RETHINKING PEN INPUT INTERACTION: ENABLING FREEHAND SKETCHING THROUGH IMPROVED PRIMITIVE RECOGNITION

A Dissertation

by

BRANDON CHASE PAULSON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2010

Major Subject: Computer Science

RETHINKING PEN INPUT INTERACTION: ENABLING FREEHAND SKETCHING THROUGH IMPROVED PRIMITIVE RECOGNITION

A Dissertation

by

BRANDON CHASE PAULSON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2010

Major Subject: Computer Science

ABSTRACT

Rethinking Pen Input Interaction: Enabling Freehand Sketching Through Improved Primitive Recognition. (May 2010)

Brandon Chase Paulson, B.S., Baylor University

Chair of Advisory Committee: Dr. Tracy Hammond

Online sketch recognition uses machine learning and artificial intelligence techniques to interpret markings made by users via an electronic stylus or pen. The goal of sketch recognition is to understand the intention and meaning of a particular user's drawing. Diagramming applications have been the primary beneficiaries of sketch recognition technology, as it is commonplace for the users of these tools to first create a rough sketch of a diagram on paper before translating it into a machine understandable model, using computer-aided design tools, which can then be used to perform simulations or other meaningful tasks.

Traditional methods for performing sketch recognition can be broken down into three distinct categories: appearance-based, gesture-based, and geometric-based. Although each approach has its advantages and disadvantages, geometric-based methods have proven to be the most generalizable for multi-domain recognition. Tools, such as the LADDER symbol description language, have shown to be capable of recognizing sketches from over 30 different domains using generalizable, geometric techniques. The LADDER system is limited, however, in the fact that it uses a low-level recognizer that supports only a few primitive shapes, the building blocks for describing higher-level symbols. Systems which support a larger number of primitive shapes have been shown to have questionable accuracies as the number of primitives increase, or they place constraints on how users must input shapes (e.g. circles can only be drawn in a clockwise motion; rectangles must be drawn starting at the top-left corner).

This dissertation allows for a significant growth in the possibility of free-sketch recognition systems, those which place little to no drawing constraints on users. In this dissertation, we describe multiple techniques to recognize upwards of 18 primitive shapes while maintaining high accuracy. We also provide methods for producing confidence values and generating multiple interpretations, and explore the difficulties of recognizing multi-stroke primitives. In addition, we show the need for a standardized data repository for sketch recognition algorithm testing and propose SOUSA (sketch-based online user study application), our online system for performing and sharing user study sketch data. Finally, we will show how the principles we have learned through our work extend to other domains, including activity recognition using trained hand posture cues.

ACKNOWLEDGMENTS

The writing of this dissertation would not be possible without the strength and guidance given to me by my Lord and Savior, Jesus Christ. In addition, this achievement would not have been completed without the love and support I have received from my wife, Stacy, as well as the rest of my entire family. I would also like to thank my family at Beacon Baptist Church for their encouragement during this process. Thank you to my committee members, Dr. Yoonsuck Choe, Dr. Ricardo Gutierrez-Osuna, and Dr. Vinod Srinivasan for all that you have taught me, both in the classroom, and through your helpful critiques of this work. Thanks to the many members of the Sketch Recognition Lab (past and present) for your help in proofreading, critiquing, and editing this work. Thank you also to Greg Sparks for taking time to proofread this document as well. Last, but not least, I would like to acknowledge my advisor, Dr. Tracy Hammond, for all of her hard work and the investment that she has made in me to further my academic career.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION 1

A. Sketch Recognition in Design 1
B. Sketch Recognition in Engineering & Education 2
C. Proposal 3

II PREVIOUS WORK IN PEN-BASED INTERFACES 7

A. Hardware Research 7
B. Human-computer Interaction 10
1. Sketch Editing 11
2. Incorporating Gestures 11
3. Beautification 13
4. Toolkits 15
5. Multimodal Systems 15
C. Recognition 16
D. High-level Recognition 17
E. Low-level Recognition 19
1. Motion-based Recognition 20
2. Appearance-based Recognition 23
3. Geometric-based Recognition 24
a. Corner Finding 25
b. Primitive Recognition 30
c. Limitations of Geometric-based Approach 31
4. Hybrid/Combination Techniques 32
F. Applications for Sketch-based Interfaces 35
1. Sketching in Design 35
a. Storyboards 37
b. 3D Modeling 37
2. Sketching in Engineering and Education 38
a. Sketching in Engineering 39
b. Sketching in Math & Science 41
c. Sketching in Language & Fine Arts 43
3. Search by Sketch 44

CHAPTER Page

III PALEOSKETCH 46

A. Introduction 46
1. First Version of PaleoSketch 49
2. Determining What Primitives to Support 49
3. Symbol Description Experiment 51
B. Data Set 53
C. The First Part of PaleoSketch: Pre-recognition 55
D. Features 58
1. New Geometric Features 59
a. General Features 59
b. Line Features 62
c. Ellipse Features 64
d. Circle Features 64
e. Arc Features 65
f. Curve Features 66
g. Polyline Features 67
h. Spiral & Helix Features 68
E. Classifier 70
F. Handling Complex Shapes 71
1. Confidence Modification 72
G. Experiment & Results 74
1. Accuracy of Complex Fits 77
H. Discussion 78
1. Complex Fits 78
2. Using PaleoSketch in a Real-world Setting 82
3. Limitations of PaleoSketch 82
I. Chapter Summary 85

IV MILITARY COURSE OF ACTION 86

A. Introduction 86
B. Previous Work in COA Systems 87
C. Methodology 89
1. Features 90
a. Rectangle & Diamond Features 91
b. Arrow Features 91
c. Dot Features 92
d. Wave Features 93
e. Gull Features 94

CHAPTER Page

f. NBC Features 95
2. Classifier 96
3. Data 96
D. Results 98
E. Discussion 98
F. Chapter Summary 100

V MULTI-STROKE PRIMITIVES 101

A. Introduction 101
B. Implementation 103
1. Graph Building 104
2. Graph Searching 106
3. Stroke Combination 108
4. False Positive Removal 109
5. Arrow Detection 111
C. Experiment 113
D. Results 115
1. Single-stroke 115
2. Multi-stroke 118
3. Complex Shapes 123
E. Discussion 129
1. Comparison to Other Methods 129
2. Multiple Interpretations 131
3. Improving Complex Interpretations 132
4. Improving Grouping 132
5. Additional Improvements 133
F. Chapter Summary 133

VI SOUSA 134

A. Introduction 134
B. Previous Efforts 134
C. The SOUSA System 135
1. Collection Studies 138
2. Verification Studies 138
D. Evaluation 141
E. Future Work 142
F. Chapter Summary 143

VII OFFICE ACTIVITY RECOGNITION 145

CHAPTER Page

A. Preface 145
B. Introduction 145
C. Related Work in Activity Recognition 149
D. Experiment 150
1. Classifiers 151
2. Feature Spaces 153
E. Results 153
1. User-Independent System 154
2. User-Dependent System 155
F. Discussion 157
G. Future Work 162
H. Chapter Summary 163

VIII CONCLUSION 164

REFERENCES 167

VITA 198

LIST OF TABLES

TABLE Page

I Accuracy results of existing recognizers on our collected set of primitive data. 54

II Accuracy results of the different feature sets using a multi-layer perceptron classifier. The first five columns represent the accuracies of the five individual feature sets. *Combined* refers to the combined feature set of CALI, HHReco, and Long. *All* refers to the combination of the *Combined* feature set, plus the Paleo features. *Modified* uses the same feature set as *All*, but also utilizes the complex confidence modification algorithm. 75

III Number of occurrences of each primitive shape in our data sets as well as the accuracy of 10-fold cross-validation on each set. The final column shows the accuracy results when we train with dataset A and test with dataset B. "Total" accuracy represents the flat average of all shape accuracies, while "weighted" average shows the average accuracy of each shape weighted by its number of occurrences. *Dataset B consisted of more lines because it included "anticipated" unit symbols, which are drawn as dashed rectangles or diamonds. 99

IV Number of each type of sketched primitive present in the collected data. Complex and polyline shapes cannot be drawn with multiple strokes. 116

V Accuracy results after 10-fold cross-validation. "Average" represents flat averages, while "weighted averages" represent averages that are weighted by the number of occurrences of each shape type. . 117

TABLE Page

VI The types of interactions we collected for our experiment. From left to right, top to bottom: drinking from a cup, dialing a telephone, picking up earphones, typing on a keyboard, using a mouse, drinking from a mug, opening a drawer, reading a piece of paper, writing with a pen, stapling papers, answering the telephone, and waving. 152

LIST OF FIGURES

FIGURE Page

1 Example high-level shape description from LADDER. 3

2 Ivan Sutherland's Sketchpad system. 8

3 Various hardware devices that allow for pen-based input. 9

4 Examples of new pen-based input devices currently being developed (from left to right): multi-touch displays [1], bimanual pen and touch displays [2], IR projected sketching systems [3], and surfaceless pen-based interfaces [4]. 10

5 Example usage of gestures in pen-based interfaces. 12

6 Example of global beautification, in which locally-beautified symbols are cleaned up based on properties related to symmetry and congruence [5]. 14

7 Example of local beautification performed by PaleoSketch [6] within the LADDER sketch recognition system [7]. 14

8 Example of real-time beautification performed as the user is drawing the stroke [8]. 15

9 Example of the distinction in labeling between low-level and high-level recognizers. Low-level recognizers assign domain-ignorant labels while high-level recognizers assign domain-specific labels. . . . 17

10 Example sketch recognition framework. 18

11 Example gesture set from Graffiti. Some of these gestures may not be natural to the user and must be learned before interacting with the system. 20

FIGURE Page

12 Motion-based methods focus on how a stroke was drawn in order to perform recognition. In this example, even though the shapes look the same visually, they would actually be considered different gestures because they were drawn with two different motions. 21

13 Example of the reference lines generated by the Ledeen recognizer. . 22

14 Appearance-based methods typically use some form of template matching to perform recognition; however, alternative variations of the same symbol would require multiple templates. 24

15 Geometric-based methods estimate an ideal shape representation (red) of the stroke (black) and then compute shape approximation errors to perform recognition. For example, a circle can be estimated by finding an approximate center (e.g., the center of the bounding box) and an approximate radius (e.g., by taking the average distance of each stroke point to the center). Lines can be estimated by simply connecting the endpoints of the stroke. 25

16 Example results of corner finding on curvilinear stroke. Results are from MergeCF [9]. 26

17 A walkthrough of the Douglas-Peucker algorithm [10]. 28

18 Example of the "straws" computed through the ShortStraw algorithm [11]. In this example, straws (a), (c), and (e) are shorter and more likely candidates for corners than the longer straws of (b) and (d). 29

19 Feature area examples of the shapes supported by the Yu and Cai recognizer [12]. 31

20 Bounding polygons that are calculated for the Apte and CALI recognizers [13, 14, 15]. 33

21 Gestures supported by the CALI recognizer [14, 15]. 34

22 Screenshots of the MARQS system [16]. 36

23 Example GUI created with the JavaSketchIT application [17]. 37

FIGURE Page

24 Examples of different 3D modeling applications that utilize pen-based input. 38

25 Examples of electrical engineering applications that utilize sketch recognition technology. 40

26 Examples of UML sketch systems. 40

27 Examples of mechanical sketches being recognized and translated into simulator models through the ASSIST application [18]. 41

28 Examples of sketch-based, math applications. 42

29 Examples of chemistry and biology applications. 43

30 Gesture set used by the Music Notepad system for composing sheet music [19]. 44

31 Example scenario of a smart notebook where both textual and sketch queries would be beneficial. 45

32 Example scenario in which higher-level context can disambiguate a lower-level interpretation. In this example, the high-level recognizer may realize that the ambiguous stroke is more likely to be a circle (which represents a wheel in this domain) rather than an ellipse because of context. 48

