Sketching Intentions: Comparing Different Metaphors for Programming Robots

Richard Davis
Stanford University
CERAS Building
520 Galvez Mall,
Stanford CA, USA
rldavis@stanford.edu

Engin Bumbacher
Stanford University
CERAS Building
520 Galvez Mall
Stanford CA, USA
buben@stanford.edu

Oceane Bel
University of Southern California
Los Angeles CA, USA
obel@usc.edu

Arnan Sipitakiat
Chiang Mai University
239 Huaykaew Rd.,
Muang, Chiang Mai,
Thailand
arnans@eng.cmu.ac.th

Paulo Blikstein
Stanford University
CERAS Building
520 Galvez Mall
Stanford CA, USA
paulob@stanford.edu

ABSTRACT

This paper introduces a new environment for programming robots and physical computing devices—the Spatial Computing Platform (SCP)—and compares it to a text-based programming environment (the Cricket Logo). The SCP simplifies the process of constructing conditional statements that link the robot's inputs and outputs together. It does this by providing the user with a virtual canvas that they can draw rectangles on using the mouse. Each rectangle represents a range of sensor values, and specific outputs can be assigned to each rectangle. When the sensor values enter into the specified range, the outputs will turn on. We designed a study with 60 youth to compare this environment to Cricket Logo, a well-known variant of Logo designed to control robotic devices. We found that participants using the spatial computing platform were able to build programs of higher complexity and make more changes to their programs over the course of an hour-long workshop.

Categories and Subject Descriptors

D.1.7 [**Programming Techniques**]: Visual Programming

General Terms

Design, Experimentation, Languages

Keywords

Spatial Computing, Robotics, Programming Metaphors

1. INTRODUCTION

Some of the first concepts that a novice learning about building programmable robots will encounter are those of input, output, and control. The robot takes in information about the world through different input sensors (e.g., light, sound, position), and acts on the world through its output actuators (e.g., motors, speakers, lights). The robot only begins to assume some measure of intelligent behavior when the sensors and actuators are linked together. While sensors and actuators can be directly connected electrically, it is more typical for the connections between sensors and actuators to be mediated by a layer of programmable logic (i.e., control). That is, the sensors and actuators are both connected to a computer or microcontroller, which is capable of running user-created programs that process inputs from the sensors and use that information to make decisions about what to do with the actuators.

Because the microcontroller is a simple computer, it is capable of being programmed using a wide range of languages. Typically, microcontrollers are programmed using a variation of the C programming language, with compilers being offered for the most popular lines of chips (AVR, PIC). However, there is a strong trend in educational robotics toward introducing children and adolescents to robotics using alternate programming languages and environments. This work is built on the assumption that programming languages like C set the barrier of entry to learning to program too high [2, 3].

Finding ways to make programming more accessible to children has been an active area of research for decades. Papert began this effort with the introduction of Logo. Since Papert, there has been an explosion of different programming languages and environments for children. These include new variants of Logo like NetLogo [11], a broad range of snap-together block-based languages [7], and data-flow languages [5].

These languages have been designed to program personal computers, which means that most of the action takes place on-screen. However, a smaller but thriving community of researchers has been working simultaneously to design similar programming environments suited for programming robots. These include variants of Logo and other text-based programming environments [9], environments that utilize snap-together blocks [4], and data-flow languages [1].

A few of these languages have introduced new metaphors for thinking about programming. For example, block-based languages like Scratch changed the metaphor of programming to "snapping together blocks representing statements, expressions, and control structures" [8:16:7]. Scratch was designed to make programming more like building with LEGO blocks, providing blocks for commands, control, triggers, and functions [8]. Because of the way that blocks snap together, it is impossible to make syntax errors. Trying out new code can be as easy as swapping out one block for another, and unused blocks can be left lying around the workspace. The choice of the LEGO metaphor reduces the barrier of entry to programming while providing the flexibility to create programs of varying levels of complexity [6]. Moreover, it changes programming into a more creative activity that allows for novices to more easily tinker with code [8].

Tinkering has been identified as an important part of learning to write computer programs [1]. Berland et al. describe tinkering as both an orientation ("the act of either having or needing no plan in the process of creating or modifying a computer program") and a set of activities ("trial and error, messing around or fussing, finding and using feedback mechanisms") [1:568].

