The WOZ Recognizer: A Tool For Understanding User Perceptions of Sketch-Based Interfaces

Jared Bott
University of Central Florida

THE WOZ RECOGNIZER: A TOOL FOR UNDERSTANDING USER PERCEPTIONS OF SKETCH-BASED INTERFACES

by

JARED BOTT

M.S., Computer Science, University of Central Florida, 2009

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2016

Major Professor: Joseph J. LaViola Jr.

ABSTRACT

Sketch recognition has the potential to be an important input method for computers in the coming years; however, designing and building an accurate and sophisticated sketch recognition system is a time consuming and daunting task. Since sketch recognition is still at a level where mistakes are common, it is important to understand how users perceive and tolerate recognition errors and other user interface elements with these imperfect systems. A problem in performing this type of research is that we cannot easily control aspects of recognition in order to rigorously study the systems. We performed a study examining user perceptions of three pen-based systems for creating logic gate diagrams: a sketch-based interface, a WIMP-based interface, and a hybrid interface that combined elements of sketching and WIMP. We found that users preferred the sketch-based interface and we identified important criteria for pen-based application design. This work exposed the issue of studying recognition systems without fine-grained control over accuracy, recognition mode, and other recognizer properties. In order to solve this problem, we developed a Wizard of Oz sketch recognition tool, the WOZ Recognizer, that supports controlled symbol and position accuracy and batch and streaming recognition modes for a variety of sketching domains. We present the design of the WOZ Recognizer, modeling recognition domains using graphs, symbol alphabets, and grammars; and discuss the types of recognition errors we included in its design. Further, we discuss how the WOZ Recognizer simulates sketch recognition, controlling the WOZ Recognizer, and how users interact with it. In addition, we present an evaluative user study of the WOZ Recognizer and the lessons we learned.

We have used the WOZ Recognizer to perform two user studies examining user perceptions of sketch recognition; both studies focused on mathematical sketching. In the first study, we examined whether users prefer recognition feedback now (real-time recognition) or later (batch recognition) in relation to different recognition accuracies and sketch complexities. We found that participants displayed a preference for real-time recognition in some situations (multiple expressions, low accuracy), but no statistical preference in others. In our second study, we examined whether users displayed a greater tolerance for recognition errors when they used mathematical sketching applications they found interesting or useful compared to applications they found less interesting. Participants felt they had a greater tolerance for the applications they preferred, although our statistical analysis did not positively support this.

In addition to the research already performed, we propose several avenues for future research into user perceptions of sketch recognition that we believe will be of value to sketch recognizer researchers and application designers.

ACKNOWLEDGMENTS

I would like to thank my wife, Maria, for her kindness, patience, and love as I slowly made progress on my dissertation. I would also like to thank my father Kevin and my mother Barbara for helping me throughout college. Thank you Dad for supporting us during my graduate work. Thanks also go out to my brothers and to my friends. Special thanks to my daughter Lilly.

Furthermore, I would like to thank my faculty advisor, Dr. Joseph LaViola, for his guidance and support. I would like to thank the members of my committee, Dr. Charles Hughes, Dr. Hassan Foroosh, and Dr. Ed Lank for their direction and guidance. Finally, I would like to thank the past and present members of the Interactive Systems and User Experience Lab for their help.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

While touchscreen computers have been popular for several years now, stylus-based input has started to make significant strides into mainstream. Tablets and smartphones often have styluses available for them. Not only are styluses gaining in popularity, but touch-based handwriting input is also common. Recent operating systems from Microsoft, Apple, and Google contain handwriting recognition applications and services, putting handwriting recognition within easy reach of users worldwide.

Within this context, sketch recognition systems have the potential to be useful in STEM education for example, because they provide an easy to understand, natural input method for complex structures. Drawing and writing are familiar acts for human beings, because we learn to draw from an early age. Using a stylus with a computer can be very similar to using pen and paper, which makes sketching a natural input mechanism for free body diagrams, chemistry bond diagrams, and algebraic equations. In addition, handwritten expressions and drawings can be a powerful component of intelligent tutoring systems and computational engines [11, 14, 31, 35, 41, 62]. The ability to take written expressions and perform complicated actions on them, such as graphing or solving them, can be enticing and exciting for users [37].

However, a major challenge with sketch recognizers is that they are error prone [25]. Thus, understanding how people perceive and tolerate these recognition errors is an important research question because the answers to this question can provide insight into how to design better interfaces for dealing with and mitigating recognizer error. For example, people view errors quite differently in terms of accuracy, and have many different reasons for liking one recognition mode or another [9].

Section 1.1 Statement Of Research Question

A major challenge in sketch recognition is understanding how people perceive recognition, in particular, because it is hard to understand what "good enough" recognition really is. How close to perfect does recognition need to be? Is a recognizer that consistently makes one or two errors as good as a recognizer that is mostly perfect, but bungles an entire sketch on occasion? How are accuracy constraints related to other recognizer properties when it comes to "good enough" recognition? In order to explore how recognition parameters affects user preferences, we need a recognizer in which these parameters can be controlled. Recognizers are not designed in this way; we cannot just take a recognizer and make it 75% or 80% accurate for all users so that we can study it. Input is often ambiguous to a recognizer in some way; the output for one user may be completely correct, but another user may received output riddled with recognition errors. As recognition accuracy changes, we expect that user perceptions of the recognition system will change, but we need to control accuracy if we want to rigorously study how they change. We cannot control recognition accuracy, and if we want to study users working with these systems, we need a way to control the various recognition and user interface parameters. Thus, it has been difficult to explore user perceptions and preferences for sketch recognition, since we cannot control how well a recognizer will perform with a given person's writing. Solving this problem is not easy and many researchers focus on achieving greater accuracy, without consideration for "good enough"; consequently, there has been little research in this area.

To this end, we designed and implemented a Wizard of Oz sketch recognition system, the WOZ Recognizer. The WOZ Recognizer allows a user to simulate realistic recognition of different types of sketches in order to perform studies on sketch recognition that would otherwise not be possible with traditional sketch recognizers. In addition to building the WOZ Recognizer, we set out to ascertain whether it was possible to build a Wizard of Oz recognition system that produced believable results and was usable for performing studies on sketch recognition.

In this work, we seek to explore and understand how people perceive sketch recognition. We believe that sketching is a useful input method for computers; drawing and writing are natural acts for human beings. We learn to draw in real and abstract ways from an early age. By harnessing a quick and easy input method, we can transform the way we interact with computers.

The field of sketch recognition is vast; we cannot hope to explore perceptions of all kinds of sketches in this work. We began our work with a study on logic diagrams, where we examined interfaces for creating logic gate diagrams. Each interface used a different form of interaction and we examined how users perceived their qualities (such as speed and ease of use). Working with logic gate diagrams led us to create the WOZ Recognizer to support studying sketch-based interfaces, when we realized that we needed to create a tool that let us control recognition accuracy and realistically simulate recognition. Once we created the WOZ Recognizer, we focused on mathematical sketching and later broadened the WOZ Recognizer to include diagram sketching.

In the world of recognition, accuracy is an important area to study, because it is imperfect and because it seems to be intrinsically linked to the usability of a recognizer or interface. Therefore, much of our work has focused on producing a controlled accuracy recognizer and examining how accuracy affects users.

We have performed two studies on mathematical sketching with the WOZ Recognizer. In the first study of mathematics-based sketching, we sought to answer whether people prefer real-time or batch recognition for mathematical sketches, and whether that preference was related to recognition accuracy or the number of mathematical expressions written. In the second study, we wanted to know whether user interest in an application affects their tolerance for recognition errors. To do this, we examined how users tolerate recognition errors for a set of four mathematical sketching applications and analyzed the data to determine whether the participants tolerated more errors for the applications they found more interesting. In these studies, we wanted to examine recognition in a controlled manner, where we can examine how users feel about very specific levels of accuracy. In this context, the WOZ Recognizer was necessary for our research, as it allowed us to control recognition accuracy. Finally, we performed an evaluative study of the WOZ Recognizer in order to examine its usability and improve it for future use.

Section 1.2 Contributions

In this dissertation we:

- Developed a tool for studying user perceptions of sketch recognizers
- Studied mathematics sketch recognition with controlled accuracy.

In order to realize these contributions, we:

- Performed a study exploring user perceptions of different input modalities with applications for creating logic gate diagrams. Most relevantly, we found that users' perceived a sketch-based application to be faster than the hybrid interface, when the hybrid interface was statistically faster.
- Performed an evaluative study of the WOZ Recognizer, where experts in sketch-based interfaces learned to use the WOZ Recognizer and evaluated it.

- In our two user studies on mathematics-based sketches, we have examined user perceptions, feelings, and preferences with regards to mathematics sketch recognition.
  - In the first study, we have found that people have a preference for real-time recognition in many situations, but not all.
  - In the second study, we found that users perceived a greater tolerance for recognition errors with mathematical sketching applications that they found more interesting or useful.

Section 1.3 Organization

In Chapter 2, we discuss work related to user perceptions of handwriting and sketch recognition, and Wizard of Oz studies. Next, we present our work on evaluating user perceptions of interfaces for logic gate diagram creation in Chapter 3, which led to the creation of the WOZ Recognizer. Following that, in Chapter 4, we discuss the WOZ Recognizer in detail, including our evaluative study. Chapter 5 presents the first study we performed using the WOZ Recognizer, which examines user preference for recognition mode and how it relates to recognition accuracy and sketch size. Our second study using the WOZ Recognizer, which examines mathematical sketching applications and user tolerance for recognition errors, is presented in Chapter 6. Next, we present a discussion of our findings and future work avenues in Chapter 7. Finally, we summarize our findings in Chapter 8 and present questionnaires, etc. in the Appendices.

CHAPTER 2: RELATED WORK

Section 2.1 Wizard of Oz

A Wizard of Oz system is one where a human performs some of the actions that a computer would normally perform, such as recognition of speech or handwriting. In a Wizard of Oz system, the presence and actions of the human (the wizard) are hidden from the general public, such that the wizard's involvement is known only to the researchers. Wizard of Oz systems in structured handwriting recognition have not been widely used due to the complexity in simulating recognition. For example, mathematics expressions have complicated interactions between symbols; small changes in parsing can create vastly different recognized expressions. Consequently, realistically simulating math handwriting recognizers has rarely been attempted.

However, Wizard of Oz systems have been developed for handwriting recognition in other domains. LaLomia performed a Wizard of Oz study to examine user acceptance of handwriting recognizers [36]. Different accuracy rates, modes of presentation, and types of input form were tested; only accuracy rates were found to have an effect on user acceptance.

Errors were randomly inserted into the output to achieve the different accuracy rates. Our work differs in its focus, as we are looking at general sketch recognition for mathematics and diagrams, and are inserting errors in a controlled manner based upon real mathematics handwriting recognizers [10]. Read et al. performed a similar experiment with children using batch recognition [53]. Neither study focused on mathematics. We think that these results are not applicable to the math domain because math has a complex structure not found in normal writing.

Oviatt et al. use a Wizard of Oz scenario to examine how students adapt their computer input while using a dual input (speech and pen) system to solve mathematical problems [51]. Specifically, they looked at how the students changed their vocal levels and pen pressure when the computer system did not recognize the students were talking to it. Anthony's work on mathematics input for intelligent tutoring explores how different modes of input affect user learning, input speed, and cognitive load among other things, and included the use of Wizard of Oz studies in some cases [4]. Anthony et al. also performed an experiment to examine how input modality affects student learning [5]. They examined two input modalities, handwriting and typing, and used a Wizard of Oz methodology to provide feedback to the participants. They found that students learned as much using handwriting as typing, but took less time to solve all of the problems using writing compared to typing. These studies focus more on the Wizard of Oz experiments, while our work focuses on a tool to help people perform Wizard of Oz experiments.

Wizard of Oz systems for simulating user interfaces have also been created. WOZ Pro is a pen-based system for prototyping user interfaces [28]. WOZ Pro uses pen input to create user interfaces, but is not focused on simulating pen-based interfaces. In contrast to WOZ Pro, SketchWizard is a Wizard of Oz tool for designing and testing pen-based prototypes [17]. It provides tools for user interface design and simulating those designs using a Wizard of Oz approach, but supports different domains and does not have controlled recognition accuracy. SketchWizard is focused on user interfaces for sketching systems, while we are focused on sketch recognition itself. UISKEI is another pen-based tool for prototyping user interfaces, with the ability to define complex UI interactions [56]. Interface elements are created through sketching in UISKEI, and behaviors are defined through the use of traditional WIMP input and gestures. In contrast to the WOZ Recognizer, these tools are largely focused on simulating user interfaces, while we are focused on simulating sketch recognition.

Wizard of Oz tools for speech recognition have also been developed. SUEDE is a speech interface prototyping tool that uses Wizard of Oz recognition [32]. ARTUR is a Wizard of Oz system for helping children with language disorders through speech training [7]. In both systems, a wizard performs the act of recognition. Jeanie is a prototype calendar using a multimodal speech and pen recognition system [23]. A series of Wizard of Oz experiments were performed in order to gauge how the system would be used.

Section 2.2 Sketching

Researchers have examined the interactions users have with pen-based interfaces. One area of pen-based interfaces that is especially interesting is sketching, where users leverage their existing notions of pens and drawing to create complex, natural 2D drawings. Structured 2D languages are well-suited for sketching as they are often complex; sketch-based interfaces have been created for many kinds of structured 2D languages. CircuitBoard is a sketch-based system for creating and working with logic circuit diagrams [68]. Gennari et al. present a system for electrical circuit diagrams, AC-SPARC [22]. SketchRead is a sketch recognition system for diagram-based sketches, such as family tree and electrical circuit diagrams, and can be extended to new domains [3]. MathPad^2 combines mathematics expressions and physics drawings in a system for mathematics problem solving [39].

Sketching is also used for other domains that are less structured. InkAnchor is a finger-based sketching system for mobile phone note-taking [55]. Taele and Hammond created two sketch-based “intelligent language workbook” interfaces for learning Japanese and Chinese, Hashigo and LAMPS [60]. These interfaces use a 2D grammar specification language to provide symbol recognition, which is also used to help evaluate the user’s writing correctness. SketchIT is a gestural and sketch-based system for structural analysis [21].

Some researchers have created new pen-based interaction techniques exploring the natural interactions users have with pens. Crossing is a technique in which strokes replace mouse movements and clicks [1]. CrossY uses the crossing technique to create a system where single strokes can be used to perform complex UI interactions [6]. Other researchers have tried to fuse different interaction techniques to create more task-oriented pen-based tools, such as DENIM [43]. DENIM is an informal pen-based sketch system created to rapidly design websites by sketching pages and interacting with them using gestures and visible controls.

Section 2.3 Diagrams

Reid et al. describe techniques for beautifying handwritten graphs [54]. While we use graph-based diagrams in the WOZ Recognizer, graph layout is performed a priori to recognition. SmartVisio is a Microsoft Visio-based system for diagram sketching that provides batch and streaming recognition and easy correction facilities [67]. DiaWoZ II is a Wizard of Oz tool for performing mathematics and engineering experiments [8]. It allows a user to input mathematics by typing using LaTeXand a set notation. A wizard evaluates the input, acting as a tutoring mathematics engine. In contrast to DiaWoZ II, we are focused on mathematics and diagram recognition, and use the Wizard of Oz paradigm to simulate recognition, not mathematical and natural language understanding. Our work differs from others that focus on user interface design by examining how recognition affects users. SketchInsight is a Wizard of Oz application for exploring pen and touch interactions for information visualization [65]. The SketchInsight work has been more exploratory in how users interacted with the application, rather than a controlled examination of how system parameters affect users. This work provided insights into the types of gestures and interactions used by participants in their study. With regards to pen input, they found that certain actions (such as rearranging) were done with touch interactions, while others like erasing were mixed between pen and touch. Our work with the WOZ Recognizer greatly differs from SketchInsight, as their Wizard of Oz system has no notion of recognition accuracy.

Forsberg et. al. investigated the applicability of pen computing in desktop environments with a diagramming task using mouse-and-keyboard and pen-based techniques [19]. This work is closely related to ours, but we are solely investigating user perceptions of pen-based interfaces without examining mouse and keyboard interfaces.

Section 2.4 Handwriting Recognition

While mathematical handwriting recognition has been explored in detail, most research focuses upon recognition algorithms. How to measure the accuracy of a recognized handwritten equation is the subject of much debate [13, 50, 58]. For instance, Anthony used Levenshtein word distance as the measurement for symbol accuracy [4]. LaViola used two metrics, a symbol accuracy metric and a position accuracy metric in [37]. We have taken a similar approach, using both a symbol and a position accuracy metric.

The effects of different accuracy rates on users are rarely explored; this is not surprising as handwriting recognizers are not designed to have controlled accuracy, making it hard to perform such studies. Field et al. examined the effect of task on recognition accuracy and looked at how gathering training data using three different tasks (isolated shape copying, diagram copying, and diagram synthesis) affected classification [18]. They found that training from different tasks' data had little effect on accuracy when using a "factory-trained" recognizer, one which has not been specifically trained for the person using it.

Some work on user acceptance of handwriting recognition has been performed, though to the best of our knowledge none has been done on mathematics. We believe that the highly structured format of mathematics may change how users feel about recognition accuracy. Frankish et al. examined how user acceptance of pen interfaces was affected by different tasks [20]. They found that some tasks, such as a database application, were accepted by users in spite of poor recognition accuracy.

Recognition feedback is another area that has been less thoroughly explored. In [38], LaViola et al. explore different methods for displaying recognized mathematical expressions and conclude that small, typeset expressions displayed below the user's writing was most preferred. However, they did not control for recognition accuracy or explore recognition feedback modes. Math Boxes presents a system for presenting recognition feedback for mathematics expressions [61]. The system displays nested, color-coded bounding boxes around the user's input, which are used to show the user how symbols are grouped together. The system was compared to offset typeset feedback and compared favorably, surpassing the effectiveness of the offset results for complex expressions. Wais et al. explored a variety of methods for triggering recognition, indicating strokes to recognize, displaying recognition feedback, and the effects of common recognition errors on user experience [64]. Their focus was upon sketch recognition and did not explore recognition accuracy, nor mathematical handwriting recognition.

Other aspects of handwriting recognition interfaces have been explored. Perteneder et al. present cLuster, a tool that assists in user selection of sketch strokes [52]. Mankoff et al. describe interface techniques to help deal with recognition errors. They also describe a toolkit they have modified that implements the techniques [46]. CorrectionPanels describes an interface component that provides various techniques that aid in error recognition, including font manipulation, text-to-speed readings of the recognized output, tooltips, and spellchecking [59].

One area we are interested in examining is differences in user perceptions of recognition when working with different types of sketch input device, such as a finger versus a stylus. In [45], the viability of an Apple iPad tablet device for sketch recognition was examined. They determined that the iPad was a viable device for sketch input by comparing it with a tablet PC. Vatavu et al. performed studies to examine user perceptions of making gestures with a stylus [63]. They found that gesture difficulty was most closely correlated with completion time.

CHAPTER 3: LOGICPAD

Section 3.1 Introduction

Our initial exploration into user perceptions of sketch-based interfaces started with a study of pen-based interfaces for creating logic gate diagrams. In previous work with sketch-based interfaces, we sought to create interfaces that closely mimicked pen and paper. This similarity to pen and paper often leads to people saying that sketch-based interfaces are the best method for structured 2D languages, such as mathematics, chemical bond diagrams, music notation, and circuit diagrams [25]. Using a pen to draw and write gives us a high degree of expressiveness, which makes writing as an input mechanism for a computer a powerful tool. Researchers investigate new pen-based interfaces and methods for new domains [29], and work is performed on making recognizers better [22,26,66]. While similarity to pen and paper give us familiarity and expressiveness, it does not show that pen-based interfaces are ideal. Are they ideal, and if so, what makes them ideal? While naturalism (similarity to pen and paper) is important, is it more essential than other factors like speed and ease of use? How should be we continue our research into pen-based interfaces for structured 2D languages?

To try to explore these questions, we devised a study to determine if people would prefer a sketch interface with 100% recognition accuracy over other computer interfaces that were faster and more traditional. We performed a study where we asked participants to use and compare three interfaces for creating logic diagrams. We used three interfaces, a sketch interface (see Figure 3.1), a drag-and-drop interface (see Figure 3.2), and a hybrid interface that combined handwritten wires and labels from the sketch interface with the spatial UI from the drag-and-drop interface and a pie menu for creating gates (see Figure 3.3).

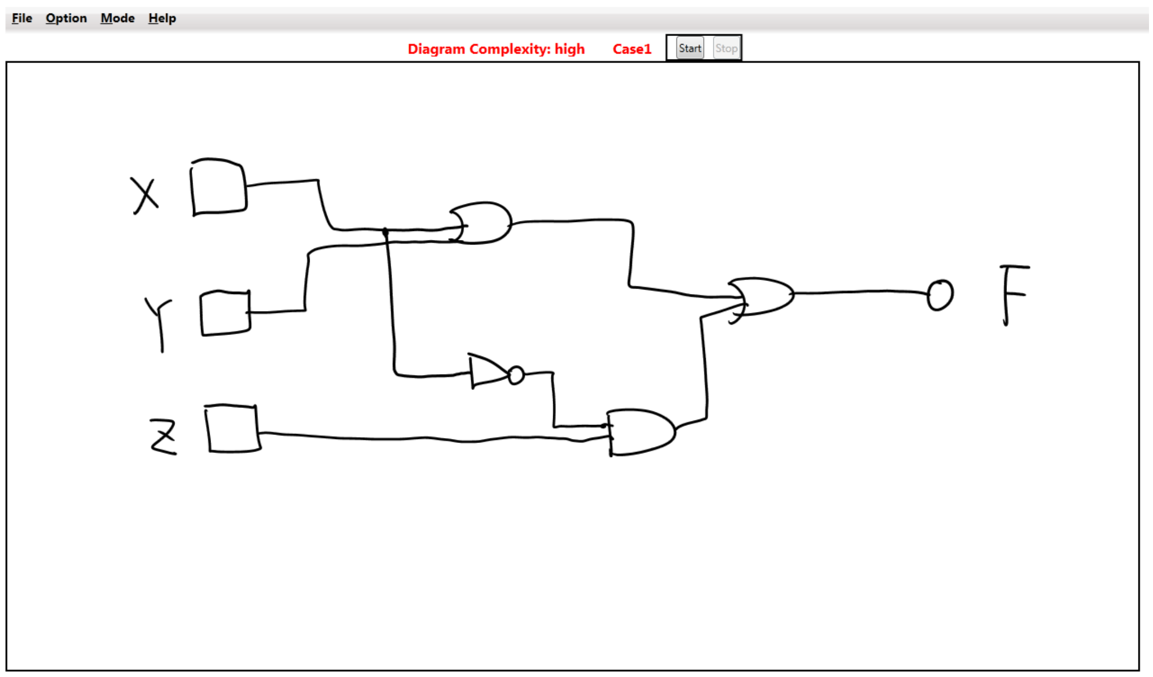


Figure 3.1: An ideal natural 100% recognition accuracy sketch interface as if using pen and paper.

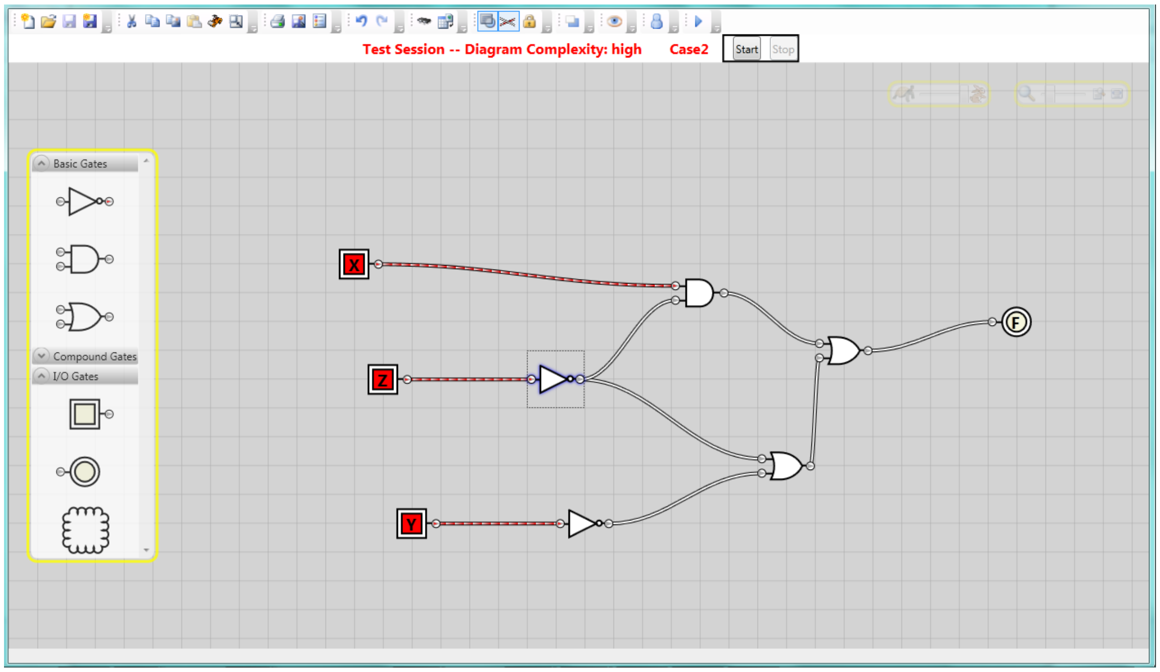


Figure 3.2: A drag-and-drop interface with a static logic gate toolbar on the left side to drag and drop gates.

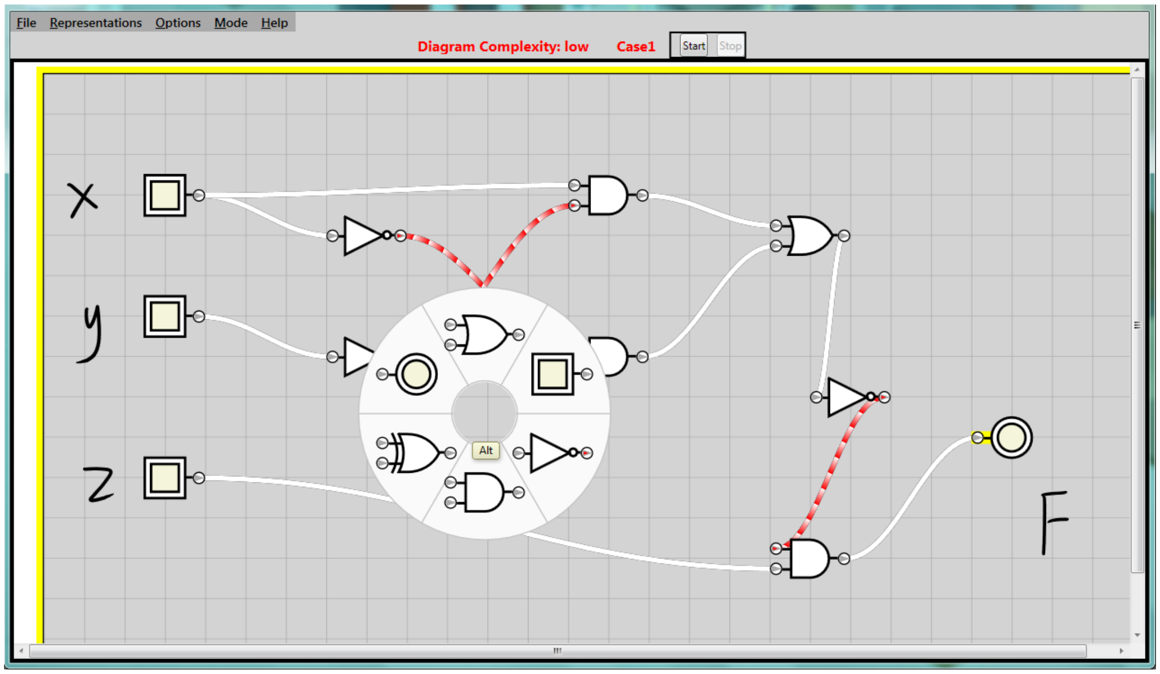


Figure 3.3: A hybrid interface, combining features from sketch and drag-and-drop, with a radial menu for creating gates.

Section 3.2 User Interfaces

Section 3.2.1 Sketch Interface

While we hypothesized that recognition accuracy plays an important role in user perceptions of sketch recognition, we chose to provide complete recognition for our study, so that accuracy would not be a factor when comparing interfaces. We are more interested in experimental validity for ideal recognition rates, than examining current state-of-the-art recognition rates. We chose to take a Wizard of Oz approach in our logic diagram sketch interface, which takes recognition accuracy out of the equation (see Figure 3.1). In order to let the sketch interface be as natural as possible, we simulate batch recognition (recognition after the completion of writing, invoked by performing an action) and delay feedback to users until they invoke recognition. With streaming latency, we have a greater risk of introducing feedback delays for each symbol that we are displaying to the user. Batch recognition allows us to remove the issue. A Wizard of Oz approach means that the sketch interface did not perform any recognition to transform the input strokes into gates, wires, or labels. Instead, the correct feedback is provided to the user once recognition has been invoked. In other words, participants sketched an entire diagram and then had the experimenter invoke recognition. Participants were free to sketch logic gates, wires, and labels as they saw fit without constraints. We implemented a scribble-erase gesture for participants to erase ink strokes. Feedback came in the form of a Boolean equation for the sketched diagram, which was part of the Wizard of Oz approach. We chose to forgo other more direct forms of recognition feedback because we wanted to eliminate distractions and to be as close to pen and paper as possible.

Section 3.2.2 Drag-and-Drop Interface

We implemented an interaction model that would be familiar to most users, a traditional WIMP-based interface [12]. We based this interface on [33] (see Figure 3.2). A static toolbar on one side of the interface provides gates that can be dragged onto the workspace. Wires are created by tapping on one terminal in a gate, and dragging the stylus to another gates's terminal. Labeling a gate was performed using the stylus and the keyboard. Wires and gates can be deleted by selecting the item(s) and using the delete key on the keyboard. Wires can also be deleted by tapping on them.

Section 3.2.3 Hybrid Interface

We developed a hybrid drag-and-drop and sketch interface (see Figure 3.3) similar to part of LogicPad [30]. We wanted to take the strengths of the sketch and drag-and-drop interfaces and combine them to create a fast, easy to use interface. With the complex symbols in a logic diagram, we thought that users would find it quicker and easier to drag a gate from a toolbar than sketch it. However, with wires and labels, we thought that it would be quicker and easier to draw them with a stylus, so we combined these two methods in our hybrid interface.

While dragging and dropping gates from a toolbar can be easy, it is not particularly efficient to drag from a static menu on one section of the screen. In order to make our hybrid interface faster and more usable for stylus input, we adopted a gesture-based method to trigger the logic gate toolbar, a radial menu [34]. The radial menu appears in-place where the tap-and-hold gesture has been performed, keeping the user from having to move across the screen to create a gate. From our pilot studies, we determined a hold time of 0.5 seconds worked best for triggering the radial menu, as it was long enough that users rarely triggered the menu by accident, and short enough that they did not have to wait long.

We used the same gate visualization in the drag-and-drop and hybrid interfaces. Most gates have two input terminals and an output terminal, and terminals are used to start and finish wires. Wires are made in much the same way as in the drag-and-drop interface: the user draws from one gates terminal to another gates terminal. In contrast to LogicPad, we developed a visualization to help users to create wires more efficiently. While creating a wire, when the stylus is near a terminal, the terminal is highlighted yellow, indicating that the user is close enough for a wire to be created by raising the stylus from the screen. Also, in contrast to LogicPad, we thought that users might want to branch wires, so they can do so by making a dot on an existing wire and drawing toward a gate. This will create a wire from the starting terminal of the branched wire.

Handwriting recognition is used to provide gate labels; after recognition, we display a typeset label in place of the ink strokes. As with the sketch interface, wires, gates, and labels can be erased with a scribble-erase gesture.

Section 3.3 Experimental Study

We conducted an experiment designed to explore the usability of our three interfaces. Based on our initial observations, we hypothesized that:

- Primary: Overall, participants will prefer the sketch interface over the hybrid and drag-and-drop interfaces.
- Secondary: The hybrid interface will be faster than the sketch and drag-and-drop interfaces.
- Secondary: The sketch interface will be rated more natural than the hybrid and drag-and-drop interfaces.

While we thought participants would spend less time copying diagrams with the hybrid interface, they would prefer the sketch interface over the hybrid and drag-and-drop interfaces because of the sketch interfaces naturalness.