33 Primitives supported by PaleoSketch. 52

34 An example of a drawn stroke that has a “tail.” 55

35 Example stroke along with its corresponding speed graph (top right), direction graph (bottom left), and curvature graph (bottom right). 56

36 Two similar polylines, drawn to different rotations, but with similar direction graphs. 57

37 Sketched primitives and their corresponding bounding boxes. Notice that all four corners of the rectangle’s bounding box are near the drawn stroke, while none are close for the ellipse, and only two are near for the line. 60

FIGURE Page

38 Direction graphs for a polyline (left) and arc (right). The points of highest and lowest direction value are circled (middle). NDDE is calculated as the length of the stroke between direction extremes (bolded in blue), divided by the entire stroke length. DCR (bottom) is the maximum change in direction (bolded in red) divided by the average change in direction across the stroke. In this example, the polyline has a lower NDDE value because its minimum direction value is in the middle of the stroke, whereas the arc has its direction extremes closer to the endpoints. Furthermore, the polyline has a large spike in its direction graph, causing it to have a much higher DCR value than the arc, which maintains a fairly smooth continuity throughout. 63

39 Estimating the center of an arc. 1) Connect the endpoints of the stroke. 2) Take the perpendicular bisector of 1. 3) Connect the endpoints of the stroke to the point where 2 intersects the stroke. 4) Take the perpendicular bisectors of the two lines from 3 and find their intersection. 65

40 Examples of correctly classified shapes. This is just a subset of the 900 sketched shapes used for testing. 76

41 Examples of complex shapes drawn with elliptical arcs, thus causing the recognizer to return a “one line, one curve” interpretation rather than “one line, one arc” (the users’ intentions). 78

42 Confusion matrix for our experiments using the optimal version of PaleoSketch. Dark areas indicate high confusion, while white areas indicate little or no confusion. 79

43 Examples of mis-classifications in the PaleoSketch system. 80

44 Examples of higher degree complex fits achieved through the integration of PaleoSketch into LADDER. 81

45 Examples of complex fits recognized using the MLP version of PaleoSketch that was trained only on “one line, one arc” samples. . . 83

FIGURE Page

46 Example LADDER domains (top: military course of action, middle: planetary physics simulation [20], bottom: football play diagram) using the PaleoSketch primitive recognizer. 84

47 Screenshot of a military course of action recognition system that utilizes our PaleoSketch primitive recognizer [21]. 87

48 Example of how COA symbols can be described hierarchically using a sketch grammar. 88

49 Example of each of the sketched COA primitives. 90

50 Example corner finding results on a single-stroke arrow. A and B represent the sub-segments that make up the arrow head, C represents the sub-segment that makes up the overtraced part of the head, and {D} represents the remaining sub-segments that make up the path of the arrow. 92

51 Example results produced by the alternative segmenter on a sketched wave. 93

52 Example results produced by the alternative segmenter on a sketched gull. 94

53 Example results produced by the alternative segmenter on sketched NBC symbols. 95

54 Subset of COA symbols used for data collection (data set A). 97

55 Examples of continuation strokes. 101

56 Examples of continuation strokes in a hand-sketched circuit diagram. 103

57 Example graph built from a set of six strokes representing two primitive shapes. 104

58 We use a searching algorithm to find the strongly connected components of the graph. These correspond to the strokes that are likely candidates to be merged. 106

59 Multi-stroke recognition flowchart. 107

FIGURE Page

60 Example of two lines that are merged by simply connecting the endpoints. In this case, the direction graph contains a discontinuity even though lines typically have a smooth direction graph. . . . 108

61 Examples of the different ways that users draw arrows: with one stroke (a), with three strokes (b), or two different variations of two strokes (c,d). 111

62 Symbols from multiple domain that were used for data collection. . . 114

63 Examples showing the way that most users drew dots (left) versus the single user that drew them with small strokes (right). 118

64 Examples of correctly merged and classified multi-stroke primitives (original sketch and combined primitive). 119

65 Examples of multi-stroke primitives that were unsuccessfully merged due to endpoint distances being too great. 121

66 Example of a user attempting to correct an error in sampling (top ellipse). 122

67 Examples of multi-stroke overtracing performed by one user. 122

68 Example of a NAND gate drawn by one particular user. This resulted in both a false positive and a false negative combination. . . 123

69 Example of a false positive generated due to the greedy nature of the algorithm. 124

70 Example of multi-stroke primitives that was merged correctly, but classified wrong. 124

71 Examples of the complex shapes encountered in the data set, along with the percentage of the time that each was drawn in a single-stroke rather than multiple strokes. 125

72 Examples of correctly interpreted complex shapes (original strokes and recognized shapes). 127

73 Example of a complex undersegmentation. 128

FIGURE Page

74 Example of complex oversegmentation. Additional lines at the tops of the helixes led to oversegmentations. 128

75 Overtraced lines within complex shapes kept them from being properly interpreted (left). 129

76 Examples of curved arrows and rectangles with overlapping sides that Tahuti has trouble recognizing. 130

77 Screenshot of the home page for the current version of SOUSA. . . . 136

78 Screenshot of the SOUSA form for creating a collection study. 137

79 Screenshot of the SOUSA data collection applet. 139

80 Screenshot of the form used to create a verification study. 140

81 Screenshot of a verification study. 141

82 Framework indicating where hand posture fits into the overall scheme of context-aware computing. 146

83 The 22-sensor CyberGlove II by Immersion Corporation [22]. 148

84 Average accuracy results of a user-independent system with varying neighborhood sizes (k) in the KNN classifier. Results for the linear classifiers are shown as lines since they are constant and do not require a varying neighborhood. 154

85 Accuracy results of a user-dependent system with varying number of training samples per interaction type. 155

86 Examples of variations in interaction with the same objects. 158

87 Confusion matrix for the 1-NN/raw feature space, user-independent system. Dark areas indicate high confusion, while white areas indicate little or no confusion. Note that the values along the diagonal are lightest for dialing, typing, and stapling. This indicates that these actions contain the most variance across different users. . . 159

FIGURE Page

88 Confusion matrix for the 1-NN/raw feature space, user-dependent system. Dark areas indicate high confusion, while white areas indicate little or no confusion. 160

CHAPTER I

INTRODUCTION

Sketching conveys much information between two people. According to Johnson et al., “sketches are quickly made depictions that facilitate visual thinking” [23]. Concepts and interactions that would take a significant amount of time to describe in words can often be explained by a simple diagram or sketch. Because of this, sketching is a powerful form of human-human communication. Recently, sketching has also become a popular form of human-computer interaction due to the rise of technologies like Tablet PCs, interactive pen displays, and electronic whiteboards.

The goal of sketch recognition is to understand the meaning behind a user’s sketch, in order to provide a richer user experience. Thus far, sketch recognition has proved to be a beneficial tool in many different design, education, and engineering domains where hand-drawn graphical diagrams are used to express information between two individuals. A full survey of the applications that utilize sketch recognition can be found in Chapter II, Section F.

A. Sketch Recognition in Design

Current designers often find themselves sketching out a rough design on paper, then translating that paper sketch into a computer-aided design (CAD) model in order to perform some form of simulation to test the viability of their design. In order to remove this extra step of translating paper sketches into computer-understandable forms through CAD tools, some designers opt to work directly with the toolbar-based authorware software. However, researchers have discovered that not only do “design-

The journal model is *IEEE Transactions on Neural Networks.*

ers spend too much time working with the [authorware] tool and not enough time exploring design ideas" [24, 25], but that constrained-input, cleaned diagrams often hinder the creativity of a designer [26]. Computer-aided design (CAD) systems that accept freely-drawn diagrams for input may encourage greater originality [27, 28]. Sketch recognition will allow designers to rapidly conceive their ideas using familiar materials like the pen, yet be able to perform computational processes without requiring a shifting of modes from the paper to the computer. Sketch recognition will allow designers to continue working in a low-fidelity space, yet achieve high-fidelity feedback.

B. Sketch Recognition in Engineering & Education

Sketch recognition has already proved itself to be a valuable tool in the field of education. Some domains that use sketch systems for educational purposes include: UML diagrams [29], circuit diagrams [30], geography [20], mechanical engineering [18, 31, 32, 33], mathematics [34], chemistry [35, 36], Kanji [37, 38], biology [39], and civil engineering [40]. A future goal of sketch recognition researchers is the development of accurate recognizers which will allow sketched diagrams to be graded for examinations. Because the correction of hand-drawn diagrams takes a significant amount of instructor time, they are typically omitted from traditional exams and replaced with easier to grade, multiple-choice questions. This is unfortunate because not only do diagrams test a student's problem-solving ability, but they are also widely used in the industry where they may one day be employed.

```
(define shape Pendulum ...
  (components
    (Circle mass)
    (Line arm))
  (constraints
    (concentric arm.p1 mass) ...)
)
```

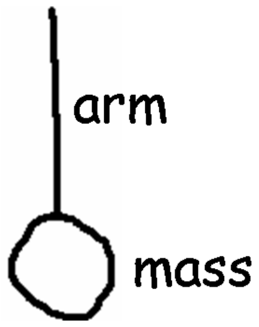


Fig. 1. Example high-level shape description from LADDER.

C. Proposal

Current state-of-the-art sketch recognizers are typically implemented in a hierarchical manner. Low-level *primitive* shapes, such as lines, polylines, ellipses, circles, and arcs, are used as building blocks for structural sketch grammars that describe more complex symbols. For example, a hand-sketched pendulum may be described as a combination of a line and a circle that meet certain geometric constraints (e.g., the line is above the circle, the endpoint of the line meets the top of the circle, and so forth), as seen in Figure 1. The main problems with existing primitive shape recognizers is that they either a) are not adequately accurate, b) do not support a large number of shapes, typically less than 5, or c) place drawing constraints on how users must input shapes.

The goal of this work is to create better low-level recognizers that support a large number of primitive shapes without sacrificing classification accuracy or requiring special drawing constraints. More specifically, we hope to create sketching algorithms that support many primitives (upwards of 18), while maintaining at least equal or better accuracy to existing state-of-the-art recognizers. By recognizing more primitive shapes, we allow high-level sketch grammars to become more expressive and capable of describing many domains. For example, in mechanical engineering diagrams, springs are typically drawn as helixes. Because existing primitive recognizers cannot classify this shape, the domain has either been unsupported by the sketch system or it uses an alternative symbol in place of the helix. This alternative symbol may not have been an intuitive substitution for the end-user. Because drawing habits are hard to break, the sketch system may have required additional learning time.

In order to achieve our goal of creating better sketch recognizers for pen-based interfaces, we aim to answer the following research questions:

1. *Can we achieve high primitive recognition accuracy without constraining a user's drawing style through the development of geometric-based features?* Current recognizers rely either on appearance-based features that are sensitive to changes in scale and rotation, or gesture-based features that are concerned solely with the motion in which a stroke is drawn (e.g., users must draw circles in a counter-clockwise direction to be recognized). We aim to see if geometric-based features can be a valid alternative for recognizing a large number of primitive shapes without sacrificing accuracy or requiring special drawing constraints. Our approach is evaluated with naturally drawn sketch data (1800 total samples) provided by 20 different users. The results are compared with those of current state-of-the-art recognizers including the Sezgin recognizer [41], $1 recognizer [42], Rubine recognizer [43], CALI recognizer [14, 15], and HHReco recognizer [44]. This question is the basis for Chapter III of this dissertation.

2. *Is our approach robust and capable of working in a sketch domain consisting of a large number of unique symbols?* Most sketch recognizers are typically evaluated on domains consisting of tens of symbols. The military course-of-action domain consists of hundreds (possibly thousands) of unique symbols. We have used our approach, PaleoSketch, to recognize sketched symbols from this domain. The

domain presented a number of challenges, including the need to introduce a new set of primitives that were necessary to fully recognize the complete gamut of symbols. This will be covered in Chapter IV.

