program on the robot and began exploring CodeIt.
- **Day 2:** Students finalized project ideas and began implementation and low-fidelity hardware modifiations.
- **Day 3:** Students tested and debugged their programs, and implemented improvements.
- **Day 4:** Students fixed final bugs and demonstrated their robots. We conducted interviews with the students.
- **Day 5:** We demonstrated other robots and performed activities. We administered a post-workshop survey.

4.2 Teaching the API

Due to the relative simplicity of CodeIt, we made the decision to teach the API via sample code. Learning from our past experience and past work by others [8], we pursued a style that minimized lecturing and maximized hands-on time with the robots and code.

On the first day of the workshop, we provided a sample program which demonstrated the use of the most common robot functions. We included detailed comments explaining each function's purpose. A portion of the 2016 CodeIt JavaScript sample code is given in Fig. 3. We asked students to make a copy of this program, modify it in any way they would like, and test it on the real robot.

Over the course of the next few days, students learned the API while developing their program. We introduced conditional statements to the students when prompted, and introduced for-loops and while-loops on Day 3 to students who required them for their programs.

4.3 Participants

Of our eleven students, three were female, and eight were male. Most were juniors and seniors in Seattle area high schools. One student had taken AP Computer Science and participated in FIRST robotics. Two had taken one or two general computing classes. One had learned HTML/CSS to create a website. The rest of the students had no prior programming experience. Our students had various disabilities or conditions including deafness, low vision or blindness, Cerebral Palsy, Muscular Dystrophy, Ollier's disease, Attention Deficit Disorder, Asperger's Syndrome, and other autism spectrum disorders or learning disabilities.

4.4 Projects

The 2015 workshop involved collaborative programming. The five students were split into three teams of two, with one student working with a teaching assistant. Each of those teams worked on a small part of a larger program which played out the scenario of Chester, a robot that serves people at a restaurant. The first team worked on greeting and escort to table, the second team worked on taking customers' orders and delivering food, and the third team managed the receipt and payment.

The 2016 workshop involved students working on individual projects. Two robots were shared between the 6 students. The different projects were as follows.

- **French Robo Expunging Device (FRED):** A cleaning robot that dusts tables, cleans whiteboards, and sweeps objects on the ground.
- **Ultravax:** A robot with a sassy personality that delivers medicine to a caretaker at the proper time.
- **Lærer:** A robot that teaches Norwegian by navigating around a room, translating the names of various objects it comes across and administering a quiz at the end.
- **Khan:** A robot that teaches mathematics and administers quizzes.
- **Sport:** A robot that acts as a pet for people who have allergies and also knows a lot of sports facts.
- **Walker:** A robot that can take a dog for a walk, for busy people with pets.

5. DATA, OBSERVATIONS, AND FINDINGS

Throughout both workshops, we took notes of our interactions with students as they worked on their projects and we recorded video-blog style interviews with each of them on the last work day of the workshop. In these brief, open-ended video interviews, we had the students describe their projects and asked them questions about their work. In addition, in 2016 we included a post-workshop questionnaire and received responses from a short interview conducted by an external supervisor on the last day. Findings based on the compiled data are presented next.

5.1 Collaborative vs. Individual Programming

In 2016, as students programmed their own robots, we saw the students engage with the robot more and feel that it is a more personal project, compared to 2015. Multiple students referred to the Turtlebot they worked on as "my robot" instead of "the robot" as compared to 2015, despite sharing it with two other students. In our post-workshop survey, a student mentioned that "not everyone wants to do the same thing," expressing that they enjoyed being able to develop their own program. This indicates that for some students unbounded creativity is key to engagement and a team environment may stifle their inventiveness.

On the other hand, collaborative programming allowed students to experience teamwork in a CS environment and delivered a more interesting and intricate robot interaction. The team effort to develop Chester in 2015 was something one student could not have accomplished alone in the limited time-frame of the workshop.