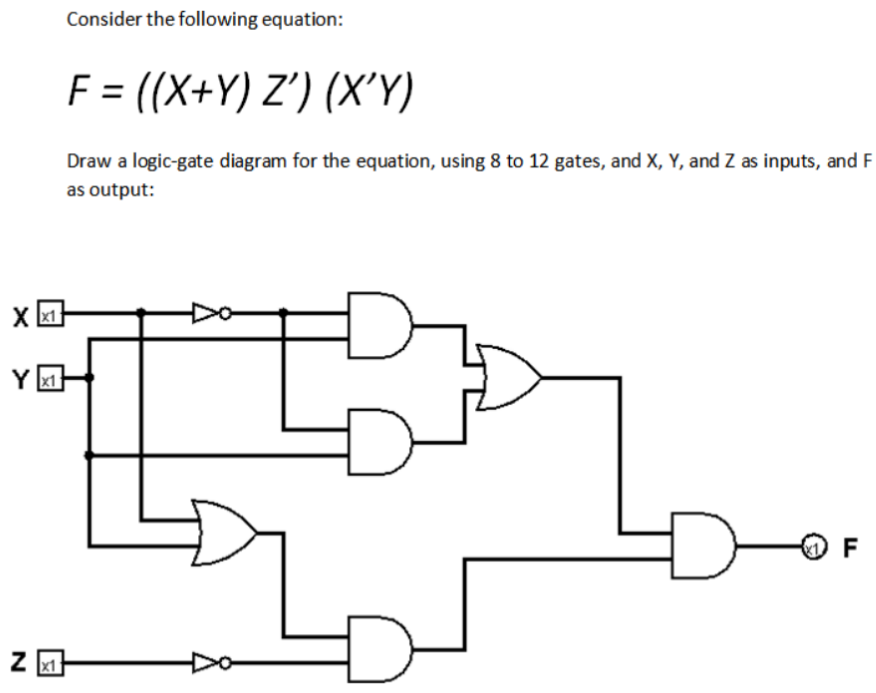
Section 3.3.1 Subjects and Apparatus

We recruited 18 participants (15 male, 3 female) to participate in our study. The mean participant age was 24, with the youngest 19 and the oldest 30. All participants were college students enrolled in electrical engineering, computer engineering, and computer science programs, and they all had taken a discrete mathematics course that covered Boolean algebra. We chose this population because they had knowledge of Boolean algebra and logic diagrams.

Participants worked on an HP Compaq tc4400, 12.1 inch tablet PC running Windows 7. The participants computer was remotely monitored by the researchers using screen-sharing software.

Section 3.3.2 Experimental Task

We asked participants to complete three copy-and-verify tasks, one for each interface. Each task was framed as a homework assignment participants were asked to grade. Each question on the assignment gave a Boolean equation and asked the "student" to draw a logic diagram for the equation (see Figure 3.4). Participants were asked to input the diagram using the specified interface and verify whether the student had drawn the diagram correctly. Therefore, each participant graded three homework assignments. Each homework assignment consisted of six questions divided into two sections, where each section was based on diagram complexity.


Consider the following equation:

$F = ((X+Y)\, Z')\, (X'Y)$

Draw a logic-gate diagram for the equation, using 8 to 12 gates, and X, Y, and Z as inputs, and F as output:

Figure 3.4: An example homework assignment problem participants were asked to grade. For each homework problem, participants were asked to copy the given diagram using the specified interface, and compare the Boolean equation given by the interface with the equation provided in the homework problem.

While copying a diagram, participants were free to input the diagrams components (wires, gates, labels) in any order they pleased. Once the participant finished inputting the diagram, and was satisfied that they had done so correctly, the researcher used the application to obtain the diagrams equivalent Boolean equation. This was done in a Wizard of Oz manner, where we did not perform recognition on the diagrams, but programmed the interfaces to provide the appropriate equation when prompted. Once the participant received the equation from the interface, they compared it with the equation given in the homework problem, and marked on the homework sheet if the diagram was correct.

We chose to use this task as it gave participants a clear motivation for why they were copying diagrams (grading a homework assignment), motivation to make sure that they had copied them correctly (not wanting to incorrectly mark a problem as incorrect), and motivation for why they would care about effort and completion time (grading many assignments takes time).

We designed eighteen pairs of Boolean equations. Each pair of equations was associated with a logic diagram, so we designed eighteen logic diagrams. For each pair of equations, one equation was equivalent to the associated diagram, and one was inequivalent. Each homework problem presented one of the two equations and the associated diagram. This simulated the student getting the homework correct (when the equation and diagram were equivalent) and incorrect (when equation and diagram were inequivalent). The diagrams were divided into two sets based upon the diagram complexity, low and high (see Figure 3.5a for a low complexity diagram and Figure 3.5b for a high complexity diagram). We decided upon these complexity level divisions by using diagrams similar to those found in introductory problems in textbooks [47] as our low complexity diagrams, and building from there for high complexity diagrams.

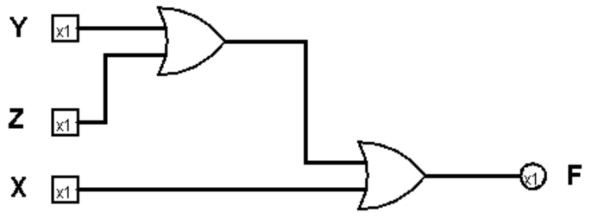
All low complexity diagrams were designed to have three input gates. The nine low complexity diagrams were further subdivided into three subsets, where all diagrams in a subset had the same number of gates and wires. One subset had 6 gates and 5 wires; the second subset had 8 gates and 8 wires, and the third subset had 10 gates and 11 wires. The nine high complexity diagrams had four input gates and were also divided into three subsets. The first subset had diagrams with 12 gates and 15 wires; the second subset diagrams had 15 gates with 18 wires, and the third subset had diagrams with 17 gates and 22 wires. The diagrams made use of input, output, AND, OR, XOR, and NOT gates. Diagrams on the homework assignment were presented to participants in a typeset form in order to provide a clear visualization. We randomly divided the diagrams from each complexity set such that each interface had three low complexity diagrams (one from each subset) and three high complexity diagrams (one from each subset). Each homework assignment was randomly assigned to have one to three (at most half) of the equation-diagram pairs inequivalent.

Section 3.3.3 Experimental Design and Procedure

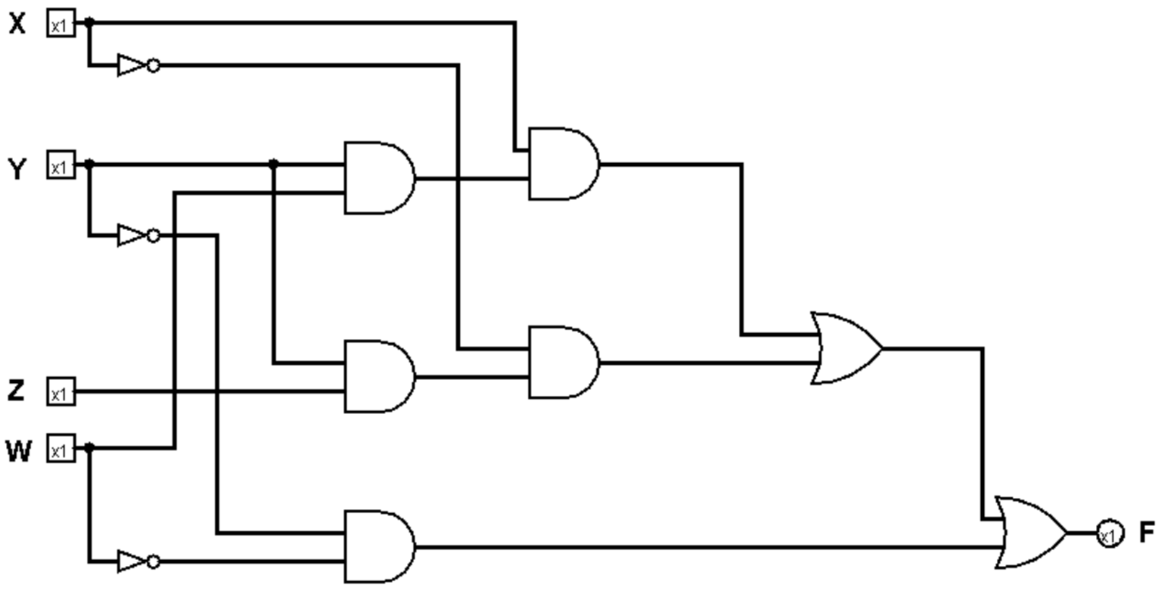
We used a 3 by 2 within-subjects factorial design where the independent variables were user interface and diagram complexity, and the dependent variable was completion time. The completion time is defined as the time interval from when the researcher pressed a start button until participants finished inputting the logic diagram and pressed a stop button.

The experiment began with participants completing a pre-questionnaire to collect demographic information, as well as their experience using a Tablet PC and their capability to solve Boolean equations.

1. How familiar are you with tablet PCs?



(a) Low Complexity



(b) High Complexity

Figure 3.5: Example logic diagrams used in our study. (a) A low complexity diagram with 6 gates and 5 wires. (b) A high complexity diagram with 15 gates and 18 wires.

2. How familiar are you with Boolean algebra? I can solve any type of Boolean algebra problems, such as Boolean minimization problems.

These last two questions were assessed using 7-point Likert scales. For the question about experience using Tablet PCs, 1 equaled unfamiliar and 7 equaled familiar. For the question about capability to solve Boolean equations, 1 equaled none and 7 equaled any. The median for both questions was approximately 5 out of 7. We asked participants to assess their own Boolean algebra capabilities because we were interested in their own perceptions of their knowledge. One limitation of our approach is that participants may have over- or underestimated their Boolean algebra abilities. For each interface, participants were shown a short video explaining how to use the interface. They were then asked to familiarize themselves with using a Tablet PC and stylus by inputting the same sample logic diagram using the three user interfaces. During the practice session, participants were encouraged to ask the researcher questions about using the interfaces. We also informed participants that their sketches would be recognized by the sketch interface without mistakes. We did so in order to build participant confidence that the sketch interface would work. This is a limitation of our work exploring proof of concepts, as this may have influenced participants to be looser in their sketches and draw faster. The researcher also monitored participants remotely to remind them about each interfaces features.

Once they had familiarized themselves with the three interfaces, each participant was given a series of tasks to perform. When ready to copy the diagram, they informed the researcher so that timing could begin. During diagram input, the researcher monitored the participants in order to make sure that they correctly copied the diagram. When they finished entering the logic diagram, the participant informed the researcher who stopped timing, and clicked the recognition button on each interface. Only after the researcher invokes recognition does the system display any feedback, which is in the form of a Boolean equation. The participant then compared the system-generated equation to the equation on the homework assignment and marked the correctness of the homework problem.

The order in which participants worked through the different interfaces was randomized and balanced such that one-third of the participants worked with each interface first, second, and last. The order of diagram complexity was also randomized and balanced across interfaces. After completing the task for an interface, participants were asked to fill out a questionnaire where they rated their agreement with seven statements on a 7-point Likert scale (for consistency, we based our statements on [37]).

1. I found it easy to use this interface to make the logic gate I want to use.
2. I found it easy to use this interface to make the wire I want.
3. I found it easy to use this interface to make the label I want.
4. I found it easy to use this interface to spatially arrange the logic gates.
5. I found this interface easy to use for creating logic gate diagrams.
6. It was quick to create the logic gate diagram.

7. It was frustrating to use this interface.

After completing all three tasks, a final interview was conducted to allow participants to summarize their perceptions and feelings on the three interfaces. Participants were also asked to rank the three user interfaces based on ease of use, speed of entry, naturalness, and overall preference and gives reasons for each ranking. Participants were verbally informed that naturalness was defined as similarity to pen-and-paper.

1. Please rank the user interfaces in order from most easy to use (1) to least easy to use (3).
2. Please rank the user interfaces in order from fastest (1) to slowest (3) on speed of entry.
3. Please rank the user interfaces in order of most (1) to least natural (3).
4. Please rank the user interfaces in order from most (1) to least (3) on overall preference.

Section 3.3.4 Results

Section 3.3.4.1 Primary Hypothesis

Did participants prefer the sketch interface over the hybrid and drag-and-drop interfaces? To analyze the overall rankings (and the three other rankings) of the interfaces, we performed a Chi-Square test on each ranking, which tells us whether there is variance in how the interfaces were ranked. In other words, these tests tell us whether participants thought the interfaces were equally easy to use, speedy, etc. (see Figures 3.9 - 3.6)). In overall ranking, we had a close race between the sketch and hybrid interfaces. The hybrid interface had nine first place rankings, while the sketch interface had seven. In second place rankings, both had eight. The drag-and-drop interface was preferred the least with fourteen third place rankings. We saw significance in the Chi-Square test on overall preference ($\chi^2_4 = 24.667$, $p < 0.001$). Our central hypothesis, that the sketch interface would be preferred was not confirmed by these rankings, as there was no clear winner in overall ranking.

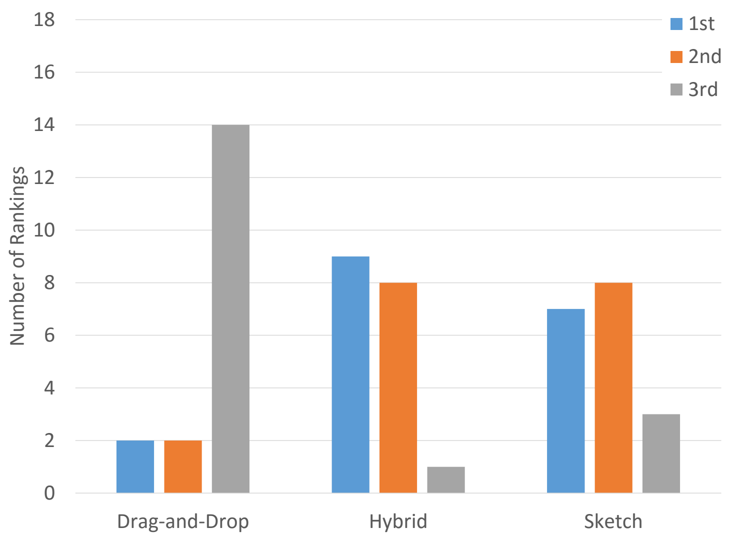


Figure 3.6: Participant rankings of the interfaces on overall preference. A ranking of first indicates greatest agreement with that criteria.

Section 3.3.4.2 Secondary Hypothesis: Naturalness

We also saw significance in the Chi-Square test on naturalness ($\chi^2_4 = 52.667$, $p < 0.001$). Fourteen out of eighteen participants ranked the sketch interface as the most natural, as it is similar to pen and paper. This confirms our second hypothesis, that participants would find the sketch interface most natural. Fourteen participants ranked the hybrid interface second in naturalness, and fifteen participants ranked the drag-and-drop interface third.

Section 3.3.4.3 Secondary Hypothesis: Speed

In terms of speed of entry, we saw significance in the Chi-Square test ($\chi^2_4 = 24.000$, $p < 0.001$). Ten participants ranked the sketch interface first and seven ranked the hybrid interface first. Ten participants ranked the hybrid interface second, and thirteen ranked the drag-and-drop interface third fastest. The participants perceptions of speed ran contrary to our first hypothesis that participants would complete the diagrams faster with the hybrid interface than the sketch and drag-and-drop interfaces. While participants **thought** the sketch interface was fastest, was it?

We analyzed the timing data (see Table 3.1) using a Repeated Measures ANOVA and used the average completion time for the three diagrams per complexity level. The ANOVA showed that interface ($F_{1.350,22.946} = 28.580$, $p < 0.001$, $\eta^2_p = 0.627$), complexity ($F_{1,17} = 353.499$, $p < 0.001$, $\eta^2_p = 0.954$), and their interaction ($F_{2,34} = 12.576$, $p < 0.001$, $\eta^2_p = 0.425$) were all significant factors. In order to see which interface was quicker, we performed paired t-tests where we looked at the times across all accuracy levels (see Table 3.2). To correct for Type I errors, we applied Holms Sequential Bonferroni Correction [27]. Contrary to the participant rankings, the hybrid interface took less time than the sketch interface and the drag-and-drop interface, which matched our first hypothesis.

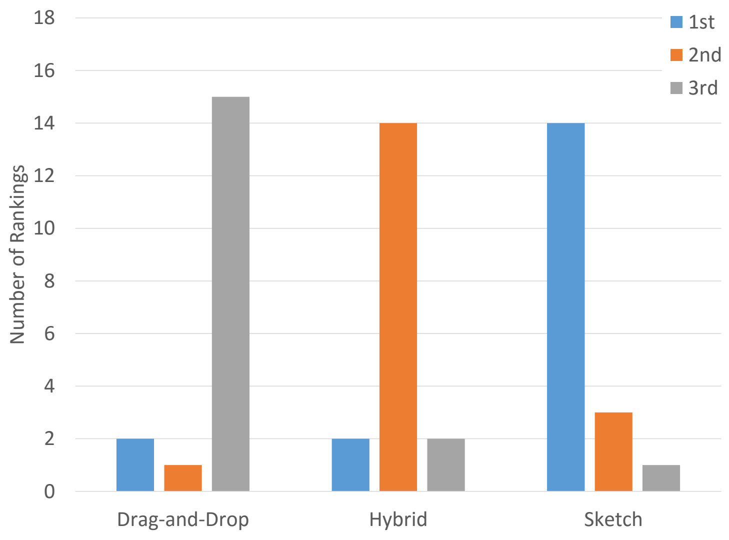


Figure 3.7: Participant rankings of the interfaces on naturalness. A ranking of first indicates greatest agreement with that criteria.

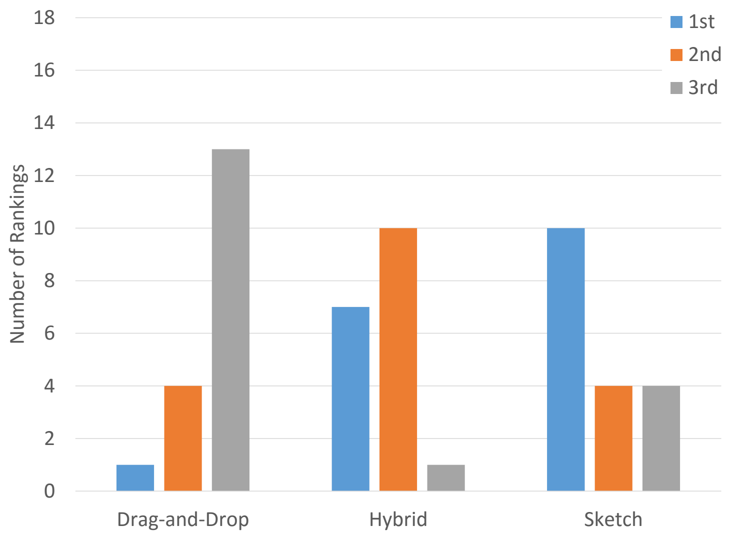


Figure 3.8: Participant rankings of the interfaces on speed of entry. A ranking of first indicates greatest agreement with that criteria.

Table 3.1: Mean completion time in seconds

Interface	Drag-and-Drop		Hybrid		Sketch	
Complexity	Mean	σ	Mean	σ	Mean	σ
Low	92.9	19.5	70.8	12.1	70.0	14.8
High	226.4	38.7	186.6	33.3	202.4	34.6
Overall	159.7	74.1	128.7	63.7	136.2	72.1

Table 3.2: T-tests showing whether each interface pair was significantly different (DD = Drag-and-Drop, S = Sketch, H = Hybrid). A negative t-value indicates the first listed interface is faster than the second.

Interface	DD-H	DD-S	H-S
Low	$t_{17} = 5.478$ $p < 0.017$	$t_{17} = 4.745$ $p < 0.025$	$t_{17} = 0.310$ $p = ns$
High	$t_{17} = 6.542$ $p < 0.017$	$t_{17} = 3.913$ $p < 0.05$	$t_{17} = 4.721$ $p < 0.025$
Overall	$t_{35} = 7.938$ $p < 0.017$	$t_{35} = 6.094$ $p < 0.025$	$t_{35} = 2.922$ $p < 0.05$

Section 3.3.4.4 Other Results

With the Chi-Square test on the ease of use rankings, we saw significance ($\chi^2_4 = 17.667$, $p < 0.005$). Many participants ranked the sketch interface as the easiest to use (10 participants), with the hybrid interface second (11 participants), and the drag-and-drop interface third, where 11 participants ranked drag-and-drop last.

After analyzing the Likert-item data using Friedman tests (see Table 3.3 for mean responses and Table 3.4 for the Friedman results), we found significance for three statements: 3, 4, and 5. All three statements related to ease of use: “I found it easy to use this interface to make the label I want”, “I found it easy to use this interface to spatially arrange the logic gates,” and “I found this interface easy to use for creating logic gate diagrams.” We then performed Wilcoxon Signed Rank Tests for each statement and corrected using Holms Sequential Bonferroni Correction (see Table 3.5). For making labels, the sketch interface was easiest to use, followed by the hybrid interface, and finally the drag-and-drop interface. Participants found the sketch interface harder to arrange gates in than in the drag-and-drop interface. This is easy to understand, as the only way to perform spatial arrangement in the sketch interface is to erase and redraw. For the statement on ease of use in making diagrams, we saw a single significant result in the Wilcoxon Signed Rank test. The hybrid interface was easier to create diagrams with compared to the drag-and-drop interface. This matches with the participant rankings of the interfaces on ease of use, where drag-and-drop was ranked last 11 times. In contrast, the hybrid interface was primarily ranked second, though it was rated highly.

We hypothesized that participants would prefer the sketch interface over the other interfaces, and speculated that they would do so because of the sketch interfaces naturalness and inspite of the hybrid interfaces speed. Our analysis of the rankings shows that the sketch interface was indeed natural and that the hybrid interface was fast, even if participants did not notice that speed. In order to provide some insight into participant reasonings on why they preferred one interface over another, we calculated Spearmans rank correlations between overall ranking and the other measurements we took (completion time, the seven ratings, and the other three rankings) (see Table 3.6 for the correlation coefficients). The correlation with the greatest coefficient was between overall ranking and the ease of use ranking. Overall ranking with naturalness was next, followed by speed. This implies that ease of use was actually more important than naturalness or speed. However, it does not tell us how much ease of use was impacted by naturalness or speed.

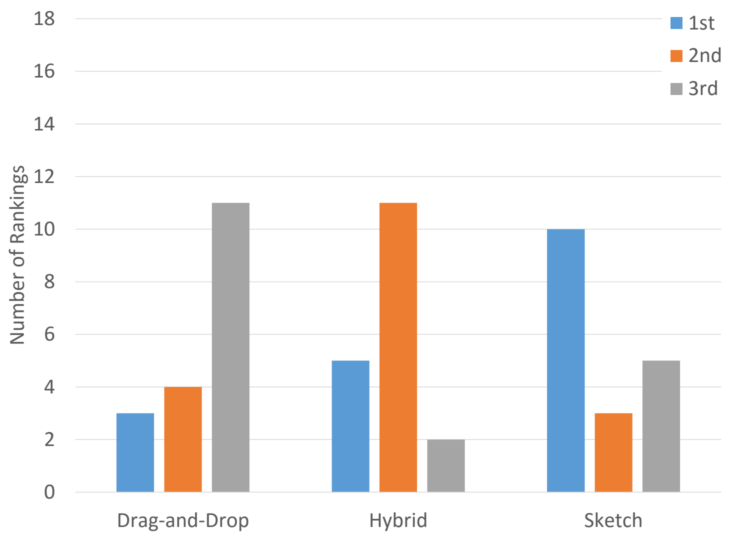


Figure 3.9: Participant rankings of the interfaces on ease of use. A ranking of first indicates greatest agreement with that criteria.

Table 3.3: Mean responses for Likert statements. Bold results indicate statements that had significant differences in responses.

Interface	Drag-and-Drop		Hybrid		Sketch	
Statement	Mean	σ	Mean	σ	Mean	σ
Easy Gate	5.72	1.447	6.17	0.924	6.39	0.698
Easy Wire	5.00	1.680	5.89	1.231	5.94	1.305
Easy Label	4.67	1.815	6.89	0.323	6.33	1.085
Easy Arrange	6.22	0.878	4.56	1.854	5.78	1.263
Easy Diagram	5.39	1.243	5.67	1.138	6.28	0.752
Quick	5.61	1.335	5.83	1.249	6.11	1.023
Frustrating	3.06	1.731	2.50	1.249	2.11	1.023

Table 3.4: Results from the Friedman test on each Likert statement. Three statements had significance (shown in bold), Easy Label, Easy Arrange, and Easy Diagram.

Statement	Chi-Square	p
Easy Gate	$\chi^2_2 = 1.9224$	$p = ns$
Easy Wire	$\chi^2_2 = 3.250$	$p = ns$
Easy Label	$\chi^2_2 = 21.382$	$p < 0.001$
Easy Arrange	$\chi^2_2 = 9.480$	$p < 0.01$
Easy Diagram	$\chi^2_2 = 7.000$	$p < 0.05$
Quick	$\chi^2_2 = 1.161$	$p = ns$
Frustrating	$\chi^2_2 = 1.846$	$p = ns$

Table 3.5: Wilcoxon Signed Rank Tests showing whether each interface pair was significantly different (DD = Drag-and-Drop, S = Sketch, H = Hybrid) on the indicated statement. A negative Z-value indicates the first listed interface was rated higher than the second.

Interface	DD-H	DD-S	H-S
Easy Label	$Z = 2.610$	$Z = 3.542$	$Z = 2.264$
	$p < 0.025$	$p < 0.017$	$p < 0.05$
	$r = 0.435$	$r = 0.590$	$r = 0.377$
Easy Arrange	$Z = 1.867$	$Z = 2.925$	$Z = 2.057$
	$p = ns$	$p < 0.017$	$p = ns$
	$r = 0.311$	$r = 0.488$	$r = 0.343$
Easy Diagram	$Z = 2.676$	$Z = 0.819$	$Z = 1.895$
	$p < 0.017$	$p = ns$	$p = ns$
	$r = 0.446$	$r = 0.137$	$r = 0.316$

Table 3.6: Correlations between overall ranking and other rankings, ratings, and completion time.

Correlation with Overall Ranking	ρ	p
Ease of Use	0.778	0.000
Naturalness	0.694	0.000
Speed	0.639	0.000
Frustration	0.411	0.002
Easy Diagram	-0.409	0.002
Easy Label	-0.397	0.003
Completion Time	0.397	0.003
Quick	-0.342	0.011
Easy Gate	-0.314	0.021
Easy Wire	-0.281	0.039
Easy Arrange	0.056	0.690

Section 3.4 Discussion

Section 3.4.1 Naturalness

Our second hypothesis that participants would find the sketch interface most natural was supported by the participant rankings on naturalness, where fourteen participants found the sketch interface most natural. As one participant wrote for why they selected the sketch interface as most natural, "sketch allows you to write as if you are drawing on paper." In contrast to the similarity of the sketch interface to pen and paper, one participant, who selected hybrid as the most natural interface, commented that "after getting used to hybrid, it was more natural to select [a] gate at [the stylus's] location, and connect inputs to gates. Sketch could arguably be more natural but took more time to make [sure] wires were correct." Another participant who also ranked the hybrid interface as most natural commented that "naturalness should be redefined in pen-based user interface for drawing logic diagram as the combination of gesture-based drag-and-drop visual controls and sketching of wires and labels." Two users who thought the drag-and-drop interface was most natural, did not like to draw too much, and felt that the drag-and-drop interface was best because it resembled their previous experience using desktop environments.

Section 3.4.2 Speed and User Perceptions

Participants worked fastest with sketch and hybrid for low complexity diagrams, and with hybrid for high complexity diagrams and overall. This matches our first hypothesis that the hybrid interface would allow participants to create diagrams most rapidly. Examining the Likert statement on quickness, the Friedman test did not show significance, meaning that we cannot reject the possibility that participants did not find a difference in speed between the interfaces (see Table 3.4). The mean response for the hybrid interface was high, so participants thought it was quick, but the other interfaces were rated fairly high as well. Quite interestingly, participants ranked the sketch interface fastest over half the time (10 participants) and the drag-and-drop interface fastest only once. We can see a disconnect between user perception of speed and actual speed. We think this can be explained by several factors.

First, when using the sketch interface, participants did not have to switch tasks; that is they only drew. In contrast, the drag-and-drop and hybrid interfaces required participants to move to or open a menu, drag and drop gates, arrange gates, and draw wires and labels. Participants likely felt that they were doing more, and doing more made them feel like they were taking more time.

Second is the issue of error blame; when sketching, we think there was a lessened perception of computer interaction (except for the erasing gesture), so participants felt as though nothing was impeding them from completing the task. Mistakes would be attributed to themselves, since they didn't feel as though they were using a computer. When using the drag-and-drop and hybrid interfaces, more complex interactions are required, and participants likely had a sense of using a computer application. When a mistake was made, participants could shift blame to the computer application, and these "external" mistakes would cause participants to feel as though they were working slower.

Finally, we think that participants may have slowed themselves down with the sketch interface by making dense drawings. This is especially true for participants who did not draw well. They had to recheck their drawings repeatedly because of their unclear drawings. When a mistake was made, a scribble-erase gesture might unintentionally overlap with several strokes, causing the participant to have to redraw various items. This disconnect in perceived speed and measured speed is also interesting because we provided a sketch interface that was always correct in its ability to recognize and understand the logic diagram. Even with 100% recognition accuracy, the sketch interface was slower than the hybrid interface for many situations.

Section 3.4.3 Ease Of Use

Section 3.4.3.1 Making Diagrams

For the statement on ease of use in making gates, the analysis did not show significance between the interfaces, which is interesting, since all three interfaces had quite different methods for creating gates (see Table 3.4). The drag-and-drop interface had a traditional method for creating gates that can be found in diagram-creation applications like Visio. A small number of participants had a few issues with dragging the gates from the toolbar, "dropping" the gate while it was still on the toolbar. Perhaps this was a result of using a stylus. Both the sketch and hybrid interfaces had a mean response of over 6, indicating that the interfaces were quite easy to create gates with. This leads us to pose the question of whether and how they can be improved upon.

Analysis of the statement on ease of use in making wires did not show significance either. Both the drag-and-drop and hybrid interfaces used a UI visualization for the wires and required participants to start and end their wires at a specific location (the gate terminals). The drag-and-drop interface required the participant to be more accurate in starting the wire at the terminal compared to the hybrid interface. The UI visualization also constrained the shape of wires to simple curves and straight lines. In contrast to the drag-and-drop and hybrid interfaces, the sketch interface allowed the participants to begin their wires anywhere and to make them look any way they wished. Finally, the hybrid interface provided a visualization (a yellow highlight) when a wire was being created and the stylus was sufficiently close to a terminal, which allowed participants to know when they had successfully created a wire. We thought the user interface differences made to the hybrid interface would cause it to be easier to use, particularly over the drag-and-drop interface. However, we cannot say whether the sketch interfaces freeform wire drawing compares favorably with the hybrid's easy wire creation.

As previously discussed, we did see significant differences in how easy it was for participants to make labels in the three interfaces. While the sketch and hybrid interfaces both allowed the participants to write their labels with the stylus, there were minor user interface differences between the two. Using the hybrid interface, participants had to write their labels a bit away from input and output gates in order to not move the gates. This accidental movement likely led to the difference in perceived ease of use between the sketch and hybrid interfaces. In contrast to those interfaces, the drag-and-drop interface required the participant to switch input modes to use the keyboard for typing the labels. Having to make a modal switch slowed down users (as did having to use the barrel button on the stylus to select the rename function from a context menu) and made it harder to use.

Section 3.4.3.2 Arranging Diagrams

While the hybrid interface provides similar mechanisms for spatial rearrangement to the drag-and-drop interface, it has a flaw in erasing gates. When scribble-erasing, if the participant starts too close to a gate, the gate will be moved instead of triggering a gesture. While participants practiced the scribble-erase gesture during the training session, they still made more mistakes in deleting gates and wires in the hybrid interface compared to the drag-and-drop interface. This might be improved through the use of mode inferencing techniques such as those described in [49].

Some participants also expressed that they preferred the sketch interface because of their own drawing abilities. In contrast, some participants drew cluttered diagrams and worried about their drawings. These participants might have benefited from recognition feedback to show them that their drawings were neat and formal enough. One participant commented that he does not like to draw, so he did not like using the sketch interface as much as the others.

Making neatly aligned diagrams can be beneficial, as it can make it easier to copy the diagram. Therefore, some participants moved the gates in order to align them. Another participant commented out that he wanted to spatially move and align the gates for every interface, and the sketch interface did not provide a way for participants to move their strokes without erasing them. Thus, aligning gates was harder than with the other interfaces, and he did not like the sketch interface.

Finally, for the hybrid interface, some participants triggered the radial menu without intending to do so, holding the stylus to the screen without moving while thinking about their next actions. Sometimes participants even generated gates when this happened. Though they had practiced bringing up the pie menu during the practice session, a few participants still had this issue. This issue should be alleviated with more practice.

Section 3.4.3.3 Frustration

Our analysis did not show significance between the interfaces for the statement about frustration (see Table 3.4). The mean responses for frustration were low for all three interfaces, indicating that there was little that was frustrating about using the interfaces, as agreement with the statement indicates higher levels of frustration. Hybrid had the lowest mean frustration rating (though, again, there were not statistically significant differences between the interfaces in our test). We think this is because it allowed participants to label gates in a natural way, spatially arrange the gates, and create gates with little work.