3. *Can multi-stroke primitives be recognized without requiring strokes to be drawn consecutively?* We want to remove as many drawing constraints from the user as possible; this includes the single-stroke constraint placed by most primitive recognizers. Unlike other multi-stroke approaches, we wish to produce an algorithm that will work with both linear and curvy strokes, will not require timeouts or special button presses to denote shape completion, and will allow stroke interspersing. We have collected isolated symbol data from a large number of real-world domains to evaluate our approach in Chapter V.

4. *How can we make the results of sketch recognition work reproducible by other researchers?* Currently, it is difficult to reproduce the results of existing sketch recognizers due to the lack of standardized benchmark data. We propose SOUSA (sketch-based online user study applets), a web-based system that will be designed to allow researchers to collect, label, and share data. By creating an online repository of sketched data, researchers can easily compare algorithms on common data sets. This not only saves researcher time in data collection, but will also advance the field of sketch recognition because algorithms can more accurately be compared. Chapter VI discusses the SOUSA system.

5. *Do the HCI principles learned from this work in sketch recognition translate to other research domains?* One of the main principles we wish to promote in sketch recognition is that users should be allowed to interact freely with a system and not be forced to do something that is unnatural to them [45]. We believe this principle translates to other research areas. We test this belief

through experiments in activity recognition. Users are asked to interact with real-world objects using a data glove. The goal will be to see if a user's hand posture can be used to determine the objects he interacts with. We have seen that, when allowed to perform studies naturally, object interaction is different across users. This experiment is covered in Chapter VII.

In summary, the work presented in this dissertation will advance the field of sketch recognition by introducing new techniques for accurately recognizing a large number of primitive shapes. This will allow sketch recognition algorithms to describe a larger number of graphical domains, which, consequently, will allow for tools and interfaces to be built for more educational, design, and engineering applications. Intelligent, sketch-based tutoring systems will allow students to sketch out diagrams and receive real-time, interactive feedback. Teachers will be allowed to test a student's problem-solving ability and receive automatic feedback through the recognition of hand-drawn sketches. Designers, who tend to produce hand-drawn sketches during the initial stages of design, can receive high-fidelity feedback through their low-fidelity prototypes. They will avoid the intermediate stages of translating a hand-sketched design into a computer-understandable form through the use of computer-aided design (CAD) tools, which perform analysis and simulation on a design.

Areas that will directly benefit from improved sketch recognition technology include (but are not limited to): computer science (UML, finite state machines, flowcharts, user interface design), mathematics, chemistry, mechanical engineering, electrical engineering (circuits), civil engineering, geography, biology, physics, and language studies (e.g. Japanese, Korean, Chinese, and Urdu).

CHAPTER II

PREVIOUS WORK IN PEN-BASED INTERFACES

Within the field of pen-based computing, there are many different research areas. While each area is typically analyzed independently of the others, they all share the common goal of providing a better experience to the users of pen-based interfaces. The research areas of pen-based computing can typically be broken into one of the following categories: hardware, human-computer interaction (HCI) studies, recognition, and applications. Hardware research focuses on creating devices for translating pen-based input into an electronic form. HCI researchers focus on how to use these pen-based devices as an effective and efficient means of input into traditional computer applications. Recognition focuses on adding the extra dimension of computer understanding to pen-based input. Finally, application researchers focus on creating novel, real-world programs that utilize existing pen-based technologies.

A. Hardware Research

The idea of interacting with computers via stylus input began in 1964 with the work of Ivan Sutherland's Sketchpad system [46], which was developed even before the invention of the computer mouse [47]. Although the system contained no form of recognition and used modal buttons and switches to input graphical models, it showed that early computer graphics researchers initially envisioned users inputting information into a computer via a pen-based medium. Figure 2 shows Sutherland interacting with his Sketchpad system using his novel light pen.

In 1969, RAND Corporation took pen-based computing one step further [48]. Unlike Sutherland's Sketchpad system, RAND Corporation's GRAIL (GRAphical Input Language) system did not require the use of modal switches to input diagrams and

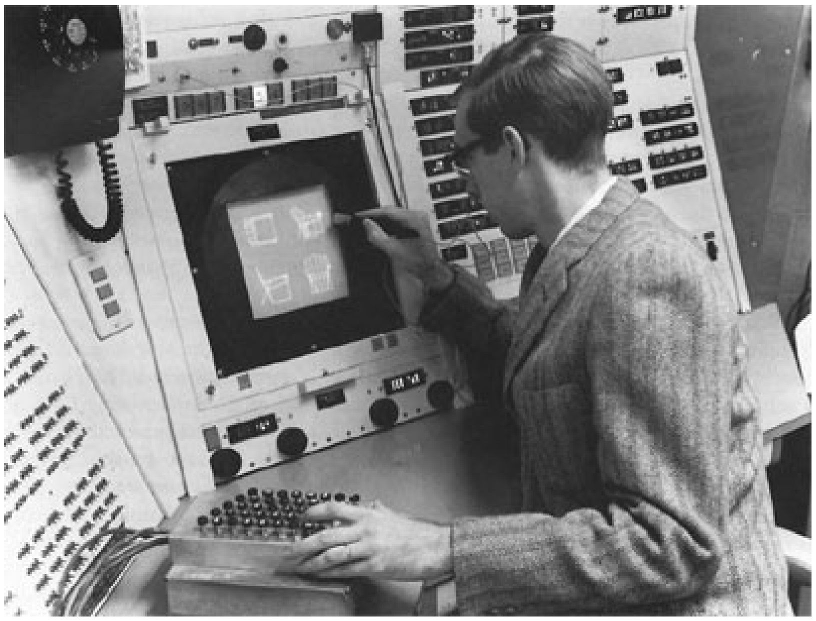


Fig. 2. Ivan Sutherland's Sketchpad system.

flowcharts. Instead, integrated software solutions were used to perform recognition based on domain-specific context.

MIT researcher, Nicholas Negroponte, arguably may have been the first to coin the term "sketch recognition" when he introduced his HUNCH system [49]. According to Negroponte, "sketch recognition is the step by step resolution of the mismatch between the user's intentions (of which he himself may not be aware) and his graphical articulations" [50]. The primary motivation of the HUNCH system was that pen-based devices should not just simply computerize a user's input, but should also aid them in their design through the recognition of the user's intention. These seminal works helped pave the way for today's modern pen-based computing research.

The current pen-based computing devices of today vary from portable handheld units to large, shared workspaces (see Figure 3). Personal digital assistants (PDAs) like the Palm Pilot and Apple Newton allowed for stylus-based input and were pre-

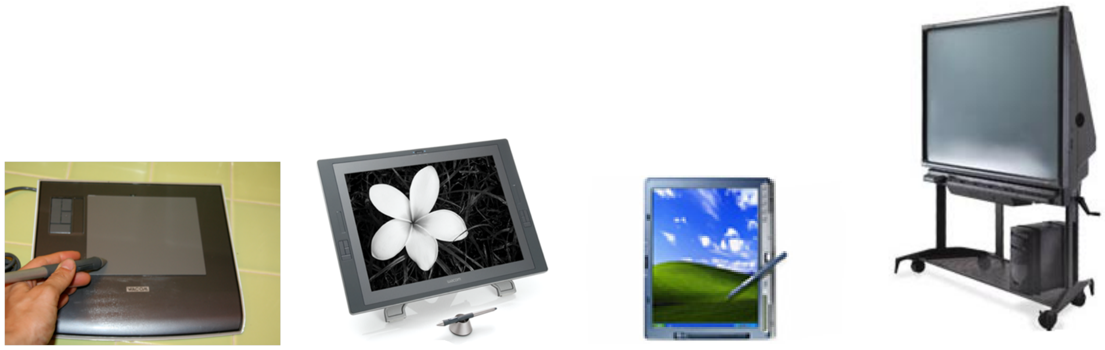


Fig. 3. Various hardware devices that allow for pen-based input.

cursors for many of the current “smart” phones that we see today [51, 52, 53, 54]. Desktop systems can allow for pen-based input either through external tablets [55], or through interactive pen displays that allow users to draw directly on the screen [56]. As laptop computers have become popular, so too have Tablet PCs, which also allow users to draw directly on the laptop monitor [57, 58]. Electronic whiteboards have also become popular, because they provide a large, shared workspace that promotes collaboration amongst users [59, 60]. For those who still prefer pen and paper over their electronic counterparts, devices exist which allow ink markings on paper to be translated into an electronic form [61, 62]. For more information on the hardware advances between today’s modern pen-based devices and those from the days of Sketchpad, we refer the reader to Meyer’s pen computing survey [63].

In addition to the pen-based devices mentioned previously, new advances in hardware have been made which could lead to new applications for interaction researchers. For example, multi-touch displays allow users to interact with programs using multi-finger gestures on the screen [1]. This concept has also been expanded to include bimanual pen and touch interaction [2]. Furthermore, some researchers have focused on creating portable projectors that allow users to sketch on any surface using an

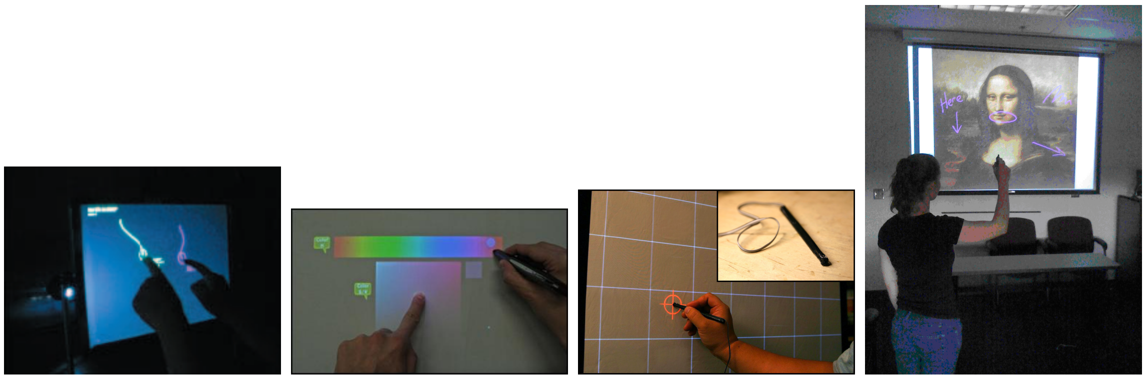


Fig. 4. Examples of new pen-based input devices currently being developed (from left to right): multi-touch displays [1], bimanual pen and touch displays [2], IR projected sketching systems [3], and surfaceless pen-based interfaces [4].

infrared (IR) pen [3]. Still, others have focused using IR technology to create writing devices that are not bound to any physical surface [4]. Devices such as these could potentially become new forms of interaction and lead to new applications for sketch and gesture recognition researchers. Figure 4 shows some screenshots of these devices.

B. Human-computer Interaction

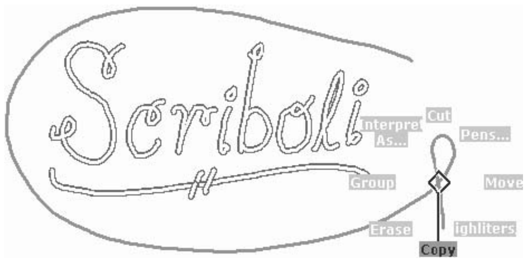
Many researchers have focused on the HCI aspects of sketching. For example, Alvarado created rules and guidelines for developing sketch recognition user interfaces (SkRUIs) [64]. Davis addressed the problem of creating “Wizard of Oz” studies for sketch-based interfaces [65]. Eoff and Hammond looked at the physical manner in which user’s sketch and realized that properties like pen tilt and pressure could be used to perform user identification [66]. Other HCI aspects of sketch-based interfaces include editing, gestures, beautification, toolkit development, and multimodal systems.