Figure 4: 2016 Post-Workshop Survey Questions

1. Before participating in this workshop, what experience did you have with computer science or robotics, if any?
2. What parts of the workshop did you like the best? Why?
3. What parts of the workshop could use improvement?
4. What part of the workshop was most beneficial to your learning?
5. What part of the workshop was most distracting or hindered your learning?
6. The Web-based CodeIt programming tool was easy to learn. [1-5]
7. I found that CodeIt allowed me to easily turn my ideas into a functional robot. [1-5]
8. I would like to use CodeIt to program robots in the future. [1-5]

5.2 Post-Workshop Survey

At the end of the 2016 workshop, students filled out an eight-question survey (Fig. 4) that evaluated the effectiveness of the workshop. Next, we discuss the most interesting results of the survey which all six students responded to.

In response to question 2, four out of six students agreed that the "best" part of the workshop was being able to program whatever they wanted on the robots. In question 6, two students rated the ease of learning as 4/5 and the rest as 5/5, indicating high levels of satisfaction with how simple CodeIt was to learn. This was an expected result of using an EUP interface, which is designed to have a low barrier of entry. One student expanded on their response to this question by adding that the reason they responded with a 4 was because at times the student needed help understanding some of the concepts.

In question 7, three students rated the ease of implementing their ideas as 4/5 and three as 5/5. One student mentioned that they did not respond with a 5 because of technical challenges with the infrastructure encountered occasionally during the workshop. These responses demonstrate the perceived expressivity of CodeIt to capture useful tasks. It also indicates that our introduction of the robot and the CodeIt API correctly communicated the robot's capabilities.

In response to question 8, three students rated their desire to use CodeIt in the future as 5/5, two as 4/5 and one as 2/5 response. The student who responded with a 2 wrote that while CodeIt is "very easy" to learn, its commands are not versatile enough for a more complex robot.

5.3 Perception about Programming

Coming in to the workshop, many students held a belief that programming would be difficult, but ultimately found it much simpler than they imagined. In a video interview, a student from the 2015 workshop reflected on her week, mentioning that at first, the sample code "looked really in-

timidating," but after working with it for a while, she learned "how any small change we made to the code affected how the robot behaved." This student came back to be a teaching assistant next year, helping us manage the workshop.

A 2016 student remarked in interactions with instructors on multiple occasions, "technology doesn't like me," expressing her discomfort and lack of confidence when working with computers. However, at the end of the workshop, she mentioned in her video interview that learning to program "was easier than [she] thought it would be," adding that programming "came easily to [her]." This sentiment was echoed by a 2015 student who remarked in his video interview that during the workshop he "grew to understand programming."

Another 2016 student, who created the Norwegian-teaching robot *Lærer* remarked in his video interview that trying to "communicate your thoughts to the robot was a pretty big learning experience," and that "being successful at these challenging things, like telling it how to move, to teach, .. was just absolutely fantastic."

5.4 Engagement with Content

We saw high levels of engagement from students using CodeIt to create complex robot behavior. This did not necessarily involve learning new CS concepts, but rather creating detailed program content using learned concepts. For example, in 2015, students spent time adding various food items to their menu which customers could order from, and discussed in detail what the robot should say and which direction it should face when it interacts with customers at the restaurant.

In 2016, one student spent hours adding new vocabulary words to his Norwegian-teaching robot, testing and iterating to ensure his robot was pronouncing words properly as it moved from object to object. The same student worked with others to develop an entirely separate program meant to make the robot appear to be possessed by a other-worldly entity, emitting bizarre noises and phrases, and drove it down the hallway, spooking some of the students in neighboring labs. Another 2016 student created a caretaker robot that "has personality." He said in his video interview that he had the most fun "getting the robot to be a little sassy." These students have progressed past the stage of simply learning the content and moved on to enjoying it, which is important to establishing long-term interest.