Section 3.4.4 Overall Ranking

Looking at the rankings, we can see that there was no overall "winner," which is contrary to our hypothesis. The hybrid interface was ranked first nine times and second eight times, and the sketch interface was ranked first seven times and second eight times. The drag-and-drop interface had fourteen third place rankings. While the hybrid interface was fastest in our timings, as previously discussed, participants felt otherwise. Instead, they found it to have speed similar to the sketch interface. The sketch interface was strongly perceived to be most natural, but the sketch interface was not significantly chosen as best overall. It is, however, clear that drag-and-drop was ranked last. Participants clearly took a number of factors into consideration when deciding on an overall winner. Participants thought the sketch interface was easy to use, quick, and natural, but did not overwhelmingly rank it best overall.

Is the naturalness of a sketch interface more important than the other factors we examined? It does not appear to be more important to several of the participants, as seven ranked the sketch interface as first in naturalness, but chose another interface as first in overall preference (all chose hybrid). Our correlations also show that ease of use was more closely correlated with overall preference than naturalness or speed, but because the sketch and hybrid interfaces did not have equivalent ease of use, we cannot say for certain.

The perceptual gaps on speed (and to some degree, ease of use) make things quite interesting. Participants who ranked sketch first overall (seven participants) also ranked sketch first for all three other criteria, except for one participant who did not rank sketch first for speed. For the drag-and-drop interface, two people ranked it first overall. One participant ranked it first for the other three criteria, and the other participant ranked it first on ease and naturalness. The hybrid interface exhibits the most interesting ranking behavior. Nine participants ranked the hybrid interface as best overall, of those, only one participant ranked hybrid first in all three other categories. Two participants who ranked hybrid first overall did not rank hybrid as first in any other category. We think this further indicates a strong gap in user perception on the various criteria.

Participants had a hard time deciding how to rank the interfaces according to participant comments. Some participants felt that the differences in overall experience between sketch and hybrid was so tiny that they could not distinguish easily between the two. Therefore, they tried to rank based on their previous rankings on other criteria. One participant who ranked hybrid as the best overall commented that “sketch was easy to use, but drawing everything out took much time, lack of ways to fix mistakes.” Another who rated hybrid as the best overall commented that “hybrid combined the way the diagram looked, how to input it, correct errors, easier than the others. Sketch was nice because it was fast for small diagrams.”

Section 3.4.5 Sketch Recognition Accuracy

Finally, after asking the participants for their rankings, we explained to each participant our motive for the experiment, and asked whether they would change their rankings if the sketch interface did not have 100% accuracy. Except for three people who rated sketch as the worst overall interface even with 100% accuracy, all other participants expressed that decreased accuracy would cause them to change their rankings. Some participants wanted to change all four rankings and place sketch as the lowest interface. Other participants wanted to change their rankings only for ease of use and overall preference, and rank sketch as the lowest, but make no change on naturalness and speed of entry. Clearly, participants thought that 100% sketch recognition accuracy was important, though it remains to be seen how dramatically it affects user perceptions. In order to provide a natural sketch interface, we need to make further improvements to recognition accuracy, as a perfectly accurate pure sketch interface was not overwhelmingly better than the hybrid interface.

As it turns out, this type of difference between user perception and measurement would be found in our follow-up works. After working with a simple Wizard of Oz system for this study, we decided to develop a system that would allow us to realistically simulate imperfect recognition accuracy. Even though sketching performed well, we presented a system that didn't make recognition mistakes. To better understanding sketch-based interfaces, we felt that we needed to include accuracy, but control it so that we can see exactly how it affects user perceptions and acceptance of sketch-based interfaces. We set out to design a Wizard of Oz system, as we felt it was the easiest way to control accuracy with realistic results.

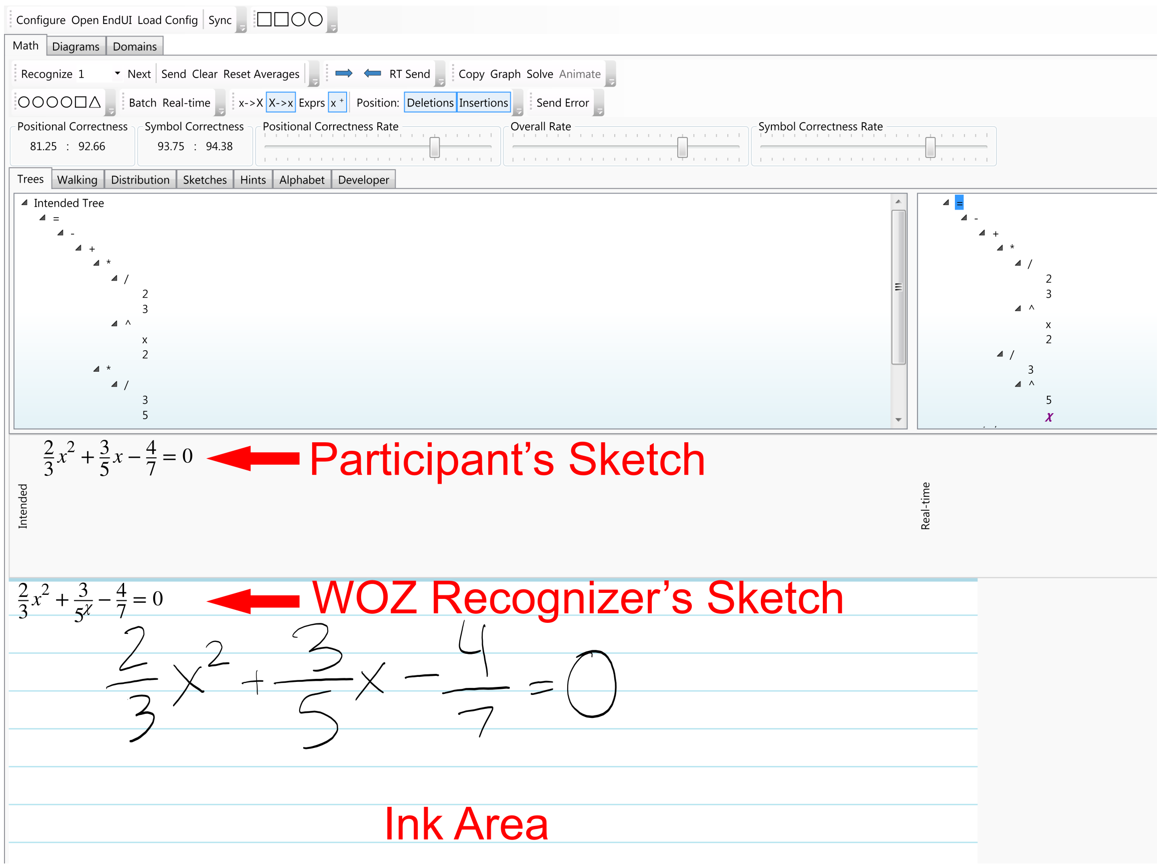
While we worked with diagrams for this study, we pivoted and changed to a similar domain for our future work, mathematics. We chose mathematics for the initial development of the WOZ Recognizer because mathematics is structured and typically requires less writing for a meaningful expression than a diagram. With mathematics, the positional relationships hold lots of meaning in comparison to many diagram domains. This constrained, rigid spacing led us to work with mathematics, as we believe that its complexity makes its results applicable to similar domains like diagrams.

CHAPTER 4: WOZ RECOGNIZER

Section 4.1 Introduction

In our work with pen-based interfaces for creating logic diagrams, we saw that the sketch-based interface provided good ease of use, speed, and naturalness. However, we simulated perfect recognition, which is atypical in real-world usage. Recognition accuracy is an important aspect of using sketch-based recognition. To explore how recognition parameters affect user preferences and experience, we need a recognizer where these parameters can be controlled. Unfortunately, recognizers are not designed in this way; we cannot control recognition accuracy for instance. To this end, we designed and implemented a Wizard of Oz sketch recognition system, the WOZ Recognizer, which generates realistic recognized sketches without performing any recognition on ink strokes. A Wizard of Oz system [12] is one where a human performs some of the actions that a computer would normally perform, such as recognition of speech or handwriting. We used this approach to build a system where a human (wizard) controls a computer system to realistically simulate a sketch recognition system. In the WOZ Recognizer, a set of pre-determined sketches are modeled and input into the system; a handwritten mathematical expression or diagram is then transformed into a recognized sketch according to an established error distribution and accuracy targets. No recognition takes place in the WOZ Recognizer. The wizard supplants an accuracy-controlled recognizer, telling the system what sketch is being simulated, choosing good recognized sketches, discarding bad ones, and working to meet an accuracy target.

Creating a Wizard of Oz system for realistic and accuracy-controlled sketch recognition is a challenging problem. The system needs to realistically simulate recognition, being able to produce results similar to real-world recognizers. Each recognition domain (mathematics, diagrams, etc.) has its own set of rules for constructing a valid sketch. These rules need to be encoded in some way into the Wizard of Oz recognition system; even though recognizers do not always produce valid output, the rules still affect the recognition errors. Recognition feedback needs to be provided to the wizard in a timely manner, and again to the user with minimal latency. Many recognition systems provide streaming recognition where feedback is provided as the user writes; simulating this type of feedback has strong latency constraints. Determine sketch recognition accuracy is a problem in itself; how does the system measure accuracy so that it can reach accuracy goals? Finally, the wizard in the system needs to receive a large amount of information and process it in a timely manner (for instance, the user's ink, the system accuracy, and the recognition output).


Configure Open EndUI Load Config Sync
Math Diagrams Domains
Recognize 1 Next Send Clear Reset Averages RT Send Copy Graph Solve Animate
Batch Real-time x->X X->x Exprs Position: Deletions Insertions Send Error
Positional Correctness 81.25 : 92.66
Symbol Correctness 93.75 : 94.38
Positional Correctness Rate
Overall Rate
Symbol Correctness Rate
Trees Walking Distribution Sketches Hints Alphabet Developer
Intended Tree
Intended
Real-time
Participant's Sketch
WOZ Recognizer's Sketch
Ink Area

$\frac{2}{3}x^2 + \frac{3}{5}x - \frac{4}{7} = 0$

$\frac{2}{3}x^2 + \frac{3}{5^x} - \frac{4}{7} = 0$

Figure 4.1: The Wizard Interface with mathematics-based sketches. A typeset version of the WOZ Recognizer's sketch is displayed below typeset version of the participant's sketch, and the user's ink is displayed in the lined section at the bottom.

The WOZ Recognizer takes a pre-defined computer representation of a sketch and simulates recognition by applying recognition errors to the representation according to error distributions. The output is shown to a wizard, who decides whether the accuracy and realism are sufficient (i.e. whether the output matches the user's writing and the errors are representative of real recognizers). The wizard also determines when to have the WOZ Recognizer display the output to the user. We are interested in sketches that have structure and complexity; therefore we started with sketches that can be modeled with graphs [24], specifically mathematics-based expressions and diagrams. While mathematics and diagrams are not especially similar, they both have an underlying structure that is two-dimensional in nature.

The WOZ Recognizer has a variety of features to make simulation of sketch recognition easy and realistic and its main features include:

1. Mathematics-based and diagram-based sketches, such as algebra, finite state machines, and flowcharts.
2. Support for a variety of diagram-based sketching domains.
3. A user interface for creating new domains.
4. Simulates batch (recognition of the whole sketch at once) and streaming recognition (recognition in pieces as it is written).
5. Supports writing multiple mathematics expressions simultaneously.
6. Gives the wizard control over recognition accuracy.
7. Allows the wizard to change many aspects of recognition errors, including the types of errors used, what symbols can have recognition errors and what those recognition errors are, and the relative prevalence of the different categories of errors.

In this chapter, we discuss how we model sketches and sketch recognition, the types of sketch recognition the WOZ Recognizer can simulate, how the WOZ Recognizer controls accuracy while producing realistic recognized sketches, how the wizard controls the WOZ Recognizer, how users interact with it, a case study examining how expert users used the WOZ Recognizer, and finally a discussion of the case study.

Section 4.2 Preliminaries

It is important that we have a clear understanding of the terminology and concepts used throughout the remaining sections of this dissertation.

Section 4.2.1 Sketch Terminology and Background

A *symbol* is a collection of ink strokes, a visual glyph that is used to represent different things other than itself, like characters, numbers, and other geometric entities (e.g. a finite state machine accept symbol, see Figure 4.2) that have meaning. A sketch is made up of a collection of symbols that relate to each other in some way. The sketch the end user is drawing is known as the *participant's sketch*. After simulation of recognition, the user is shown the *WOZ Recognizer's sketch*, the participant's sketch with recognition errors inserted into it.

For a given type of sketch (a *sketch domain*), the *grammar* sets a set of structural rules for combining symbols into valid strings, while the *alphabet* defines the symbols that can be combined using the structural rules. This notion of grammars and alphabets is fairly common in sketch recognition and is used frequently in handwriting and sketch recognition domains [3, 16, 26, 57]. Alphabets and grammars are used by the WOZ Recognizer to produce sketches. As an example, mathematics-based sketch domains, like stoichiometric equations, have a defined alphabet of symbols that need to be recognized and parsed into meaningful structures. For diagrams, a symbol alphabet is composed of node and edge symbols, which are typically shapes, and label symbols which are characters. These two pieces, the alphabet and the grammar, are key to error simulation in the WOZ Recognizer.

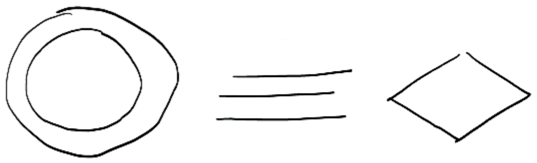


Figure 4.2: Examples of symbols composed of geometric shapes. A finite state machine's Accept State symbol, a bond diagram's Triple Bond symbol, and a flowchart's Decision symbol.

For error simulation we need to define an *error alphabet* and an *error grammar*. To create an error alphabet, for each symbol, we define a set of symbols for which a recognizer might mistakenly recognize the correct symbol (e.g. *a* for *9*, *l* for *1*, or *v* for *u*). We also define a probability that each "substitute" symbol might replace the correct symbol. Finally, we define a probability for each symbol that it might not be recognized, because we saw such errors in real mathematics recognizers. We also take the same concept of incorrect recognition substitutions and use it to create an error grammar. For example, in mathematics, a symbol that should be a superscript might reasonably be recognized as being in a regular position, or a number might not be included under a root symbol. With some types of diagrams, moving a symbol higher in the diagram can change the meaning, such as in a family tree, where ancestors appear above descendants. Thus, for each rule in the grammar, we have a set of substitution rules that might be applied, and, for each rule, the probability that it will be applied. These collections of substitutions are important, because they allow us to realistically simulate handwriting recognition for sketches.

Section 4.2.2 Recognition Error Types

For a given sketch, a *correct symbol* is the symbol found at a specific node in the model of the participant's sketch. The *recognized symbol* is the symbol in the analogous node in the model of the WOZ Recognizer's sketch. The recognized symbol can be correct or incorrect, depending on whether an error was introduced. We define four categories of symbol errors: segmentation errors, similarity errors, wild errors, and no-recognition errors (see Figure 4.3). Segmentation errors are those for which the recognizer has incorrectly segmented a set of strokes. We then have two subcategories, over-segmentation and under-segmentation. In over-segmentation, there are more recognized symbols than there are correct symbols. Under-segmentation is the converse, with less recognized symbols than correct symbols. With similarity errors and wild errors, segmentation is correct, but there is still an error in recognition. Similarity errors have a correct symbol and a substitute symbol that are visually similar or written similarly. Case errors, where an upper-case and lower-case symbol have similarity, are a common type of similarity error. With wild errors, there does not appear to be a relationship between the correct and substitute symbol. Finally, no-recognition errors occur when a symbol is missed by a recognizer and no output is given for the input strokes.

Errors in grammar result in symbols changing position in relation to other symbols. For instance, a label can be associated with the wrong edge, or a symbol can be a superscript when it should not be. While symbol errors are similar across different sketch domains, grammatical errors generally have little similarity across domains. Only when there are similar grammatical rules in two domains would we expect to see similar recognition errors. The WOZ Recognizer supports an extensive set of position errors for mathematics-based sketches; with diagram-based sketches, only label association errors are supported (where a label is associated with the wrong node or edge). The grammar for mathematics covers a wide range of expressions, so we have an extensive error grammar in the WOZ Recognizer. Diagram-based sketches have great variety in their grammars, so a single error grammar does not cover many diagram domains. Thus, we could not include a single error grammar that covered many sketch grammars. Instead, we have a simple grammar for diagrams; the primary rule is that labels can be disassociated from the correct location in the diagram.

Additionally, the relative positions of symbols in mathematics-based sketches are highly important; a small change in position can cause a great change in meaning in the sketch, while rearranging the nodes in a finite state machine diagram makes no change in meaning. Even without extensive grammar errors, we felt that we could support a wide variety of diagram-based sketches since many types of diagrams do not have spatial rules in their grammars.

Mathematics-based sketch grammar error categories include: enclosure errors, script errors, division errors, and wrong position errors (see Figure 4.4). Enclosure errors deal with a symbol that should be enclosed in some structure (like parentheses), but are not; there are several sub-types of enclosure errors that are associated with specific structures, like parentheses or roots. Superscript errors involve a symbol that appears as a superscript when it should not. Not-super- and not-subscript errors are the converse, a symbol should be a super- or subscript, but is not. Two error categories deal with division structures, numerator and denominator errors; each category describes an error where a symbol is not placed in its the appropriate position around a division symbol. In some cases, these division errors can be simple with one or two symbols coming out of the division structure; in other cases, the entire division structure is lost. Another category, wrong positions, is for errors where some symbol is not in the correct position, for example, e^{-x} becoming $-e^{x}$.

Section 4.2.3 Accuracy

In order to control recognition accuracy, we had to first decide what exactly a recognition error was and how "wrong" a sketch was when there were recognition errors; in other words, how to measure accuracy. Measuring the recognition accuracy of a sketch has been done in several ways [4,13,37,50,58]. After considering several accuracy analysis methods, two separate metrics were chosen, position accuracy and symbol accuracy. These metrics tell us how close the WOZ Recognizer's sketch is to the participant's sketch. The most important factor in our accuracy metric choices was that measurements had to be performed by the WOZ Recognizer, as it needed to know a potential WOZ Recognizer's sketch's accuracy in order to meet its accuracy targets. Another important factor in choosing our metrics is that they had been used in other sketch-related research [4,37]. Additionally, some of the other metrics we considered gave accuracies that were confusing and unintuitive to users. It is also important to note that a recognizer's accuracy is not defined by a single sketch. Sketches vary in size, complexity, and structure; having one error in a small sketch produces a different accuracy rate than in a sketch with twice as many components. Thus, the WOZ Recognizer measures the accuracy of the current sketch (*per-sketch accuracy*) and the accuracy across a set of sketches (*completed sketch accuracy*).

Section 4.2.3.1 Position Accuracy

To measure position accuracy, for each correct-recognized symbol pair in the models of the participant's and WOZ Recognizer's sketches, we find all the positions for the correct symbol and all the positions for the recognized symbol and compare these positions. When a position is missing for the recognized symbol, it counts as an error. The WOZ Recognizer also counts extra positions as recognition errors (extra positions typically occur due to a failure by a recognizer to align multiple symbols to the same baseline, resulting in one or more symbols gaining extra position, such as super- or subscript position). The sum of all positions for all of the correct symbols gives the total number of positions in a sketch. The number of correct positions in the model of the WOZ Recognizer's sketch is summed and divided by the total number of positions to give us the position accuracy.

For example, the sketch $2x$ becomes 2^x. In the original sketch, each symbol is in a single position, for two total positions. In the WOZ Recognizer's sketch, the *2* is still in the correct position, but the *x* has an extra position, so the sketch is 50% accurate.

Section 4.2.3.2 Symbol Accuracy

We chose two approaches to measure symbol accuracy because symbols are arranged in different ways between diagrams and mathematics. Diagrams have Latin alphabet symbols for labels, which are generally longer one-dimensional strings of symbols, while mathematics symbols are arranged in two-dimensional strings. We decided to measure symbol accuracy using Levenshtein word distance [42] for diagram labels, since it is a good fit for one-dimensional strings. However, mathematical sketches' two-dimensional layout is not a good fit for Levenshtein word distance. Instead, for mathematics-based sketches, symbol accuracy is measured by the number of correct symbols divided by the total number of symbols. With the shape-based symbols in diagram-based sketches, we devised two accuracy metrics depending upon the type of errors used. With errors in the component shapes, we count the number of changes in shapes (see Figure 4.5); with symbol-only errors, we used the same process as mathematics-based sketches.

- Segmentation Errors
  - Over-segmentation: Substitute $|o$ for p
  - Under-segmentation: Substitute w for vv
- Similarity Errors: Substitute *5* for s
  - Case Errors
    - Lower-for-Upper (LfU): Substitute u for U
    - Upper-for-Lower (UfL): Substitute O for o
- Wild Errors: Substitute f for *7*

Figure 4.3: Examples of the symbol error categories.

$$2x \rightarrow 2_x \tag{4.1}$$
$$3y \rightarrow 3^y \tag{4.2}$$
$$x_i \rightarrow xi \tag{4.3}$$
$$a^b \rightarrow ab \tag{4.4}$$
$$\frac{1}{x^2+x} \rightarrow \frac{1}{x^2}+x \tag{4.5}$$
$$\sqrt{a^2+b^2-1} \rightarrow \sqrt{a^2+b^2}-1 \tag{4.6}$$
$$x^{-a} \rightarrow -x^a \tag{4.7}$$
$$\frac{a+2h}{d} \rightarrow \frac{a+2}{d}h \tag{4.8}$$
$$\frac{a+b}{c} \rightarrow a+b-c \tag{4.9}$$

Figure 4.4: Examples of some types of mathematics errors. (4.1) subscript error, (4.2) superscript error, (4.3) not subscript error, (4.4) not superscript error, two enclosure errors, (4.5) division error, and (4.6) root error, and (4.7) wrong position error. Divisions can also have errors like (4.8) and (4.9).

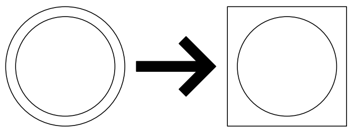


Figure 4.5: With geometric diagram symbols, symbol errors can occur in the defined symbols or in the component shapes that make up the symbol. With shape-based errors, one geometric shape is substituted for another, as in the circle in this accept state is replaced by a square. This type of errors can produce symbols that are undefined for the sketch domain. The symbol accuracy for this example is 50%, as one shape was changed out of two shapes.

Section 4.3 The Recognition Process

Using the WOZ Recognizer to simulate recognition can be divided into four phases: pre-processing, simulation, output generation, and simulating streaming and batch recognition.

Section 4.3.1 Pre-processing

In order to build the WOZ Recognizer, we performed a one-time pre-processing process that examined real recognizers. We examined the output of real mathematics-based and diagram-based recognizers and based the errors created by the WOZ Recognizer on our observations. We observed the errors we saw in two mathematical handwriting recognizers [48, 69], categorized them, and extrapolated to obtain some broader categories. We based our substitute symbols and rules on the data we collected. We collected recognized sketches from three people who each used both recognizers. We felt that we could collect this data from only a small number of people, as only a small sample was needed to see what error types working recognizers made. Each participant wrote a set of 43 equations [37] using each recognizer. The recognized expressions were then categorized and tallied in order to create an error distribution (see Figures 4.7 and 4.8). The total number of symbol and position errors per user was rather close (298 versus 281 versus 264). We based the WOZ Recognizer's default error distribution on this data. We obtained the probabilities of the rules by looking at the data we obtained from the real recognizers. However, these probabilities were tweaked for our preliminary testing, because we wanted some rules to be more or less frequent, since some categories had very high probabilities. For instance, the subscript/superscript rules were highly probable in our previous studies, but they were even more probable in the collected data.

We implemented script positioning errors, enclosure errors, division errors, and general "wrong position" errors for the WOZ Recognizer. The wizard is able to define the probability for these kinds of errors through the Wizard Interface (see Section 4.4.2 for details).

Section 4.3.2 Simulating Recognition

In order to simulate recognition, we need to have a sketch domain (an alphabet, a grammar, an error alphabet, and an error grammar), and set a number of simulation parameters, including recognition mode (batch or streaming) and recognition accuracies. Once these inputs are set, recognition can occur. Recognition occurs through a combination of the wizard's input and the modules shown in Figure 4.6. The data flow between the Participant's Interface and the Wizard Interface is shown in Figure 4.9.

Section 4.3.2.1 Accuracy Regulator

The process of recognition begins with the Accuracy Regulator, which measures sketch accuracy and determines the number of recognition errors needed to meet the target recognition accuracy. Graphs are used to represent sketches; the two-dimensional structure of the sketch is encoded in the graph structure. Each node in the graph represents a symbol in the sketch, such as an *x*, a circle, or a division symbol. We use the traditional approach for mathematics expressions, parse trees, where mathematics operators are internal nodes in the tree and other symbols are leaf-nodes [2]. In other words, parse trees are a tree structure that represents the syntactic structure of a mathematical expression. We use regular graphs for diagrams, where different symbols are stored in the nodes and edges.

After receiving the target accuracies and the model of the participant's sketch, the Accuracy Regulator counts the number of symbols and positions in the sketch. These numbers are multiplied by the target accuracies to obtain the number of symbols and positions that must be correct to meet the accuracy (and the number of recognition errors of each type to create). When multiple sketches are involved, the Accuracy Regulator looks at accuracy rates, the current sketch's numbers, and the completed sketches' numbers to determine the number of symbols and positions that need to be correct to meet the target accuracy rates. The number of errors is then given to another component of the WOZ Recognizer, the Error Creator.

Section 4.3.2.2 Error Creator

The Error Creator is responsible for determining the types of errors to create and inserting the errors by altering the sketch. Once the Accuracy Regulator has given the number of symbol and position errors to the Error Creator, the Error Creator determines what errors to insert and where to insert them. The Error Creator first chooses and applies all position errors before moving on to symbol errors. This is done because position errors might be removed symbols from the sketch. When *n* position errors are needed, an error category is sampled from the error grammar's error distribution with respect to the positions available in the sketch. Should a category not be possible for a sketch, another category is sampled from the distribution until an acceptable category is found. All of the nodes in the sketch that could be affected by this category are collected and randomly ordered. A node is then chosen and the Error Creator tries to apply the error at that node. If some error application rules are broken (as described in Section 4.3.2.2.1), another node is selected until either the category is applied or the possible nodes are exhausted. Should the nodes be exhausted, another error category is sampled from the distribution and the process continues.

While substitution rules (position errors) are only applied in appropriate situations (e.g. a symbol can change from a superscript position to a normal position, but not to a subscript position), they are chosen based upon the position error distribution. With diagram-based sketches, the distribution is simply the position error rate, since there is a single type of position error, label association errors. The position error distribution for mathematics-based sketches defines the relative prevalence of the various kinds of substitution rules (subscript errors, not-superscript errors, etc.).

Symbol errors are applied in much the same way; the nodes with visible symbols are randomly ordered and the first node is selected. The Error Creator checks if there are any substitute symbols in the error alphabet for the chosen symbol. If an error can be applied to a symbol in the node, an error is randomly chosen according to the substitute symbols defined in the error alphabet. Once *n* symbol errors have been placed, the Error Creator has finished working on the sketch.

Diagram-based sketches can have a special type of symbol error. Since shape-based symbols are composed of geometric shapes, symbol errors can be configured to act upon the component shapes instead of the entire symbol. This allows for potentially nonsensical symbols to appear in the sketches, or sets of shapes that have no defined meaning.

Section 4.3.2.2.1 Restrictions on Errors In order to simplify some of the errors that users receive, and to maintain a level of realism in the errors the recognizer generates, we chose to restrict or not implement a few error categories in our recognizer. Some errors are hard to predict; from a users point of view, it is not always obvious what went wrong. For our studies with the WOZ Recognizer, we wanted to simulate a recognizer that would seem sensical to the participants, where they might be able to understand why an error occurred. Additionally, with errors that seem random, it is harder to determine where the errors should be created. We chose not to implement under-segmentation errors, because of the difficulty in implementing such errors in a realistic manner. Under-segmentation errors are harder to simulate because they require more complex examinations of the parse tree to identify potential locations for the errors. The wizard also has increased cognitive burden to determine if the ink strokes provide an acceptable basis for the under-segmentation.

We enacted several restrictions based upon domain knowledge on the kinds of errors the WOZ Recognizer creates for mathematics-based sketches and the conditions under which these errors can be created. In most cases, the wizard is able to disable these restrictions if needed. We restricted super- and subscript errors to one or two symbols; not-super- and not-subscript errors have the same restriction. We enacted this restriction as we wanted the WOZ Recognizer to produce errors that affected only a small number of symbols and to help the recognizer to meet its accuracy targets; placing many symbols in a superscript position in a superscript error would result in a large decrease in position accuracy. Similarly, errors of enclosure are restricted to a few symbols; we enacted this rule for the same reasons as with script errors, accuracy and small errors. Super- and subscript errors can only be created on non-operator symbols and operators with two or less operands (two or less children nodes in the tree).

Symbol errors also have a few restrictions; an error where a symbol is over-segmented into multiple symbols can only occur if that symbol is marked as being written in multiple strokes by the wizard. We included case errors in our error alphabet, but we also allow these errors to be enabled or disabled since the problem of case recognition is challenging.

Section 4.3.2.3 Recognition Accuracy Monitoring

Once the sketch has been created by the Error Creator, it is given to the Accuracy Regulator to tally the recognition accuracy. Simply put, the Accuracy Regulator counts the number of symbols and positions in the model of the WOZ Recognizer's sketch that are the same as in the model of the participant's sketch. In addition to per-sketch accuracies, the completed sketch accuracies are also calculated. The symbol and position accuracies are calculated by the Accuracy Regulator and passed along to the UI to be displayed to the wizard.

While the WOZ Recognizer tries to reach each accuracy target for every sketch, hitting an exact percentage is not always possible since sketches have great variability in size and complexity. Thus, the target accuracy is reached across multiple sketches, not necessarily on an individual sketch basis. A single symbol error in one sketch may reach 75% symbol accuracy and 95% accuracy in another sketch.

Since some errors affect more than one symbol or position, the target accuracy is not always met by the Error Creator. There is an element of randomness involved in these cases, thus, the wizard must manipulate the system by regenerating the WOZ Recognizer's sketch until they receive satisfactory accuracy. Symbol accuracy rates rarely miss the target by any great amount since there are many items in the error alphabet. However, position errors can need more manipulation when dealing with certain structures like division, where large errors can be introduced.

Diagram-based sketches have two symbol accuracy targets, one for label symbols (the label accuracy target) and one for graph symbols (the symbol accuracy target). We used two targets for these sketches, as these different kinds of symbols might be handled by different recognition systems in a real recognizer, and we wanted to give wizards the option to simulate such a thing.

Section 4.3.2.4 Error Alteration

A recognition error might not always be a great fit for the user's writing, so the WOZ Recognizer can alter specific errors. When an alteration is made, the specified error is given to the Error Creator by the Wizard Interface. The error is examined and viable alternate categories/substitutes for the error are determined (e.g. subscript error becomes a superscript error, or a different substitute symbol is chosen). For position errors, the error categories were chosen so that the recognition rate would not change. An alternate error is chosen at random according to the error distributions and the new recognition error is substituted for the old one in the list of recognition errors. The Error Creator then takes the original sketch and re-applies the previously generated recognition errors in order as described in Section 4.3.2.2. Finally, the revised sketch is given to the Output Generator to be displayed and to the Accuracy Regulator for recognition accuracies to be updated.

Section 4.3.2.5 Error Correction

Users can correct recognition errors with the Participant's Interface. As discussed in Section 4.4.2, the user can erase their writing and rewrite it. The wizard is able to select the part of the sketch that was rewritten and remove the recognition errors contained therein. When an error is selected to be removed, the Error Creator is given the selected error. It then removes the error from its list of errors for the sketch. As with error alteration, the Error Creator takes the original sketch and re-applies the previously determined recognition errors in order. This recreates the sketch without the recognition error that was removed. The sketch is then given to the Output Generator to update the UI. The sketch is not given to the Accuracy Regulator unless the sketch has not been transmitted to the Participant's Interface, as previously discussed.

Section 4.3.3 Output Generator and UI

After the Accuracy Regulator examines the sketch, it is passed to the Output Generator for a typeset version to be created. The Output Generator takes the graph-based sketch representation and creates a well-formatted visual representation of the sketch. Both the model of the Participant's sketch and the model of the WOZ Recognizer's sketch are taken by the Output Generator and parsed into their visual representations. The positional relationships between symbols in a mathematics-based sketch are automatically inferred based upon the mathematics. The layout algorithm used is based upon LaTeX formatting rules [69]. Since the WOZ Recognizer is focused on copy tasks, where the user copies a sketch from a handout, each diagram-based sketch must have its visual layout (the positional relationships between nodes) defined by the wizard in order for the WOZ Recognizer to display it. This makes it easy to have the same diagram on screen as on paper without the wizard having to manipulate the diagram's layout. While this does not allow for the user to create their own diagrams, it does allow for easy batch and streaming recognition simulation.

