Intuito: Opportunistic Tangible Programming by Demonstration for Physical Components

Rawan Alharbi
Northwestern University
Evanston, IL 60208, USA
rawan.alharbi@northwestern.edu

Nabil Alshurafa
Northwestern University
Evanston, IL 60208, USA
nabil@northwestern.edu

Michael Horn
Northwestern University
Evanston, IL 60208, USA
michael-horn@northwestern.edu

CHI'17 Extended Abstracts, May 6-11, 2017, Denver, CO, USA.
ACM ISBN 978-1-4503-4656-6/17/05.
http://dx.doi.org/10.1145/3027063.3053264

Abstract
As computer programming becomes more important to various fields and disciplines and as it is more commonly taught in education settings, the number of end-users with basic programming experience is increasing. The importance of being able to easily and quickly develop programs has prompted research in "opportunistic" programming methods. This research contributes to this domain by introducing a platform called Intuito, designed for programming physical components (sensors and actuators) through direct "programming by demonstration" techniques. Our approach is to offer users a tangible system that maps the user's actions with sensors and actuators into editable text-based code by inferring the user's intentions. We present our initial hardware and software prototype with an in-lab preliminary evaluation of this system.

Author Keywords
PBD; Sensors; Tangible Interaction; End-user Programming, opportunistic programmers; intelligent interfaces; prototype

ACM Classification Keywords
H.5.2. User Interfaces: Input devices and strategies; Interaction styles; Prototyping; User-centered design. []

Introduction

When it comes to prototyping and building ideas in physical computing, toolkits like Raspberry Pi[4] and Arduino[1] have lowered the barrier of entry for non-engineers who know little about electronics. These kits allowing novices to program microprocessors using a higher-level language and well documented libraries. However, despite these advantages, novice users still face barriers to entry while programming physical components [2].

Many online resources provide novice users with sample code to help them in understanding how to program physical component. Tutorials can be helpful for illustrating a concept, especially if they are well documented. However, even the best tutorials might not be at the right level for all novice users or they might not address the exact problem that a user is attempting to solve. This often leads novices to follow an "opportunistic programming" approach by searching online to see how others implement different ideas and functionality and then subsequently copying and pasting code snippets from different sources to create an end product. In this case, users have to search for code, identify the boundaries of the code snippet that is to their interest and then modify it as needed, which has proven to be a challenge for novice programmers [6].

To mitigate the errors resulting from copying code from multiple sources, this paper offers techniques that enable opportunistic programmers (novice, intermediate or expert) to generate code samples through programming by demonstration rather than borrowing from online sources. It introduces a platform called Intuito that aids in translating users' ideas into code. The goal of Intuito is to enable the opportunistic programmer to focus on the triggers and actions of the interaction, rather than worry about the depths of language syntax to translate ideas into code that the computer can understand. Intuito produces starter text code mapped from the demonstrated events. With Intuito, users first demonstrate an idea by capturing the state of each trigger or action (e.g. when the button is pressed, then turn on the light). The system then analyzes the captured events and produces text code that automates the demonstrated process. The code can be further edited and modified by the opportunistic programmer.

The primary contributions of this paper are: (i) The design of a tangible modular interactive method of programming by demonstration for physical modules aimed at helping opportunistic programmers in translating their ideas into text code that can be reused and modified. (ii) Results from our preliminary testing with 13 participants. These findings should help inform the design of future toolkits that uses tangible programming by demonstration methods to aid opportunistic programmers.

Background and Related Work

Opportunistic programmers (both novice and expert) care more about the speed and ease of development rather than structure and robustness [8]. A research study [3] shows how opportunistic programmers utilize the internet heavily to create mash-up programs. They engage in just-in-time learning to seek clarification of a concept or to remind them about a syntax or a function. In the research, they found that opportunistic programmers spend 19% of their time on the internet with 1/3 of their code copied from different web pages. Opportunistic programmers rely heavily on example code as it serves as a starting point for their project rather than starting from scratch. This approach of reusing and modifying code examples is not an approach used by opportunistic programmers only. Educators have used scaffolding with code templates to allow the student to experiment and visualize the logic by modifying specific param-

eters and rerunning the code [12]. The code provided by the educator is carefully picked and tailored to the students, unlike the opportunistic programmer who has to look for different snippets of code and merge them into one program. The aim of Intuito is to create this starting code by using the programming by demonstration method, where they manually demonstrate the desired inputs and corresponding outputs of a system by interacting with Intuito, instead of spending a lot of time looking for snippets of code and then merging them to create the desired program.