In an effort to make tinkering with robots easier, the authors developed a new metaphor for programming robots: Spatial Computing [3, 10].

This metaphor drastically simplifies the process linking the robot's inputs and outputs through conditional logic. The Spatial Computing Platform (SCP) reduces this activity to drawing rectangles on a virtual canvas. Each rectangle represents a range of sensor values, and specific outputs can be assigned to each rectangle. When the sensor values enter into the specified range, the outputs will turn on. In order to evaluate the SCP we designed a study where we compared the SCP to a Logo programming environment (Cricket Logo). This study is described in detail below.

2. A NEW METAPHOR: SPATIAL COMPUTING

2.1 Design

The Spatial Computing Platform (SCP) is a simple interface aimed to help youth tinker with sensors, actuators, and the conditional statements that link them together. It is a web-based interface that provides users with a canvas where they can draw rectangles with the mouse. Each of these rectangles corresponds to a region in 2-D sensor space (Figure 1).

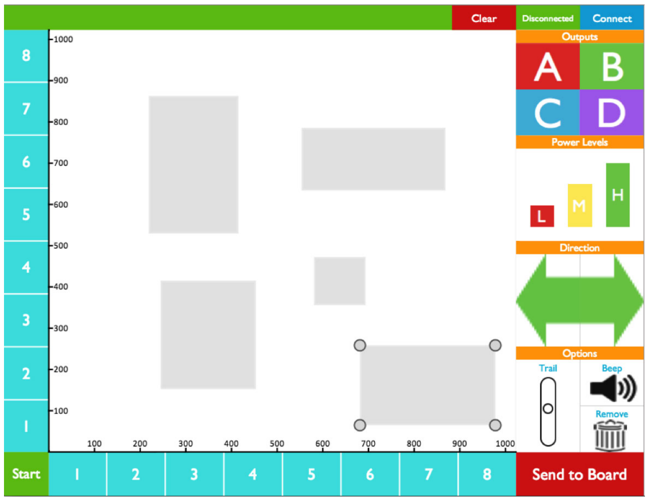


Figure 1: The Spatial Computing Platform

The SCP is designed to work with the GoGo Board, which provides 8 sensor inputs and four actuator outputs (Sipitakiat et al., 2004). These 8 sensors are represented along the x- and y-axes of the SCP canvas. Clicking on the sensor number activates that sensor and displays its value as a green dot on the canvas. When selecting one sensor on the x-axis and another on the y-axis, the green dot is able to move through a two-dimensional sensor space that captures all of the combined sensor values.

Once a rectangle has been drawn on the interface, the user can drag an output letter (A, B, C, D) from the top-right onto the region. Once the output has been assigned, the power (low, medium, high) and polarity (forward, reverse) can be set by dragging their icons onto the region. When the sensor values combine so that the green dot enters into a rectangular region, the assigned output turns on. For example, drawing a rectangle on the canvas and assigning it to an output as shown in Figure 2 below is functionally equivalent to a relatively complex conditional statement in Logo (Figure 3).

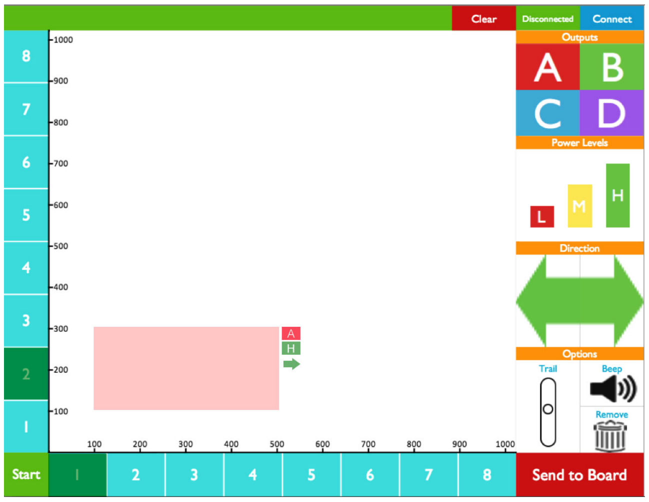


Figure 2: A simple sensor-actuator assignment

```
to controlrobot
    ab, off
    forever [
        if (sensor1 > 100) and (sensor1 < 500) [
            if (sensor 2 > 100) and (sensor2 < 300)
                [ a, setpower 7; a, thatway; a, on]
            ]
    ]
end
```