For mathematics-based sketches, a tree-view of the graph structure is also used to facilitate various wizard interactions with the sketch. The Output Generator converts both the model of the participant's sketch tree and the model of the WOZ Recognizer's sketch into visual representations of the trees (see Figure 4.10).

Finally, the UI is given the accuracies calculated by the Accuracy Regulator to be displayed (see Figure 4.11).

Section 4.3.4 Streaming and Batch Recognition

When the command to send a sketch to the Participant's Interface is received, several things happen. First, the Accuracy Regulator commits the current sketch's accuracies to the completed sketch accuracies (the Accuracy Regulator keeps count of the number of correct symbols/positions and the total number of symbols/positions). While the Accuracy Regulator has previously calculated the per-sketch/completed sketch accuracies, the per-sketch accuracies are malleable until transmission, as simulation might be performed on the sketch again. Transmission is considered the cutoff point for re-simulating recognition. The sketch is then given to the Network Component which creates and sends a message containing the sketch to the Participant's Interface. The Participant's Interface uses the Output Generator to convert the sketch into the visual representation (as in Section 4.3.3). Depending on the recognition mode, the structures are immediately parsed by the display system and shown to the user (streaming recognition), or are held until the recognition process has been invoked by the participant (batch recognition).

In streaming recognition mode, the wizard selects a subset of the sketch to send to the Participant's Interface that corresponds to what the user has drawn. This process is described in Section 4.4.2. Once the subset has been selected, the *Streaming Selector* create a subgraph of the sketch's graph. The subgraph is then given to the Output Generator to be shown to the wizard. When the transmission command is received, the subgraph is sent to the Participant's Interface, where the Output Generator converts it to be displayed to the user.

With mathematics-based sketches, the Streaming Selector is given a parse tree and a number n (the number of symbols to include) as input. The Streaming Selector traverses the parse tree in a manner according to how the expression was typically written in our preliminary data collection and selects the first n symbols. The traversal algorithm is a combination of pre-order and in-order traversal. For example, for the expression $-n$, the - is the parent node of the n, and is selected before the n. In $m-n$, the - is the parent of the m and the n, but is selected after the m, but before the n. The Streaming Selector selects the nodes from the tree and constructs a new tree from the subtree.

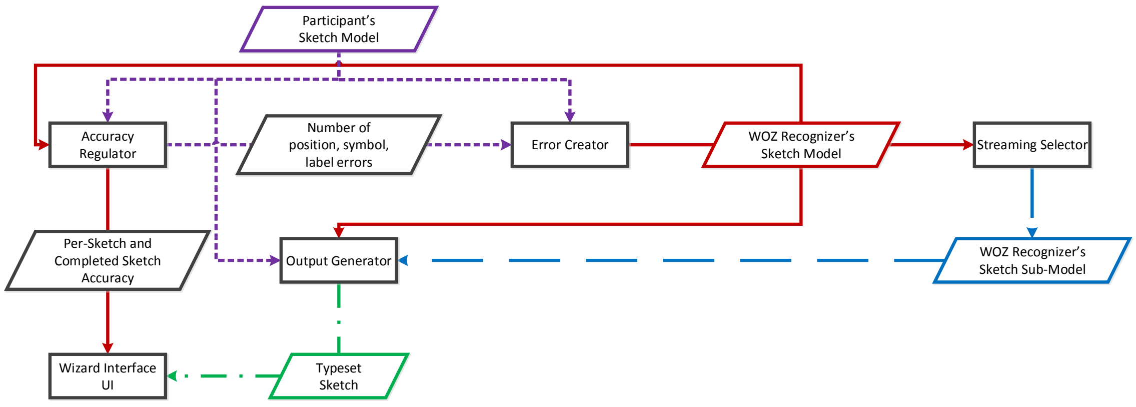

Participant's Sketch Model
Accuracy Regulator
Number of position, symbol, label errors
Error Creator
WOZ Recognizer's Sketch Model
Streaming Selector
Per-Sketch and Completed Sketch Accuracy
Output Generator
WOZ Recognizer's Sketch Sub-Model
Wizard Interface UI
Typeset Sketch

Figure 4.6: Data flow throughout the WOZ Recognizer. The rectangles indicate components in the WOZ Recognizer, while the rhomboids indicate data. The short dashed purple lines show where the model of the Participant's sketch is passed. The solid red lines show the flow of the model of the WOZ Recognizer's sketch, and the green lines with dashes and dots show the passing of the typeset sketches. The long dashed blue lines show the flow of the modified sketch model during streaming recognition to the various components of the WOZ Recognizer.

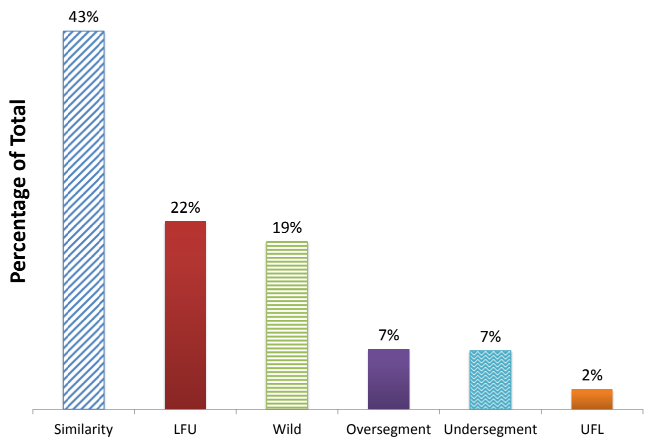


Figure 4.7: Distribution of symbol errors (Figure 4.3) from our preliminary data collection. The default error alphabet in the WOZ Recognizer is based upon this distribution.

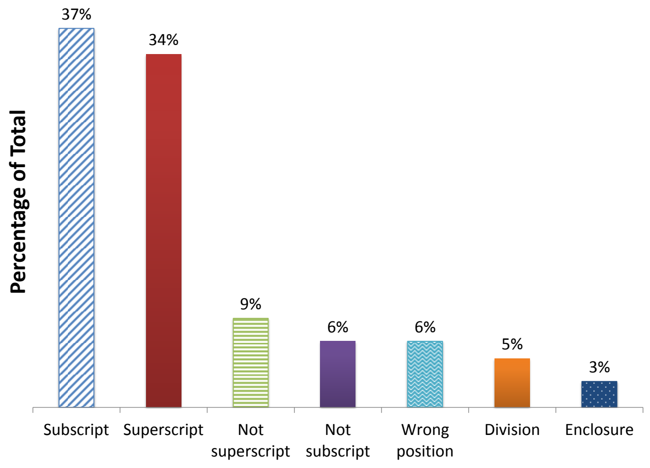


Figure 4.8: Distribution of position errors from our preliminary data collection. The categories are sorted in order of most prevalent to least prevalent. The default error grammar in the WOZ Recognizer is based upon this distribution.

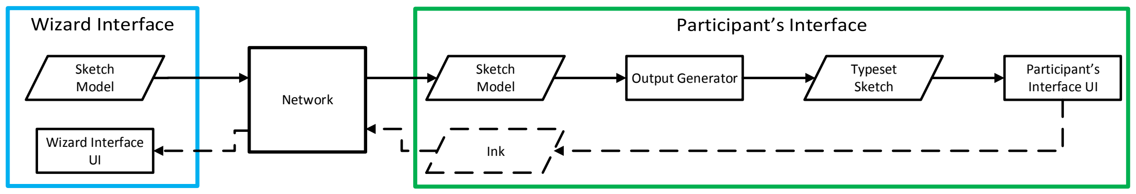


Figure 4.9: Primary data flow between the Wizard Interface and the Participant's Interface. The rectangles indicate components in the WOZ Recognizer, while the rhomboids indicate data. The Wizard Interface sends sketch models to the Participant's Interface, where they are given to the Output Generator and converted into a visual representation. The visualization is then displayed in the UI. Ink strokes are sent from the Participant's Interface to the Wizard Interface where they are displayed in the UI. In addition, various control information, such as recognition mode and UI events are sent between the two interfaces.

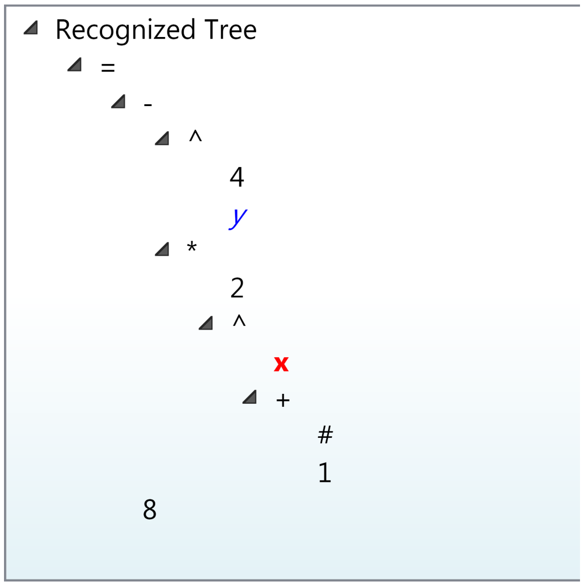


Figure 4.10: Tree-view of a WOZ Recognizer's sketch. Symbol errors are shown in blue and position errors in red. When both types of errors are rooted at the same symbol, the symbol is shown in purple. Some special symbols are used to represent extra information for the Error Creator and Output Generator, such as the # symbol, which signals an operator that normally has two operands, but only has one.

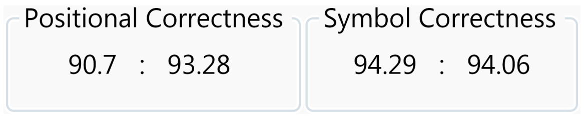


Figure 4.11: Symbol and position accuracy UI elements. For each type of accuracy (symbol, position, label), each UI widget shows the per-sketch and completed sketch accuracies. The per-sketch accuracy is shown first followed by the completed sketch accuracy.

Section 4.4 User Interfaces

Two applications compose the WOZ Recognizer, the Wizard Interface for the wizard and the Participant's Interface for the user. Using the Participant's Interface, a user sketches using a stylus; these ink strokes are transmitted to the Wizard Interface so that the wizard can monitor what is being written. The Wizard Interface uses the model of the participant's sketch to generate a model of the WOZ Recognizer's sketch which is sent to the Participant's Interface where it is displayed to the user in a typeset form. The WOZ Recognizer requires two workstations, the wizard station and the recognition station (see Figure 4.12 for the wizard station and Figure 4.13 for the recognition station). In contrast to the recognition station, the wizard station does not require any stylus input, because there are times when both hands are needed for keyboard interactions and we found it harder to pick up and put down a stylus compared to grabbing a mouse.

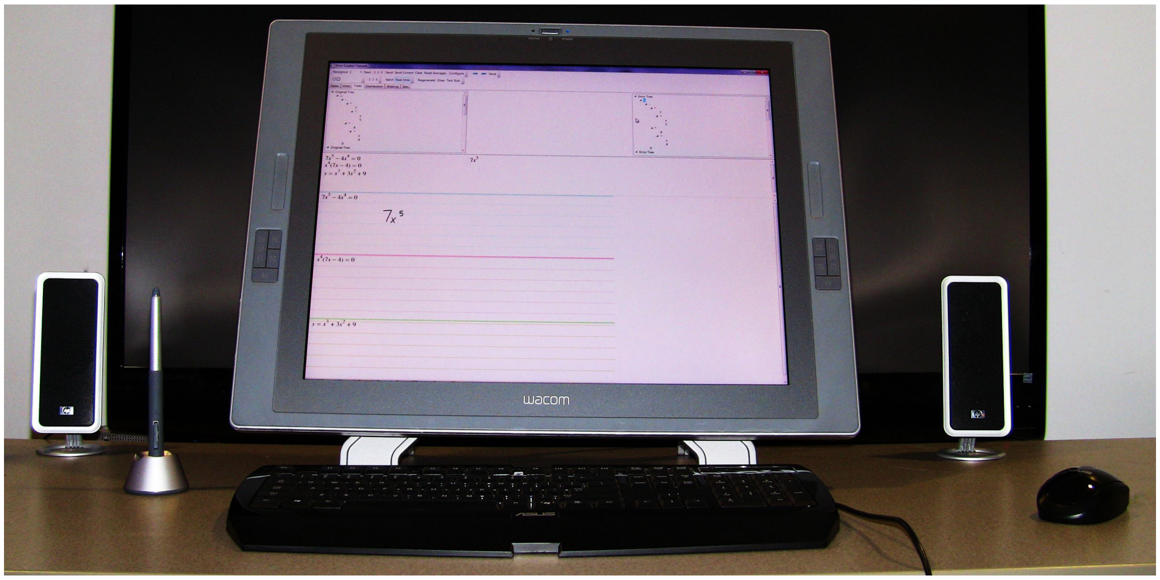


Figure 4.12: The wizard station uses two monitors, one to run the Wizard Interface, the other to show the Participant's Interface to the wizard. The Wizard Interface is shown on the large gray central monitor.

Figure 4.13: The recognition station is a tablet PC.

Section 4.4.1 Participant's Interface

The user interface for the Participant's Interface was designed to be simple to use. For the user, the WOZ Recognizer acts similarly to traditional recognizers like [48, 69]. There is a writing area with a horizontally lined background to help guide the user's writing (see Figure 4.14). The background is divided into multiple sections when the user will be working with multiple sketches at once, each section marked with a thick horizontal line at its top. The user can erase ink with a scribble erase gesture [69], or clear the whole writing area with a button press. The typeset visualization of the WOZ Recognizer's sketches appear vertically below the ink (based on results from [38]); as the user writes, the visualizations are repositioned so that they are clear and legible.

The Participant's Interface contains a minimal set of user interface elements, but was designed so that elements could be easily changed. The primary component of the interface is a canvas that allows the user to write (and communicates with a network layer to send the ink to the Wizard Interface) and displays the sketch visualizations (which are given to it by the network layer). In our previous work with the WOZ Recognizer, we extended the Participant's Interface to include controls for performing some functions with the recognized mathematics (such as copying, graphing, and solving, see Figure 4.15 for the extended UI).

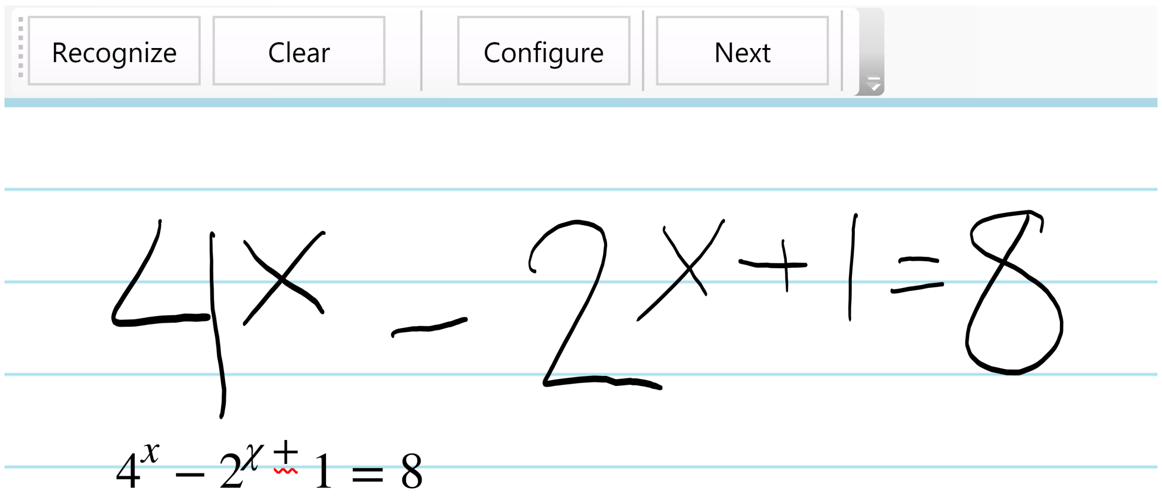


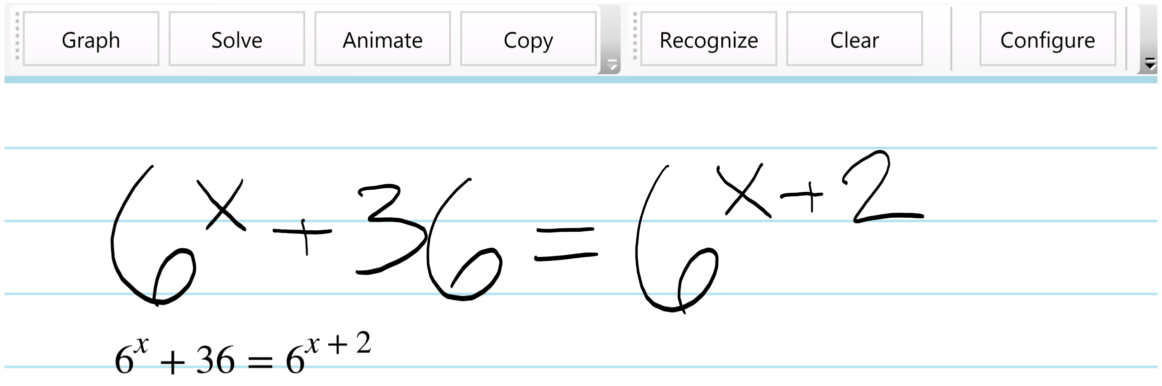
Figure 4.14: Participant's Interface with a mathematics single expression set. The recognized expression is shown below the user's writing. There is a symbol error in this sketch; the second *x* has been recognized as a χ. There is also a position error on the *1*; the *1* is not in a superscript position.

Section 4.4.2 Wizard Interface

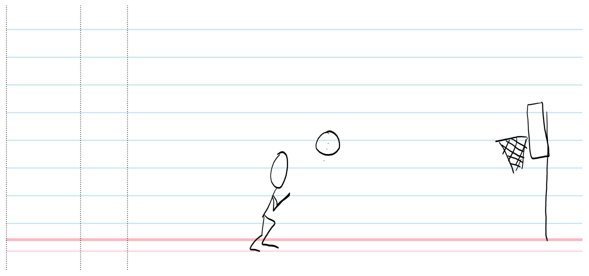
The wizard plays a very important role in simulating recognition, rejecting and choosing WOZ Recognizer's sketches, controlling when to send a sketch in batch recognition mode, and controlling the streaming recognition flow. Not all WOZ Recognizer's sketches are appropriate for every situation; perhaps a symbol error is inappropriate for the way the user wrote the symbol, or the wizard wants to avoid certain error types for the current sketch. The wizard can reject a sketch by generating a new one. Choosing an appropriate WOZ Recognizer's sketch is done by sending it to the Participant's Interface.

Compared to the Participant's Interface, the Wizard Interface has many sophisticated features. The wizard is presented with a visualization of the graph-structures of the model of the participant's sketch and the model of the WOZ Recognizer's sketch (see Figures 4.10 and 4.16). For mathematics-based sketches, parse trees are used; for diagram-based sketches, a typeset visualization of the diagram is used. Below the sketches, the user's ink is displayed. The wizard generates WOZ Recognizer's sketches by pressing a button or using keyboard shortcuts.

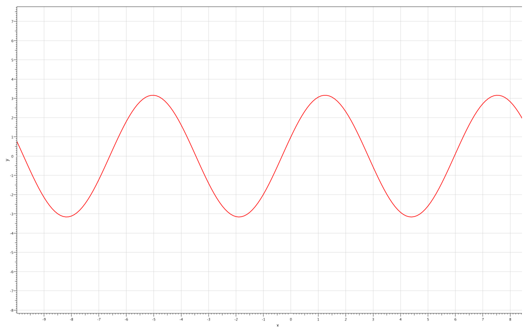
Sliders are used the control the different target accuracies; as previously described in Section 4.3.2, the target accuracies are used to determine the number of recognition errors.



(a) Extended User Interface



(b) Animate



(c) Graph

Figure 4.15: The extended user interface used in a study with the WOZ Recognizer. The Participant's Interface allowed the user to animate an illustration, copy a mathematics expression, graph an equation, or solve a series of equations.

By changing a slider and re-simulating recognition, the wizard can increase or decrease the symbol or position accuracy of a sketch. Accuracies are displayed for each type of accuracy (symbol, label, position), with the current sketch's accuracy followed by the completed sketch accuracy (see Figure 4.11 for an illustration of the accuracy for the first sketch in a set).

Once the WOZ Recognizer has created the sketch, it is displayed for the wizard to examine. If the wizard is unhappy with the sketch and how closely it meets the target accuracies, they can re-simulate recognition right away. Re-simulating recognition is performed in the same way as simulating recognition; the wizard uses a keyboard shortcut or a GUI button. As the user writes using the Participant's Interface, their ink strokes are displayed for the wizard on the Wizard Interface. If the errors in the WOZ Recognizer's sketch do not seem reasonable for the user's writing, the wizard can also choose to re-simulate recognition (although this is difficult to do during streaming simulation). With batch recognition, the wizard is able to transmit the WOZ Recognizer's sketch to the Participant's Interface prior to the user invoking recognition. The WOZ Recognizer's sketch will be held until recognition is invoked. This helps to minimize latency.

We included controls to allow for re-simulation of specific recognition errors to give the wizard some control during streaming. The wizard can select a recognition error from the parse tree (for mathematics) and select to re-simulate it (using keyboard, mouse, or GUI controls). This has the effect of cycling through recognition errors for that location in the tree. For example, where a superscript error has been introduced, re-simulating that error might swap it to a subscript error, as in Figure 4.17.

Streaming recognition is a more complicated process for the wizard than batch recognition. The wizard must tell the WOZ Recognizer which symbols have been written. As previously discussed, the system can traverse the parse tree for mathematics-based sketches to construct a parse tree for the first n symbols. The wizard tells the system the number of symbols to include using a keyboard shortcut (to increment or decrement the number of symbols) or using GUI buttons. The UI reflects this by showing the wizard what the output of the subtree would look like (i.e. for the parse tree for x^2+1, traversing two symbols would display x^2 in typeset format). Essentially, streaming recognition is performed by the wizard generating the n^{th} subtree for the n^{th} symbol the user writes. This subtree is then displayed to the wizard and can be sent to the Participant's Interface for display to the participant too.

There are many different possibilities for how someone can write a mathematics-based sketch. However, we found that most people tend to write in a few specific ways, so we implemented a few variations in traversing the parse tree. For instance, with division, people usually write the numerator, then the division symbol, and finally the denominator. Sometimes people will write the division symbol followed by the numerator and the denominator.

In addition, the wizard determines when to send the streaming output to the user. The wizard must balance latency (the desire to send input immediately, but not to send it too early), and realism (respond to how the user writes). In contrast to streaming recognition, batch recognition mode allows the wizard to largely eliminate latency (by preemptively sending the output to the user, where it is then held until the user invokes recognition), and affords the wizard more time to find realistic recognition errors.

When multiple mathematics-based sketches are to be simulated simultaneously, the wizard must select which parse tree to work with, for both batch and streaming recognition. This is as simple as selecting the correct root node in the Wizard Interface.

Transmitting the output of batch recognition to the Participant's Interface is performed with a keyboard shortcut or GUI button that sends the entire output. For streaming recognition, a separate shortcut and button are provided that only send what has been selected using the previously described streaming controls. These two controls provide the wizard with increased flexibility over what they send. For instance, if the wizard is a bit behind in streaming mode and the participant has finished writing the sketch, the wizard can use the keyboard shortcut for batch sending to send the entire output at once, catching them up to the user. With batch recognition, if the participant writes the first three symbols and invokes recognition, the wizard can respond by using the streaming controls to select the first three symbols and send with the keyboard shortcut to send the streaming output. The WOZ Recognizer takes care of immediate or buffered display output on the Participant's Interface based upon the selected recognition mode (batch or streaming).

Since the wizard may re-recognize a sketch several times in order to get a "good" sketch, a sketch is not permanently added to the set measurements until the WOZ Recognizer's sketch is sent to the Participant's Interface. This gives the wizard some ability and discretion to generate WOZ Recognizer's sketches that meet their criteria for "goodness" or recognition accuracy. When the wizard is generating and choosing sketches, they may not immediately get a sketch they find appropriate; in such a case, tweaking the error rates may be required. With certain types of expressions, like divisions and roots, this is a greater problem, which is generally due to position errors being larger than one or two positions and the great variations in the ways symbols are written.

With diagram-based sketches, it is easier for the wizard to send specific symbols because the spatial relationships between symbols are fixed. The wizard selects symbols to send during streaming recognition by clicking on the diagram visualization. As the symbols are selected, the graph structure builds up.

Users correct the recognizer's "mistakes" by erasing the "bad" symbols and rewriting new ones. We decided that error corrections would at best correct the error, and at worst replace the error with another error; the wizard has no way to introduce new errors to a WOZ Recognizer's sketch without generating a completely new sketch. This behavior is a limitation of our framework, which we may address in the future. However, we think that this behavior is also reflective of how an accurate and "good" recognizer would behave, as it would likely have the correct symbol in its n-best list [40] and update accordingly.

Error corrections can be tied to the recognition error rate; when the target accuracy is *n%*, and wizard tries to correct an error, *n%* of the time the error will correct, while the rest of the time it will not. Since some symbols can be written different ways (for instance, there are two common variants for the number *2*), the wizard can choose to regenerate a specific symbol error, which will cycle through the substitute symbol list (still based upon the relative probability of each substitute defined in the error alphabet). As well, the wizard can choose to regenerate specific position errors, such as a superscript error changing to a subscript error.

For mathematics-based sketches, the wizard removes errors from the WOZ Recognizer's sketch by selecting a node from the parse tree and pressing a button; this removes any errors rooted at the selected node. With diagram-based sketches, the wizard can click on parts of the visualization of the WOZ Recognizer's sketch to remove errors. A list of the various errors is also presented so that the wizard can easily and quickly remove a specific error. When a label becomes dissociated, the wizard can move it around the graph to make its position more realistic in relation to the user's handwriting.

The wizard has control over the distribution of position and symbol errors through the Wizard Interface. The wizard can create error alphabets through a GUI and save them to XML files. Diagram-based sketches can be saved as XML documents; a separate document is used for the diagram layout, which allows for multiple layouts to be made for each graph structure. Mathematics-based sketches are saved as MathML documents. The WOZ

Recognizer has built-in support for three diagram domains, finite state machines, chemistry bond diagrams, and flowcharts (see Figure 4.16 for examples). To load mathematics expressions into the system, the wizard can specify a configuration XML file containing a list of MathML files. The files are loaded into the WOZ Recognizer, parsed, and converted into the WOZ Recognizer's internal format. The sketches are available through a dropdown box of loaded sketches. Diagrams are loaded individually by selecting the two appropriate XML files using a file chooser dialog. If the diagram's domain is not currently loaded into the WOZ Recognizer, the wizard will need to select the domain definition XML file so that the diagram can be visualized and recognition errors defined.

The wizard interface provides several notifications to the wizard in the form of colored shapes that change color when an event occurs (the shapes can be seen in Figure 4.1). A notifier is setup for when the participant invokes recognition, presses the "next" button to go to the next sketch, simulation errors, and for recognition accuracy. The simulation error notifier blinks red when the wizard tries to remove a recognition error and then error is not removed as previously described. The accuracy notifiers give the wizard an at-a-glance view of the per-sketch and completed sketch accuracies, with four to six notifiers depending on the sketch type (two for symbol accuracy, two for position accuracy, and two for label accuracy). Each accuracy notifier shows red, yellow, or green depending on how close the accuracy is to the target.

Section 4.4.2.1 Domain Interface

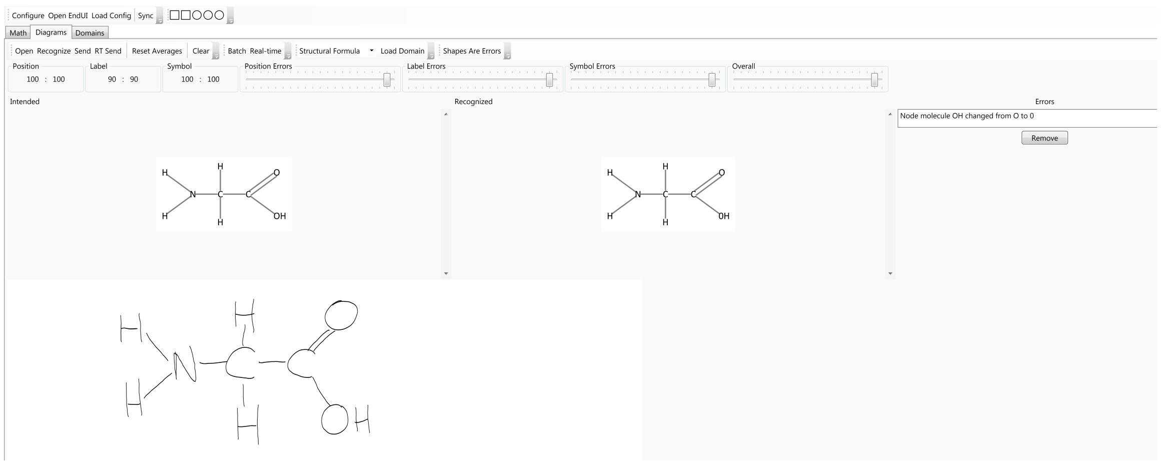
The wizard is able to define new diagram-based sketch domains through the Domain Interface. A domain is composed of a symbol alphabet, a shape alphabet, a label alphabet, and the error alphabet for symbols and shapes. Latin alphabet symbol errors (for labels) are defined in a separate GUI. The Domain Interface allows wizards to create symbol alphabets and error alphabets, and to define the distribution of errors in the grammar. For example, the finite state machine domain has a symbol alphabet of three symbols, state, accept-state, and transition. The state symbol is a circle, the accept-state symbols are a pair of concentric circles, and the transition symbol is an arrow. The shape alphabet is then only the circle and arrow shapes. We have defined one symbol error, where the accept-state symbol is recognized as the state symbol. The WOZ Recognizer provides various interface elements for defining new domains (see Figure 4.18). A separate interface component is used to position the component shapes relative to each other.

For traditional symbols, the wizard can type in a symbol and add it to the alphabet. They can also define substitute symbols for the correct symbol and the relative frequencies of the substitute symbols. For shape-based symbols, the wizard is able to combine the base geometric shapes supported by the WOZ Recognizer into new symbols. The WOZ Recognizer supports node symbols composed of rectangles, ellipses, triangles, and diamonds. Edge symbols can be composed of lines and arrows. For each shape, they can define its height and width and then arrange the shapes into the symbol by moving the shapes on a canvas. The shapes can be snapped to each other, if desired. Finally, the wizard affects the error grammar by entering the relative frequencies of the substitution rules.

Once the wizard has set the alphabet, error alphabet, and error grammar rule frequencies, the domain can be saved to an XML file and loaded into the WOZ Recognizer. The alphabet is also used with sketches to create the typeset version of a sketch. Each symbol in a diagram-based sketch is referred to by name; the alphabet is used to translate that name into shapes and their layout.

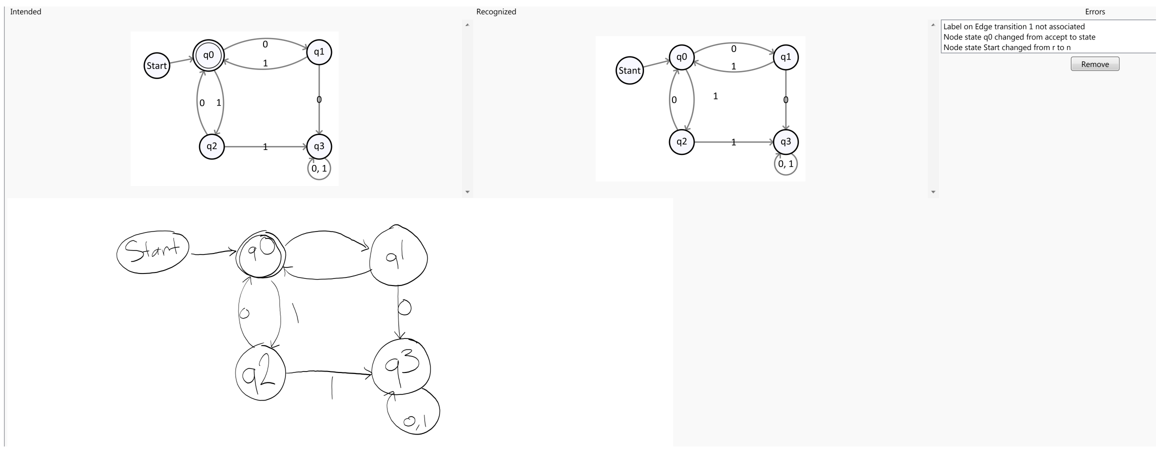
We used the Domain Interface to create the finite state machine, chemistry bond diagram, and flowchart domains included with the WOZ Recognizer. Part of the flowchart domain can be seen in Figure 4.18.