Programming by demonstration (PBD) is the method of translating user's actions demonstrated on a graphical or physical components into a program. PBD systems differ in how they represent and generalize the recorded demonstrated event [10]. Some PBD systems represent the output as visual before-after rules [14] and others have no representation other than playing the recorded actions [5, 13]. Although this simplified representation will not need a computer to display the recorded events, it will limit users' ability to edit or share their recorded actions. Since computers are more widely available and the number of people with some basic programming skills is increasing, we chose to represent the captured events visually as triggers and actions along with an event-driven text code to support the opportunistic programmer.

PBD systems try to infer and predict what the user's demonstration means using heuristics or artificial intelligence, while other systems save the recorded events as is without trying to infer user intention [11]. Systems that use heuristics implement rules that help in recognizing the demonstrated event, most of these rules are encoded when the PBD system is created, but these rules can be extended anytime by the developer or the user. Machine learning algorithms can be used to infer programs and improve the PBD system by providing examples [7], but they fail to explain to the user how to code such an interaction. Intuito uses heuristics guided by tangible interaction methods to infer and produce event-based text programs that control triggers and action components.

Intuito User Scenario

Sarah is a student who lives a sedentary lifestyle. She sits in front of her computer all day, and she is starting to notice that she gets headaches and experience fatigue more often. She decides to take a 10-minute break every hour whenever she is working in front of her computer. She tries setting her alarm to ring after an hour, but Sarah keeps ignoring the alarm by hitting the snooze button. She soon realizes that this method does not work for her, and she starts thinking about other ways to interrupt her workflow to take a break. She remembers one effective method that has always worked was her brother interrupting her concentration by waving his hand in front of the computer. Sarah decides to automate this solution for herself by attaching a hand model to a servo which will start to wave if the chair sensed her setting for too long. She has some programming skills, so she decided to look for example code online and finds lots of examples. She does her best in navigating and copying sample code, but when her code still generates many confusing error messages, and she starts to get frustrated. Sarah remembers that someone gave her an Intuito programming by demonstration kit as a gift where she can simply demonstrate her actions, and it will get translated to code. She decides to use Intuito, and her first step is to select Intuito's trigger and action components that she needs (a pressure sensor and a servo with a hand attached). Then she places them in the desired position (pressure sensor on the seat and servo in front of the screen). She opens Intuito's software and selects the start recording button. Then she sits on her chair and holds

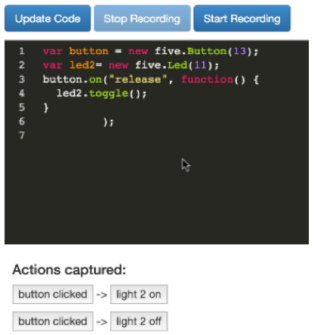

Figure 1: A Screen shot of Intuito's software showing the start recording, stop recording and update buttons, captured events and the generated output code.

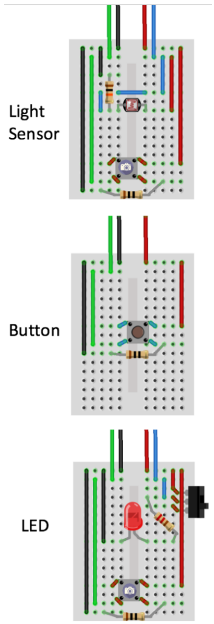

Figure 2: Illustration for some of Intuito's trigger and action components.

the pressure sensor's capture button for 3 seconds. After that, she positions the servo at 0 degrees and clicks on the servo's capture button. Then she moves the servo to 180 degrees and clicks on the servo's capture button again. She then stops the recording using the stop recording button in Intuito's software. Intuito provides Sarah with the code, and she now can test her program right away. She sits on the chair, and after three seconds the servo hand started to move immediately. However, Sarah wants to trigger the servo movement after 60 minutes instead of 3 seconds. Sarah does not need to go back and demonstrate the action by holding the capture button for an hour; she can just modify the generated code and change the code to trigger the action after an hour (replace 3 seconds with 3600 seconds) then click on update code button. Sarah is happy with her solution; she feels that it is working for her because this is an interruption that triggers a pleasant memory, motivating her to take a break.

Designing Intuito

Components

In designing our programmable physical components, we built on the experience and knowledge of others who used the modular approach [9] to introduce a new modular design that enables programming by demonstration. Figure 2 shows an illustration of some of the components we prototyped. Each component features a capture button to capture the state of the component (e.g. capture light value using the light sensor). The LED has an extra switch which allows the user to change the state of the LED (on or off). Other components (light sensor, servo, pressure sensor, and button) do not need it as the state can be changed by manipulating the component (e.g. moving the servo) or the environment (turning the light off to capture low light value). The components are plugged into an Arduino Shield using a 4-pin connector.