1. Sketch Editing

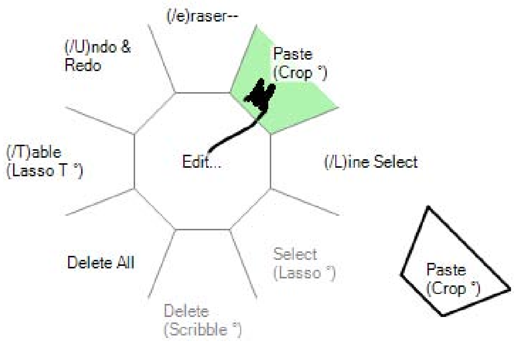
Due to the fact that the pen is used both to create elements and to modify elements, many HCI studies have focused on how to edit sketch elements once they have been drawn (e.g., scale, translate, rotate, cut, copy, delete). Saund and Moran's PerSketch system focused on utilizing the principles of Perceptual Organization in order to perform complex editing tasks [67]. PerSketch followed a draw/select/modify paradigm and used a modal button press to switch between drawing and editing modes. Saund and Lank would later show that a modeless switch was possible using pen trajectory and context [68]. The principles of Perceptual Organization were also used in ScanSribe to form composite groupings of prime sketch elements and also showed to be promising for grouping text strokes [69, 70]. Other sketch editors used recognition themselves to perform different editing tasks. For example, Zhu et al. recognized drawn circles along a stroke which could then be used as control points for stroke manipulation [71].

2. Incorporating Gestures

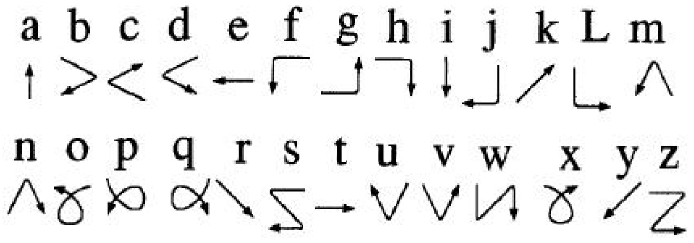
Many systems have used sketched gestures to perform editing tasks since gestures have been shown to be easier to remember than textual commands [72]. For example, Hinckley et al.'s Scriboli system used a pigtail gesture at the end of a lasso, where the end direction of the pigtail denoted a different operation (move, cut, copy, delete) [73]. Zeleznik and Miller showed how prefix flicks and postfix terminal punctuation (e.g., pause, tap, click) could be used to disambiguate sketched gestures from creative ink [74]. Sketched gestures have also been used as substitutes for alphanumeric characters in order to write simple textual letters [75]. Examples of these systems are seen in Figure 5.



(a) Scriboli's pigtail gesture and associated pie menu [73].



(b) Zeleznik's prefix flicks (shown as slashes) and postfix taps (shown as degree signs) [74].



(c) Unistroke gesture substitutes for alphanumeric characters [75].

Fig. 5. Example usage of gestures in pen-based interfaces.

3. Beautification

Another area of concern for HCI researchers has been beautification, both on the local stroke level and global sketch level. Beautification is the process of taking the user's rough strokes and translating them, through recognition, into a more formal form. Even though people prefer high-fidelity, beautified output because of its more professional look [28], it has been shown that users are more creative with the original, informal strokes, because the sketch feels more flexible and capable of being modified [27].

Some algorithms have focused on the beautification of entire sketch symbols and the relationships between them such as symmetry, congruence, collinearity, and horizontal and vertical alignment [5, 76]. This can be considered a form of *global sketch beautification* (see Figure 6). Others have focused on beautifying on a stroke by stroke basis [6, 12, 41] (i.e., *local beautification*, as seen in Figure 7). Beautification of strokes can either be triggered manually by the user or can occur automatically after the stroke has been drawn. In some cases, strokes can be beautified in real-time while the user is still drawing the stroke, as seen in Figure 8 [8].

The time of beautification is important to HCI researchers. For example, if results are shown too soon, it may distract the user. However, if results are delayed, misrecognitions will not be noticed until later in the sketching process, and recognition errors could propagate. In this dissertation, algorithms that can recognize and beautify basic shapes are presented. While the algorithms return beautified shape interpretations, the time of beautification is left to the application developer who uses the recognition algorithms. For more information on the advantages and disadvantages of different forms of beautification, please see [77, 78].

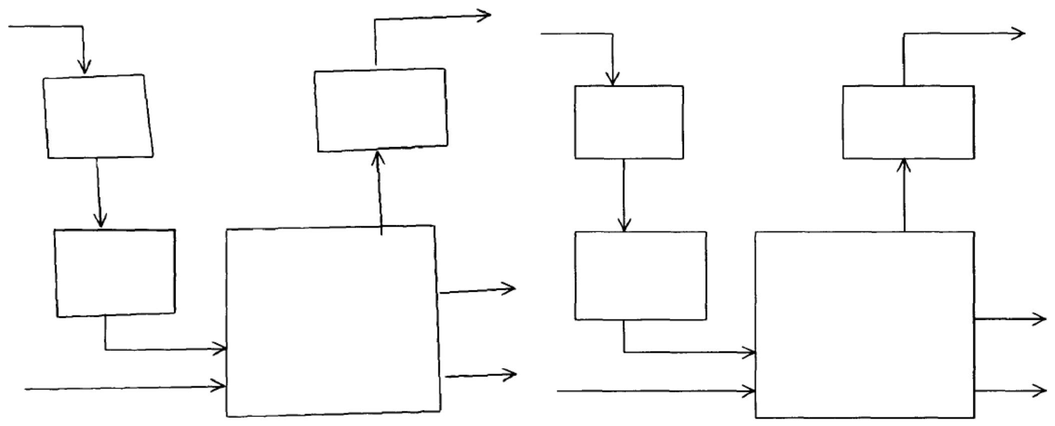


Fig. 6. Example of global beautification, in which locally-beautified symbols are cleaned up based on properties related to symmetry and congruence [5].

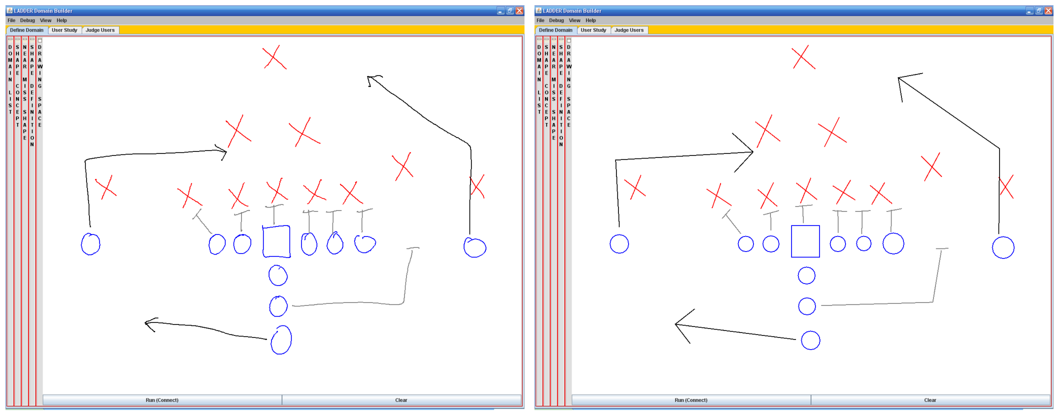


Fig. 7. Example of local beautification performed by PaleoSketch [6] within the LADDER sketch recognition system [7].

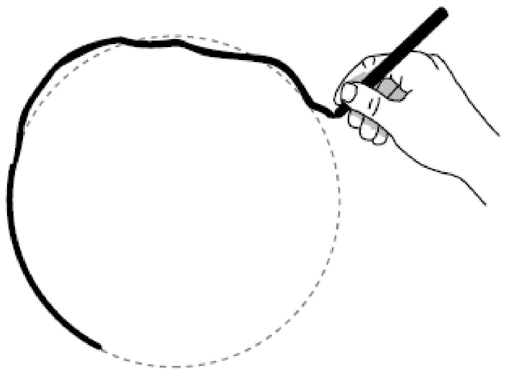


Fig. 8. Example of real-time beautification performed as the user is drawing the stroke [8].

4. Toolkits

In order to aid application developers, many researchers have created toolkits for supporting various sketch tasks. To support gestures in sketch-based interfaces, Wobbrock et al. developed an easy-to-implement algorithm for recognizing user-defined, sketched gestures [42]. Tools, such as "quill," can be used to further help developers as it was able to analyze gesture sets to determine which gestures may easily be confused with one another [79]. Other toolkits, like Burlap, have supported models of ambiguity and provided mechanisms for selecting alternative interpretations when sketches are misrecognized [80, 81]. Still, other toolkits have supported the entire creation of informal ink-based applications with full recognition and editing capabilities [82, 83].

5. Multimodal Systems

Finally, because sketch recognition is still far from perfect, many researchers have looked into combining sketch with other modes on input, like speech, to better recognition. For example, Kullberg created a calendar application that could be edited using a combination of gesture recognition and speech [84]. Speech has also been used in sketch-based mechanical engineering diagrams to aid recognition, as well as to help specify force and motion parameters [32, 85]. Sketch and speech have further been utilized in many military course of action applications [86, 87].

C. Recognition

Recognizing an entire sketch is a complex task, because sketches are "ambiguous," "dense," and "replete" [25]. Often a sketch conveys much more than just a single piece of information. UML diagrams show class objects and the relationships between them. Mechanical schematics show not only devices and bodies, but the forces and interactions between them. Because a sketch typically contains multiple objects and forms of information, it is commonplace to use hierarchical schemes to perform recognition. A UML diagram gives a snapshot of the inner working of an entire piece of software. However, this diagram is made up of class objects and interactions, which are represented as rectangles and arrows. These rectangles and arrows can be broken down further into a set of lines. And taken to the extreme, lines can be broken down even further into points. In this dissertation, the term *high-level recognition* refers to the process of finding meaningful, domain-specific symbols within a drawn sketch (e.g., a class object, an anchored body, an OR gate). *Low-level* or *primitive recognition* refers to the process of interpreting an individual stroke, or group of strokes, as a basic, domain-ignorant shape (e.g., line, ellipse, curve). The differences in labeling between high-level and low-level recognition can be seen in Figure 9.

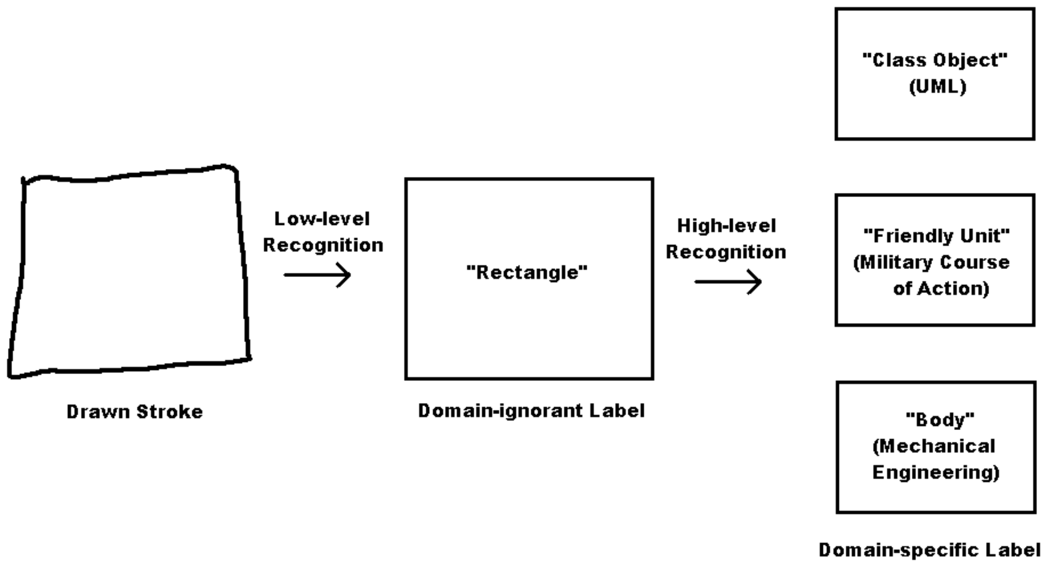


Fig. 9. Example of the distinction in labeling between low-level and high-level recognizers. Low-level recognizers assign domain-ignorant labels while high-level recognizers assign domain-specific labels.