As discussed previously, the finite state machine alphabet has three symbols, state, accept-state, and transition. State and accept-state are node symbols, while transition is an edge symbol. Transition symbols are presented as labels on the edges. Chemistry bond diagrams show the bond structure of molecules and compounds. There are four symbols in the chemistry bond alphabet. There is a single symbol for nodes, molecule, which represents a molecule or compound. The molecule symbol has no shape representation, only a label, such as *C*, *N*, or *OH*. Three symbols are given for edges, singlebond, doublebond, and triplebond. Each symbol has one line per bond.


Configure Open EndUI Load Config Sync
Math Diagrams Domains
Open Recognize Send RT Send Reset Averages Clear Batch Real-time Structural Formula Load Domain Shapes Are Errors
Position 100 : 100
Label 90 : 90
Symbol 100 : 100
Position Errors
Label Errors
Symbol Errors
Overall
Intended
Recognized
Errors
Remove

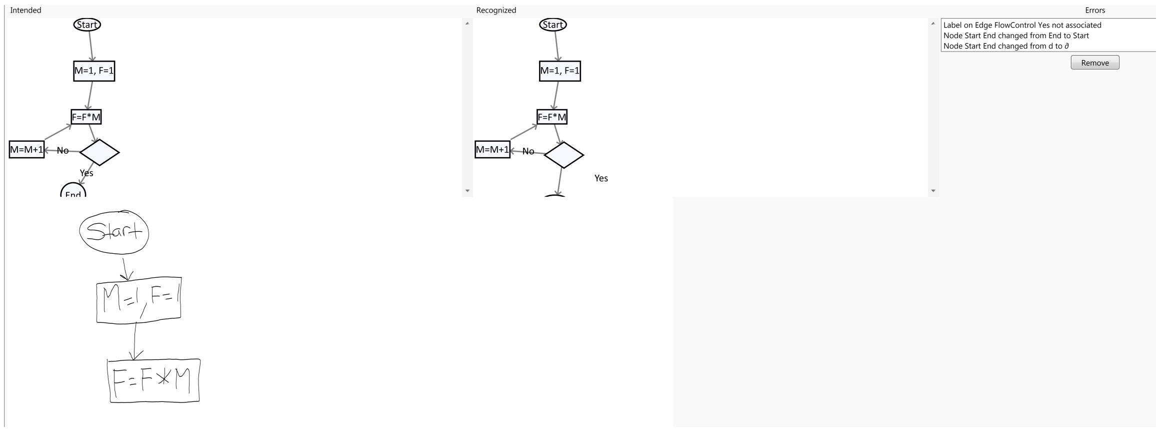
Node molecule OH changed from O to 0

(a) Chemistry Bond Diagram



Label on Edge transition 1 not associated
Node state q0 changed from accept to state
Node state Start changed from r to n

(b) Finite State Machine



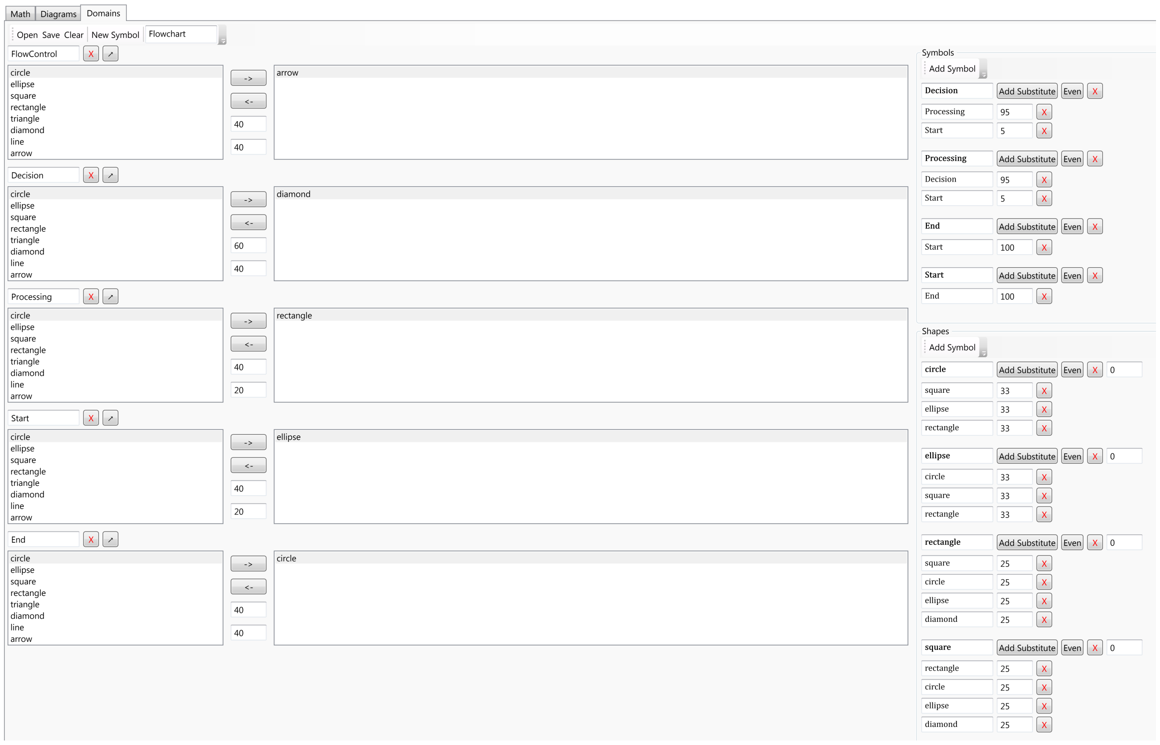
Label on Edge FlowControl Yes not associated
Node Start End changed from End to Start
Node Start End changed from d to ∂

(c) Flowchart

Figure 4.16: Examples of the built-in diagram domains with the Wizard Interface. In the Wizard Interface, the participant's and the WOZ Recognizer's diagrams are displayed near the top. The user's ink is displayed below.

$$\frac{2x}{x^2-1} \to \frac{2x}{x^{2^{-1}}}$$
$$\frac{2x}{x^2-1} \to \frac{2x}{x^{2_{-1}}}$$
$$\frac{2x}{x^2-1} \to \frac{2x}{x^{2-1}}$$

Figure 4.17: Example of how the WOZ Recognizer might cycle through possible position recognition errors targeted at the symbol *1*.



FlowControl
circle, ellipse, square, rectangle, triangle, diamond, line, arrow
40
40
arrow

Decision
circle, ellipse, square, rectangle, triangle, diamond, line, arrow
60
40
diamond

Processing
circle, ellipse, square, rectangle, triangle, diamond, line, arrow
40
20
rectangle

Start
circle, ellipse, square, rectangle, triangle, diamond, line, arrow
40
20
ellipse

End
circle, ellipse, square, rectangle, triangle, diamond, line, arrow
40
40
circle

Symbols

Decision
Processing 95
Start 5

Processing
Decision 95
Start 5

End
Start 100

Start
End 100

Shapes

circle 0
square 33
ellipse 33
rectangle 33

ellipse 0
circle 33
square 33
rectangle 33

rectangle 0
square 25
circle 25
ellipse 25
diamond 25

square 0
rectangle 25
circle 25
ellipse 25
diamond 25

Figure 4.18: New diagram-based sketch domains can be defined by the wizard using the Domain Interface. Symbols are defined using a variety of shapes (left side). On the right side, symbol substitutes are defined, as are shape substitutes.

Table 4.1: Mean accuracies for the WOZ Recognizer. The experiment shows that the WOZ Recognizer is sufficiently accurate in achieving its target accuracy.

		90%	95%	99%
Single	Position	89.0 $\sigma = 2.39$	95.0 $\sigma = 0.815$	99.1 $\sigma = 0.303$
	Symbol	90.0 $\sigma = 0.723$	94.9 $\sigma = 0.605$	99.0 $\sigma = 0.366$
Multiple	Position	90.0 $\sigma = 1.94$	94.8 $\sigma = 1.13$	99.1 $\sigma = 0.218$
	Symbol	90.4 $\sigma = 1.43$	95.1 $\sigma = 0.550$	99.0 $\sigma = 0.550$

We tested the WOZ Recognizer's accuracy in reaching its target accuracies for mathematics sketches by running it 24 times at three different recognition accuracies (90%, 95%, 99%) on both single and multiple expression sets (the expression sets used in [9], primarily composed of algebraic, geometric, and calculus expressions). Depending upon the set size, five expressions or five groups of three expressions composed each run. The test results show that the WOZ Recognizer is very close to achieving the target accuracies (see Table 4.1).

Section 4.6 Expert User Case Study

To better understand the effectiveness of the WOZ Recognizer's user interface and the processes used to simulate both math and diagrammatic recognizers with the tool, we conducted a small case study using two sketch recognition experts. These two experts have expertise in both building recognition systems and in studying the effects of recognition accuracy on user performance and acceptance of sketch-recognition technologies. They used the WOZ Recognizer to simulate recognition for mathematics and diagram-based sketches. For each type of sketch, two fifteen minute tasks were performed, one for batch recognition and one for streaming recognition. Each expert started with the mathematics task. For each task, the expert user acted as the wizard, controlling the WOZ Recognizer, while a member of our research lab copied sketches from a handout using the WOZ Recognizer's Participant's Interface. During each 15 minute task period, as many sketches as possible were simulated. For mathematics-based sketches, the target recognition accuracies were changed every five sketches. With diagram-based sketches, the target recognition accuracies were changed every three sketches. The sketches used in the study can be seen in Appendices A and B. Target accuracies were randomized for each participant; the accuracies targeted were 99%, 97%, 95%, 92%, 90%, and 87%.

Each expert user participated in four sessions, simulating mathematics recognition twice and diagram recognition twice with each session lasting between 1.5 to 2.5 hours. Upon arrival for the first two sessions, each expert user learned to use the various features of the WOZ Recognizer and practiced using the features, which typically took a minimum of one hour to complete. Once the tutorial portion had been completed and they felt comfortable with the WOZ Recognizer, the expert users performed the two testing tasks for a given session (i.e., streaming and batch for mathematics, streaming and batch for diagrams). After each session we asked each expert user to comment on the WOZ Recognizer's user interface, the processes used for simulating both mathematics and diagrammatic recognizers, any problems they encountered and how they solved them, and their overall reaction to how the tool can be used for both recognition perception research and recognizer development (see Appendices C, D, and E for the questionnaires we used). In addition to their comments, we measured streaming simulation latency, the number of mistakes they made in sending recognition feedback for streaming recognition, and how closely they were able to match the recognition target accuracies, all important measures in studying the WOZ Recognizer's simulation capabilities.

Our experimental setup consisted of two workstations, the recognition station and the wizard station. The recognition station was an HP Compaq tc4400, 12.1 inch tablet PC running Windows XP Tablet Edition. For the wizard station, a 21 inch monitor displayed the Wizard Interface for the WOZ Math Recognizer and a secondary 17 inch monitor showed the participant's screen. A desktop PC with two Intel Core i7 920 processors at 2.67 GHz and 9 GB memory running Windows 7 powered the wizard station.

Section 4.6.1 Results

We asked both experts whether they thought the WOZ Recognizer would be useful for doing sketch-related studies. Both thought it would be useful for mathematics. Expert A felt that it would be useful for studies on user perceptions and how they are affected by changing accuracy. Expert B was a bit apprehensive about diagrams, but thought that it was "okay for math recognition." With regards to diagrams, Expert A thought that they could create the diagrams that they work on using the domain interface.

Section 4.6.1.1 Ease of Use

Expert A thought that the mathematics part of the WOZ Recognizer was "surprisingly easy to use." They also felt that identifying errors and correcting them was easy. The details of recognition were also "easy to customize." They considered the recognition to be realistic "depending on the given accuracy." With diagrams, it was "easy to create realistic recognition errors at a specified accuracy." Expert B also felt recognition errors were realistic. Their largest issue with realism was that the WOZ Recognizer introduces no-recognition errors, as they felt that a recognizer should always produce some output for an ink stroke. Interestingly, the real recognizers we examined for mathematics sometimes produced no-recognition errors. In addition, the WOZ Recognizer gives the wizard the ability to tune the frequency of no-recognition errors, so they can be disabled in order to emulate such a recognizer. Being able to correct recognition errors "on the fly" was also a positive aspect of the WOZ Recognizer. Controlling accuracy was easy as well as fixing recognition errors for diagrams.

Some issues the experts had were alleviated through continued use and learning that the different control methods (keyboard versus mouse, etc.) made things easier. For instance, after the second mathematics session, when asked if they found it easier than the first session, Expert B said “Yes, because last time, for making corrections I was right-clicking and then clicking the menu. This time I was alt-clicking.” When Expert B moved on to diagrams, they thought that removing recognition errors with diagrams was easy “thanks to the keyboard shortcut.”

Overall, diagrams required more effort, as there were large sketches at times, and the point-and-click interface for selecting symbols for streaming recognition could be cumbersome. Expert B felt it was “not terribly difficult,” but “not terribly easy” either, and that it required a lot of clicking.

Section 4.6.1.2 Problem Areas

We asked the participants to provide feedback on what they considered the most negative aspects of the user interface. Streaming mode for mathematics was a particular area of frustration. One expert (Expert B) found it difficult to keep track of the subset of the sketch that had been sent to the user. “While I’m sending I’m looking at first the progression of this thing [typeset sketch] and simultaneously [the user’s writing]. Sometimes I go ahead and the user hasn't gotten that far and I need to wait or go back." While the wizard is presented with the selected subset of the sketch, we do not show the wizard whether it has been sent already. "It may be that I've sent some of it and then I moved to something else and I was looking at it and I forgot about it." We can address this issue either by providing a visual notification when the selected subset has not yet been sent, or by highlighting the part that has been sent.

Identifying what the user had erased and rewritten was problematic for both experts for mathematics and diagrams for batch and streaming modes. With streaming mode, it was considered a greater issue, since the wizard has to look at the user's writing and the tree or diagram to select and remove the recognition errors. For example, Expert A commented "In [streaming], when you correct your error, sometimes I can't see it, so I feel rushed." The wizard has to switch their focus between the user's ink and the sketch in order to see what has been erased and rewritten and then find the errors and remove them. It was suggested that after something was erased, new ink strokes should be colored a different color to make them easy to identify.

Similarly, both experts thought it was harder to identify the regions of the parse tree that needed to be edited for error correction. The error list in the diagram UI largely eliminated this issue, as it provided a simple list of recognition errors and selecting one highlighted the symbols in the diagram, which helped to differentiate between identical errors on two different nodes, labels, or edges.

Expert B found the user interface to have a learning curve and not be very approachable, as it is “complicated.” However, they said that “after you use it for a little while, it is okay.” For instance, after the second mathematics session, they felt that “Removing errors is easier today.” They exhibited frustration at times in keeping track of what was happening. They felt that once they started lagging behind, it was hard to keep up. Expert B thought that the notifications for events (such as recognition being invoked) should be improved, as they were too small. Expert A also had an issue with the notifications, as they commented “I forgot to look at [the notifications].” By improving the existing notifications and providing visual feedback for what has been sent, we think that we can sufficiently improve the issues the experts had with keeping track of the inputs and outputs of the WOZ Recognizer.

Streaming recognition of diagram labels was confusing for Expert B, as the selected symbols were indicated by increasing the weight of the font. Selection of the symbols in the label also caused issues for Expert B, as they would sometimes select too many symbols, cycle through the entire label, or forget to select the last symbol in the label. However, Expert B did think that the control scheme provided enough fine-grained control.

Expert B also wanted to be able to use a typeset mathematics sketch (similar to the diagram interface) instead of using the tree-view. However, once the expert had used the diagram part of the WOZ Recognizer, they commented that selecting parts of a diagram sketch was “harder than math,” due to the dexterity required in selecting symbols. Expert

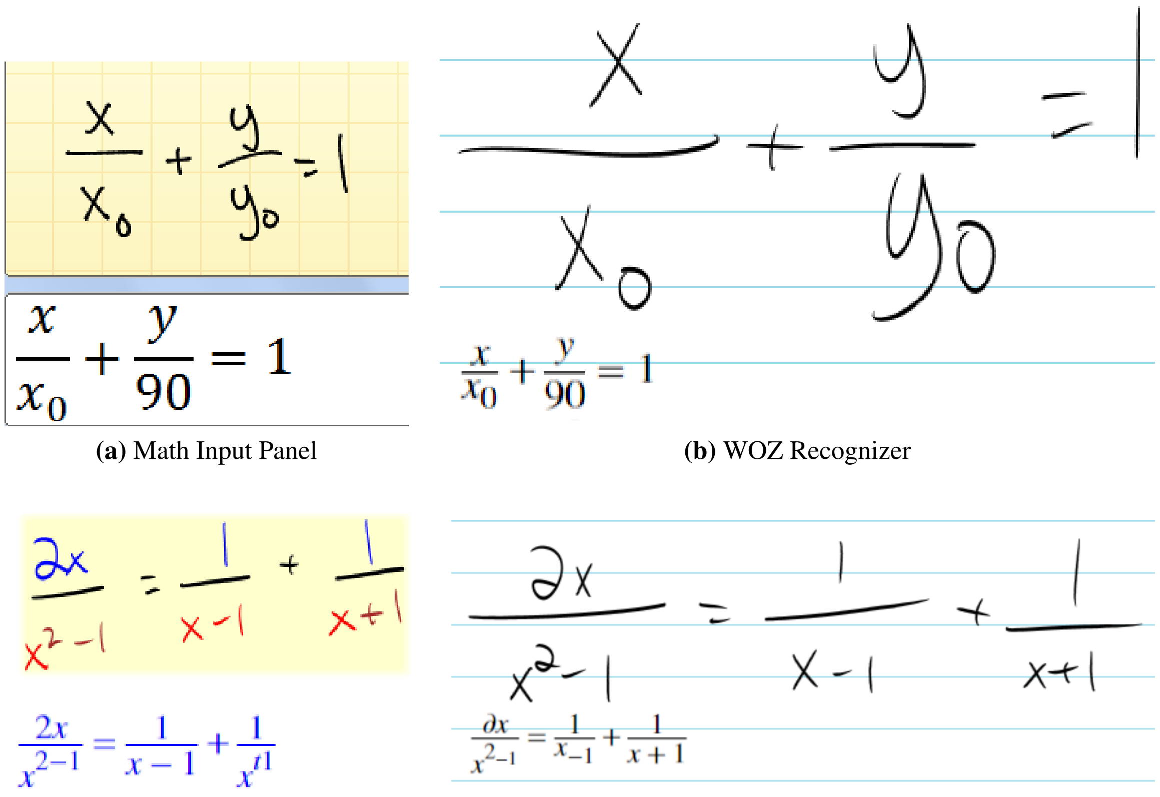
A also found some of the diagrams symbols hard to select. Both experts had a particular issue with bond diagrams because they have bond symbols composed of parallel lines that requires dexterity to select the individual lines. In addition, bugs they encountered in streaming recognition and error correction. For instance, when there was a label error and a shape error at the same node in a finite state machine diagram, removing the label error corrects the shape error.

One area we expected to have some problems was the display for the user's ink, as the screen resolution of the Wizard Interface was insufficient to show the entirety of the Participant's Interface. This causes the wizard to have to scroll to view the lower region of the ink area. We asked the experts for their feedback on the ink area and confirmed that it was an issue for larger diagrams. One resolution would be to scale the ink so that it fits in the available area; another is to scroll the ink area when the writing is displayed off-screen or the user scrolls down themselves.

One of our experts thought that the user interface for showing the recognition accuracy should have been larger; the controls for viewing and changing accuracy were also not always visible to the wizard with mathematics, as they are on a different tab than the tree view. This caused the participants to spend less time looking at the completed sketch accuracies and adjusting the target accuracies. The participants were also careless about selecting the WOZ Recognizer's sketches with accuracy in mind; instead they focused on quickly selecting a sketch and to some degree realism. We plan to change the user interface to emphasize the accuracy controls and improve notifications related to the completed sketch accuracies.

Section 4.7 Discussion

The WOZ Recognizer can produce sketches similar to real mathematics recognizers as demonstrated in Figure 4.19. An example of a mathematics-based sketch of a stoichiometric equation which was simulated with high position and symbol accuracy is shown in Figure 4.20.


$\frac{x}{x_0} + \frac{y}{90} = 1$
$\frac{x}{x_0} + \frac{y}{90} = 1$
$\frac{2x}{x^{2-1}} = \frac{1}{x-1} + \frac{1}{x^{t1}}$
$\frac{\partial x}{x^{2-1}} = \frac{1}{x_{-1}} + \frac{1}{x+1}$

(a) Math Input Panel **(b)** WOZ Recognizer

(c) StarPad **(d)** WOZ Recognizer

Figure 4.19: Examples of mathematics handwriting recognition from Microsoft's Math Input Panel [48] and StarPad [69], and similar results from the WOZ Recognizer. This shows that the WOZ Recognizer can produce realistic WOZ Recognizer's sketches.

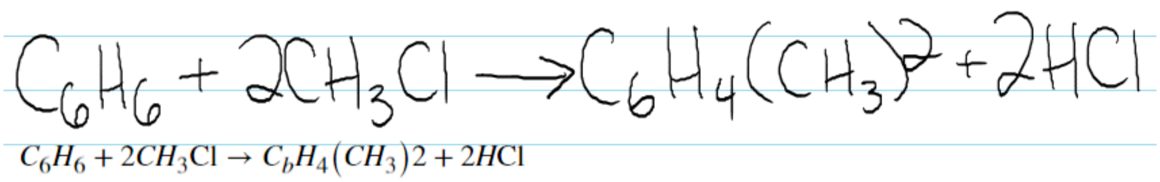


Figure 4.20: Example of a WOZ Recognizer's sketch using a stoichiometric equation. Two errors are present in the recognition, a *6* is changed to a *b* and a *2* is no longer a superscript.

CHAPTER 5: NOW OR LATER

Section 5.1 Introduction

When we developed the WOZ Recognizer, we started by developing its mathematics-based sketching capabilities. Mathematical handwriting recognizers are complicated, near-inscrutable black boxes from a user's point of view. A small change in the way one writes a symbol can have a large effect on recognized expressions. While mathematical handwriting recognizers are mostly generalized in terms of the mathematics they support, they can also show a narrow focus in their preference for one result over another. We cannot expect that mathematical handwriting recognition will achieve perfect accuracy in the short term, perhaps not even in the long term. Not only is there great variation in handwriting, but mathematics add a 2D layout component to the problem. Consequently, we need to examine how best to allow users to focus on entering mathematics and not on the recognizer's inaccuracies.

Invoking handwriting recognition comes in two flavors, now (streaming recognition) and later (batch recognition). Streaming recognition provides rapid feedback, which allows a user to examine the computer's understanding of their input early in the process. However, streaming handwriting recognition can be a distraction for some users, causing them to slow down to fix mistakes; conversely, it can provide valuable feedback by allowing a user to see what went wrong and when. Batch recognition, where the user writes an entire expression (or group of them) and then manually invokes recognition, can allow a user to write without distraction, but provides no recognition feedback during writing, often requiring the user to perform some action to invoke recognition. Batch recognition also requires visual processing and subsequent correction of a potentially large amount of errors at once. The parameters in mathematics recognizers greatly affect their usability and their users' feelings.

To explore these issues, we performed a study to see how users felt about these recognition modes. In order to perform the study we used the WOZ Recognizer (see Figures 4.12 and 4.13). Twenty-four users participated in our experiment with both recognition modes, three recognition accuracy levels, and different expression set sizes.

Section 5.2 Experimental Study

Our primary concern in performing this study was determining participant preference for recognition mode. Prior to the experiment, we formulated several hypotheses:

- As recognition accuracy increases, user preference for streaming recognition will increase. Participants will prefer immediate feedback when it is mostly correct.

- Accuracy will affect user feelings. Participants will report less frustration and distraction, and more ease in writing and correcting when working with more accurate recognizers.
- Equation set size will affect user feelings. Participants will report increased distraction and frustration with the larger set size. The increased number of errors will be more to find and fix at a time, making it more frustrating and distracting.

Section 5.2.1 Subjects and Apparatus

We recruited 24 college students from the general university population (18 male, 6 female) to participate in our study. The participants ages ranged from 18 to 31. Twelve participants had previous experience using tablet PCs, while eight had used some form of handwriting recognition software, and three had used mathematical handwriting recognition software. We had one left-handed participant. The experiment took approximately 1.5 to 2 hours to complete and each participant was paid 10 dollars for their time.

Our experimental setup consisted of two workstations, the participant station and the wizard station. The participant station was an HP Compaq tc4400, 12.1 inch tablet PC running Windows XP Tablet Edition (see Figure 4.13). The participant station was cordoned off from the wizard station in order to remove distractions for the participant and minimize any noises from the wizard. The sound of a fan was also played during the experiment to further minimize the sounds. For the wizard station, a 21 inch monitor displayed the Wizard Interface for the WOZ Recognizer and a secondary 17 inch monitor showed the participant's screen (see Figure 4.12). A desktop PC with two Intel Core i7 920 processors at 2.67 GHz and 9 GB memory running Windows 7 powered the wizard station. Two people were required to administer the experiment, a proctor and a wizard.

Section 5.2.2 Experimental Task

Participants were asked to complete six writing tasks; each consisted of writing expressions varied in size, at one of three recognition accuracy levels, which apply to both symbol and position accuracy, and in one of two recognition modes. Expression accuracy is measured in two ways, symbol accuracy and position accuracy. Symbol accuracy is measured as the number of correct symbols divided by the total number of symbols, and position accuracy is measured by dividing the correct number of parsing decisions by the total number of parsing decisions (see Section 4.2.3 for details). For each accuracy level, we used the same accuracy targets for each measure, symbol and position. The three accuracy levels (90%, 95%, and 99%) were chosen because we felt they were reasonable accuracy levels; any lower and users would likely find them too hard to use. That is to say that 90% accuracy is the minimum accuracy that we believe that users might find tolerable. We based our initial accuracy levels on established accuracy thresholds [20,36]. We then performed pilot studies to determine our minimum accuracy for the experiment. Two tasks were performed at each accuracy level; one task used a set of single expressions (single equation set) consisting of five expressions written individually, and the other task had a set of multiple expressions (multiple equation set) consisting of five groups of three expressions. For each task, participants performed two subtasks, each time writing the same expression set, once in batch recognition mode, once in streaming recognition mode. Overall, each participant performed six tasks, writing 120 expressions in total (60 unique expressions, each written twice).

Each task has its own set of expressions, so we constructed six equation sets. Single equation set tasks had five separate expressions and multiple equation set tasks had five groups of three equations. For the experimental tasks, we designed our expression sets to be in all lowercase; capitalization errors were also disabled. In real recognizers we examined [48, 69], changing case through erasing and rewriting was problematic at best; recognizers tend to solve this problem by providing functionality to allow the user to choose from a list of alternate recognized expressions. Consequently, we chose to avoid this issue altogether.

Section 5.2.3 Experimental Design and Procedure

We used a 3 by 2 by 2 within-subjects factorial design, where the independent variables were recognition accuracy, recognition mode, and set size. Two recognition modes, batch recognition and streaming recognition, were included; two set sizes were used, one equation by itself and three equations together. We felt that three equations as a group were a good compromise in terms of time spent writing and group size. The dependent variables were user preference for recognition mode and distraction level, which were determined through a questionnaire given after each recognition task.

A proctor guided the participants throughout the experiment, giving them questionnaires, finding mistakes in what they wrote, and performing interviews. First, the participants were given a pre-questionnaire. The pre-questionnaire asked the participants for their age, gender, which hand they write with, as well as whether they had ever used a tablet PC, handwriting recognition, or a mathematics recognizer. They then practiced using the different recognition modes. Participants were then given a preliminary task to familiarize themselves with the recognizer interface. During the explanation of the study and the interface, participants were told that they would experience different recognition accuracies during the experiment. While working with batch recognition mode, participants wrote and corrected two expressions, and then proceeded to write and correct a multiple equation group of three expressions in streaming recognition mode. Participants were then given a series of tasks to perform. The order in which participants worked through the different tasks was randomized and counterbalance such that one-third of the participants received the 90% accuracy tasks first, one-third received the 95% accuracy tasks first, and one-third received the 99% accuracy tasks first. The presentation of the multiple equation set task or the single equation set task first was also counterbalanced. As each task has two subtasks, one in streaming recognition mode and one in batch recognition mode, three of the tasks were performed with the streaming recognition subtask first, and three of the tasks were performed with batch recognition first. The participants were instructed to find their mistakes and correct them before moving on to the next expression; the proctor pointed out errors that they missed when necessary.

The post-task questionnaire asked subjects to rate their agreement with four statements on a seven point Likert scale, where 1 was Strongly Disagree, 4 was Neutral, and 7 was Strongly Agree:

- Easy to write: It was easy to write the expressions.
- Easy to correct: It was easy to correct the expressions when necessary.
- Frustration: It was frustrating writing and correcting the expressions when necessary.
- Distraction: I was distracted from writing expressions by the recognition system.

An interview after each task pair was also given, which asked which recognition mode the participant preferred for the previous task pair. In a final interview after all tasks were completed, we asked participants a few brief questions about the two recognition modes, such as whether they changed the way they wrote during the experiment, whether they watched the streaming recognized expressions as they wrote, and what they thought about how the recognized expressions were displayed.

We designed our expression sets so that they had similar numbers of symbols and positions (see Table 5.1). Most expressions were basic polynomial equations. Within the multiple equation sets, there were trigonometric equations in each set. The single equation sets all had at least one trigonometric equation and one integral. Example expressions can be seen in Figure 5.1.

Section 5.2.4 Results

We examined user's preferences for batch or streaming recognition for each expression set size at each accuracy level using chi-square tests (see Table 5.2). Most participants did not exclusively prefer one recognition mode to the exclusion of the other. Eighteen participants preferred batch for at least one of the six tasks and twenty-three participants preferred streaming for at least one of the six tasks. For multiple equation sets, there was a clear preference for streaming recognition at all three accuracy levels ($\chi^2_1 = 10.67$, $p < 0.05$). For single equation sets, at 90% accuracy there was also a preference for streaming recognition ($\chi^2_1 = 5$, $p < 0.05$), but there was no clear preference at higher accuracy levels. Using contingency tables, we examined participant preference for recognition mode. Looking at accuracy, there was no significance in preference across the three accuracy levels ($\chi^2_1 = 1.48$, $p = 0.48$). When we looked at equation set sizes, there was statistical significance ($\chi^2_1 = 7.91$, $p < 0.005$), meaning that there was a difference in preference for recognition mode between the single and multiple equation set tasks.

Table 5.1: Position and symbol totals for single and multiple equation sets. We balanced the sets so that they had similar position and symbol counts.

Multiple Expressions	Set 1	Set 2	Set 3
Positions	310	321	317
Symbols	223	205	204
Single Expression	**Set 1**	**Set 2**	**Set 3**
Positions	220	213	214
Symbols	98	102	99

$$3x^3 + x^2 + 1 = 8 \tag{5.1}$$
$$\int\int xy^2 + x^2y\, \mathrm{d}y\, \mathrm{d}x \tag{5.2}$$
$$p(t) = \cos(t-e) + \sin(t+k) \tag{5.3}$$

Figure 5.1: Example expressions used in our experiment.

Table 5.2: User preference statistics for batch and streaming recognition for each accuracy level and task set size. In most cases, there was a statistical preference for streaming recognition.

		Batch	Streaming	χ^2	p
90%	Single	6	18	6	$p < 0.05$
	Multiple	4	20	10.67	$p < 0.01$
95%	Single	11	13	0.167	$p = 0.683$
	Multiple	4	20	10.67	$p < 0.01$
99%	Single	10	14	0.667	$p = 0.414$
	Multiple	4	20	10.67	$p < 0.01$

For each subtask, we asked participants to evaluate the recognizer's symbol and position accuracy for the expression set they had just written. The mean accuracies are displayed in Table 5.3. Participants perceived the recognition accuracy to be no less than 5% below the actual recognition accuracy. Additionally, for lower accuracy levels, participants thought the multiple equation set tasks had lower accuracy than the single equation set tasks.

Table 5.3: Mean perceived recognition accuracies. Participants showed a clear underestimation of accuracies and had a greater underestimation for the multiple expression set tasks.