Interface

We implemented an interface (Figure 1) that allows the user to start and stop a recording session, see the captured events and update and modify the generated code. The *record button* in the interface will activate the recording mode, listening to any state capture events triggered by the components. The *event captured area* will show the state and the name of the component being captured as feedback. When the stop recording interface button is clicked, the server will implement the algorithm detection logic, and the result will be displayed in the code area where the user can edit or add to the code and deploy it again by clicking on *update code*. The components communicate with the interface via Musical Instrument Digital Interface (MIDI) messages sent via a USB cable connected to the Arduino.

Detection Algorithm

Figure 3 shows the steps needed to generate an event-driven code which will light up the LED whenever the light in the room is low, or whenever the button is clicked using the programming by demonstration method. In this example, the user wants to program the LED to light up when the room is dark and turn off when there is light in the room. In this case, the light sensor will be the trigger component, and the LED will be representing the action component. One way of demonstrating this example will be by capturing the light sensor value when the room is dark, and then the user clicks the capture button for the LED 4 times to capture the blinking state (on, off, on and off). The following are the steps used to in the detection algorithm: *State Capture* (capture the states of the components in the program array) , *Segmentation* (segment each independent event that consist of triggers and actions components), *Analysis* (analyze each segment and its components to infer logic) , *Code Generation and Implementation* (generate code for each segment based on the analysis).

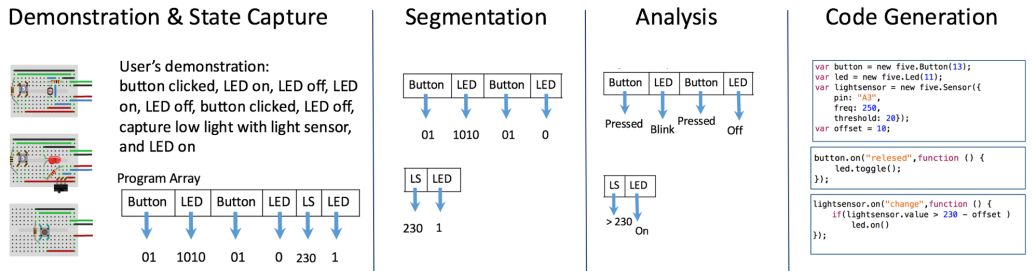

Figure 3: Intuito's detection algorithm steps

Preliminary Evaluation

To evaluate our system, we recruited 13 participants (6 men and 7 women) with different levels of programming skills. Four participants considered themselves novice or beginners and nine consider themselves intermediate programmers. Eight participants had never programmed a microcontroller before.

Participants were asked to do four programming tasks shown in 1 using the programming by demonstration method. The final output of the programming task was shown in a video format to avoid any textual or verbal cues that may bias the participant's interpretation. Participants were asked to perform the task in the exact order. We standardized the programming tasks to understand variability in participants' demonstration. During the experiment, we recorded the participants, the computer screen, and the components for further annotations to understand how the participant used the system.

All participants completed the programming tasks along with the code modification task successfully. From the video of the experiment and the user interviews, we summarize our findings in the form of design guidelines that will help inform other researchers who are interested in tangible programming by demonstration.

Visualizing the captured events aid in demonstration

As soon as the participant captured an event, it was rendered and displayed on the screen, giving the users a chance to remember the actions that they performed and to catch any mistakes in their demonstration sequence. Many participants made mistakes while demonstrating and, in such case, they have repeated the demonstration. An edit function where the user can remove unwanted recorded events using Inutito's software and if needed they can replace the deleted events with a new event that they will record by demonstrating it on the components will be more user-friendly.

Visualizing the captured events details helps in constructing code landmarks

Participants found that the names and values of captured events displayed next to the code facilitated locating the corresponding code. It was easy for participants to change numbers but some were hesitant to change the logic in the

Task #	Description of PBD task and the following code modification task
0	When the button is pressed, the LED state is toggled (this is a warm-up task)
1	When the button is pressed, the LED blinks. Change the speed of blinking.
2	When the button is pressed twice, the LED blinks. Change it to button clicked 3 times instead.
3	Only when the light sensor value is high, and the button is pressed the servo moves. Change it to when the light is low and make the servo move faster
4	When the button is pressed the green LED will light up first, and then the red LED. Change the order to be red first and then green.