D. High-level Recognition

Most sketch recognition systems have employed a framework similar to the one shown in Figure 10. Drawn strokes are interpreted as one of any supported shape primitive. These recognized primitives are then given to the high-level recognizer which attempts to identify domain-specific constructs. In order to support a large variety of sketch domains, many algorithms have utilized shape grammars to define the symbols of a given domain [83, 88, 89, 90, 91, 92]. These grammars define symbols in terms of a set of shapes that meet particular spatial and geometric constraints (e.g., Figure 1).

Calhoun et al.'s system recognized symbols using semantic networks and a dictionary of learned shape definitions [90]. These shape definitions described the symbols in terms of lines and arcs and the geometric relationships between them. Other languages, like LADDER, have supported more primitive shapes and were capable of

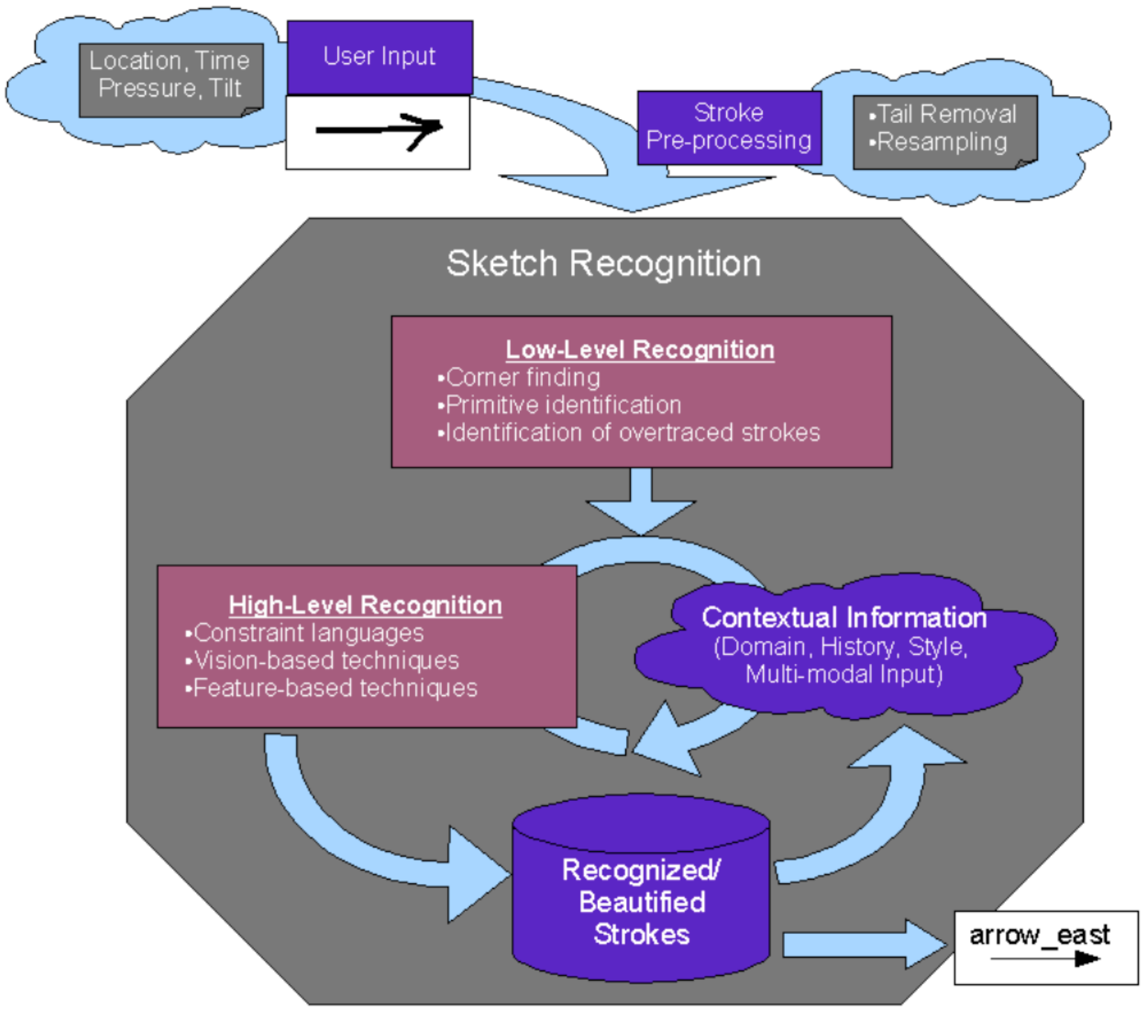


Fig. 10. Example sketch recognition framework.

expressing other details of the sketch like editing and display properties [83]. In some cases, systems have utilized additional, domain-specific context to help aid recognition [30, 93, 94, 95, 96]. Constraints can also be defined using statistical models, such as a Bayesian network, to provide confidence values for a particular symbol interpretation [97]. Because symbol definitions can be tedious to define, some researchers have created methods for automatically generating shape grammars from drawn examples [98, 99, 100].

In addition to shape grammars, other high-level paradigms exist as well. Sharon and van de Panne used constellation models from computer vision to model the spatial relationships between the components of a sketched symbol [101]. Sezgin noticed that people often tend to draw symbols in a particular order [102]. For example, when drawing stick figures, users tend to start with the head, followed by the torso, and then the limbs. Therefore, he used Hidden Markov Models (HMMs) to help uncover the temporal patterns in users' drawing. These temporal patterns have also been used by other researchers to create sketch applications that adapt to one particular user [103]. Temporal and contextual information have also been captured in other systems using conditional random fields (CRFs) [104].

E. Low-level Recognition

Low-level recognition is the process of assigning a domain-ignorant label to a stroke or group of strokes. This label represents the name of one of many primitive shapes which are used as part of the building block vocabulary for high-level shape grammars. As the number of supported primitive shapes increases, so too does the expressiveness of the high-level shape grammar. Thus, it is important to not only have accurate primitive recognizers, but to also have primitive recognizers that support many differ-

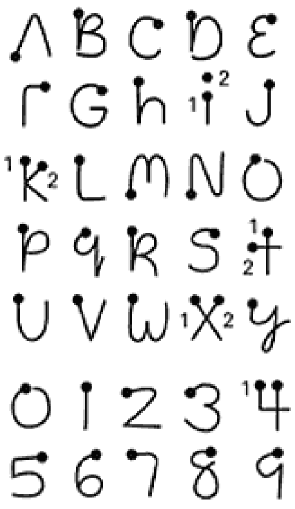


Fig. 11. Example gesture set from Graffiti. Some of these gestures may not be natural to the user and must be learned before interacting with the system.

ent shapes. These recognizers may either require training for the end-user to perform, or require no prior sketch examples at all. Some may be *online* algorithms that utilize the timing or ordering of sketched points, while others are *offline* and require only the x and y pixels of the sketch. The classes of low-level recognizers have traditionally been broken down into three categories: *motion-based*, *appearance-based*, and *geometric-based* recognizers.

1. Motion-based Recognition

Motion-based recognizers got their start from algorithms meant to interpret gestures that represented either editing commands [43] or alphanumeric characters [75, 105, 106, 107], but have also been used as primitive recognizers in some sketch systems. Because these gesture sets are typically large and may not be initially obvious, there is often a learning curve associated with new users (see Figure 11). Motion-based recognizers concern themselves with classifying shapes based on how the individual

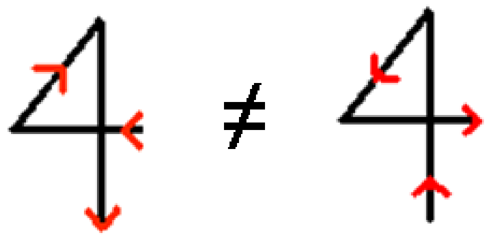


Fig. 12. Motion-based methods focus on how a stroke was drawn in order to perform recognition. In this example, even though the shapes look the same visually, they would actually be considered different gestures because they were drawn with two different motions.

strokes were drawn, and not on what the stroke actually looks like, as seen in Figure 12. Therefore, these types of algorithms often place constraints on how users must draw particular shapes in order to be recognized. For example, a user may naturally draw circles using a clockwise motion, but may be forced to draw circles in a counter-clockwise direction in order to be recognized. Drawing habits like these are hard for new users to break. Often times, users are also required to pre-train the system and must provide numerous example sketches before using the application.

In 1991, Dean Rubine proposed a gesture recognition toolkit, GRANDMA, which allowed single-stroke gestures to be learned and later recognized through the use of a linear classifier [43]. He proposed thirteen features that could be used to describe any single-stroke shape. Such features included the sine and cosine of the starting angle of the stroke, length and angle of the bounding box diagonal, total and absolute rotation, and maximum speed. He also provided two techniques for rejecting bad gestures. To date, Rubine's recognizer is probably the most widely used motion-based algorithm in sketch and gesture recognition [19, 79, 82, 84, 97, 108, 109, 110].

Rubine's work was later extended by Long et al. [111], who determined a new feature set that consisted of eleven of Rubine's features along with six of their own.

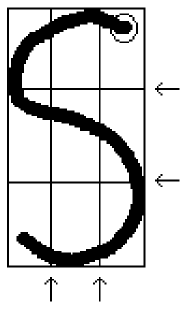


Fig. 13. Example of the reference lines generated by the Ledeen recognizer.

This feature set was chosen after multi-dimensional scaling was used to determine the most relevant features. Like Rubine's recognizer, a linear classifier was used to classify single stroke shapes. Other modifications of Rubine's algorithm include the changes made in InkKit, in which features related to absolute size were replaced with ratios instead [112].

The Ledeen recognizer formed a 3x3 grid from the bounding box of a drawn stroke, or set of strokes [105]. As seen in Figure 13, a 3x3 grid generates four interior reference lines. The recognizer calculated the number of times the stroke intersected each reference line, and made note of the starting location of the stroke. It also looked at the angles between strokes in the case of multi-stroke gestures. Based on this information, the algorithm performed recognition. The recognizer from the Electronic Cocktail Napkin system used a similar recognizer, but also included information related to the aspect ratio of the strokes, as well as, the number and location of corners [93, 94, 95].

In order to avoid feature selection and discovery in gesture recognition, Choi et al. used manifold learning to recognize multi-stroke gestures [113]. Kernel Isomap is utilized, along with a novel dissimilarity metric that accounts for both spatial and temporal information in a stroke. By using the features extracted through manifold learning, the authors were able to show that their approach outperformed the Rubine recognizer on datasets consisting of alphabetic characters, numbers, and mathematical symbols.

2. Appearance-based Recognition

Some systems have looked to computer vision to help solve the sketch recognition problem. These appearance-based algorithms have focused strictly on how a sketched shape looks, and have used some form of template-matching to compare a candidate symbol to others in a learned catalog; the timing and ordering of points is completely ignored.