Single		90%	95%	99%
Batch	Symbol	84.3 $\sigma = 8.12$	88.6 $\sigma = 7.98$	94.0 $\sigma = 4.59$
	Position	79.5 $\sigma = 10.8$	86.6 $\sigma = 7.23$	91.4 $\sigma = 5.37$
Streaming	Symbol	85.6 $\sigma = 7.64$	88.9 $\sigma = 6.84$	93.8 $\sigma = 4.59$
	Position	79.0 $\sigma = 11.5$	84.6 $\sigma = 7.87$	91.6 $\sigma = 4.82$
Multiple		**90%**	**95%**	**99%**
Batch	Symbol	77.3 $\sigma = 16.8$	84.7 $\sigma = 11.8$	94.2 $\sigma = 3.91$
	Position	76.0 $\sigma = 17.1$	80.8 $\sigma = 12.4$	92.6 $\sigma = 4.95$
Streaming	Symbol	80.0 $\sigma = 12.9$	89.1 $\sigma = 7.04$	93.8 $\sigma = 4.38$
	Position	73.2 $\sigma = 13.8$	85.2 $\sigma = 7.72$	91.8 $\sigma = 6.63$

From the interviews we performed with each participant at the experiment's end, twenty-two participants (91%) reported that they had changed the way they wrote during the experiment. Mostly, people commented that they changed certain aspects of their writing in order to correct perceived errors in the way they wrote (based upon the recognized expressions). Several people reported that they changed the way they wrote super- and subscripts in order to correct those errors; this is not surprising as superscript and subscripts were common in our expressions (see Section 4.3.1 for more information on the distribution of errors in the WOZ Recognizer). We also asked participants whether they watched the streaming results as they wrote; twenty-one participants reported they had (87%). Of those who did, twelve reported they watched the results, but not all the time during the streaming tasks.

To analyze the data collected for each task, we performed an analysis using Friedman and Wilcoxon Signed Rank tests on the Likert item data [15]; we also performed a post-hoc correction using the Holm's Sequential Bonferroni adjustment [27]. For these statements, we compared the data across recognition mode at each accuracy level and set size, across accuracy levels for each recognition mode and set size, and across set size at each recognition mode paired with the two higher accuracy levels (95% and 99%). Average responses can be found in Tables 5.4 through 5.6 (recall 1 = strongly disagree, 7 = strongly agree).

Section 5.2.4.1 Easy To Write

As we expected, there were some significant differences in how easy participants found it to write the expressions at different accuracy levels. The Friedman test for the ease of writing expressions showed significance ($\chi^2_{11} = 63.613$, $p < 0.001$). Increased accuracy made it easier to write the expressions. For the single equation set and batch recognition, increasing accuracy always led to a mean increase in participants' reported ease in writing; the 99% accuracy tasks were easier than the 90% ($Z = -3.358$, $p < 0.0167$) and 95% accuracy tasks ($Z = -2.271$, $p < 0.025$), as were the 95% accuracy tasks compared to the 90% accuracy tasks ($Z = -2.000$, $p < 0.05$). The multiple equation set also had some significant differences in ease of writing based upon accuracy; with the batch recognition mode, the 99% accuracy tasks were easier than the 95% accuracy ($Z = -3.307$, $p < 0.0167$) and 90% accuracy tasks ($Z = -2.274$, $p < 0.025$). Streaming recognition mode produced two significant results; 99% accuracy had greater reported ease in writing than 90% accuracy ($Z = -3.402$, $p < 0.0167$) and 95% accuracy tasks had greater reported ease than 90% accuracy tasks ($Z = -3.080$, $p < 0.025$). Comparing across set size, participants found it easier to write single expressions than multiple expressions using streaming recognition and 90% accuracy ($Z = -2.862$, $p < 0.0167$). No other comparisons for accuracy, nor for set size nor recognition mode were significant after applying the post-hoc Bonferroni correction.

Table 5.4: Mean ease in writing expressions. Participants generally found it easy to write expressions regardless of recognition mode.

Set Size	Mode	Accuracy	Mean	σ
Single	Batch	90%	5.875	0.900
		95%	6.208	0.932
		99%	6.542	0.721
	Streaming	90%	6.000	1.216
		95%	6.083	0.929
		99%	6.375	0.711
Multiple	Batch	90%	5.875	1.262
		95%	5.667	1.204
		99%	6.458	0.779
	Streaming	90%	5.375	1.279
		95%	6.125	1.076
		99%	6.292	0.908

Section 5.2.4.2 Easy To Correct

The Friedman test on ease of correction also showed significance ($\chi^2_{11} = 50.656$, $p < 0.001$). As with ease in writing expressions, significant differences in the ease in correcting expressions were found when comparing higher accuracy tasks with lower accuracy tasks. With both the single equation sets and multiple equation sets, participants found it easier to correct higher accuracy tasks than lower accuracy tasks. With batch recognition mode and the single equation sets, participants found it easier to correct expressions at 99% accuracy than at 90% accuracy ($Z = -2.561$, $p < 0.0167$). Using streaming recognition and the single equation sets, participants reported greater ease in correcting expressions at 99% accuracy than at 90% accuracy ($Z = -2.967$, $p < 0.0167$) and at 95% accuracy ($Z = -2.240$, $p < 0.025$). When working with the multiple equation sets and batch recognition mode, 99% recognition accuracy made it easier to correct expressions than 90% accuracy ($Z = -2.662$, $p < 0.0167$). The final significant differences in ease of correction were found with the multiple equation sets and streaming recognition mode; participants found increased ease in correction with 99% accuracy than with 90% ($Z = -2.818$, $p < 0.0167$) and 95% accuracy ($Z = -2.303$, $p < 0.025$). No other comparisons were significant after the Bonferroni correction.

Table 5.5: Mean ease in correcting expressions. As with writing expressions, participants generally found correcting expressions easy regardless of recognition mode.

Set Size	Mode	Accuracy	Mean	σ
Single	Batch	90%	5.708	1.122
		95%	6.167	0.917
		99%	6.417	0.776
	Streaming	90%	5.792	1.141
		95%	5.833	1.090
		99%	6.375	0.875
Multiple	Batch	90%	5.375	1.377
		95%	5.625	1.408
		99%	6.250	1.152
	Streaming	90%	5.708	1.367
		95%	5.833	1.308
		99%	6.375	0.770

Section 5.2.4.3 Frustration

For participant reported frustration levels, the Friedman test showed significance ($\chi^2_{11} = 88.66$, $p < 0.0001$). Comparing frustration levels across different accuracies again produced significant differences. Increasing accuracy decreased frustration. When writing in batch recognition mode and the single equation sets, 99% accuracy was less frustrating than 90% accuracy ($Z = -4.122$, $p < 0.0167$) and 95% accuracy ($Z = -2.358$, $p < 0.025$); 95% accuracy was also less frustrating than 90% accuracy ($Z = -2.263$, $p < 0.05$). One significant difference in frustration levels was found when participants used the single equation set and streaming recognition, 99% accuracy was less frustrating than 90% accuracy ($Z = -3.102$, $p < 0.0167$). There were also significant differences across accuracy levels when participants used the multiple equation sets. When working with batch mode, participants reported less frustration when 99% accuracy was used than when 90% ($Z = -3.206$, $p < 0.0167$) or 95% accuracy was used ($Z = -2.428$, $p < 0.025$). Working with streaming recognition, participants reported being less frustrated at 99% accuracy than at 90% accuracy ($Z = -3.868$, $p < 0.0167$) and at 95% accuracy ($Z = -3.216$, $p < 0.025$). They also reported less frustration at 95% accuracy than at 90% accuracy ($Z = -2.829$, $p < 0.05$). No other comparisons across accuracies were significant after post-hoc correction.

Comparing frustration levels across set sizes produced a single significant result. Writing expression groups was more frustrating than writing a single equation; this proved significant with streaming recognition and 90% ($Z = -2.904$, $p < 0.0167$). All other comparisons were not found to be significant.

Section 5.2.4.4 Distraction

We found significant differences in distraction levels using the Friedman test ($\chi^2_{11} = 32.74$, $p < 0.001$). Comparing across recognition mode, for the single equation set at 95% accuracy, participants found batch recognition less distracting than streaming recognition ($Z = -2.326$, $p < 0.025$). For the multiple equation set at 90% accuracy, participants reported being less distracted using batch recognition than using streaming recognition ($Z = -2.809$, $p < 0.0167$). Comparing across expression set size showed one significant result; when using streaming recognition at 90% accuracy, participants reported being less distracted using the single equation set than with the multiple equation set ($Z = -2.309$, $p = 0.0167$). Comparing distraction levels across accuracies did not reveal any significant results.

Table 5.6: Mean frustration levels. Lower recognition accuracies lead to higher levels of frustration.

Set Size	Mode	Accuracy	Mean	σ
Single	Batch	90%	3.083	1.349
		95%	2.417	1.742
		99%	1.708	0.859
	Streaming	90%	2.958	1.517
		95%	2.291	1.517
		99%	1.917	0.974
Multiple	Batch	90%	3.375	1.837
		95%	3.042	1.601
		99%	2.333	1.523
	Streaming	90%	3.833	1.685
		95%	2.917	1.530
		99%	1.958	1.042

Table 5.7: Mean distraction levels. Distraction levels did not exhibit much variance.

Set Size	Mode	Accuracy	Mean	σ
Single	Batch	90%	1.833	1.129
		95%	1.417	0.776
		99%	1.458	0.833
	Streaming	90%	2.083	1.283
		95%	1.875	1.076
		99%	2.250	1.359
Multiple	Batch	90%	1.625	0.875
		95%	1.750	1.113
		99%	1.458	0.833
	Streaming	90%	2.417	1.586
		95%	2.250	1.622
		99%	1.833	1.274

Section 5.3 Discussion

Contrary to our hypothesis, it is clear that recognition accuracy had little impact on user preference for recognition mode, as preference did not generally vary with accuracy. In other words, at different accuracies, participants did not prefer batch or recognition more. Instead, the expression set's size influenced recognition mode preference. As mentioned earlier, we hypothesized that at low accuracies, participants would prefer batch recognition, and as recognition accuracy increased, participants would increasingly prefer streaming recognition over batch recognition. Our experiment's results do not support this hypothesis. We think that participants preferred streaming recognition for the multiple equation set tasks and the 90% accuracy single equation set task, because there were more errors to correct and streaming recognition provided immediate feedback on errors, allowing participants to immediately and easily find and correct recognition errors. As accuracy increased in the single equation set tasks, finding all the errors became easier since there were fewer to find.

In contrast to participant preference for recognition mode, study participants found it easier to write and correct and were less frustrated at higher recognition accuracies, as can be seen in Tables 5.4 through 5.6. Interestingly, distraction levels presented an anomaly; in two cases, the mean distraction levels increase from 95% accuracy to 99% accuracy. Only when participants used streaming recognition and the multiple equation set did we see a downward trend in distraction across all three accuracy levels as we expected. In the case of streaming recognition with the single equation set, participants reported greater distraction levels at 99% accuracy than at 90% and 95% accuracy. This trend runs contrary to our expectations that increased accuracy would lead to decreased distraction, as fewer errors would distract participants from copying the expressions.

One other anomaly presented itself in the Likert scale responses. When writing the multiple equation set and using batch recognition mode, the mean reported ease in writing the expression decreased from 90% recognition to 95% recognition (but went back up for 99% recognition accuracy). Perhaps this is attributable to some aspect of the expression sets, such as a larger number of exponents or subscripts. It is unclear the exact nature of these anomalies; we will have to experiment further to determine the cause.

As with our previous work, we designed our study with a focus on experimental validity rather than ecological validity. To this end, we chose recognition rates that spanned a range that we determined might be acceptably accurate and controlled accuracy between users, rather than examining real recognizers and measuring their accuracy post-hoc. With real recognizers, we would expect to see accuracy variance between users, which we want to factor out, so that we can see the effect of specific accuracy levels on user perceptions.

CHAPTER 6: APPLICATIONS AND ACCURACY

Section 6.1 Introduction

Following our first study, where we examined user preference for “now or later” recognition, we thought that we might explore how recognition accuracy intermingled with the application being used for recognition. That is, if the purpose of an application is really interesting, does that affect how tolerant the user is of recognition errors? With traditional handwriting recognition, one paper examined this idea [20], while another paper examined a similar idea [36]. As with our previous work using the WOZ Recognizer, we focused upon mathematics sketching with the hypothesis that mathematics sketches would produce similar results as diagram sketches.

Handwritten mathematical expressions are a powerful component of intelligent tutoring systems and computational engines [14, 35]. The ability to take a written expression and perform complicated actions on them can be extremely enticing and exciting for users. User interest in the application they are performing can affect how they use software and what they are willing to tolerate from it. In this experiment, we explore the impact of recognition inaccuracy during mathematics handwriting recognition with various mathematics applications.

Twenty college students took part in our study, where they wrote mathematics equations and experienced four applications using the recognized mathematics, including, copying, making a graph, animating, and solving.

Section 6.2 Experimental Study

In order to explore how people were willing to tolerate (explicitly or implicitly) errors in mathematics handwriting recognition, we performed a user study where participants were presented with four applications relating to mathematics recognition. Recognition accuracy was decreased over the course of each application. Participants were asked to write various equations for each application and evaluate whether the recognizer's accuracy was acceptable to them for the application. We hypothesized that participants would be more tolerant of recognition errors with applications they considered more interesting or useful as shown with other handwriting domains [20, 36].

Section 6.2.1 Subjects and Apparatus

We recruited twenty college students (11 female, 9 male), ages 18 to 19, to participate in the study. All the students were part of a program designed to help students succeed in STEM disciplines and were taking an applications of calculus class offered by the program. In order to be placed in the calculus course, the students had to show interest in the program and were evaluated by the program to ensure they had a sufficient background in the mathematics needed for the course. The students selected for our study were chosen specifically for their interest in mathematics. We selected this population because we felt the applications we chose would appeal more strongly to them. Additionally, we needed to use algebra equations for some of the applications, so we chose a population who would have a reasonable understanding of algebra. We could have chosen another population, such as mathematicians. However, they already have a tool set (computer algebra systems, etc.) that allows them to solve challenging and complex mathematical problems. As a result, mathematicians or engineers are arguably a less useful sample because they have a special skill set already. However, students need good tools to support problem solving and assignment completion. We think this is where mathematical expression recognition can have the most benefit.

Nineteen participants self-identified as freshmen, while one participant self-identified as a sophomore. One participant had previously used handwriting recognition software, though no participants had used mathematics handwriting recognition software. The experiment took approximately one hour to complete and each participant was paid ten dollars for their time.

Two people administered the study, a proctor and a wizard. The proctor interacted with the participants, while the wizard sat across the room, controlling the recognition application. Participants were not informed that the wizard was involved in the study. Participants wrote on an HP Compaq tc4400, a 12.1 inch tablet PC running Windows 7 (see Figure 4.13). The recognition application had the look of paper with horizontal lines to help participants write. The wizard controlled the experiment using a Dell Precision T3500 desktop PC with Windows 7 and a pair of monitors. The primary monitor was used for the WOZ Recognizer software which allowed the wizard to control recognition, accuracy, and other parts of the experiment. The wizard also had a live view of the participant's screen through the second monitor. In addition, the proctor had a monitor showing the participant's screen, in order to easily check that the correct expressions were being written by the participant and that all mistakes were corrected before an application was invoked. A video camera recorded their comments and reactions during the experiment.

Section 6.2.2 Experimental Task

Participants performed four separate tasks involving mathematics handwriting recognition: *animate*, *copy*, *graph*, and *solve*. We chose the four applications because we felt they were representative of the tasks that are performed by students in classrooms and can be executed in mathematics recognition systems such as MathPad^2 [39] and MathBrush [35]. The applications represent base and common actions performed on mathematics in education settings. Students graph equations using pen and paper or graphing calculators in classrooms around the world; manipulating and solving an equation for a variable is also an integral part of mathematics. Complicated computer animations and rudimentary hand-drawn animations are shown in physics courses to illustrate problems. When writing papers or turning in homework, students often have to type mathematics expressions using complicated equation editors or inline input languages. These applications are all examples of tasks that students would do in math and physics classes at various levels in learning and in doing homework. By choosing applications that would be familiar and meaningful to students, we hoped to elicit strong feelings about the tasks.

For each task, participants wrote one or two math expressions on the tablet PC at a time. Correction of mistakes (both in recognition and writing) was performed by scribble erasing and rewriting the erased symbols; we decided not to include n-best lists for correction in order to simplify the recognition interface (see Figure 6.1). After correcting all mistakes in recognition and writing, they invoked the application being tested. For the *animate* task, participants also drew a shape using a single stroke before invoking recognition (see Figure 6.2). The stroke was then animated according to the equations written. With the *copy* task, the recognized expression was converted to MathML so that it could be pasted as an equation in Microsoft Word 2007 (see Figure 6.3). Participants had a choice between using a real or a virtual keyboard to paste the expression into Word. Most participants used the virtual keyboard. The *graph* task evaluated and plotted the expression on a 2D Cartesian coordinate system (see Figure 6.4). We chose to plot the expressions on a graph with bounds of positive and negative fifty in each direction because the graphs were hard to differentiate when we varied the bounds to fit the expressions. Lastly, the *solve* task numerically solved the expression for the primary variable and displayed the answers below the expression (see Figure 6.5 for an example). Only solutions in the real numbers were displayed to participants. Streaming recognition was simulated by the wizard, so participants received feedback as they wrote.

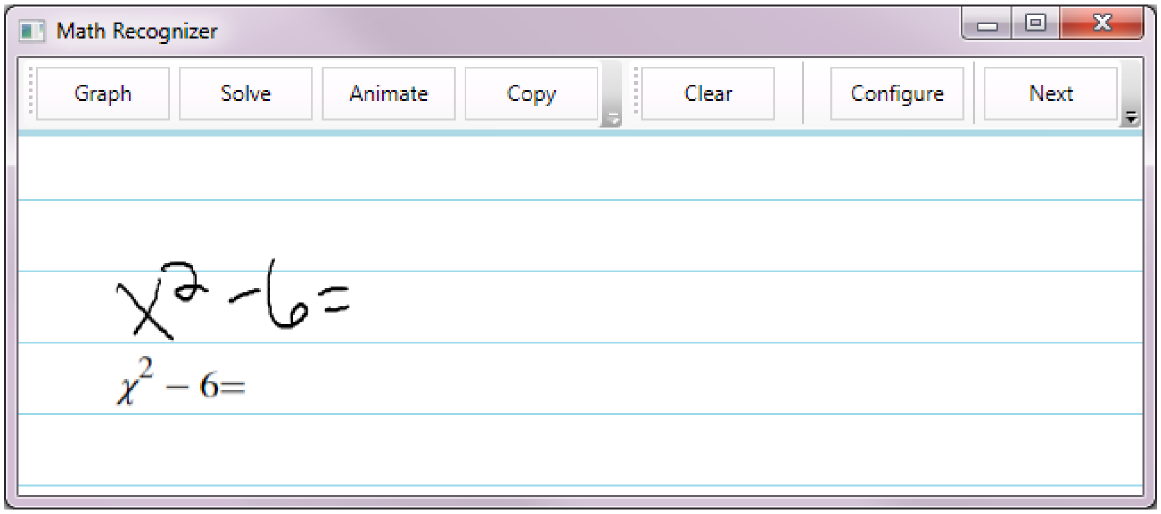


Figure 6.1: The user interface for the participants was kept simple. Buttons are used to invoke the different applications. The writing area is lined to give the look of paper and recognized expressions are displayed just below the participant's writing. In this expression, the recognizer has made an error in recognizing the 'x' as a χ.

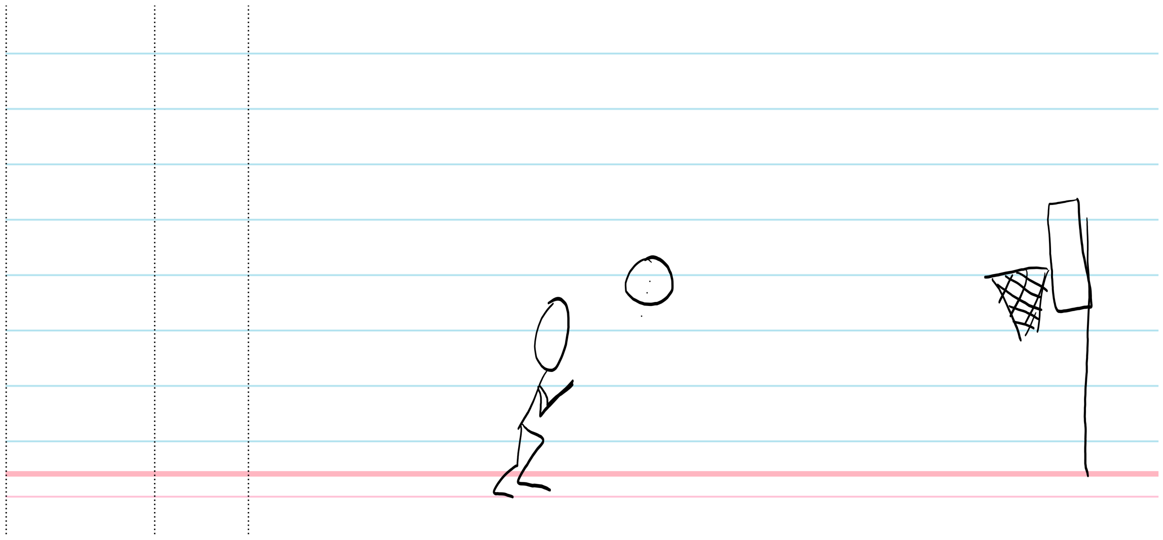


Figure 6.2: An example *animate* task; the ball is being animated. Most participants drew simple shapes rather than complex ones as illustrated here. The writing area is divided into two sections, the top blue lined section for the first expression and the bottom pink lined section for the second expression.

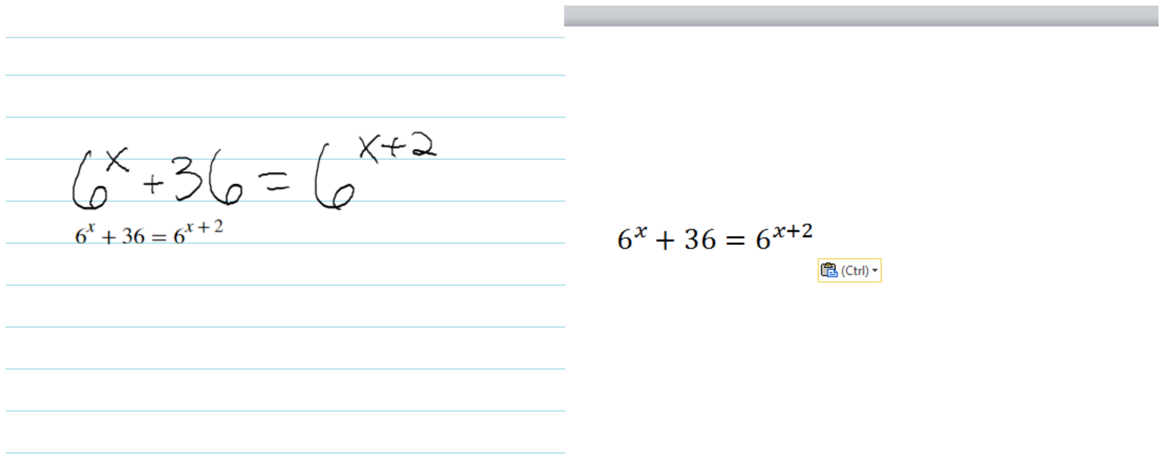


Figure 6.3: An example *copy* task. An expression is shown in the recognition application on the left and pasted into Microsoft Word on the right.

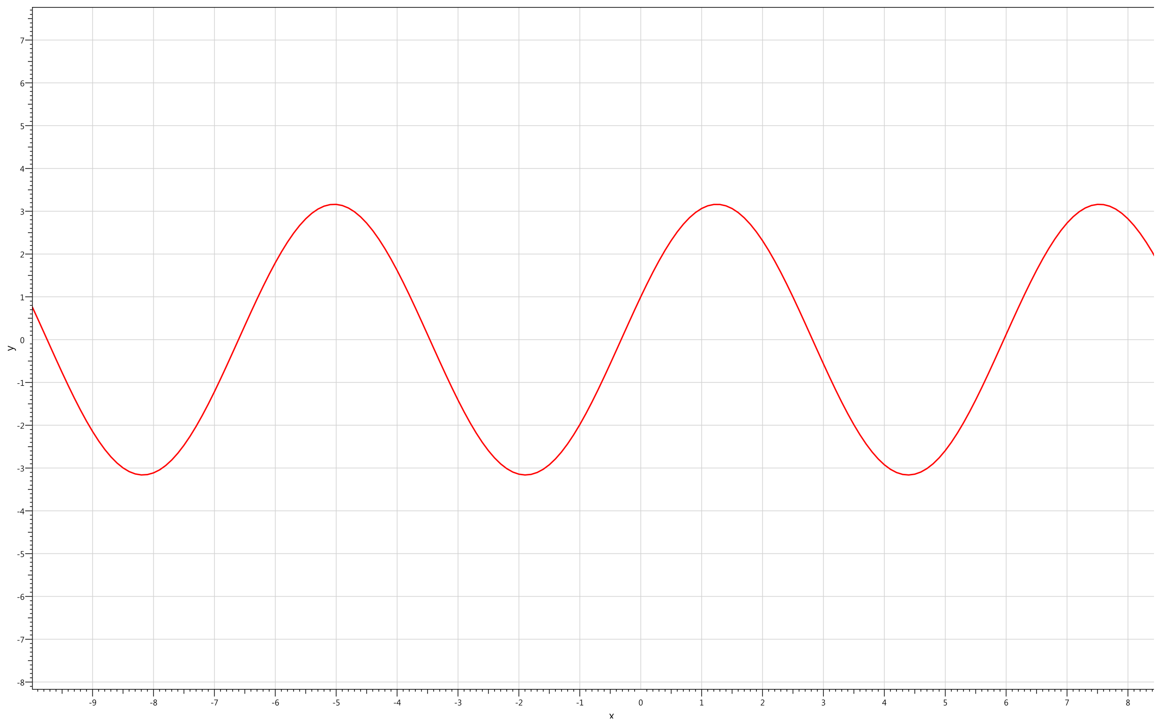
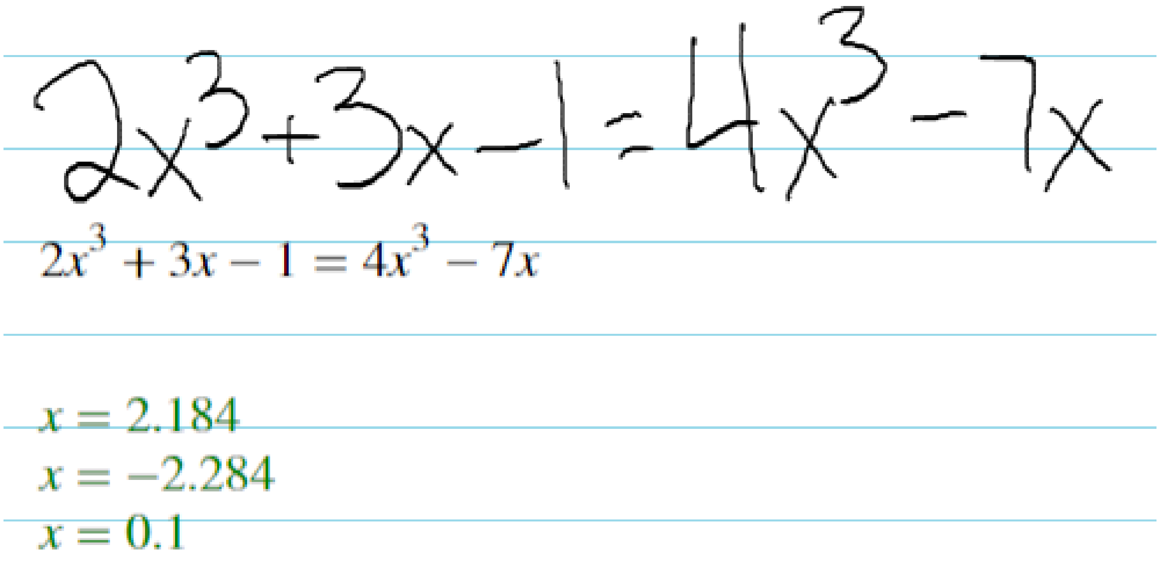


Figure 6.4: An example *graph* task with a trigonometric equation. Participants were able to move around the graph and zoom in and out. Unlike the *animate* and *solve* applications, the *graph* application opens in a separate window.



$2x^3 + 3x - 1 = 4x^3 - 7x$

$x = 2.184$
$x = -2.284$
$x = 0.1$

Figure 6.5: An example *solve* task. Handwriting recognition is shown in black below the handwriting. Solutions to the expression are shown in green below.

We divided the accuracy space into ranges of four percent and a group of expressions was associated with each range for each task, (i.e. all participants experienced the same expressions for the *animate* task at 99-96% accuracy). The accuracy ranges in order were 100%, 99-96%, 95-92%, 91-88%, 87-84%, and 83-80%. For each range, the midpoint was targeted, calculated as the minimum accuracy plus one-half the range, i.e. 98% for the 99-96% range ($96\% + 2\%$). We decided to use accuracy ranges in order to limit the length and number of mathematical expressions participants had to write. We chose to use 4% ranges because it gave a good balance of similarity between accuracies in the range, ease of attaining the accuracy, and quickness of exploring the accuracy space.

Similar to [9], we are using two accuracy metrics, symbol and position accuracy; they both target the same accuracy, but are independently controlled. Symbol accuracy is simply measured by the number of correct symbols divided by the total number of symbols. As with symbol accuracy, position accuracy uses two expressions, the original expression (the intended expression) and the recognized expression. For each intended-recognized symbol pair in the intended and recognized expressions, we find all the positions for the intended symbol and all the positions for the recognized symbol and compare these positions. When a position is missing for the recognized symbol, it counts as an error. The sum of all positions for all of the intended symbols gives the total number of positions in a sketch. The number of correct positions in the recognized expressions is summed and divided by the total number of positions to give us the position accuracy. A group of expressions could have symbol accuracy within the target range, but position accuracy outside the range.

To give a better understanding of what percentage accuracy means, for a set of two expressions used for the animate task, there are 60 symbols and 61 positions. At 98% accuracy, we would expect one symbol error and one position error; changing to the next accuracy range, which targets 94% accuracy, we would expect to see four symbol errors and four position errors. Decreasing to the next accuracy range, which is centered at 90% accuracy, there would be about six symbol errors and six position errors. An expression is shown for three accuracy ranges in Figure 6.6. Equation 1 shows the expression without any errors. Equation 2 has one symbol error and one position error, and Equation 3 has three symbol and three position errors.

For the *copy*, *graph*, and *solve* tasks, recognition accuracy was decreased after every three expressions written. The *animate* task presented expressions in pairs, one controlling the animation's x-axis component, the other the y-axis; after two pairs of expressions (two animations), recognition accuracy was decreased. Each task has its own set of relevant expressions (see Figure 6.7 for example expressions). At the beginning of each task, accuracy was reset to 100%.

$$\frac{8x^3}{11} - \frac{2}{x^4+2x^2} + \frac{1}{x} \tag{6.1}$$

$$\frac{8x^3}{1)} - \frac{2}{x4+2x^2} + \frac{1}{x} \tag{6.2}$$

$$\frac{8x3}{11} - \frac{2}{x4+2\chi 2} + \frac{1}{\chi} \tag{6.3}$$

Figure 6.6: An example expression recognized at the 100% accuracy range (1), the 95-92% accuracy range (2), and the 91-88% accuracy range (3).

During the error correction phase, when a participant erased a symbol and rewrote it to correct a recognition error, n% of the time it corrected, where n is the current accuracy range's midpoint. This had the effect of error correction not working for the participant every time, which simulated the same error occurring again after the participant tried to make a correction.

$$x(t) = 38\cos(\frac{\pi}{4})t \tag{6.4}$$
$$y(t) = 122\sin(\frac{\pi}{4})t - 5t^2$$
$$\frac{8x^3}{11} - \frac{2}{x^4+2x^2} + \frac{1}{x} \tag{6.5}$$
$$y = -4x^2 + x \tag{6.6}$$
$$4^x - 2^{x+1} = 8 \tag{6.7}$$

Figure 6.7: Example expressions used in our experiment. 4.) An expression pair for the *animate* task. 5.) An expression for the *copy* task. 6.) An expression for the *graph* task. 7.) An expression for the *solve* task.

Section 6.2.3 Experimental Design and Procedure

A proctor guided participants throughout the experiment, showing them how to use the recognizer and the applications, giving them questionnaires, and asking questions. Throughout the experiment, participants were videotaped and were asked to think aloud. First, the participants were given a questionnaire in order to collect demographic information, such as their age, gender, and math experience. The proctor then explained and demonstrated the four tasks, starting with the *copy* task, moving to the *graph* task, the *solve* task, and finally the *animate* task. While explaining the tasks, the proctor gave an example of how each application might be used during the participant's school career. For example, *animate* might be useful in a physics class and *copy* might be useful for a math class where homework must be typed. After the demonstrations, the participants were given a questionnaire asking them to rank the applications in order of most (1) to least interesting (4) and in order of most to least useful. The questionnaire also asked the participants to rank their agreement with two statements for each task, "I find the ... task interesting," and "I find the ... task useful" on a seven point Likert scale, where 1 represented Strongly Disagree, 4 was Neutral, and 7 represented Strongly Agree.