5.5 Disability in the Background

By allowing students to write code on their own laptops, we were able to mitigate many accessibility challenges that would normally be present when teaching robot programming to a group of students with disabilities. Providing alternative input and output modalities on the robot allowed students to make the robot they programmed as accessible as possible to the rest of the group. For instance, most students in the 2016 workshop displayed status messages on the robot's screen while also using text-to-speech to verbalize the same message, such that both low-vision and hard-of-hearing students could be aware of the robot's status. Similarly, in the 2015 workshop the robot programmed by the students could take food orders both directly from the screen or with speech input, making it accessible for students in wheelchairs with limited reach and dexterity and students with low-vision.

Given that the workshop was run in the context of the DO-IT summer program centered around disability, we expected that participants would program the robot to address challenges they face due to their disabilities. Instead, participants chose to work on developing robots that addressed general problems (*e.g.,* cleaning, forgetting to take/losing medications) or problems they faced which are independent of their disabilities (*e.g.,* studying math). Many explored their personal interests (*e.g.,* pets, sports, languages).

5.6 Computer Science Concepts

The workshop introduced students to basic CS concepts. In 2015, students learned variables, conditionals, loops, events, and arrays, and they learned to write simple functions. In 2016, students learned the same concepts, except for events.

Despite the differences between the two offerings of the workshop, the kinds of mistakes made by students were similar. Most common mistakes were syntactic; such as mismatched quotation marks, brackets and braces, and incorrect use of the assignment operator (=) instead of the equality operator (==), among other similar novice programmer mistakes. Many of these mistakes could be avoided with a more advanced IDE with autocomplete and error marking.

Other mistakes had more to do with semantics. For example, some students in the 2016 workshop wanted to ask a question to the user and timeout after sixty seconds. They would write the following:

```
// Javascript semantic mistake
answer = robot.askMultipleChoice(
           "What is your favorite color?" ,
           ["Indigo", "Jade"]);
waitForDuration(60);
```

What the above code actually does is wait indefinitely until it receives a response to the multiple choice question, and then waits for sixty seconds no matter the response. The correct solution takes advantage of the optional final parameter for the askMultipleChoice function:

```
// Javascript corrected
answer = robot.askMultipleChoice(
           "What is your favorite color?" ,
           ["Indigo", "Jade"],
           60); // final parameter is timeout
```

Common robot-related errors encountered both years were bugs that had to be tuned by experimentation, such as determining how much to rotate the robot to face someone in a particular seat, and how long to wait in between spoken sentences for natural interaction. For example, the student who was making the robot speak in Norwegian made several iterations to achieve the correct phonetic pronunciation.

6. DISCUSSION

Overall, we found that individual programming created a more personalized and engaging work environment compared to a team programming project. Hence, we recommend this approach. Key resources that made this approach possible were the small ratio of students to robots (3:1), the ability to switch between two student's programs instantaneously on the robot, the small ratio of students to instructors/teaching assistants (1:1), and a large lab space that allowed stations for testing different robots and recording video blogs. Although we think the workshop could possi-

bly be scaled so long as these resources are also scaled, we believe that larger group activities (*e.g.,* discussion about robot jobs, discussion about feasibility and usefulness of brainstormed ideas) might not engage all students at the same level if the group gets too large. We ran this workshop with twelfth-grade students, but believe that high school students of all ages, as well as with older middle school students, could participate as well.

One way to further scale our workshop is to enable teachers, who do not necessarily have expertise in CS, to run them in their communities. While we think that the robot platforms are not yet robust enough to enable this, we see potential for using simulated robots that can be programmed similarly. We are currently working on an Hour of Code module that involves programming a simulated mobile manipulator robot (as opposed to the simple simulated toy robots previously used in Hour of Code). In future work, we would like to better understand the impact of using human-scale robots for education, as opposed to smaller, toy robots or virtual agents.

In our workshops we taught the robot API by having students edit a simple example program, as shown in Fig. 3. In our opinion, this worked very well. Due to the simplicity of the API, it was faster to allow students to experiment with tweaking parameters than to actually perform a lecture explanation of each function. That being said, as the API grows, we would need to select a subset of the functionality to put into a sample program, so as not to overwhelm the students with information.

