Teaching Introductory Programming Concepts Through a Gesture-Based Interface

Lora Streeter(✉) and John Gauch(✉)

University of Arkansas, Fayetteville, AR 72701, USA
{lstrothe, jgauch}@uark.edu

Abstract. The goal of our research is to create and evaluate a visual and gesture-driven interface to teach computer programming to non-traditional programmers, typically school-age children. By making the interface more enjoyable for young students, we hope to keep students engaged and increase their attention span while learning how to program.

Our system combines components from Google's Blockly, a visual block programming language with drag-and-drop puzzle pieces, and Microsoft's Xbox Kinect, which is used to perform skeletal tracking. We created pre-defined gestures to correspond to program functions and available actions, which were compiled from a survey conducted of over 100 grade-school students over three years who had very little to no programming experience before we met. After learning how to use Blockly and having a basic understanding of simple programming logic, the students were asked to create intuitive gestures for common programming constructs, while both standing up using full body movement, and sitting down at a desk, using only their hands. The specific programming constructs included in the survey were loops, conditionals, run program, and undo.

To detect the gestures, we have implemented and evaluated a number of gesture matching algorithms. One challenge is that the size, shape, and path of the gestures varied considerably, so the data has to be normalized for any comparisons.

Keywords: Blockly · Gesture matching · Kinect · Programming
Quantization · Visual programming

1 Introduction

Computer programming is an important field given the rapid advance of technology in recent years, but traditional programming classes are not necessarily directed to the masses. They are typically tailored to people who are either already interested in programming or are old enough (or have enough self-discipline) to sit in a class, take notes, and then go home and experiment with what they have learned in front of a screen. Could there be a better way to teach them? What if, instead of having them sit down and type, they could have a more interactive experience? These are two of the questions that motivate the current research.

C. Stephanidis (Ed.): HCII Posters 2018, CCIS 852, pp. 116–123, 2018.
https://doi.org/10.1007/978-3-319-92285-0_17

Programming is an integral part of the technologically driven society, so there is a need to provide a better way to teach programming to a broader audience. It is an essential career skill that teaches problem solving skills that are broadly useful. Hence, there is a need to grab the attention of non-traditional programmers, draw them into the world of computing, and hold their interest.

Programming with a language meant solely for text analysis or numerical computation can easily discourage people since learning syntax and rules is typically not as enjoyable as manipulating images and completing fun tasks. When people become immersed in learning and having fun, they are more likely to continue onto more challenging topics. It was discovered during some very controlled trials for first-time university level programmers that using a visual language like Alice increased retention by 41%, and the average grade in the class rose an entire letter grade [1].

The goal of our research is to create and evaluate a visual and gesture-driven interface to teach programming to non-traditional programmers, typically school-age children. We have discovered intuitive gestures for specific programming constructs for children, and are working toward recognizing pre-defined gestures. We are also working toward answering the question "Can we teach programming effectively without a mouse?" by using a gesture-driven interface utilizing the Kinect as an input device integrated with Google's Blockly. This paper discusses these intuitive gestures and our gesture matching algorithms.

2 Background and Related Work

2.1 Visual Programming Languages

Visual programming languages have a long history starting around the 1970's and have explored a number of programming paradigms and interfaces. They usually incorporate icons, have a drag-and-drop interface, or are mouse or graphics based – such as Google's Blockly, MIT's Scratch, Carnegie Mellon's Alice, and Berkley's Snap. Google's visual editor, Blockly, enables users to create programs by using a mouse to drag-and-drop connecting puzzle piece blocks together to accomplish a series of goals. After the user has completed a goal, Blockly shows them how many lines their program would have taken in JavaScript. Each task the user is given can be solved with the information they have been provided, and each puzzle builds on the previous in each game.

2.2 Gestural Programming Languages

A gestural programming language is one that takes input as a movement of the hands, face or other parts of the body instead of keyboard or mouse input [2]. Gesture-based languages can involve multi-touch gestures on a tablet device [3], using an image or video as input [4, 5], determining finger locations using a data glove [6], or manual selection of symbolic markers to control a robot [7]. Consumer devices, like mobile phones, tablets, and controller-free sensors such as the Kinect and LeapMotion, all of which are equipped with many sensors, cameras and multi-touch screens, have opened up new possibilities of promoting gestural programming to the broader public.