Hse and Newton performed recognition using Zernike moments [44]. Other algorithms utilized histograms either at the stroke level [114] or the pixel level [115] to perform recognition. In order to handle variations in scale and orientation, algorithms, such as the congealing algorithm proposed by Learned-Miller, may apply various affine transformations to symbols before comparing templates [116]. Wobbrock et al. used a template-matching algorithm that can handle differences in orientation by rotating symbols along their "indicative angle," the angle between the centroid of the stroke and its starting point [42]. When performing template matching, many algorithms will use a simple Euclidean distance metric, while others may use more complex distance metrics like Hausdorff distance [117, 118, 119, 120]. Rather than perform template matching directly on a particular sketch, Ouyang and Davis performed template matching on feature images that are generated based on stroke orientation along set of reference angles [121].

Although these appearance-based systems have the advantage of increased drawing flexibility and extensibility, they also have a notable downfall. The main disadvantage of these appearance-based recognizers has been their inability to handle shapes

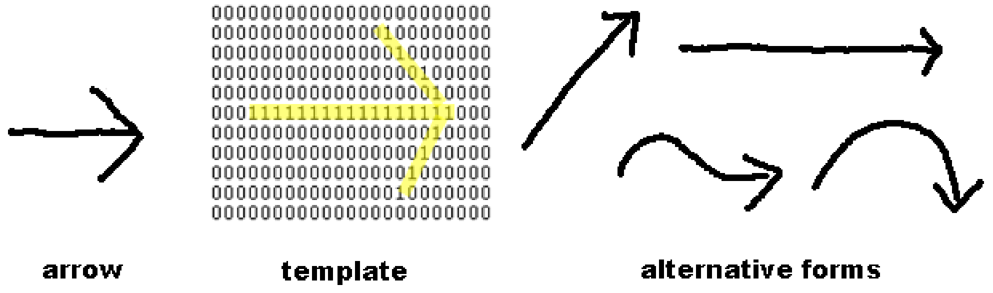


Fig. 14. Appearance-based methods typically use some form of template matching to perform recognition; however, alternative variations of the same symbol would require multiple templates.

that can be drawn in an arbitrary number of configurations; thus, they require a separate template for each variation of a shape. Imagine an arrow that can have an unlimited number of acceptable paths (straight line, curved, squiggled, etc.), as seen in Fig. 14. Essentially, the training space for these arbitrary symbols would be infinite, requiring the user to provide explicit examples of every possible configuration of the shape. In most domains, this would be boundless and unreasonable.

3. Geometric-based Recognition

Because gesture-based techniques place drawing constraints on users, and because appearance-based techniques have trouble handling shapes of arbitrary configuration, a shift to the use of geometric-based recognizers has occurred. To allow for variance in scale and orientation, these types of recognizers use geometric formulas to describe basic building block shapes, or primitives (see Fig. 15). An additional benefit of this approach is that primitive shapes can be automatically beautified, because the parameters needed to perform beautification are also used for recognition.

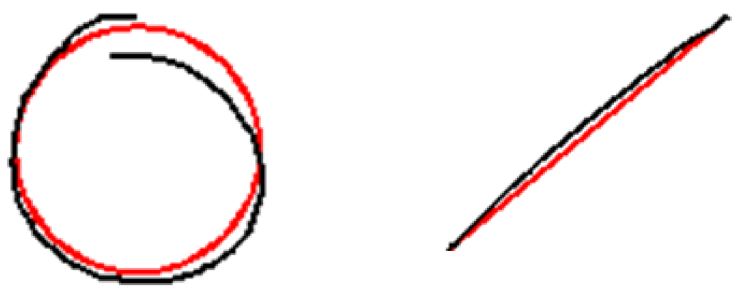


Fig. 15. Geometric-based methods estimate an ideal shape representation (red) of the stroke (black) and then compute shape approximation errors to perform recognition. For example, a circle can be estimated by finding an approximate center (e.g., the center of the bounding box) and an approximate radius (e.g., by taking the average distance of each stroke point to the center). Lines can be estimated by simply connecting the endpoints of the stroke.

a. Corner Finding

One of the important sub-processes of geometric-based techniques is called *corner finding*. This has also been referred to as *segmentation* [122], *fragmentation* [123], *vertex detection* [9, 12, 41], or cusp detection [45]. The goal of corner finding algorithms is to break down a single stroke into a its most basic primitives. For example, if a user draws a square, then the algorithm's goal is to return the four, linear sub-segments. Some algorithms are meant to work solely on polyline strokes, while others can be used on curvilinear segments (e.g., Figure 16). The corners of strokes provide important perceptual information when performing geometric-based recognition.

Polyline corner finders are meant to reduce a stroke down into its basic linear sub-components. One of the first, and most popular, polyline corner finding approaches is the Douglas-Peucker algorithm [10]. This algorithm was originally intended to reduce the number of points along a contour in cartography map representations. A visual example of the algorithm can be seen in Figure 17. One of the downfalls of the Douglas-Peucker algorithm is that it runs in quadratic time - $O(n^2)$ - in the number

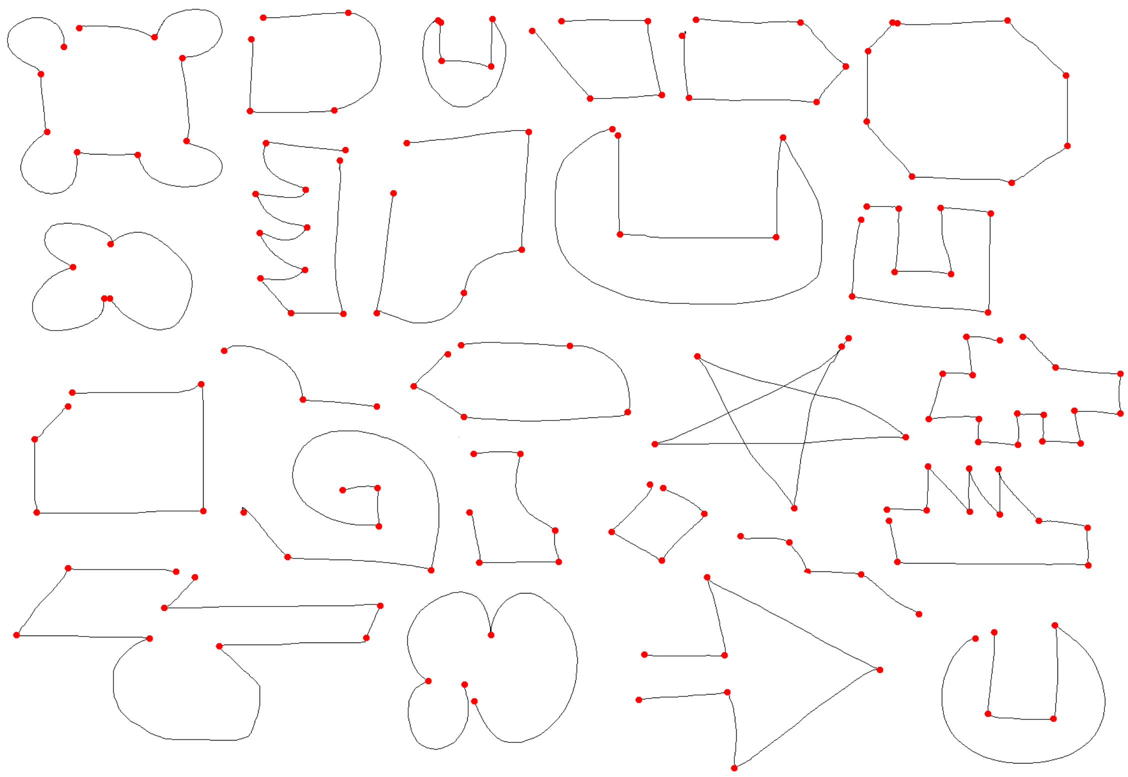


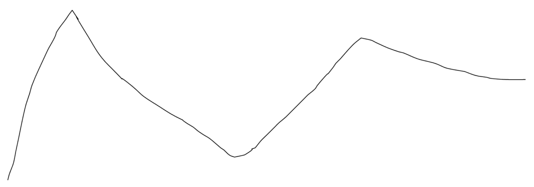
Fig. 16. Example results of corner finding on curvilinear stroke. Results are from MergeCF [9].

of points in the stroke. This led to additional modifications by other authors, which have since improved the algorithm to a *O(n log n)* running time [124].

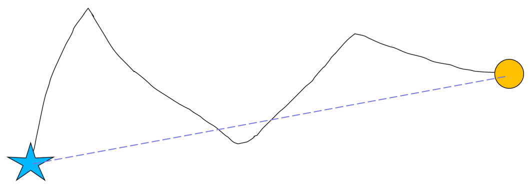
ShortStraw is a more recent polyline corner finder that has the extra benefit of being easy to implement [11]. It starts by first resampling the points of the stroke so that each point is equidistant to its neighbor. Next, a series of "straws" are computed within a fixed window size around a candidate corner (Figure 18). Those points with the shortest straws are considered the ideal corners of the stroke.

One of the most significant discoveries of the ShortStraw paper, was that existing corner finding approaches did not factor in false positives into their accuracy calculations. Thus, they were not penalized for oversegmentations (i.e., those that produced too many corners). To account for this, the authors of ShortStraw developed a new "all-or-nothing" accuracy measure that penalizes for false positives. Essentially, this accuracy denotes the percentage of the time that a stroke is segmented correctly with no missing corners and no additional corners. With this measure they determined ShortStraw was 74% accurate, an improvement over the 30% accuracy of the next best corner finder. Since its publication, many researchers have implemented improvements to the ShortStraw algorithm, some of which have increased all-or-nothing accuracy to above 99% [125].

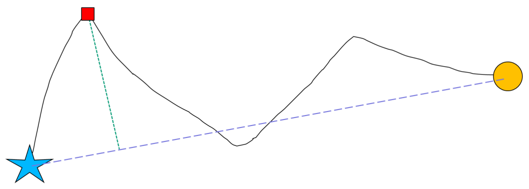
Unlike the Douglas-Peucker and ShortStraw algorithms, some corner finders also work for curvilinear strokes as well as polyline strokes. Although some have used sophisticated techniques, such as dynamic programming, to perform segmentation [123], most have used simple approaches related to curvature and speed calculations. The *curvature* of a given point is defined as the change in direction at that point, where *direction* is the arctangent angle of the change in y over the change in x. Equations 2.1 and 2.2 show the formulas typically used for direction and curvature. In these equations, n represents the index of the current point, k denotes a neighborhood



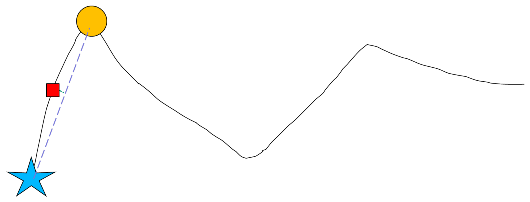
(a) The user draws a stroke.



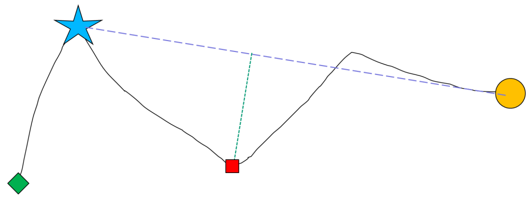
(b) The algorithm begins by selecting the endpoints as corners. One is selected as an anchor (star), and one is selected as a floater (circle). Next, an ideal line is generated between the two points.



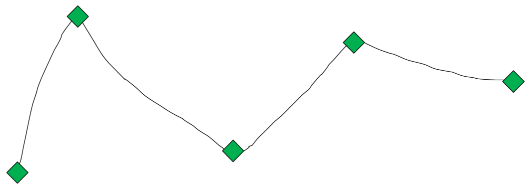
(c) The point that is furthest away from the optimal line is tested as a candidate corner. If this point is far enough away from the optimal line (within some threshold), then it is added as a corner and becomes the new floater.