Figure 3: The Logo code needed to achieve the same effect as drawing the rectangle in Figure 2

3. METHODS

3.1 Participants

We participated in a day-long series of workshops about electronics, building and robotics organized by a local charter school. The workshop took place on a Saturday and was widely advertised in the region for youth aged 8-12. The workshop series was free and open to the public, which meant that the participants did not necessarily all attend the same school (we did not collect data on their school). The data for this study was collected in the robotics workshop, which our group designed. This is an hour-long workshop that we gave three times over the course of the day to 60 participants in total. There were 6 computer/robot stations available, which meant that the participants worked in groups of two to four. Participants were free to choose their work station. 45 out of the 60 participants filled out a post-survey about their background and experiences in the workshop. The average participant age was 10 years old (SD = 1.9). 19 out of the 45 children were girls. 31 out of the 45 participants had tried programming before, and 27 out of 45 had prior experience building robots. Almost all of children gained their prior experience from working with LEGO Mindstorms. None of the children were familiar with Cricket Logo or the SCP.

3.2 Research Design

During the workshop facilitated by two of the authors, participants worked in groups of two and three at stations with a laptop computer connected to a GoGo board-powered robotic. The robot was equipped with two infrared sensors on the bottom (for sensing dark and light strips) and two motors connected to the front wheels. The participants were able to upload new programs from the laptop to the robot over a standard USB cable.

Each workshop followed the same schedule. In the first 15 minutes, one of the researchers gave a short lecture on sensors and actuators, and guided the participants through the construction of a simple, working program using either the Cricket Logo programming environment or SCP. The program instructed the robot to move forward while it was on a light surface and to stop when it was on a dark (colored black) surface. After that, the participants were challenged to create a program that would allow the robot to follow a black strip on the ground. Participants worked on this problem for 40 minutes. Finally, in the last five minutes of the workshop, consenting participants filled out a post-survey.

In the first of the three workshops, participants worked with Cricket Logo, a text-based programming environment for the GoGo board. In the second and third workshops the participants used the new Spatial Computing Platform (described in section 3 above). Aside from the programming environment, nothing else was changed between workshops. This structure resulted in a simple experimental design with two conditions: Text-based Logo or Spatial programming environment (Figure 4).

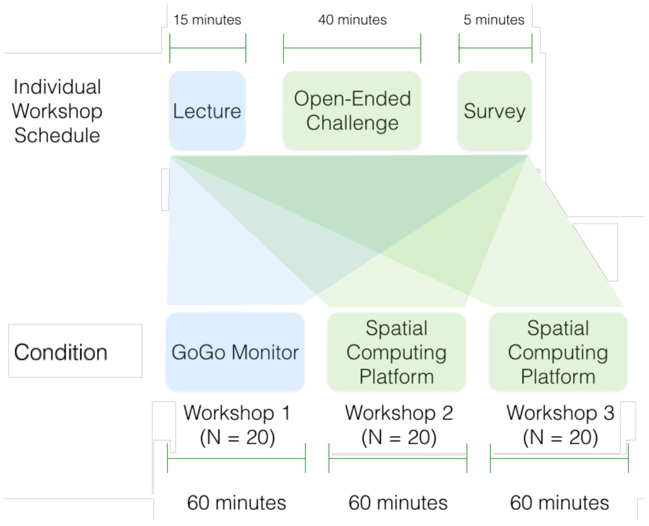


Figure 4: Experimental Design

3.3 Measures

We had two data sources: the post-survey and the video recordings of the laptop screens. The post-survey consisted of 14 questions asking participants about the age, gender, prior programming experience, and the experience in the workshop. As mentioned previously, 45 out of the 60 participants filled out the survey. We performed basic statistical tests on the surveys, finding no different significant differences in these measures between the two conditions. The laptop screens were recorded using Camtasia Studio. Due to a glitch in the recording software, we ended up with 10 screen recordings in total, 3 of the workshop using Cricket Logo, and 7 of the remaining two workshops. We coded the videos for the types of interactions participants had with the software, the number of different programs uploaded to the robot, and the number and types of changes made to each program. The unit of analysis was the participant group at a computer station.