After completing the questionnaire, participants were asked to write and correct two mathematical expressions. Once they completed the expressions, the first task was started. The proctor informed participants that recognition accuracy would change throughout the course of each task. Task orderings were randomized and counterbalanced throughout the study. For the *animate* task, since users had to write two expressions, the screen was divide into two sections, marked by different colored lines. The participants were required to write the first equation in the top section and the second in the bottom section (see Figure 6.2). Participants were instructed to copy the expression(s) from a sheet of paper, correct any mistakes in recognition (or what they wrote), and then invoke the application being tested. The proctor pointed out mistakes to correct when necessary. When a participant did invoke an application with errors and the recognized expression did not make sense for the application, an error message was displayed telling the participant to look for and correct any remaining recognition mistakes.

When it was time to change the accuracy, the proctor asked the participant whether the accuracy for the most recent expressions had been acceptable for using the application in a real-world setting like the one described during the proctor's demonstration of the applications. The proctor also discussed the participants answers with them in order to gauge why they did or did not find the accuracy acceptable. Once the number of recognition mistakes was found to be unacceptable, testing for the task was stopped and the next task was started. Finally, when all tasks were completed, the proctor performed a semi-structured interview where they asked the participants whether they felt that how interesting or useful they found each application affected their willingness to tolerate errors while using the applications.

The wizard plays an important role in the experimental task, using the WOZ Recognizer to generate and select recognized expressions and simulate streaming recognition. Just prior to the participant writing an expression, the wizard generates and selects a recognized expression that meets the accuracy requirements and handwriting characteristics of the participant. For instance, when a participant writes "+" symbols such that they resemble "t" symbols, the wizard may choose to generate recognized expressions until one is generated that has a "t" substituted for a "+". In this way, the wizard tries to generate expressions that match the participant's handwriting characteristics. Once the wizard has selected a recognized expression, streaming recognition is simulated by the wizard selecting and sending subparts of the expression as they are written by the participant. When the participant scribble-erases a symbol, the wizard “backs up” and sends an expression without the erased symbols if possible. Details on using the WOZ Recognizer can be found in 4.

Section 6.2.4 Results

It is important for our experiment that participants had different feelings about the four applications. If all four applications have the same interest to a participant, then we could not expect to see any differences in accuracy thresholds. Thus, participants were asked to rank the applications in order of how interesting they found the application and how useful they found the application (see Table 6.1 for rankings on interest and Table 6.2 for rankings on usefulness). We performed a Chi Square test on each set of rankings; for the rankings on usefulness, there was a statistical difference in usefulness among the applications ($\chi^2_3 = 34.80$, $p < 0.0001$). There was also a statistical difference in interest among the applications ($\chi^2_3 = 41.60, p < 0.0001$). This implies that there were some applications that participants found more interesting and more useful than others.

Participants were also asked to rate their agreement with statements about each application using Likert scales. We analyzed this data using the Friedman and Wilcoxon Signed Rank tests [15] and performed a post-hoc correction using Holm’s Bonferroni adjustment [27]. For the statements about interest in the applications, the Friedman test showed significance ($\chi^2_3 = 12.48$, $p < 0.01$); for the statements about the usefulness of the applications, the Friedman test also showed significance ($\chi^2_3 = 15.34$, $p < 0.005$). A summary of the Likert data is shown in Table 6.3. Nineteen of the participants had at least some variation in both their perceptions among the applications; one participant rated all four applications the same on the Likert scales for interest, but had variations among their ratings of usefulness. When we examined if there were applications that participants found more useful than others, two applications stood out, the *solve* and *graph* applications. The *solve* application was significantly more useful than the *animate* application ($Z = -2.968$, $p < 0.008$), as was the *graph* application ($Z = -2.809$, $p < 0.008$). On the contrary, no applications were found to be significantly more interesting than any other.

When a participant finished all four tasks, we asked them whether they thought how interesting or useful they found an application affected their tolerance for recognition errors for that task. Ninety percent (eighteen of twenty) of the participants answered affirmatively. Many participants thought there had been an effect on their tolerance, while a few were not sure whether there had been, but thought that there should have been an effect. One of the two participants who did not think that there had been an effect felt that the application being used was irrelevant to recognition errors, while the other participant replied that they felt the same way about all four applications, so there was no difference in tolerance due to interest in an application.

Table 6.1: Total number of rankings of *interest* for each position for each application. The *copy* application was found to be least interesting.

Ranking	Animate	Copy	Graph	Solve
1st	9	0	5	6
2nd	2	4	12	2
3rd	2	6	2	10
4th	7	10	1	2
Median	2	3.5	2	3

Table 6.2: Total number of rankings of *usefulness* for each position for each application. The *copy* application was also found to be least useful.

Ranking	Animate	Copy	Graph	Solve
1st	4	1	6	9
2nd	3	1	11	5
3rd	5	8	2	5
4th	8	10	1	1
Median	3	3.5	2	2

Table 6.3: Responses to the Likert statements. A larger number represents greater agreement with the statements about the application.

	Interest		Usefulness	
Application	Mean	Std. Dev.	Mean	Std. Dev.
Animate	6.15	1.531	4.85	1.725
Copy	4.75	1.713	5.65	1.565
Graph	5.70	1.342	6.45	0.887
Solve	5.80	0.951	6.55	0.686

To analyze participants' tolerance for recognition mistakes, we looked at the number of accuracy ranges a participant experienced before they stated that the accuracy was unacceptable. With this data, we performed the Friedman test twice (see Tables 6.5 and 6.6 for summaries). We sorted the data by the participant's ranking of interest in the applications (sorted from most to least interesting) and by the ranking of usefulness for the applications (also from most to least useful) and did the Friedman test for each sorting. The tests showed no statistical differences amongst the four applications; that is, participants did not tolerate more recognition errors for applications they thought were more useful ($\chi^2_3 = 2.21$, $p = .530$) or interesting ($\chi^2_3 = 4.62$, $p = .202$).

We also recorded the amount of time it took a participant to write an equation, correct it, and press the button to invoke the appropriate application. The timing data is a measurement of the time to write and correct an expression, and other than the variations in the expressions used for the different tasks, the task itself plays no part in the timing data. The timing data shows that there was a real effect of recognition errors in terms of the time it took participants to obtain a correct recognized expression. As accuracy decreased, the time to obtain a correct recognition increased, as we would expect. The mean time per accuracy range for each application is shown in Table 6.7.

Table 6.4: Average lowest tolerable accuracy for each application in percentiles.

	Animate	Copy	Graph	Solve
Mean	92.78	90.89	92.28	89.51
Std. Dev.	5.56	4.62	4.32	4.91

Table 6.5: The mean number of ranges each participant was willing to tolerate. Each participant's accuracy tolerance is grouped together according to the rankings of their interest in the applications from most to least interesting.

Ranking	Mean	Std. Dev.
1st	3.55	1.099
2nd	3.25	1.020
3rd	3.65	1.137
4th	3.60	1.142

Table 6.6: The mean number of ranges each participant was willing to tolerate. Each participant's accuracy tolerance is grouped together according to the rankings of how useful they found the applications from most to least useful.

Ranking	Mean	Std. Dev.
1st	3.30	1.031
2nd	3.55	1.317
3rd	3.70	1.129
4th	3.50	0.946

Table 6.7: Mean time in seconds to write an expression, correct any recognition mistakes, and invoke the application. Times are shown for each accuracy range and application.

	Animate		Copy		Graph		Solve		Overall	
Range	Mean	Std. Dev.	Mean	Std. Dev	Mean	Std. Dev	Mean	Std. Dev.	Mean	Std. Dev.
1 (100%)	22.08	7.08	20.27	11.02	20.75	10.47	21.60	9.61	21.09	9.82
2 (99% - 96%)	19.88	8.34	32.13	21.45	20.95	7.62	18.93	8.10	23.26	14.18
3 (95% - 92%)	23.58	7.49	31.81	14.08	28.24	13.05	22.42	9.82	26.84	12.26
4 (91% - 88%)	32.77	12.42	33.41	13.21	38.71	12.91	34.52	12.92	35.03	13.02
5 (87% - 84%)	42.52	18.27	39.55	22.90	30.12	11.31	30.24	12.46	35.60	18.00
6 (83% - 80%)	33.02	14.21	33.74	11.74	53.03	26.43	31.75	8.99	48.93	16.83

CHAPTER 7: DISCUSSION AND FUTURE WORK

Section 7.1 Discussion

In the study described in Chapter 3, despite the absence of recognition errors, participants did not rate/rank the Wizard of Oz sketch interface as the best interface for all criteria. This shows that there are areas where pure sketch interfaces can be improved. For instance, some symbols are hard to write or take a long time to write. Other interfaces (such as menus, toolbars, or gestures) can provide better ease of use for such situations, because the task complexity is reduced. Some tasks, like organization and layout are problematic for sketch-based interfaces the closer they are to pen and paper. Providing a form of transformative feedback might improve sketch-based interfaces, such as cleaning up strokes, aligning structures, spacing symbols (such as in [61]), or providing widgets that can be moved around.

Perhaps from a performance point of view, pure sketch-based interfaces are not always the best interface for a structured 2D language. However, from a preference point of view, there is still a powerful aesthetic that makes them useful even when they are not the most efficient.

Since many of the problems participants had with the sketch interface dealt with spatial arrangement, sketch interfaces will have to provide tools to help users in this regard. For instance, when sketching logic diagrams, drawing many intersecting wires can be tricky, particularly when you are going back to make sure you drew correctly. Visualizations of wire intersections could make complex diagram creation easier for users. In addition, by utilizing domain knowledge, sketch interfaces could help spatial arrangement by inferring how wires should be connected. One way in which we might further explore these interfaces is examining how often users make errors with the different types of interfaces; perhaps naturalness will lead to fewer errors and a greater feeling of satisfaction. Another research avenue to explore is looking at another type of diagram creation task (i.e. synthesis), where the interfaces might perform differently in terms of ease of use, speed, and naturalness.

Another question raised by the LogicPad study in Chapter 3 relates to the drag-and-drop interface. A stylus is not necessarily the best device for pointing tasks, like those used in the drag-and-drop interface [44]. How then, do the sketch and hybrid interfaces compared to the drag-and-drop interface when it is used in a desktop setting?

After performing this study, we saw that further research into understanding user perceptions of pen-based interfaces was needed; we were intrigued by the differences we saw between our participants' perceptions and our measurements. Our work with LogicPad inspired us to develop a Wizard of Oz system for studying user perceptions of sketch-based interfaces, which would eventually become the WOZ Recognizer described in Chapter 4. Initially, we decided to tackle a similar problem domain to diagrams, mathematics, and focus on simulating recognition with controlled accuracy so that we could study the effects of accuracy on user perceptions.

The systematic study of mathematics-based sketches and recognition required to create and design the WOZ Recognizer allowed us to model sketches and the recognition process. From our work with the WOZ Recognizer, we feel that all recognition errors are not equivalent to a user; the greater the difference from the expected result, the more we expect that an error will bother a user. An interesting line of investigation would be to examine whether certain types of errors are linked together.

As discussed in Chapters 5 and 6, two studies have used the WOZ Recognizer to explore user perceptions of mathematics handwriting recognition. In these studies, controlled mathematics accuracy was simulated for a series of copying tasks. Feedback from these studies has helped us to refine the WOZ Recognizer, particularly in streamlining the Wizard Interface and designing user interface components that enable the wizard to quickly generate WOZ Recognizer's sketches and respond to variations in user input. When we began to use the WOZ Recognizer for our first study, we quickly realized that there was variation in the order people wrote mathematics structures as discussed in Section 4.4.2. In order to accommodate these variations, we implemented a user interface component where the wizard can choose the order in which the Streaming Selector traverses the parse tree. The WOZ Recognizer had a simpler UI during the first study, so viewing the accuracy information was easy. As we expanded its capabilities for the second study, we realized that the wizard would often have a different tab open, so they would not be able to quickly see the accuracy information. We addressed this problem with the introduction of our notification system, where we have UI components that act like lights that turn on when something important happens. After each study, we expanded the configurability of the recognition simulation of the WOZ Recognizer, so that we can better control where and when errors are placed. While the first study showed us that always re-simulating recognition of the entire mathematics expression made streaming recognition difficult when the user's handwriting didn't match the WOZ Recognizer's sketch, the second study forced us to solve the problem, since it was exclusively streaming recognition. We introduced the ability to alter specific errors to address the issue.

As described in Chapter 4, we performed an evaluative study of the WOZ Recognizer using experts on sketch-based recognition and interfaces, which happened after refining the WOZ Recognizer through the two studies described in Chapters 5 and 6. We feel that the users were receptive to the WOZ Recognizer and were able to successfully learn to use it in a short time. We think the WOZ Recognizer is useful for studying how people interact with sketch-based systems including exploring their perceptions of how recognizers behave. It allows us to study and understand pen-based interfaces for sketching, by controlling recognition aspects and giving a developer the ability to create interfaces where recognition is controlled. By using the WOZ Recognizer in such a way, a researcher or developer can examine or study sketch-based interface with regards to recognition accuracy, which we think is important. The WOZ Recognizer helps to explore complex sketch domains where the pen-based input is the primary focus of interaction, which is most similar to pen and paper interaction. We think that these kinds of interfaces are useful and should be explored rigorously. The WOZ Recognizer is also useful in designing recognizers, because it can help to provide lower bounds on accuracy which can be important when thinking about trade-offs between real time interactivity compared to the amount of computation.

In addition to applications of the WOZ Recognizer to academic research, we believe that the WOZ Recognizer would be useful for recognizer and application designers. One important question that we believe the WOZ Recognizer would be useful in answering is what “good enough” recognition is. Researchers can use the WOZ Recognizer to examine user acceptance of recognition at various accuracy rates and conditions, and see how this is affected by other recognition properties.

When a designer is thinking about supporting a particular task for some recognition domain, they can use the WOZ Recognizer to see how users respond to the task and the kinds of recognition errors they experience, without having to implement an entire recognizer. We think that the WOZ Recognizer could be useful in the early prototyping stages of the UI design process to get a feel for how the interface is working in connection with recognition. It can be done iteratively as a way of testing out different design decisions in terms of the various recognition strategies the designers want to use.

In the preliminary work for our studies using the WOZ Recognizer, we noticed that when working with mathematics-based sketches, while users may write out of the expected order, we found that this problem could generally be solved by delaying sending the WOZ Recognizer's sketch to the user until they have reached a point in which the n^{th} parse tree matches what has actually been written. This is not truly real-time, but still gives streaming recognition. This delayed streaming recognition also can occur when a user is writing too fast for the wizard to keep up, when multiple symbols are written in a single stroke, and when users frequently change the order in which they write structures.

Latency in simulating recognition is important and directly related to the wizard, as they control when feedback is sent to the Participant's Interface. We believe that we can reduce the latency in the Participant's Interface receiving feedback through some user interface changes. When streaming, a button press (or key press) is required for each symbol, including implied symbols like multiplication or superscripts. The experts in our case study did not like this and had a hard time adjusting to it. We could skip over these implied symbols to reduce latency from user confusion and from having to perform two actions. Sometimes the wizard will fall behind the user's writing and batch multiple symbols into a single send. While this isn't necessarily noticeable for a copy task, it does have the effect of increasing latency. Subjectively, we think that the participants in our case study were close to achieving sufficient latency for a copy task, as users will typically be focusing on a piece of paper or their own writing. Users will notice when latency is extremely large, i.e. nothing was been recognized, or once they have finished copying, as their attention is no longer split between three things (reading the sketch, writing the sketch, reading the recognizer's output). However, when they are copying, splitting their attention between multiple things, they will be more tolerant of latency.

In the study presented in Chapter 5, it was clear that recognition accuracy had little impact on user preference for recognition mode, as preference did not generally vary with accuracy. Often, participants expressed that batch recognition was better for single expressions, but that streaming was preferable for multiple expressions. One streaming recognition aspect that participants liked was its immediacy; it gave them immediate feedback and they were able to immediately correct and adapt their writing styles. For tasks at 90% recognition accuracy, some participants felt that batch mode was tedious, as there were many errors and it was hard to remember which errors they had corrected before hitting the recognize button. This was especially true with the multiple equation set, since there were more errors to correct, which forced participants to spend a long period performing error correction. Additionally, participants stated that correcting all their mistakes at once was time consuming. Unfortunately, we did not time how long it took each participant to write and correct the expressions, so we cannot verify or refute this perception. This brings us back to an inherent issue with batch recognition; it requires a period of intense visual identification of errors. We think that this explains why there was a stronger preference for streaming recognition for the low-accuracy multiple equation task compared to the higher accuracy tasks.

Participants often stated that they wanted to spend time writing expressions; interestingly, this was often used as a reason for preferring both batch and streaming recognition. We think that differences in what distracted participants explains this contradiction. Participants who were distracted by recognition errors as they wrote would find it easier and faster to write and then recognize; participants who were not distracted would not have to sit through an error correction cycle of finding and rewriting incorrectly recognized symbols. Since participants reported lower distraction levels using batch recognition over streaming recognition, we can speculate that distraction played little part in user preference for recognition mode. Additionally, there may have been differences in writing cycle perceptions; some participants may have included error correction as part of writing, while others did not. Those who viewed error correction as separate would likely view any period of solely correcting errors as "not writing."

In general, participants expressed a desire for faster recognition and correction; that is, they wanted to see recognized expressions immediately and fix mistakes immediately. During the final interview with the participants, we asked them whether they would prefer a version of batch recognition where they could press the recognize button after each expression or correction, over the batch recognition performed in the study. This alternate batch recognition would give more immediate recognized expressions and the ability for users to see error correction results almost immediately. Thirteen participants expressed that they would like that version of batch, ten participants stated that they preferred the implemented version, and one stated that it was situation dependent. We think that this is related to the change in preferences with expression set size. Batch delayed feedback until all expressions had been written. We believe that the delay between writing and feedback was too long. Having the participants recognition each expression after writing might see similar results to single expression tasks.

Participants consistently underestimated the position and symbol recognition accuracy for each task. The perceived accuracies were fairly consistently 5% or greater in error. What is most interesting is that participants had a greater underestimation of accuracies for the multiple equation tasks at low recognition accuracies. Additionally, participants exhibited a greater variation in perceived recognition accuracy for the multiple equation set than the single equation set (and variation decreased as the real accuracy increased). One explanation might be that participants saw several expressions with errors and viewed all the errors as effecting one expression; participants may have also had a harder time evaluating the number of symbols and positions for multiple expressions at once.

Although it is not the primary task that most users will perform while using mathematical handwriting recognition software, we chose to have participants perform a copying task during the study, as we felt that it was a representative of one type of task that users perform in real-world situations (such as in educational settings). For instance, students will copy down equations while doing their homework and teachers might copy them down while creating a test. Additionally, using a copying task allowed us to control the experiment and what participants wrote.

We feel that knowing user perceptions about the two recognition modes are more important for user interface design than an objective quantitative measurement of the modes, such as user speed. Consequently, a subjective quantitative experiment was performed; objectively measuring those perceptions would have greatly increased the setup's complexity and the time required of each participant.

In Chapter 6, we saw that participants in the study thought that they tolerated more recognition errors for applications for which they had a preference. However, most participants did not tolerate more recognition errors for their most useful or most interesting applications. Along with a yes or no answer, participants were asked to give reasons why they were or were not willing to tolerate the level of errors they had just experienced. One of the most common explanations for a negative response was that there were too many errors to correct. Participants did not want to spend the time to correct so many errors. Some participants also stated that using another method for performing the task, such as a calculator for graphing, would be quicker, so there was no reason for using the handwriting recognition application. Another common reason for no longer tolerating the number of recognition errors was frustration. Common stated sources of frustration were repeated problems recognizing certain symbols, a large number of errors, and the amount of time it took to find and correct all the errors. Working on the copy task, one participant stated "I think that would be about my tolerance because it just kept making more errors. Every time I drew a two, it would make a 'd' or some sort of symbol," and that they would "probably just write it down instead of using this to copy it." In this case, the participant was unfamiliar with the symbol the recognizer was giving, which caused some confusion and increased frustration. A common positive response was that it was easy to use the handwriting recognition, and there were few and easy to correct errors; for instance one participant stated "you can change it pretty quickly if it's just one [error]." A common response from participants was that they were becoming increasingly frustrated with the recognition errors, but that they were not a large enough problem yet.

Why did participants think they tolerated more recognition errors for the "quality" applications? It is possible that time to completion played a role in how long participants tolerated low accuracy results. Perhaps participants ascribed good attributes to the applications that they considered to be good? We theorize that participants had intolerance for recognition errors mitigated by the continuous feedback provided by the system in the form of ink strokes and the typeset recognition.

For some participants, time was a factor in their decision to tolerate recognition errors. For instance, one participant felt they needed to write slower in order to write neat enough for the recognizer to understand their handwriting, which caused them some annoyance. For other participants, the time to find and correct errors was important. A large number of errors, and having to make repeated attempts to correct errors meant it took a long time to be able to invoke the application, and participants were often frustrated by how long they felt it took. For instance, one participant stated “it’s kind of starting to get more tedious with writing them, because [the recognizer] keeps doing things wrong.” In their decisions to tolerate a level of recognition accuracy, some participants explicitly compared the time to perform the task using another method to the time to the handwriting recognition application. They were willing to tolerate errors so long as the task time was in the handwriting recognition application’s favor.

In speaking with the participants after they completed all four tasks, many participants compared using a handwriting recognition application to another application for the same purpose as a factor in deciding to tolerate recognition errors. When performing a task with another application or tool was easy (graphing with a calculator), participants were less willing to tolerate recognition errors. In cases like solving, most participants were not familiar with other tools beyond pen and paper, and felt that the *solve* task was thus highly useful and easy to use. Since solving an expression using pen and paper also involves writing the expression, usually multiple times and through several steps, going through error correction was viewed as easier and quicker to many participants. The time to correct errors was often mentioned; rather than talking about the overall time to write and correct, participants tended to talk about the time to correct. The time to correct was important to them; having easy correction would greatly help participants feel using mathematics handwriting recognition was viable.

Examining the quantitative data from the two participants who did not feel as though their interest in a task affected their willingness to tolerate recognition errors, we found that they tolerated less errors from the tasks they were interested in. One participant ranked the *animate* task their most interesting and most useful task. The participant tolerated one less range for the *animate* task than for the other tasks. The other participant tolerated one less range for their two most useful tasks, but did not have the same effect for their most interesting tasks.

One participant regularly had to type expressions for entry into a web-based homework system and found the experience of typing expressions very frustrating, because the system required a very specific entry format. For expressions with multiple parsings in the entry language, the participant might enter the "wrong" parsing and be marked incorrect. The participant thought that mathematics handwriting recognition would be useful for such systems since it was so tedious typing the expressions and recognition errors were relatively obvious and easy to correct.

What role did feedback play in the study? An application's quality is a function of its feedback. Better feedback, whether that be faster, more easy to understand, more useful, or nicer looking, decides how we view an application. In that sense, the usefulness or interestingness of each application was a measurement of the extra feedback provided by the application.

We wonder how interactivity plays in a role in how participants tolerated recognition errors. Were the more interactive applications better tolerated? The visual applications, *animate* and *graph*, provided a level of entertainment to the participants that made them feel as though they tolerated more errors. One participant said that it was worth putting in the extra time correcting mistakes to see what happens with animation and graphing. Another participant commented that even though the *animate* task had several errors in recognition, it was still interesting to watch the shape "fly around," and that the application would keep people wanting to use it just to watch the animations.

We think that our results inform the debate on the speed versus accuracy tradeoff in recognizer design. Our experiment shows recognizer designers just how important people find error correction. People want easy and quick error correction; in some ways, correction is a replacement for accuracy. Participants were fairly tolerant to recognition errors, as the average accuracy for the last range that participants found acceptable was 91.36% ($\sigma = 4.99$). As previously discussed, some participants commented during the experiment that the accuracy was no longer tolerable because of the long time to correct. The time to correct is directly related to recognition accuracy in two ways, a greater number of errors and the failure of an error to be corrected. That is, lower accuracy led to an increase in errors and a greater number of corrections per error; having to correct the same mistakes repeatedly would likely cause a lot of frustration for the participants in addition to the time sink. The additional time to correct errors was fairly significant; changing from the first (where no correction was necessary) to the sixth accuracy range caused a greater than 50% increase in the time to write and correct the expression, and invoke the application (see Table 6.7).

In [20] and [36], a low tolerance for recognition errors in normal handwriting recognition was found. In [36], 97% accuracy was the minimum accuracy deemed acceptable. In contrast, we found a significantly higher tolerance for recognition errors with mathematics handwriting recognition. We think this is at least partially attributable to the amount of writing involved; mathematics expressions are relatively short in comparison to writing a paragraph. We expect that similar tolerance for recognition errors would be found for regular writing with short entries, such as names or dates.

Frankish et al. [20] also suggest that recognizer performance plays a role in how well-liked handwriting recognition applications are and that the interest level in an application also affects its success. Prior to executing our experiment, we hypothesized that mathematics recognition would follow along these lines, that participants would tolerate more recognition mistakes for applications they found more interesting or useful; while participants felt that they did tolerate more recognition mistakes for the applications they liked, the quantitative data does not support this finding.

We propose several factors that might contribute to this disconnect. For one, participants were not made explicitly aware of how many accuracy ranges they experienced. While participants were asked how they felt about the accuracy after each range, they were not directly told how much accuracy had changed or what the recognition accuracy was for the previous expressions. So while participants might have done the same number of ranges for two tasks, they might not have realized that they were experiencing very similar accuracies. Also, they were not likely to be keeping track of how many ranges they experienced for each task, so when they thought about how the applications affected their tolerance for recognition mistakes, it is unlikely they correctly estimated how many ranges they experienced for each task. In [9], participants consistently misjudged recognition accuracy for positions and symbols. As recognition accuracy decreased, the amount they misestimated increased, so a 1% change in recognition accuracy might contribute to a greater perception of mistakes. This disconnect between perception of accuracy and the computed accuracy might cause participants to think that they experienced quite different accuracies between tasks. In addition, different error types are likely to give different feelings; a change from a superscript to a regular position is likely less frustrating and annoying than a change from outside a root to under a root. People may feel that the "more annoying" errors have a greater effect in terms of accuracy.

Finally, although we found significance in differences between applications using the participants' responses to the Likert statements (usefulness and interest), we think that the differences between the means are still somewhat small. We think that if there was a larger contrast between some of the applications (for instance an average of six for *animate* and two for *copy*), we might potentially see a starker contrast in accuracy tolerance between the applications.

Section 7.1.1 Feedback Drives Perceptions

One powerful insight we gleaned from our studies is how application feedback drives user perceptions of the application. Applications communicate to us through feedback; we interpret the feedback in various ways and form an understanding of what the application is communicating. While most applications provide some visual feedback when we provide them input, we think that seeing a computer display our handwriting gives us a form of feedback that we relate to more closely. Compared to a keyboard, we are directly and dextrously manipulating an object when we write. The act of writing takes us closer to our application and makes us feel that the application understands us in a more human way. The constant communication that we get simply from writing lessens the important of the other forms of feedback we receive (although it does not eliminate the need for it). We hypothesize that the feedback provided by writing lessens the importance of accuracy compared to other input methodologies. By the time we receive the forms of feedback that differentiate the applications, we have already dealt with the communication issues. When the feedback cycle is more tightly integrated between input and application purpose, we might expect to see a greater effect on error tolerance.

The feedback cycle from writing and seeing our strokes displayed by the application allows us to delay other forms of recognition feedback. We expect that this will be most powerful when we have high input frequency (low delays between inputting sketch pieces) as we will naturally stay focused on a single cognitive task. Once the user has completed a cognitive task, they will want to receive feedback before continuing on to another task. With the study presented in Chapter 5, the user had to switch cognitive tasks between each expression in the multiple expression sketches; the form of batch recognition we used causes the user to have a context switch where they receive no feedback from the application. A lack of timely feedback can make us feel as if the application isn't listening, even if we haven't asked the application for a response. When this happens, we cannot tell if the application understands what we are telling it. So long as the application's response that it understands us (sketch recognition) comes in a reasonable timeframe, our applications can provide delayed feedback.

Earlier, we hypothesized that writing gives users different feelings about communicating than other forms of computer input such as typing. With our study presented in Chapter 3, all three interfaces provided participants with constant visual feedback. Yet, many participants felt like they completed logic gate diagrams faster with the Sketch interface. We hypothesize that this is due to the difference in how we receive feedback from writing in comparison to other input methods. Another hypothesis is that the Hybrid interface introduces delays in feedback that weren't present in the Sketch interface. For instance, the Hybrid interface used a marking menu where the participant had to hold the stylus in a single location for a bit of time before the menu opened. This introduces a delay in the feedback received by the user, where they have no indication that their communication with the computer is understood. The user has communicated with the computer, but are forced to wait for acknowledgement and a response before they can continue communication. In contrast, the Sketch interface only provided immediate feedback until recognition was invoked. Finally, we hypothesize that the context switching required by the user to provide different forms of input when using the Hybrid interface contributes to the perceptions of speed.

The harder it is for a user to find a recognition error, the more important recognition accuracy becomes. When the user is primed to look for recognition errors, the presence of errors becomes less important. When the burden of finding and correcting errors becomes large compared to the task of inputting, the user is unlikely to accept the recognition accuracy. If a recognizer has an estimation of recognition confidence, instances of low confidence should be expressed to the user through visual feedback. This visual feedback will help the user to more easily check the recognition and also feel more confident about the recognition process.

Our hands are highly responsive, able to nimbly manipulate, and have many nerves. Do we receive more sensory input from writing than typing? Does writing take more brain power than typing? If so, do these nervous system traits cause us to treat actions done with greater dexterity and control differently than simpler actions? We also receive large amounts of haptic feedback from our hands. Do we get more or getter feedback from writing than typing?

Section 7.1.2 Mathematics and Diagrams

We believe that our results are not only applicable to mathematics, but to other structured input languages like sketches and diagrams. Our belief is that the complexity in these different domains will provide similar reactions to errors in recognition. The two-dimensional relationships between symbols in sketches and diagrams provide a structure that is similar to the structure in mathematics; while the structure in mathematics is quite complex, we think that the general complexity of the domains is enough for our study's results to apply. We think that people would find the same tediousness in correcting sketch recognition errors as in correcting mathematics errors, as the errors often involve some mental processing of just what went wrong during recognition. We think that people may also be more cognizant of errors in writing than in shapes or other symbols, as we deal with letters and words on a daily basis. In that respect, mathematics handwriting recognition represents the extreme end of sketch and handwriting recognition, being both familiar and complex.

Section 7.2 Future Work

Although we have been able to successfully use the WOZ Recognizer to explore user perceptions of sketch recognition, there still remains work to do to improve the software. It supports simulation of copying tasks, where the user is writing sketches that are displayed in front of them. In order to support a thinking task, where the wizard would not know the sketches the user writes beforehand, we would like to develop the WOZ Recognizer to be able to generate realistic WOZ Recognizer's sketches on a piece-by-piece basis. As the user writes, the wizard would input the sketch into the system, which would generate recognition errors that persist as more symbols are added. Also, we are interested in applying graph layout algorithms to graph-based diagrams to support non-copying tasks. We plan to include a mechanism for wizard-defined error grammars in the future. We are also exploring the possibility of expanding the WOZ Recognizer to support other domains and areas besides mathematics and diagram-based diagrams. One avenue for future works is expanding the graph backend to construct diagrams and sketches that do not have explicit edges. One aspect of probabilistic errors that we have not implemented in the WOZ Recognizer is that some symbols are more likely to generate errors than others. In the WOZ Recognizer, all symbols in the error alphabet are equally like to generate symbol errors.

Currently, should a user draw something unexpected (like a doodle), the wizard has no way to respond with anything but the pre-determined output. We are considering giving the wizard an interface for choosing their own output so that something can be sent in such circumstances.

We have considered expanding the WOZ Recognizer to other types of input, like gestures. We could take a simplified approach to recognition with gestures, since multiple gestures do not typically have spatial relationships between them, and only focus on the error alphabet part of simulating recognition errors. Perhaps multi-touch gestures would provide additional complexities to simulate.

Previous work [45], has examined the viability of touch-based tablets for sketch recognition. We wish to take this further and examine whether, for instance, users are more tolerant of recognition errors when using a touch-based device, rather than a stylus-based device.

Based on feedback from our case study evaluating the WOZ Recognizer, we plan to make several changes to the WOZ Recognizer. First, the user interface for mathematics-based sketches places the controls for accuracy (and the widgets that display the per-sketch and completed sketch accuracies) and the controls for streaming recognition on separate tabs. We will consolidate the user interface to place the accuracy controls where they will always be usable without changing tabs. Second, we plan to introduce a mechanic where the system knows how many sketches are in a set and can determine the total number of errors allowable for the set. This will help the wizard to monitor and control accuracy and prevent early sketches from having too many recognition errors. Third, we plan to improve the notifications in the WOZ Recognizer to help the wizard to monitor recognition accuracy at a glance. Fourth, we plan to move the various aspects of the UI so that the wizard does not have to change their focus as much, including moving the typeset version of the WOZ Recognizer's sketch under the user's ink and swapping the tree view locations of the Participant's sketch and the WOZ Recognizer's sketch. Fifth, we plan to improve the area that displays the user's ink so that the wizard does not have to manually scroll to see if the user wrote off the screen. Sixth, we would like to improve streaming simulation by skipping invisible symbols (like implied multiplication) so that the wizard has an easier and quicker time selecting the correct piece of the sketch. Seventh, with regard to diagrams, we plan to improve the mechanisms for selecting symbols in the sketch since it can be hard to select small, thin lines using the mouse. Finally, we plan to implement some changes to improve the wizard's ability to tell when strokes have been erased and rewritten, and to improve their ability to identify what parts of the sketch have been sent during streaming recognition.