(d) Next, the algorithm repeats steps (b) and (c). In this case, the candidate corner is not far enough to be added.



(e) The floater moves to the end of the stroke and the previous floater becomes the new anchor. Steps (b) and (c) are then repeated.



(f) The process continues until the anchor reaches the end of the stroke.

Fig. 17. A walkthrough of the Douglas-Peucker algorithm [10].

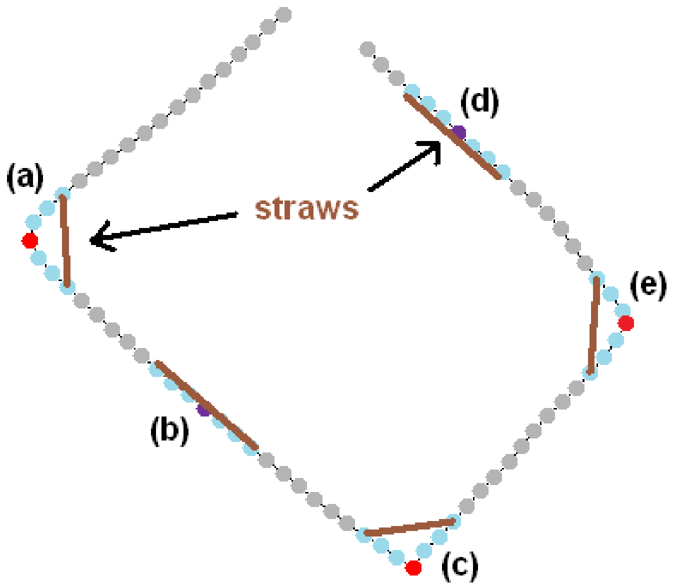


Fig. 18. Example of the "straws" computed through the ShortStraw algorithm [11]. In this example, straws (a), (c), and (e) are shorter and more likely candidates for corners than the longer straws of (b) and (d).

size around a point (typically 1 or 2), D is a function that computes the path length between two points, and φ is a shift function that ensures that curvature values remain between π and $-\pi$. This shift function keeps the curvature graph continuous.

$$d_n = arctan(\frac{y_{n+1} - y_n}{x_{n+1} - x_n}) \tag{2.1}$$

$$c_n = \frac{\sum_{i=n-k}^{n+k-1} \varphi(d_{i+1} - d_i)}{D(n-k, n+k)} \tag{2.2}$$

Some algorithms have searched for corners simply by finding the points of highest curvature [12], but some have also observed that corners can also be found at the points of lowest speed [41, 126]. These points are typically found using global thresholds that are a function of the mean or median values. In some cases, these thresholds are lowered in order to produce many false positive corners, which is then followed up with a post-process stage that merges corners that have little effect on the overall error of the fit [9]. Rather than use global thresholds, Kim and Kim took an alternative approach and utilized local information, such as local convexity and local monotonicity, to find corners [122]. However, because of the resampling required by this approach, oversmoothing can cause some corners to be missed [9].

b. Primitive Recognition

After corner finding has taken place, each sub-segment can then be recognized as a primitive shape. Sezgin, Stahovich, and Davis described a single-stroke algorithm that was composed of an approximation stage, a beautification stage, and a recognition stage [41]. Corners were detected by finding the points of highest curvature, along with the points of lowest speed. Hybrid fits between these two sets of points were calculated using an average-based filtering technique. Next, the error of each fit was determined by using an orthogonal distance squared error. The system was limited to only a few primitive shapes: lines, curves, ellipses, and complex fits (those composed of a combination of lines and curves). For complex fits and vertex approximation, the authors reported an accuracy of 96%.

Yu and Cai presented an alternative single-stroke primitive recognizer [12]. Corners were detected using only curvature data. The most significant contribution of Yu and Cai was the introduction of their feature area error metrics (see Figure 19). The shape set of the Yu and Cai recognizer was expanded to include lines, polylines, circles, ellipses, arcs, and helixes. For primitive shapes, they achieved near 98% accuracy; however, they admit to not having produced the recognition rates of Sezgin for complex shapes.

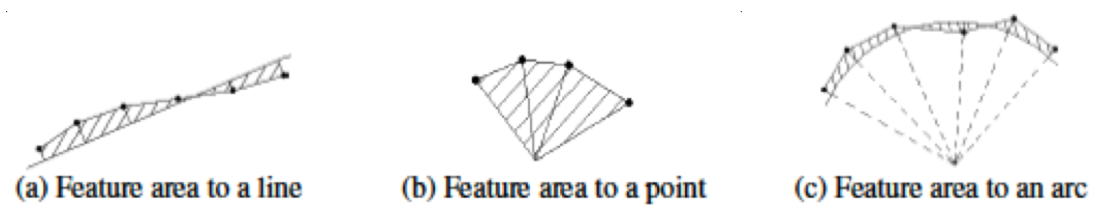

(a) Feature area to a line
(b) Feature area to a point
(c) Feature area to an arc

Fig. 19. Feature area examples of the shapes supported by the Yu and Cai recognizer [12].

c. Limitations of Geometric-based Approach

The primary benefit of the geometric-based approach is its ability to describe the symbols of many different domains. Because geometric formulas are used to describe shapes, problems related to scale and orientation often become non-issues. Furthermore, the geometric-based approach is the least limiting on the user. It normally does not require special drawing constraints like the gesture-based paradigm, nor does it require user-specific training. Because of these benefits, it is the primary technique used within this dissertation. Additional motivation for this approach is given in Chapter III.

Although basic visual vocabularies are often followed by users when sketching [127, 128, 129], there are specific types of drawing that are not well-suited for the geometric-based approach. These problem areas typically arise when dealing with non-geometrical shapes that are hard to describe, such as curvy shapes and blobs, handwriting (i.e., text), and shading.

Even though a curve is normally a supported primitive by most low-level recognizers, it is hard for users to describe high level symbols that are composed of many different curves. This is because a) curves can be drawn to an infinite number of degrees, b) control points are hard to define for curves, c) any basic shape can technically be described in terms of a geometric sequence of curves, and d) it is hard to distinguish where one curve starts and another one begins. Most geometric-based recognizers cannot handle artistic drawings because of this specific limitation. Therefore, sketches of animals, people, trees, and others containing abstract shapes are not supported.

Handwritten text is also not typically handled by geometric, sketch recognizers, because it often is drawn in a specific way that may not always follow the same convention across all users. Furthermore, some letters and numbers are hard to describe geometrically. Because of this, most sketch systems handle handwriting through external means either by using the keyboard [29] or by designating special areas as "text-only" where the strokes drawn in that area are sent to an outside handwriting recognizer [130]. Since there exist recognizers that perform well on only iconic shapes (i.e., sketch recognizers), and since there are recognizers that perform well on just text (i.e., handwriting recognizers), some researchers have recently begun to analyze means of distinguishing text strokes versus shape strokes and have shown promising results [131, 132, 133, 134].

Like complex curves, shading is also not supported by the geometric-based approach and is better suited for a vision-based approach because of its arbitrariness. However, a couple of exceptions to this rule do exist. Most recognizers can detect dense areas of points within a stroke, which can denote things like scribbles or filled-in regions [135]. Furthermore, shapes like overtraced lines and ellipses can still be recognized because their geometric formulas are still intact. Overtracing mainly becomes a problem when it occurs within complex shapes or multi-stroke shapes.

4. Hybrid/Combination Techniques

Other algorithms have attempted to utilize the benefits of each type of low-level recognition paradigm by combining approaches in an ensemble manner. For example,

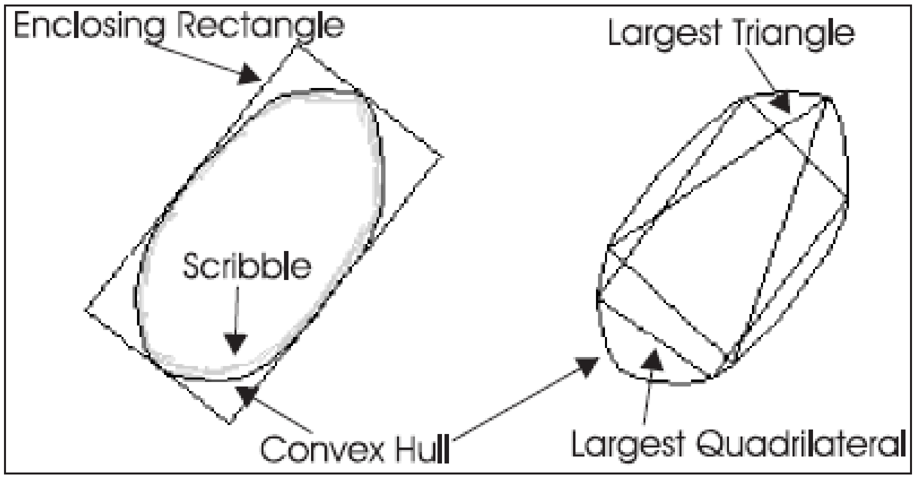


Fig. 20. Bounding polygons that are calculated for the Apte and CALI recognizers [13, 14, 15].

GLADDER attempted to determine which recognizer, either Rubine [43] or PaleoSketch [6], should be used to interpret each stroke as it entered the recognition system [136]. It did this by utilizing a rejection method that first checks the Mahalanobis distance to any of the Rubine-defined glyphs. If this distance exceeded a threshold, then the PaleoSketch recognizer was used instead. The authors found that they chose the correct recognizer and returned the correct result 80% of the time.

Another hybrid approaches have focused on calculating convex hulls and other various bounding polygons (Figure 20) and compare their ratios, which are referred to as shape "filters" [13, 14, 15]. In this sense, these approaches are similar to geometric-based techniques. However, these algorithms are offline (similar to appearance-based techniques), and do not utilize the timing and ordering of stroke points. Apte used these shape filter ratios as features in a decision tree to recognize lines, ellipses, circles, diamonds, triangles, and rectangles with an accuracy of 97.5%. Multi-stroke primitives were also supported as long as the strokes used to compose the shape were drawn consecutively, and within a certain time threshold. The CALI recognizer

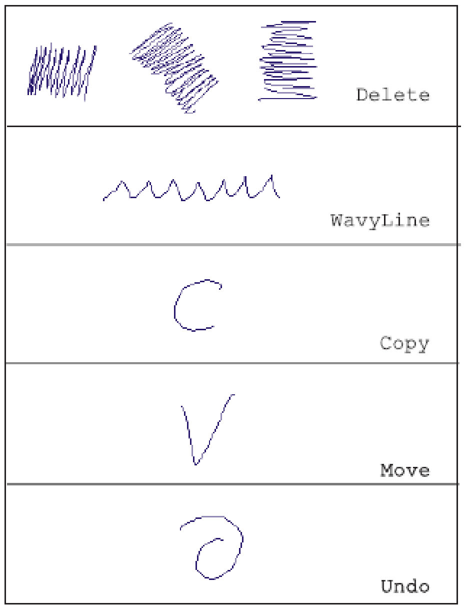


Fig. 21. Gestures supported by the CALI recognizer [14, 15].

extended the work of Apte to include the recognition of the previously supported shapes and arrows, in addition to a set of uni-stroke gestures, seen in Figure 21 [14, 15]. The CALI recognizer used a fuzzy logic classifier and achieved recognition rates around 93%. A primary distinction between this approach and the traditional geometric-based approach is that shapes like diamonds, rectangles, triangles, and arrows are recognized wholistically, whereas with a geometric-based approach, these would first be segmented into lines by a corner finder, recognized as individual lines by a primitive recognizer, and then combined and formed into a specific polygon by a high-level recognizer.