4. RESULTS

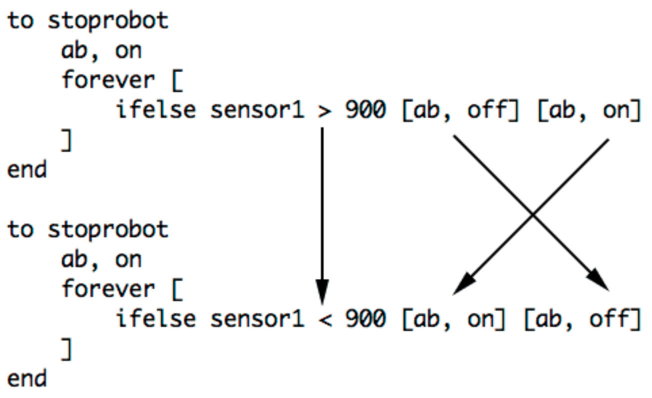
Because there were no significant differences between the groups' experiences in the workshop, we have omitted those results from this section. Instead, we focus specifically on our findings from the video coding process. We present exploratory statistical analyses, as the small number of groups does not allow for any confirmatory analysis.

4.1 Differences in Program Complexity

In the lecture portion of the workshop, participants constructed a working program along with the instructor. This program used the input from a single sensor to control the power to a single output. In order to measure the level of program complexity, we looked at the number of conceptual edits participants made to the starter code and the number of sensors used in the programming code. A conceptual edit is an edit adds a new piece of functionality to the code (e.g., modifying the number of sensors, adding new conditional statements). Participants in the Logo group made a smaller number of edits on average than those in the SCP group. More importantly, only one group using the Logo environment added input from a second sensor to their code, while all the groups using the SCP integrated information from both sensors into their programs.

4.2 Conditional Confusion in Logo

All of the groups using the Logo programming environment appeared to have trouble with construction of conditional statements linking sensor values to actuator outputs. For example, while trying to get their robot to follow the black line one of the groups changed the direction of the inequality while also swapping the conditional clauses. This resulted in new code that appeared to be different but was functionally equivalent (**Figure 5**).


```
to stoprobot
    ab, on
    forever [
        ifelse sensor1 > 900 [ab, off] [ab, on]
    ]
end

to stoprobot
    ab, on
    forever [
        ifelse sensor1 < 900 [ab, on] [ab, off]
    ]
end
```

Figure 5: Changing the inequality and swapping the conditional clauses results in the same functionality

4.3 Rate of Programming

We measured the rate or programming by recording each time a group successfully uploaded working code to the robot. For an upload to be counted, it had to be different from the previous upload and compile successfully. We found significant differences between the two groups on this measure. The group using Logo uploaded new programs at a significantly slower rate, 0.16 new working uploads per minute (SD=0.09) than the group using the SCP, who uploaded 0.40 new working programs per minute (SD=0.12); $t(5.42) = 3.357$, $p = 0.02$.

5. CONCLUSION

Youth in the workshop who worked with the Spatial Computing Platform (SCP) were more likely to make complex changes to their starter code while also uploading new versions of their code to their robots at a higher rate. Both of these findings provided evidence that the SCP succeeded in making it easier to tinker with the sensors, actuators, and the control structures that link them together. These findings raised two further questions. First, what were the features of the SCP that made it better for tinkering? Second, did participants using the SCP also have a better understanding of the logic connecting the sensors and actuators in the robots they were programming? In order to answer these questions we will return to the videos from the workshop.

To answer the first question we can analyze differences in the software design as well as evidence from the videos. The text-based programming environment provides different tabs for users to interact with the GoGo Board. The Logo tab is where participants write the Logo code and upload it to the GoGo Board. The other tab, the GoGo Console, provides a simple, non-programmable control panel for the GoGo Board. This tab contains buttons and sliders that allow the user to directly interact with the GoGo Board by turning motors on and off, changing their direction, and reading the values of the sensors. All of the groups in the text-based condition repeatedly navigated to this control panel to change the settings of their robots, even after having uploaded Logo code that governed the robots' behavior. There was an interesting behavioral difference within groups between the two tabs: In the Logo tab, the participants were careful, barely making any changes to the code beyond flipping signs and changing numbers. In the Console tab on the other hand, they clicked buttons and dragged sliders frequently and with ease.