Table 1: Programming tasks that the participant performed in the experiment using Intuito

code. We suggest adding clear editable logic symbols (e.g. sensor value > 887, instead of sensor value: 887) in the feedback area and to encourage users experimentation.

Feedback to handle over demonstration

Some participants demonstrated the minimum required events (e.g. demonstrating blinking by turning the LED on and off once), and others did more (e.g. demonstrating blinking by turning the LED on and off three times). One way to avoid having the user over record is to show feedback after a predefined minimum set of recorded events is performed. For example, the LED on, LED off events should be replaced with a blink event in the feedback area to show the user that there is no need for further demonstration. Two participants thought that they had to demonstrate a *"don't do this case"* in Task 2 (the LED components blinks only after pushing the button component twice). Those two participants captured the first button push, and then they captured an LED off state to show a *"don't do this case"* meaning button do not turn the LED now. Then they pressed the button again to demonstrate the second push and after that, they demonstrated the blinking state of the LED. The *"don't do this case"* have confused our system because Intuito interpreted the demonstrated events like the following: whenever the button is pressed then the LED will either blink or will stop blinking if it was already blinking which is not what the user intended to do. Participants who demonstrated the extra *"don't do this case"* treated Intuito as if it was a kid that does not know how to count or as someone whom they do not have a common language to communicate with. One solution to that problem can be by emphasizing that the user has recorded two events in the captured events area or by telling the user to do the minimum demonstration as if they were acting it out rather than trying to explain it to the computer.

Conclusion

We have introduced Intuito, a platform that enables opportunistic programmers to generate a sample code for their idea by demonstrating. Through our preliminary study and observations, we presented design guidelines that can help other researchers in tangible programming by demonstration field.

Future Work

In future work, we plan to perform an unstructured user study where we ask people to brainstorm projects that they want to build using our platform. We will expand our physical components modules and add Bluetooth capabilities to it. We hope to can integrate the components with other nonphysical functionality (ex. send a tweet, or save in a database) to extend the interaction realm. We also plan to implement this programming by demonstration method in programming smart home and wearable devices.

References

[1] Arduino AG. 2016. Arduino. https://www.arduino.cc/. (2016). [Accessed: 2016-07-01].

[2] Tracey Booth and Simone Stumpf. 2013. End-user experiences of visual and textual programming environments for Arduino. In *International Symposium on End User Development*. Springer, 25–39.

[3] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. Two studies of opportunistic programming. In *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*. ACM Press, New York, New York, USA, 1589.

[4] Raspberry Pi Foundation. 2016. Raspberry Pi. https://www.raspberrypi.org/. (2016). [Accessed: 2016-07-01].

[5] P. Frei, V. Su, B. Mikhak, and H. Ishii. 2000. Curlybot: designing a new class of computational toys. *Proceed-*

ings of the SIGCHI conference on Human factors in computing systems 2, 1 (2000), 129–136.

[6] Paul Gross and Caitlin Kelleher. 2010. Non-programmers identifying functionality in unfamiliar code: strategies and barriers. *Journal of Visual Languages & Computing* 21, 5 (2010), 263–276.

[7] Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. 2007. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*. ACM Press, New York, New York, USA, 145.

[8] Björn Hartmann, Scott Doorley, and Scott R. Klemmer. 2008. Hacking, mashing, gluing: Understanding opportunistic design. *IEEE Pervasive Computing* 7, 3 (2008), 46–54.

[9] Edwin L Hutchins, James D Hollan, and Donald A Norman. 1985. Direct manipulation interfaces. *Human–Computer Interaction* 1, 4 (1985), 311–338.

[10] Henry Lieberman. 2001. *Your wish is my command: Programming by example*. Morgan Kaufmann.

[11] Brad A Myers and Richard McDaniel. 2001. Sometimes you need a little intelligence, sometimes you need a lot. *Your Wish is My Command: Programming by Example. San Francisco, CA: Morgan Kaufmann Publishers* (2001), 45–60.

[12] Taro Narahara. 2015. Design exploration through interactive prototypes using sensors and microcontrollers. *Computers & Graphics* 50, 0 (2015), 25–35.

[13] Hayes Solos Raffle, Amanda J Parkes, and Hiroshi Ishii. 2004. Topobo: a constructive assembly system with kinetic memory. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 647–654.

[14] D Canfield Smith, Allen Cypher, and Larry Tesler. 2001. Novice programming comes of age. *Your Wish Is My Command: Programming by Example* (2001).