In addition to the structure domains we currently support, we think that gestures, in particular multi-touch gestures would be a good fit for a Wizard of Oz toolkit like the WOZ Recognizer. Multi-touch gestures present their own interesting challenges in terms of complexity; multiple strokes occur simultaneously and multiple gestures may be occurring at once. While gestures and sketch recognition can use batch recognition, and sketch recognition may occur in a streaming manner, processing multi-touch input can be continuous, requiring constant processing, classification, and feedback. By providing a Wizard of Oz system for multi-touch gesture recognition, we can better explore the limitations and preferences in gestural interfaces.

In Chapter 5, participants engaged in a copying task, which is not the most common mathematics task, but still an important one. A thinking task where the participants are doing something with the mathematics is an important area that we need to explore, especially in intelligent tutoring applications; people's preferences may change when the task changes. One way in which we might simulate a thinking task is by combining a copying task and a distraction task.

For our study in Chapter 6, we tried to keep position and symbol accuracy in the same range for a group of equations. We want to explore how users react to low accuracy of one type and not the other; intuitively, we think that users will be more bothered by position errors than by symbol errors and we wish to see if our hypothesis proves true. Relatedly, we think that some categories errors may cause greater amounts of frustration and a greater perception of errors than others. Thus, we want to be able to perform an experiment on how a category of error affects accuracy tolerance (for instance, capitalization errors). Another area of interest is exploring how participants respond to an application that has very low interest to them; we think that they might have significantly decreased tolerance for an application that they find uninteresting and useless. Finally, we want to see whether participants' feelings about the applications changed as they used them over the course of the experiment.

With regards to recognition feedback, we wonder how assembly-line type tasks will work with sketch recognition. From our work with recognition feedback, it seems as though users aren't comfortable with writing everything and then going back for an error correction phase. Do we need to take task breaks so often that such an assembly-line style of input is unsuitable?

CHAPTER 8: CONCLUSION

In this dissertation, we first presented our work with pen-based logic gate diagram interfaces, which led to the creation of our Wizard of Oz system for studying user perceptions of sketch-based interfaces. Next, we presented the WOZ Recognizer, a Wizard of Oz system for simulating sketch recognition and an evaluative user study of the system. Through two user studies, we have explored user perceptions of sketch recognition. While we have initially explored mathematics-based sketching, we plan to expand into other types of sketches in our future work. In our first study, we examined whether there is a link between recognition mode, recognition accuracy, and number of sketches. We found that users preferred real-time recognition for a number of situations, but had no clear preference for others. In our second study, we examined whether there is a link between recognition accuracy and application; we posited that users would be more tolerant of recognition mistakes when they used a more interesting or useful mathematical sketching application. Our results show that users felt they were more tolerant for the applications they liked the most. Finally, we presented a discussion of our findings while working with the WOZ Recognizers and ideas for future research into user perceptions of sketch recognition systems. Our goal in creating the WOZ Recognizer was to develop a tool that would facilitate research into sketch recognition systems, where recognition parameters are controlled and the research is methodical. We believe the WOZ Recognizer and the research we performed provide valuable contributions to our knowledge of sketch systems and provide an avenue for future research.

APPENDIX A: MATHEMATICS EXPRESSIONS

$$[t!]x_\alpha = \sqrt{\beta + 137.2} \tag{A.1}$$

$$\mathrm{Pb(OH)_4 + 2H_2SO_4 \rightarrow Pb(SO_4)_2 + 2H_2O} \tag{A.2}$$

$$\frac{2x}{x^2-1} = \frac{1}{x-1} + \frac{1}{x+1} \tag{A.3}$$

$$\mathrm{Fe + 2H^+ \rightarrow H_2 + Fe_2^+} \tag{A.4}$$

$$y = 4\sqrt{7} + \frac{x}{2} \tag{A.5}$$

$$y^3 + 3py + q = 0 \tag{A.6}$$

$$a^4 + b^4 = (a^2 + \sqrt{2}ab + b^2)(a^2 - \sqrt{2}ab + b^2) \tag{A.7}$$

$$\mathrm{S_8 + 24F_2 \rightarrow 8SF_6} \tag{A.8}$$

$$\frac{\mathrm{d}}{\mathrm{d}x} \log_b x = \frac{1}{x \ln b} \tag{A.9}$$

$$\mathrm{2Na + Cl_2 = 2NaCl} \tag{A.10}$$

$$\frac{\mathrm{d}}{\mathrm{d}x} \sec x = \frac{\cos(x)\,0 - 1(-\sin(x))}{(\cos(x))^2} \tag{A.11}$$

$$\int_2^4 y^2 - x^2 \,\mathrm{d}x \tag{A.12}$$

$$v'(t) = -\omega^2 A \cos(\omega t) \tag{A.13}$$

$$\tag{A.14}$$

$$[t]!\mathrm{TiO_2} + 2\mathrm{Cl_2} + \mathrm{C} \rightarrow \mathrm{TiCl_4} + \mathrm{CO_2} \tag{A.15}$$

$$\int (\cos(\theta))^2 \,\mathrm{d}\theta = \frac{1}{2}\theta + \frac{1}{2}\sin(\theta)\cos(\theta)\,\frac{\partial y}{\partial x} = mn^2\sqrt{p} \tag{A.16}$$

$$\int -3\mathrm{e}^{3x+12}\,\mathrm{d}\theta = -\mathrm{e}^{3x+12} + C \tag{A.17}$$

$$g'(x) = \frac{4x-2}{x^2+1} \tag{A.18}$$

$$\cos\left(\frac{a}{2}\right) = \sqrt{\frac{s(s-a)}{bc}} \tag{A.19}$$

$$\frac{\mathrm{d}}{\mathrm{d}x}\cos(x) = -\sin(x) \tag{A.20}$$

$$(\mathrm{NH_4})_2\mathrm{PtCl_6} = \mathrm{N_2} + \mathrm{NH_3} + \mathrm{HCl} + \mathrm{Pt} \tag{A.21}$$

$$\int \mathrm{e}^{-x}\,\mathrm{d}x = -\mathrm{e}^{-x} \tag{A.22}$$

$$\mathrm{AgNO_3} + \mathrm{H_2SO_4} \rightarrow \mathrm{Ag_2SO_4} + \mathrm{HNO_3} \tag{A.23}$$

$$\mathrm{C_2H_5OH} + 3\mathrm{O_2} = 2\mathrm{CO_2} + 3\mathrm{H_2O} \tag{A.24}$$

$$\mathrm{P_4(s)} + 6\mathrm{Cl_2(g)} \rightarrow 4\mathrm{PCl_2(l)} \tag{A.25}$$

$$m_5 + m_p + m_d - m^5 + m^d + m^g \tag{A.26}$$

$$\int \frac{\mathrm{e}^{ax}}{b + ce^a x}\,\mathrm{d}x = \frac{1}{ac}\log(b + ce^a x) \tag{A.27}$$

$$\tan(3a) = \frac{-(\tan(a))^3 + 3\tan(a)}{-3(\tan(a))^2 + 1} \tag{A.28}$$

$$\mathrm{C_3H_8} + 5\mathrm{O_2} \rightarrow 4\mathrm{H_2O} + 3\mathrm{CO_2} \tag{A.29}$$

$$y = \sqrt{x^3 + 9x} \tag{A.30}$$

APPENDIX B: DIAGRAMS

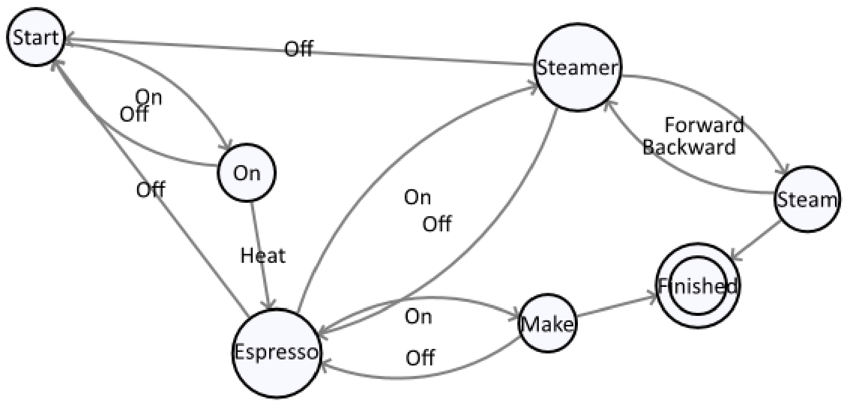


Figure B.1

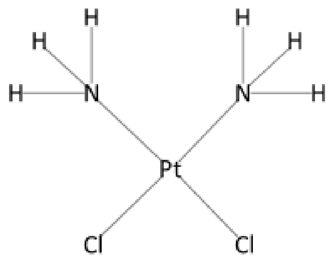


Figure B.2

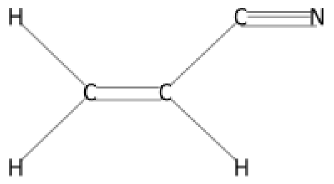


Figure B.3

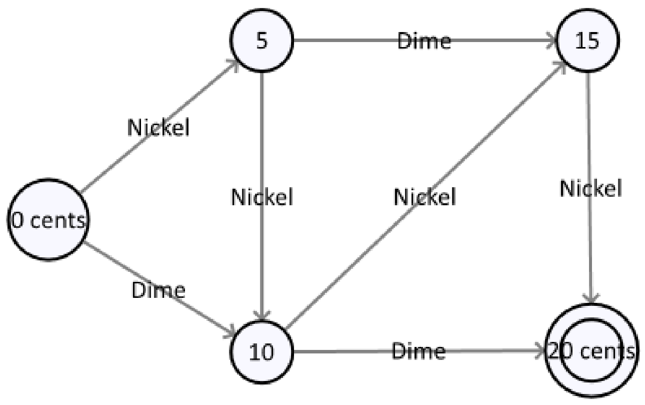


Figure B.4

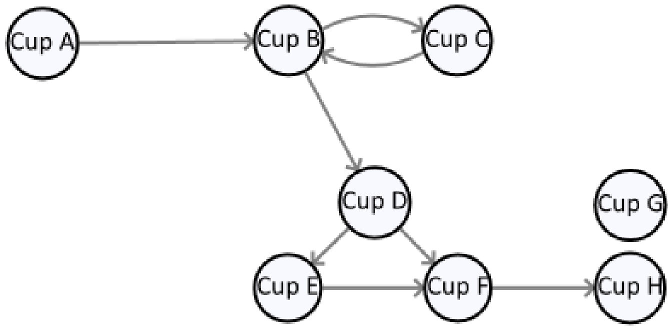


Figure B.5

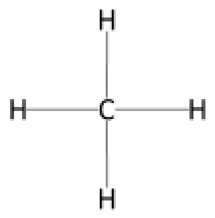


Figure B.6

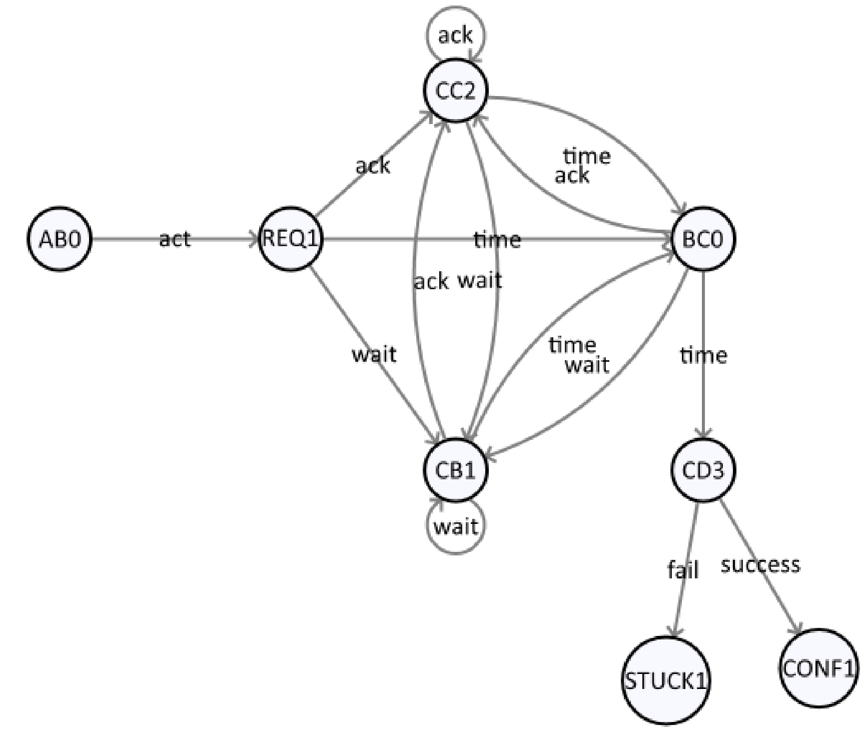


Figure B.7

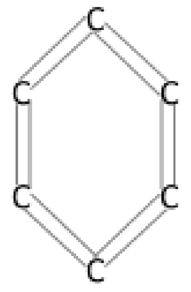


Figure B.8

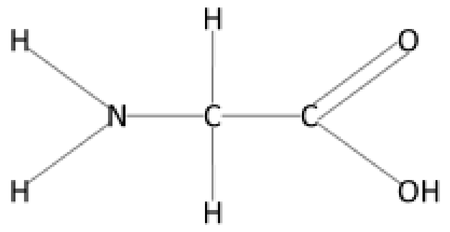


Figure B.9

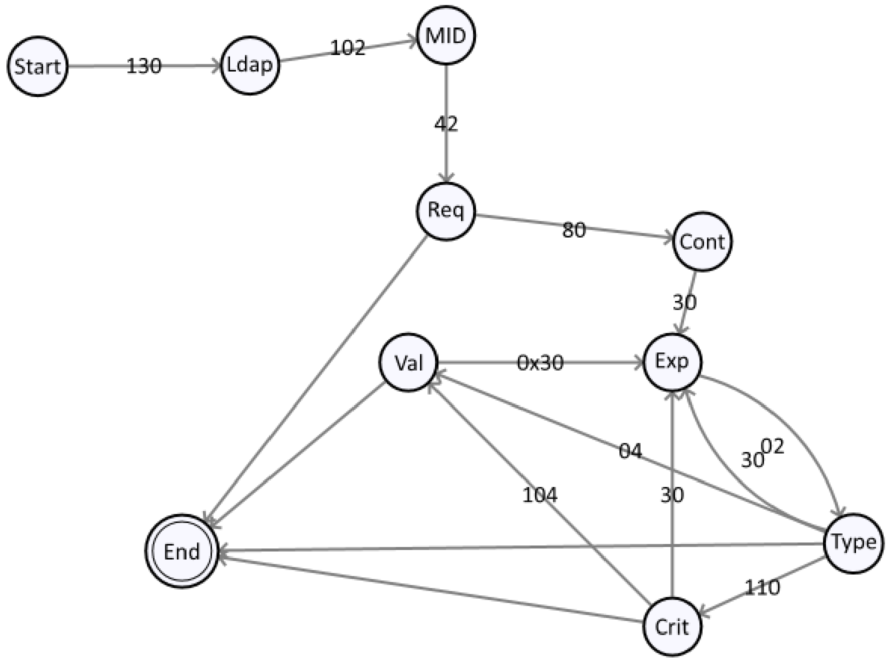


Figure B.10

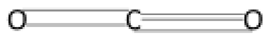


Figure B.11

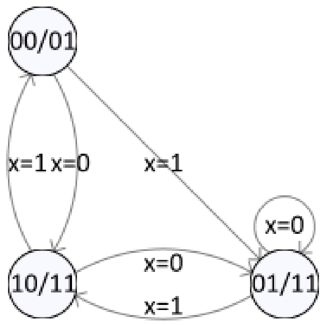


Figure B.12

APPENDIX C: PRE-QUESTIONNAIRE

1. What is your age?
2. What is your class rank (freshman, sophomore, etc.)?
3. What is your major?
4. What is your gender?
5. What hand do you write with?
6. Rate your experience with mathematical handwriting recognition software (1 through 7).
7. Rate your experience with sketch recognition software (1 through 7).
8. Rate your interest in sketch recognition (1 through 7).

APPENDIX D: POST-TASK QUESTIONNAIRE

1. I found it easy to recognize a sketch (1 through 7).
2. I found it easy to remove an error from the WOZ Recognizer's sketch.
3. I found it easy to watch the users writing.
4. I was distracted by the users writing.
5. I found it easy to set the recognition accuracy.
6. I found it easy to perform batch recognition.
7. I found it easy to perform streaming recognition.
8. I found it challenging to keep up with the users writing with streaming recognition.
9. I found it difficult keeping track of recognition accuracy.
10. I was able to quickly generate a WOZ Recognizer's sketch that matched the users writing.
11. I found the WOZ Recognizer's sketches to be realistic.
12. I was able to quickly adapt to changes in the users handwriting.
13. I was able to easily tell what I was sending during streaming recognition.
14. I found it difficult to find recognition errors to correct them.

APPENDIX E: POST-STUDY QUESTIONNAIRE

1. Overall Reaction to the WOZ Recognizer (1 through 7)
    (a) Terrible through Wonderful
    (b) Difficult through Easy
    (c) Frustrating through Satisfying

1. Mathematics (overall) (1, Hard through 7, Easy)
2. Recognizing Sketches
3. Batch Recognition
4. Streaming Recognition
5. Selecting part of sketch to send
6. Identifying Recognition Errors
7. Removing Recognition Errors

1. Diagrams (overall) (1, Hard through 7, Easy)
2. Recognizing Sketches
3. Batch Recognition
4. Streaming Recognition
5. Selecting part of sketch to send

6. Identifying Recognition Errors

7. Removing Recognition Errors

1. Using the WOZ Recognizer would make it easy to perform studies on mathematical sketch recognition (1, Strongly Disagree through 7, Strongly Agree).

2. Using the WOZ Recognizer would make it easy to perform studies on sketch recognition.

1. List the most positive aspects of the user interface.

2. List the most negative aspects of the user interface.

1. Overall, please rate the WOZ Recognizers ease of use (1, Hard through 7, Easy).

2. Is there anything you would like to see the WOZ Recognizer do that it currently does not support?

LIST OF REFERENCES

[1] Johnny Accot and Shumin Zhai. Beyond fitts' law: Models for trajectory-based hci tasks. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, CHI '97, pages 295–302, New York, NY, USA, 1997. ACM.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[3] Christine Alvarado and Randall Davis. Sketchread: a multi-domain sketch recognition engine. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*, UIST '04, pages 23–32, New York, NY, USA, 2004. ACM.

[4] Lisa Anthony. *Developing Handwriting-based Intelligent Tutors to Enhance Mathematics Learning*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, 2008.

[5] Lisa Anthony, Jie Yang, and Kenneth R. Koedinger. Benefits of handwritten input for students learning algebra equation solving. In *Proceedings of the 2007 conference on Artificial Intelligence in Education: Building Technology Rich Learning Contexts*

That Work, pages 521–523, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[6] Georg Apitz and François Guimbretière. Crossy: A crossing-based drawing application. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, UIST '04, pages 3–12, New York, NY, USA, 2004. ACM.

[7] Olle Bälter, Olov Engwall, Anne-Marie Öster, and Hedvig Kjellström. Wizard-of-oz test of artur: a computer-based speech training system with articulation correction. In *Proceedings of the 7th international ACM SIGACCESS conference on Computers and accessibility*, Assets '05, pages 36–43, New York, NY, USA, 2005. ACM.

[8] Christoph Benzmüller, Helmut Horacek, Ivana Kruijff-Korbayová, Henri Lesourd, Marvin Schiller, and Magdalena Wolska. Diawoz-ii: a tool for wizard-of-oz experiments in mathematics. In *Proceedings of the 29th annual German conference on Artificial intelligence*, KI'06, pages 159–173, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] Jared N. Bott, Daniel Gabriele, and Joseph J. LaViola, Jr. Now or later: An initial exploration into user perception of mathematical expression recognition feedback. In *Proceedings of the 8th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, pages 125–132. ACM, August 2011.

[10] Jared N. Bott and Joseph J. Laviola, Jr. The woz math recognizer: A mathematics handwriting recognition wizard of oz tool. Technical Report CS-TR-11-03, University of Central Florida, 2011.

[11] Sarah Buchanan, Brandon Ochs, and Joseph J. LaViola Jr. Cstutor: a pen-based tutor for data structure visualization. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, SIGCSE '12, pages 565–570, New York, NY, USA, 2012. ACM.

[12] Bill Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[13] Kam-Fai Chan and Dit-Yan Yeung. Error detection, error correction and performance evaluation in on-line mathematical expression recognition. *Pattern Recognition*, 34(8):1671 – 1684, 2001.

[14] Salman Cheema and Joseph J. LaViola, Jr. Applying mathematical sketching to sketch-based physics tutoring software. In *Proceedings of the 10th international conference on Smart graphics*, SG'10, pages 13–24, Berlin, Heidelberg, 2010. Springer-Verlag.

[15] W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, December 1998.

[16] G. Costagliola, V. Deufemia, and M. Risi. Sketch grammars: a formalism for describing and recognizing diagrammatic sketch languages. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 1226 – 1230 Vol. 2, aug.-1 sept. 2005.

[17] Richard C. Davis, T. Scott Saponas, Michael Shilman, and James A. Landay. Sketch-wizard: Wizard of oz prototyping of pen-based user interfaces. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, UIST '07, pages 119–128, New York, NY, USA, 2007. ACM.

[18] Martin Field, Sam Gordon, Eric Peterson, Raquel Robinson, Thomas Stahovich, and Christine Alvarado. The effect of task on classification accuracy: using gesture recognition techniques in free-sketch recognition. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, pages 109–116, New York, NY, USA, 2009. ACM.

[19] Andrew S. Forsberg, Andrew Bragdon, Joseph J. LaViola, Sashi Raghupathy, and Robert C. Zeleznik. An empirical study in pen-centric user interfaces: Diagramming. In *Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling*, SBM'08, pages 135–142, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[20] Clive Frankish, Richard Hull, and Pam Morgan. Recognition accuracy and user acceptance of pen interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '95, pages 503–510, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[21] Li Ge and Falko Kuester. *SketchIT: A Sketch-Based Modeling Environment for Structural Analysis*, chapter 104, pages 833–840. American Society of Civil Engineers,

2013.

[22] Leslie Gennari, Levent Burak Kara, Thomas F. Stahovich, and Kenji Shimada. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics*, 29(4):547–562, August 2005.

[23] R. A. Goubran and C. Wood. Building an application framework for speech and pen input integration in multimodal learning interfaces. In *Proceedings of the Acoustics, Speech, and Signal Processing, 1996. on Conference Proceedings., 1996 IEEE International Conference - Volume 06*, ICASSP '96, pages 3545–3548, Washington, DC, USA, 1996. IEEE Computer Society.

[24] A. Grbavec and D. Blostein. Mathematics recognition using graph rewriting. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, pages 417–, Washington, DC, USA, 1995. IEEE Computer Society.

[25] William I. Grosky, Robert Zeleznik, Timothy Miller, Andries van Dam, Chuanjun Li, Dana Tenneson, Christopher Maloney, and Joseph J. LaViola. Applications and issues in pen-centric computing. *IEEE MultiMedia*, 15(4):14–21, October 2008.

[26] Tracy Hammond and Randall Davis. Ladder, a sketching language for user interface developers. *Computers & Graphics*, 29(4):518 – 532, 2005.

[27] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):pp. 65–70, 1979.

[28] Christopher D. Hundhausen, Anzor Balkar, Mohamed Nuur, and Stephen Trent. Woz pro: a pen-based low fidelity prototyping environment to support wizard of oz studies. In *CHI '07 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '07, pages 2453–2458, New York, NY, USA, 2007. ACM.

[29] Joaquim Jorge and Faramarz Samavati. *Sketch-based Interfaces and Modeling*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[30] Bo Kang and Joseph LaViola. Logicpad: A pen-based application for visualization and verification of boolean algebra. In *Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces*, IUI '12, pages 265–268, New York, NY, USA, 2012. ACM.

[31] Kourtney Kebodeaux, Martin Field, and Tracy Hammond. Defining precise measurements with sketched annotations. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, pages 79–86, New York, NY, USA, 2011. ACM.

[32] Scott R. Klemmer, Anoop K. Sinha, Jack Chen, James A. Landay, Nadeem Aboobaker, and Annie Wang. Suede: a wizard of oz prototyping tool for speech user interfaces. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, UIST '00, pages 1–10, New York, NY, USA, 2000. ACM.

[33] S. Kollmansberger. Logic gate simulator. Computer program, June 2011.

[34] Gordon Kurtenbach and William Buxton. User learning and performance with marking menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, pages 258–264, New York, NY, USA, 1994. ACM.

[35] George Labahn, Edward Lank, Scott MacLean, Mirette Marzouk, and David Tausky. Mathbrush: A system for doing math on pen-based devices. In *Proceedings of the 2008 The Eighth IAPR International Workshop on Document Analysis Systems*, pages 599–606, Washington, DC, USA, 2008. IEEE Computer Society.

[36] Mary LaLomia. User acceptance of handwritten recognition accuracy. In *Conference companion on Human factors in computing systems*, CHI '94, pages 107–108, New York, NY, USA, 1994. ACM.

[37] Joseph J. LaViola, Jr. An initial evaluation of mathpad^2: A tool for creating dynamic mathematical illustrations. *Computers & Graphics*, 31:540–553, August 2007.

[38] Joseph J. LaViola, Jr., Anamary Leal, Timothy S. Miller, and Robert C. Zeleznik. Evaluation of techniques for visualizing mathematical expression recognition results. In *Proceedings of graphics interface 2008*, GI '08, pages 131–138, Toronto, Ont., Canada, 2008. Canadian Information Processing Society.

[39] Joseph J. LaViola, Jr. and Robert C. Zeleznik. Mathpad^2: a system for the creation and exploration of mathematical sketches. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 432–440, New York, NY, USA, 2004. ACM.

[40] Joseph J. LaViola Jr. and Robert C. Zeleznik. A practical approach for writer-dependent symbol recognition using a writer-independent symbol recognizer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1917–1926, November 2007.

[41] WeeSan Lee, Ruwanee de Silva, Eric J. Peterson, Robert C. Calfee, and Thomas F. Stahovich. Newton's pen: A pen-based tutoring system for statics. *Computers & Graphics*, 32(5):511 – 524, 2008.

[42] V I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[43] James Lin, Mark W. Newman, Jason I. Hong, and James A. Landay. Denim: Finding a tighter fit between tools and practice for web site design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, pages 510–517, New York, NY, USA, 2000. ACM.

[44] I. Scott MacKenzie, Abigail Sellen, and William A. S. Buxton. A comparison of input devices in element pointing and dragging tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 161–166, New York, NY, USA, 1991. ACM.

[45] S. MacLean, D. Tausky, G. Labahn, E. Lank, and M. Marzouk. Is the ipad useful for sketch input?: a comparison with the tablet pc. In *Proceedings of the Eighth*

Eurographics Symposium on Sketch-Based Interfaces and Modeling, SBIM '11, pages 7–14, New York, NY, USA, 2011. ACM.

[46] Jennifer Mankoff, Scott E. Hudson, and Gregory D. Abowd. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '00, pages 368–375, New York, NY, USA, 2000. ACM.

[47] Morris Mano and Charles Kime. *Logic and computer design fundamentals*. Prentice Hall, Inc., Upper Saddle River, New Jersey, USA, 1997.

[48] Microsoft. Math input panel. Computer program, 2009.

[49] M. Negulescu, J. Ruiz, and E. Lank. Exploring usability and learnability of mode inferencing in pen/tablet interfaces. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, SBIM '10, pages 87–94, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[50] Fathallah Nouboud and Réjean Plamondon. Document image analysis. chapter On-line recognition of handprinted characters: survey and beta tests, pages 342–355. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[51] Sharon Oviatt, Colin Swindells, and Alex Arthur. Implicit user-adaptive system engagement in speech and pen interfaces. In *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI '08, pages 969–978, New York, NY, USA, 2008. ACM.

[52] Florian Perteneder, Martin Bresler, Eva-Maria Grossauer, Joanne Leong, and Michael Haller. cluster: Smart clustering of free-hand sketches on large interactive surfaces. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software &#38; Technology*, UIST '15, pages 37–46, New York, NY, USA, 2015. ACM.

[53] Janet C. Read, Stuart MacFarlane, and Chris Casey. 'good enough for what?': acceptance of handwriting recognition errors by child users. In *Proceedings of the 2003 conference on Interaction design and children*, IDC '03, pages 155–155, New York, NY, USA, 2003. ACM.

[54] Peter Reid, Fred Hallett-Hook, Beryl Plimmer, and Helen Purchase. Applying layout algorithms to hand-drawn graphs. In *Proceedings of the 19th Australasian conference on Computer-Human Interaction: Entertaining User Interfaces*, OZCHI '07, pages 203–206, New York, NY, USA, 2007. ACM.

[55] Yi Ren, Yang Li, and Edward Lank. Inkanchor: Enhancing informal ink-based note taking on touchscreen mobile phones. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 1123–1132, New York, NY, USA, 2014. ACM.

[56] Vinícius C. V. B. Segura, Simone D. J. Barbosa, and Fabiana Pedreira Simões. Uiskei: a sketch-based prototyping tool for defining and evaluating user interface behavior. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '12, pages 18–25, New York, NY, USA, 2012. ACM.

[57] M. Shilman, H. Pasula, and S. Russel R. Newton. Statistical visual language models for ink parsing. In *Proceedings of the AAAI Spring Symposium on Sketch Understanding*, pages 126–32, 2002.

[58] Steve Smithies, Kevin Novins, and James Arvo. A handwriting-based equation editor. In *Proceedings of the 1999 conference on Graphics interface '99*, pages 84–91, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[59] Ryan Stedman, Michael Terry, and Edward Lank. Aiding human discovery of handwriting recognition errors. In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, ICMI '13, pages 295–302, New York, NY, USA, 2013. ACM.

[60] Paul Taele and Tracy Hammond. *The Impact of Pen and Touch Technology on Education*, chapter Enhancing Instruction of Written East Asian Languages with Sketch Recognition-Based "Intelligent Language Workbook" Interfaces, pages 119–126. Springer International Publishing, Cham, 2015.

[61] Eugene M. Taranta and Joseph J. LaViola, Jr. Math boxes: A pen-based user interface for writing difficult mathematical expressions. In *Proceedings of the 20th International Conference on Intelligent User Interfaces*, IUI '15, pages 87–96, New York, NY, USA, 2015. ACM.

[62] Stephanie Valentine, Francisco Vides, George Lucchese, David Turner, Honghoe Kim, Wenzhe Li, Julie Linsey, and Tracy Hammond. Mechanix: A sketch-based

tutoring system for statics courses. In *The Twenty-Fourth Conference on Innovative Applications of Artificial Intelligence*, 2012.

[63] Radu-Daniel Vatavu, Daniel Vogel, Gry Casiez, and Laurent Grisoni. Estimating the perceived difficulty of pen gestures. In Pedro Campos, Nicholas Graham, Joaquim Jorge, Nuno Nunes, Philippe Palanque, and Marco Winckler, editors, *Human-Computer Interaction INTERACT 2011*, volume 6947 of *Lecture Notes in Computer Science*, pages 89–106. Springer Berlin Heidelberg, 2011.

[64] Paul Wais, Aaron Wolin, and Christine Alvarado. Designing a sketch recognition front-end: user perception of interface elements. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, SBIM '07, pages 99–106, New York, NY, USA, 2007. ACM.

[65] Jagoda Walny, Bongshin Lee, Paul Johns, Nathalie Henry Riche, and Sheelagh Carpendale. Understanding pen and touch interaction for data exploration on interactive whiteboards. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2779–2788, December 2012.

[66] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 159–168, New York, NY, USA, 2007. ACM.

[67] Jie Wu, Changhu Wang, Liqing Zhang, and Yong Rui. Smartvisio: Interactive sketch recognition with natural correction and editing. In *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, pages 735–736, New York, NY, USA, 2014. ACM.

[68] Shane W Zamora and Eyrún A Eyjólfsdóttir. Circuitboard: Sketch-based circuit design and analysis. In *IUI Workshop on Sketch Recognition*, 2009.

[69] Robert Zeleznik, Timothy Miller, Chuanjun Li, and Joseph J. Laviola, Jr. Mathpaper: Mathematical sketching with fluid support for interactive computation. In *SG '08: Proceedings of the 9th international symposium on Smart Graphics*, pages 20–32, Berlin, Heidelberg, 2008. Springer-Verlag.