The participants using the SCP seemed similarly at ease while drawing rectangles and assigning different outputs to them. However, the SCP is functionally equivalent to the Logo tab, as it is a programmable environment. Thus, the SCP appears to provide the ease of use that comes with a non-programmable control-panel with the functionality of a programming environment.

Looking for the similarities between the GoGo Console and the SCP will help draw out the beneficial features of the SCP. First, in both the GoGo Console and the SCP, users can only make changes using the mouse. This has the effect of reducing the programming language to little more two buttons and gesture. When compared to the programming in Logo, this drastically reduces the number of actions a user needs to make to achieve the same level of functionality (compare Figure 2 and 3). Second, neither the GoGo Console nor the SCP can be put into a broken state. Participants in the Logo group made a number of very minor syntax errors (e.g., mistakenly using a vertical bar instead of a square bracket) that crashed their programs with compiler errors. This type of mistake was not possible in the SCP.

Regarding the second question, there is some evidence that the SCP provided a more transparent pathway for participants to understand how their code was controlling the robot's behavior. In the SCP, the sensor values are represented by a green dot moving across the canvas that updates in real time, meaning both the program output and the programming environment are located on the same screen. Users can see that when the dot moves into different rectangular regions, the outputs assigned to those regions turn on. The youth in our study recognized that the green dot changed along with the amount of light reaching the IR sensors. For example, after only 12 minutes, one of the boys using the SCP exclaimed "Oh, it's when that thing *(the green dot)* goes in there *(the rectangle)* that's when it *(the motor)* goes." He then checked his understanding by drawing very small rectangles around the green dot. Traditional text-based environments do not provide this layer of feedback on top of the Logo code. In order to see the sensor values within most Logo environments the user has to leave the programming tab. In our study, we observed one group using the Logo environment that blindly tried different cutoff thresholds rather than clicking back and forth between the tabs to check the threshold values against the actual sensor values. We interpreted this behavior to mean that the participants in this group did not understand that the selection of threshold values has to be matched to the actual sensor values. This might suggest that this group also had a weaker understanding of the connections between the sensors, code, and actuators.

In sum, the SCP fostered tinkering by providing a simpler and more resilient way to achieve the same functionality as programming in Logo. Furthermore, by displaying the sensor values directly on top of the logical layer in real time, it provided a more transparent pathway for youth to understand how their code controlled their robot's behavior. We believe that these are two important features for a programming environment that introduces children to programming microcontrollers.

6. REFERENCES

[1] Berland, M. et al. 2013. Using Learning Analytics to Understand the Learning Pathways of Novice Programmers. *Journal of the Learning Sciences*. 22, 4 (Oct. 2013), 564–599.

[2] Blikstein, P. (2013, June). Gears of our childhood: constructionist toolkits, robotics, and physical computing, past and future. In *Proceedings of the 12th International Conference on Interaction Design and Children* (pp. 173-182). ACM.

[3] Blikstein, P. and Sipitakiat, A. 2011. QWERTY and the art of designing microcontrollers for children. *Proceedings of the 10th International Conference on Interaction Design and Children* (2011), 234–237.

[4] Millner, A. and Baafi, E. 2011. Modkit: blending and extending approachable platforms for creating computer programs and interactive objects. *Proceedings of the 10th International Conference on Interaction Design and Children* (2011), 250–253.

[5] Puckette, M. and others 1996. Pure Data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer Music Concerts*. (1996), 37–41.

[6] Resnick, M., & Silverman, B. 2005. Some reflections on designing construction kits for kids. In *Proceedings of the 2005 conference on Interaction design and children (pp. 117-122). ACM.*

[7] Resnick, M. et al. 2009. Scratch: programming for all. *Communications of the ACM*. 52, 11 (2009), 60–67.

[8] Resnick, M. and Rosenbaum, E. 2013. Designing for tinkerability. *Design, make, play: Growing the next generation of STEM innovators*. (2013), 163–181.

[9] Sipitakiat, A. et al. 2004. GoGo board: augmenting programmable bricks for economically challenged audiences. *Proceedings of the 6th international conference on Learning sciences* (2004), 481–488.

[10] Sipitakiat, A. (2007). *Giving the Head a Hand* (Doctoral dissertation, Massachusetts Institute of Technology).

[11] Wilensky, U. and Evanston, I. 1999. NetLogo: Center for connected learning and computer-based modeling. *Northwestern University, Evanston, IL*. (1999), 49–52.