2.3 Kinect

The Kinect is a popular, inexpensive, three-dimensional camera that has revolutionized human-computer interaction by having depth and RGB cameras in the same easy-to-use unit. Although the Kinect was originally developed for video game input, it has been used as an input device for a wide range of applications, both because it provides a hands-free interface and because it provides physical engagement for users. The OpenNI framework provides an open source API for the Kinect that tracks and reports the positions of fifteen different skeletal joints, including hands.

3 Methodology

3.1 Student Population and Experimental Design

We have had the opportunity to teach and work with a number of young students who are interested in computer science and engineering, and to get feedback from them about gestures they thought were most natural for specific programming concepts. Using this information, we developed a prototype gesture-based visual programming interface that captures gestures using the Kinect and transfers this information to Blockly to create and run programs.

The students who participated in this segment of the study attended various engineering summer camps held at the University of Arkansas for rising 6^{th}–12^{th} graders. Some students arrived with absolutely no programming experience, while others had used a few languages already. Very few, if any, of the students had used Blockly before, although around half of the students had utilized Scratch in the past. For this experiment, we focused on Blockly.

The students got the opportunity to work with Blockly and then filled out a survey at the end of the session about the concepts they learned, and whether they would prefer to program while standing and using full body movements, or sitting and using only hand gestures. They were also asked to create their own gestures for the following programming concepts for both sitting and standing. The questions were left relatively open-ended to allow the students to either describe in words the gestures/movements they would make, or to draw the path of the gesture, or a combination of both. These programming concepts include:

- If statement
- If/else statement
- For loop
- While loop
- Run program
- Undo previous action

After filling out the survey, the students got the opportunity to record their gestures into the computer using the Kinect. Our capture gesture program is written in the language Processing that allows the user to draw a gesture with their hand and the Kinect. The program recorded the coordinates of all of the user's joints, even though all of our data was drawn one-handed, and made note of which joints comprised the

gesture path (typically just the right or left hand). The gesture coordinates were saved into a text file, and a screenshot of the gesture path was saved into a jpg.

3.2 Comparing Gestures

Gesture Capturing

The gesture capture program code was designed to track the full skeleton, because when the research was started, we were not sure if standing and using full body movements or sitting and only using hand gestures would be more popular with students. In fact, 56% of all students surveyed would rather sit and use hand gestures, so we limited our subsequent research to that. However, when we asked the students to design their own gestures for both sitting and standing to see what kinds of results we got, and even when the student was standing and had the ability to use full body movements, the majority of students still chose a two dimensional, one handed gesture.

Common Gestures Among Students

While the students were given relatively open-ended prompts to generate gesture shape information, there were six distinct gestures that appeared consistently throughout the surveys. The most popular gesture was a circle or loop gesture, with 162 gestures being drawn or described under the "for loop" or "while loop" options, either while standing or sitting. 27 students chose a spiral gesture, with almost half of them choosing this gesture for one of the loop options while standing. An additional 22 students chose a "thumbs up" gesture, with 15 of them choosing it to run the program while they were seated at the computer. Finally 18 students chose a wave gesture, while 29 students chose either the figure eight or infinity sign. The rest of the gestures described or drawn were less common or unique.

Therefore, we decided to limit our gesture vocabulary to the circle/loop, infinity sign, figure eight, and wave. The spiral is very similar to a loop, so for the sake of simplicity and distinct gestures, we decided not to focus on it. The code we are currently using for skeletal tracking does not track individual fingers, so the thumbs up gesture was also discarded.

3.3 Gesture Matching Algorithms

An important part of controlling the computer with gestures is being able to recognize when a gesture has been drawn. By far, the most popular gesture created by students was a circle motion for a loop gesture. So to refine the gesture matching algorithms, it was decided to concentrate on matching the loop gesture first.

Template Matching for Loop Gesture

We started with a simple template matching algorithm, where a two-dimensional array was set up with a perfect circle plotted onto it. Instead of drawing the circle as a 1x1 pixel line, we used a 5x5 pixel mask to give the user's gesture some wiggle room so as to not require an exact 1-to-1 match. An advantage of using template matching is that it does not matter where the gesture starts or stops, if it is drawn clockwise or counter-clockwise, or how fast it is drawn. Plotting the user's gesture against the

template and measuring hit and miss percentage should give us a good idea of whether or not the user's gesture matches the template.

The number of coordinates varied greatly between gestures and users, so we normalized all gestures to have the same dataset size of 50 uniformly sampled coordinates from the original saved gesture. We discovered there was no discernable loss of matching precision when comparing 50 pared coordinates instead of up to 195 for the longest recorded gesture.