7. CONCLUSION

We present an experience report about two offerings of a robot programming workshop targeted at high-school students with disabilities. Our goal was to inspire an interest in CS among students by enabling them to program a functional mobile robot via an end-user programming interface. We found that an EUP tool can indeed make an advanced robotic platform, such as the Turtlebot, accessible to novice programmers in an educational setting. We think that this pairing of EUP tools with advanced robots is effective in developing students' confidence in their ability to program and in developing interest in Computer Science through applications they can relate to. Our workshop allowed students to program a robot to do useful tasks effortlessly, changing their potentially false perceptions about the complexity and difficulty of programming robots. More generally, we believe that EUP interfaces that are designed to be easy to learn and require little investment to create something interesting and useful, are ideal when exposing students to CS without intimidating them.

8. ACKNOWLEDGMENTS

Thanks to the DO-IT center, especially Debra Zawada, our teaching assistants (DO-IT "interns"), and to our students (DO-IT "Phase 2 scholars"). This work was funded by the National Science Foundation, Awards IIS-1552427 "CAREER: End-User Programming of General-Purpose Robots" and EEC-1444961 "AccessEngineering."

9. REFERENCES

[1] M. Burnett. Visual programming. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 1999.

[2] P. Cox, C. Risley, and T. Smedley. Toward concrete representation in visual languages for robot control. *Journal of Visual Languages & Computing*, 9(2):211–239, 1998.

[3] E. Hamner, T. Lauwers, D. Bernstein, I. Nourbakhsh , and C. F. DiSalvo. Robot diaries: Broadening participation in the computer science pipeline through social technical exploration. In *AAAI Spring Symposium on Using AI to Motivate Greater Participation in Computer Science*, March 2008.

[4] J. Huang and M. Cakmak. Code3: A system for end-to-end programming of mobile manipulator robots for novices and experts. In *ACM/IEEE International Conference on Human Robot Interaction (HRI)*, 2017.

[5] J. Huang, T. Lau, and M. Cakmak. Design and evaluation of a rapid programming system for service robots. In *ACM/IEEE International Conference on Human Robot Interaction (HRI)*, pages 295–302, Piscataway, NJ, USA, 2016. IEEE Press.

[6] S. Kim and J. Jeon. Programming lego mindstorms nxt with visual programming. In *IEEE International Conference on Control, Automation and Systems (ICCAS)*, pages 2468–2472, 2007.

[7] A. Ko, B. Myers, and H. H. Aung. Six learning barriers in end-user programming systems. In *IEEE Symposium on Visual Languages and Human Centric Computing*, pages 199–206, 2004.

[8] B. Lester. Robots' allure: Can it remedy what ails computer science? *Science*, 318(5853):1086–1087, 2007.

[9] H. Lieberman, F. Paternò, M. Klann, and V. Wulf. *End-User Development: An Emerging Paradigm*, pages 1–8. Springer, 2006.

[10] M. J. Matarić, N. Koenig, and D. Feil-seifer. Materials for enabling hands-on robotics and stem education. In *In AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education*, 2007.

[11] B. Myers. Visual programming, programming by example, and program visualization: a taxonomy. In *ACM SIGCHI Bulletin*, volume 17, pages 59–66, 1986.

[12] B. Nardi and J. Miller. *The spreadsheet interface: A basis for end user programming.* Hewlett-Packard Laboratories, 1990.

[13] R. B. Osborne, A. J. Thomas, and J. R. Forbes. Teaching with robots: A service-learning approach to mentor training. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*, pages 172–176, 2010.

[14] F. Riedo, M. Chevalier, S. Magnenat, and F. Mondada. Thymio ii, a robot that grows wiser with children. In *IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, pages 187–193, 2013.

[15] J. Weinberg and X. Yu. Robotics in education: Low-cost platforms for teaching integrated systems. *Robotics & Automation Magazine, IEEE*, 10(2):4–6, 2003.

[16] J. Wong and J. Hong. Making mashups with marmite: towards end-user programming for the web. In *Proceedings of the ACM conference on Human factors in computing systems (SIGCHI)*, pages 1435–1444, 2007.