Minimum/Maximum Scaling Algorithm

To use a template matching algorithm, we needed to normalize our data so that it fits onto the template. Our first attempt was to take the gesture, find the minimum x and y values, and the maximum x and y values, and linearly scale the gesture in both directions independently to fit the template.

As the coordinates for the drawn gesture are being plotted onto the loop gesture template, the number of hits and misses are counted, and then using these, the percentage match is calculated. If the percentage is above a certain threshold, the drawn gesture should be recognized as a match to the template gesture.

Standard Deviation Scaling Algorithm

While the minimum/maximum scaling algorithm works relatively well, one problem is that it is designed to match a saved gesture, not a real-time gesture. The files we are using for comparison were started when the user was ready to start and finished when they completed the gesture, so the minimums and maximums are relatively accurate to create a bounding box for the gesture. However, when trying to compare a gesture from a real-time feed, there is no predefined start and stop, so we still need to be able to detect it. For example, if the user reaches up and scratches their head before drawing a loop, how can we detect that and ignore it as irrelevant data?

Our first implementation was to try using standard deviation in the hope that it would help eliminate the irrelevant "straggler" coordinates before and after the intended gesture. The thought was that we should be able to take something like the number "9" and be able to detect where the loop section starts and ends without including the leg of the number. To this end, the x_{mean} and y_{mean} were calculated separately to come up with both x_{stddev} and y_{stddev}. Using this mean and standard deviation, we defined a bounding box that was k_x standard deviations units wide and k_y standard deviations tall, and we used those values to be the floor and ceiling of x and y coordinates of the gesture respectively.

Sector Quantization

Sector quantization is simply taking a large set of data and condensing it down to a smaller, more easily understandable set of data. We follow the path of the gesture and note which sectors it appears in and in what order. We have the paths of clockwise and counter-clockwise loops, figure eights, and infinity symbols predefined in the gesture matching code, and we compare each user provided gesture path to it, and then accept the largest matching percentage as the most likely candidate.

The current version of the code is written so that the predefined paths circle around twice. Instead of just 2, 3, 6, 9, 8, 7, 4, 1, 2 (see Fig. 1A) for a clockwise loop, the path is doubled to 2, 3, 6, 9, 8, 7, 4, 1, 2, 3, 6, 9, 8, 7, 4, 1, 2 so sub-lists of the arrays can be

utilized without having to wrap the array index back to zero. Because the predefined paths are almost twice as long as the actual path we are trying to match, either the match percentage must be doubled, or the threshold of matching must be halved. We chose to go with the latter; therefore, the matching percentages look lower, but they are being compared to a predefined gesture that is almost twice as long as the expected gesture. An ideal loop gesture would visit eight or nine sectors, depending whether it starts and ends in the same one, or stops just short of completing the path. Our predefined loop gesture visits 17 sectors, visiting each one twice, and the initial sector three times. Thus, a valid loop gesture match should have at least a 47% match percentage (8 sectors/17 possible locations).

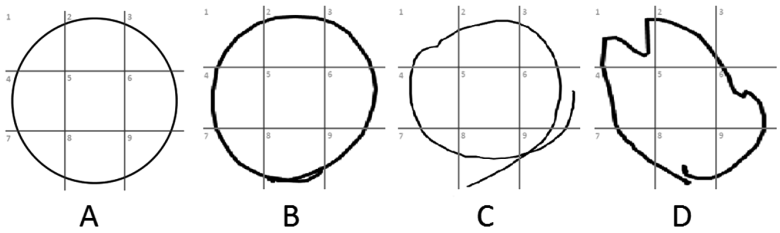

Fig. 1. Quantized gestures with sector grid; A. Ideal circle, and B–D. Three user gestures

4 Results and Analysis

4.1 Data General Analysis

Using the data collected in 435 gesture files, we have discovered that the average number of coordinates recorded for a gesture is around 94. This additional data was collected by giving young adults interested in programming or computer networking specific directions for what to draw ("draw a loop"), but not specifying where to start, which direction to go, or which hand to use. With the information in Table 1, we should be able to look at around 150 coordinates at a time to decide whether the person's movement contains a recognizable loop gesture. For the purposes of this paper, we will only be looking at the matching accuracy of the loop gesture.

Table 1. Data gathered

	Files	Total coordinates	Average coordinates	Minimum	Maximum
Figure eight	119	10973	92	42	195
Infinity	122	11953	97	39	185
Loop	129	10173	78	38	157
Spiral	40	5890	147	97	193
Wave	27	2138	79	53	131
All	437	41127	94	38	195

We tried a simple minimum/maximum scaling to linearly scale the gesture to a template. When the gesture was well-drawn and easily recognizable as a circle, this algorithm worked well (Fig. 1B). However, when the gesture had significant overlap (Fig. 1C), or was drawn shakily (Fig. 1D), the gestures did not match as well.

Next, to remove "straggler" points that were not part of the gesture itself, we tried a standard deviation scaling algorithm with many different values for sigma. Unfortunately, there was no one good value for sigma across all gestures, and the matching percentage rarely went above 50.

We are currently working with quantization, cutting down our gesture path to a three-by-three grid of nine total sectors. Requiring the gesture to match at least 47% of the path's sector order we expect to see for a loop gave us 124 matching files out of our test group of 129 (96% success rate). The percent match rate for each of these three algorithms for our sample three gestures (see Fig. 1B, C and D) is shown in Table 2.

Table 2. Percents matched for gestures

	Total coordinates	Template matching				Sector quantization
		Minimum/Maximum Scaling		Standard deviation scaling - all data		
		All data	Pared data	Sigma = 1.75	Sigma = 1.9	
B	68	91.18%	94.00%	12.50%	16.96%	58.82%
C	75	49.33%	48.00%	30.67%	6.67%	64.71%
D	112	33.04%	34.00%	11.48%	16.39%	52.94%

By requiring at least a 90% match for the template matching algorithms, and at least a 47% match on sector quantization to confirm that a drawn gesture matches what our program expects to see for a predefined gesture, template matching was not nearly as successful as sector quantization. When we break template matching into two subsections, none of the standard deviation scaling options give us a match, while only a very precise loop is matched using the minimum/maximum scaling. By using sector quantization, all three user drawn gestures are matched as a loop gesture.

While the minimum/maximum scaling algorithm gave better results on a consistent basis than the standard deviation scaling approach for template matching, we have discovered that sector quantization holds a lot of promise for future matching efforts.

5 Conclusion and Future Work

The goal of our research is to create and evaluate a visual and gesture-driven interface to teach programming to non-traditional programmers, typically school-age children. Based on surveys given to several groups of young students, we discovered that when given the option between sitting and using hand gestures, or standing and using full body movements, the majority of students choose a one-handed, two dimensional gesture, regardless of sitting or standing. This was counter-intuitive since we expected the students to use three-dimensional space and/or utilize more than their dominant hand.

Our current system has a limited gesture vocabulary that includes loops, figure eights, infinity signs, and waves. We would like to expand this in the future and allow students to create their own gestures for key programming constructs.

We have also implemented and evaluated several gesture matching algorithms. While template matching was not as successful as we were hoping, sector quantization holds a lot of promise and we are currently incorporating this into our application. When our gesture-based programming interface is completed, we would like to evaluate the effectiveness of this approach with additional groups of young prospective programmers.

References

1. Moskal, B., Lurie, D., Cooper, S.: Evaluating the effectiveness of a new instructional approach. SIGCSE Bull. **36**(1), 75–79 (2004). https://doi.org/10.1145/1028174.971328
2. Hoste, L., Signer, B.: Criteria, challenges and opportunities for gesture programming languages. In: Proceedings of EGMI, pp. 22–29 (2014)
3. Lü, H., Li, Y.: Gesture coder: a tool for programming multi-touch gestures by demonstration. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2012), pp. 2875–2884. ACM, New York (2012). http://dx.doi.org/10.1145/2207676.2208693
4. Kato, J.: Integrated visual representations for programming with real-world input and output. In: Proceedings of the Adjunct Publication of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST 2013 Adjunct), pp. 57–60. ACM, New York (2013). https://doi.org/10.1145/2508468.2508476
5. Kato, J., Igarashi, T.: VisionSketch: integrated support for example-centric programming of image processing applications. In: Proceedings of Graphics Interface 2014 (GI 2014), Canadian Information Processing Society, Toronto, Ont., Canada, Canada, pp. 115–122 (2014)
6. Kavakli, M., Taylor, M., Trapeznikov, A.: Designing in virtual reality (DesIRe): a gesture-based interface. In: Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts (DIMEA 2007), pp. 131–136. ACM, New York (2007). http://dx.doi.org/10.1145/1306813.1306842
7. Dudek, G., Sattar, J., Xu, A.: A visual language for robot control and programming: a human-interface study. In: Proceedings of the International Conference on Robotics and Automation ICRA, Rome, Italy, April 2007