MARQS (Media Albums Retrieved by Query Sketch) is a system that utilized a combination of gestural features (e.g., bounding box aspect ratio), visual features (e.g., pixel density), and geometric-based features (e.g., number of corners, average curvature) to perform recognition of more artistic symbols [16]. The goal of the

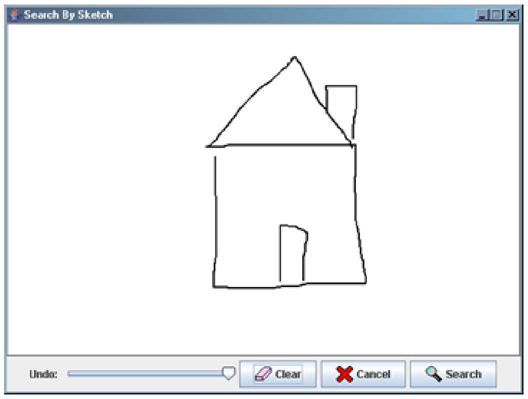
MARQS system was to allow users to retrieve multimedia albums based on their input sketch symbols (Figure 22). It required only a single example of each sketch prior to use, and provided an online learning mechanism that utilized information related to selected queries to better train the algorithm over time. The system was able to return the correct result within the first page of options 98% of the time. It was also shown that the average search rank decreased over time, indicating that the system performed better as the user interacted with the application.

F. Applications for Sketch-based Interfaces

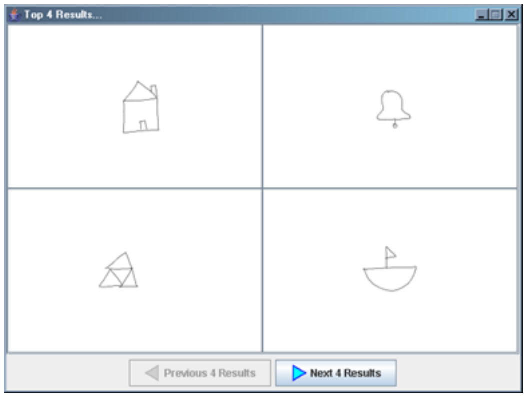
Many applications have been developed that utilize sketch recognition to provide a better or more unique user experience. These applications can be broken down into three categories, based on function: design, engineering and education, and search by sketch.

1. Sketching in Design

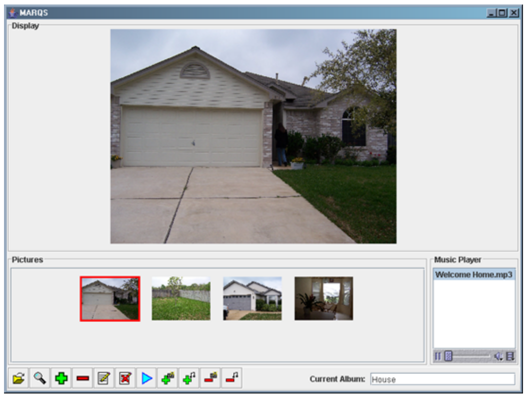
Sketching is a primary input modality during the initial stages of design, because of its informal nature and support for flexibility and creativeness. Most sketch interface developers who work in the realm of design focus on creating tools that support basic computer conveniences (e.g., copy, paste, save) while maintaining the basic creative affordance of pencil and paper. Some even create algorithms for selecting the appropriate design tools based on context [137]. Furthermore, researchers have shown that informal sketching tools support the exploration of a design more so than traditional authorware tools [24]. Because of the focus on the informal nature of sketching, some of these applications may or may not utilize actual sketch recognition.



(a) The user draws a query sketch.



(b) The system returns a list of results. Once the user selects the result they intended, their query sketch is added as a training sample for the associated media album.



(c) The media album associated with the query sketch is returned and displayed.

Fig. 22. Screenshots of the MARQS system [16].

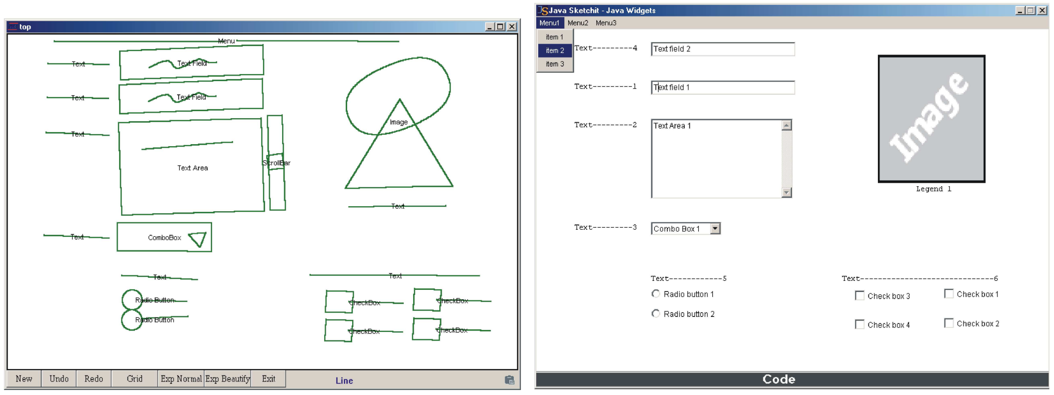


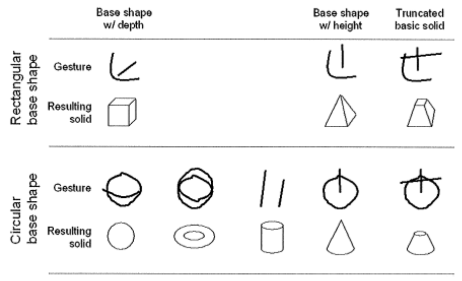
Fig. 23. Example GUI created with the JavaSketchIT application [17].

a. Storyboards

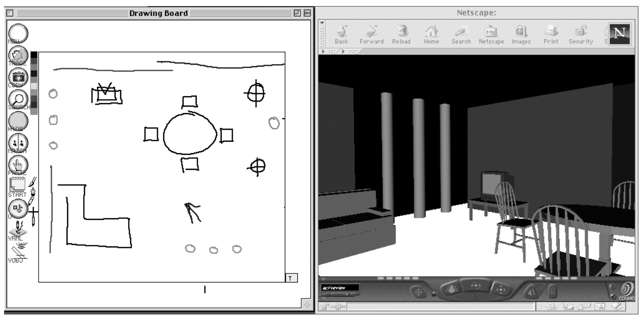
One of the most frequently used tools employed by designers is storyboards. In a recent survey of designers, Myers et al. discovered that 88% of designers use storyboards and 97% of those designers began with sketching. SILK is a toolkit that incorporated recognition and allowed users to sketch basic storyboards and interfaces [108]. Gestures were also avaiable to allow users to perform operations like delete, move, copy, group, and cycle. DENIM built off of the concepts from SILK to create an application that supported the design of websites [138, 139]. Finally, JavaSketchIT allowed users to sketch out graphical user interface (GUI) components which could be recognized with the aid of the CALI recognizer [14, 15] to produce an actual Java GUI, as seen in Figure 23 [17].

b. 3D Modeling

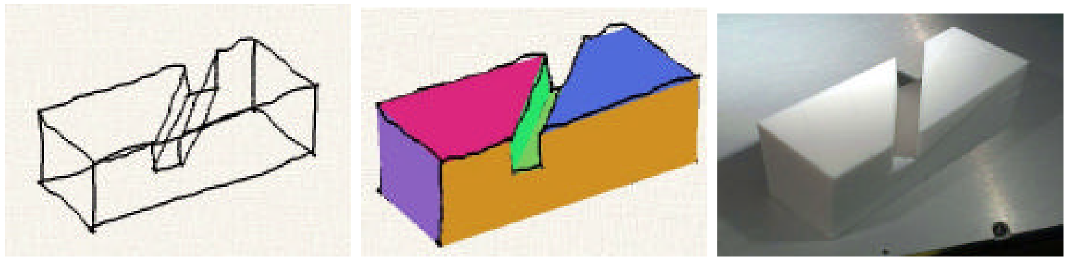
Another principle use of pen-based interfaces in design is 3D modeling. Many researchers have looked into the creation free-form 3D objects generated from 2D sketches. Some systems have taken 2D strokes and interpreted them as a 3D model



(a) Bimber et al.'s gesture set used to create 3D objects [143].



(b) Do's VR Sketchpad system that recognizes 2D shapes which represent 3D objects in a VRML scene [144].



(c) Lipson and Shpitalni's system for recognizing 2D sketches as 3D models which can then be externalized into physical models using 3D printing [147].

Fig. 24. Examples of different 3D modeling applications that utilize pen-based input.

directly [140, 141, 142], while others have recognized simple 2D gestures that represented various 3D forms [143, 144, 145]. With Do's VR Sketchpad, 2D sketches were recognized and interpreted as objects like chairs, tables, and sofas in order to create a 3D VRML floor plan [144, 146]. In some cases, the 3D models generated through sketching can be externalized into physical models using commercial 3D printers [147]. Figure 24 shows some of these systems in work.

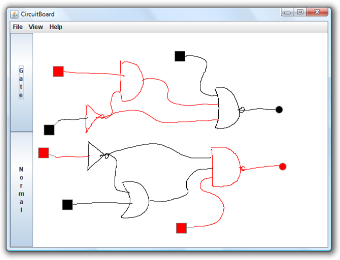
2. Sketching in Engineering and Education

Diagrams are prevalent throughout engineering and education domains due to their ability to convey a lot of information quickly and efficiently. To support the creation of these diagrams, computer-aided design (CAD) tools have been developed which provide a toolbar-based, point-and-click interface. Studies have shown that users not only perform operations faster with sketch-based interfaces, but that sketch-based interfaces also promote creativity better than the traditional CAD systems [24]. Therefore, the goal of most of these applications is to provide a top-level interface that works in conjunction with existing CAD tools.

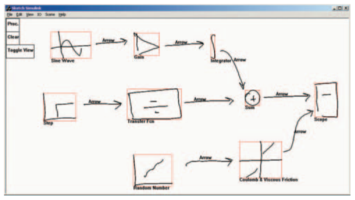
a. Sketching in Engineering

Many different engineering domains have benefited from sketch recognition technology, including electrical engineering [30, 119, 148], software engineering [29, 110, 130, 149, 150, 151, 152], mechanical engineering [18, 31, 32, 153, 154], and even civil engineering [40]. For example, systems like AC-SPARC [148] and Sim-U-Sketch [119] allowed users to sketch electrical circuit diagrams that could be translated into models that were usable in external simulators like SPICE and Matlab's SimuLink. Others, performed simulation directly by allowing users to specify the input values for sketched logic diagrams [155]. Figure 25 shows some examples of electrical engineering applications that utilize sketch recognition.

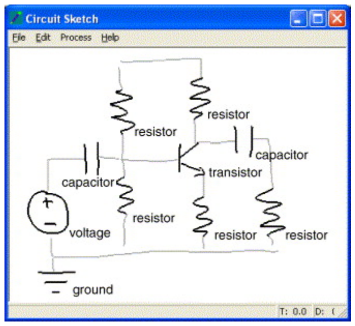
Another application of sketch recognition lies in software engineering. The Unified Modeling Language (UML) is a standardized language for modeling the interactions and design of software components. Many UML models, such as use case, class, and sequence diagrams are capable of being supported with sketch recognition. MaramaSketch provided a sketch interface on top of the existing Marama plug-in for the Eclipse integrated development environment (IDE) [130, 150]. Other software, such as Tahuti, recognized UML class diagrams and could translate them into Rationale Rose models, which have the capability to generate Java code automatically [29]. Figure 26 provides some screenshots of example UML sketch systems.



(a) The CircuitBoard system that performs simulation on sketched circuit logic diagrams [155].

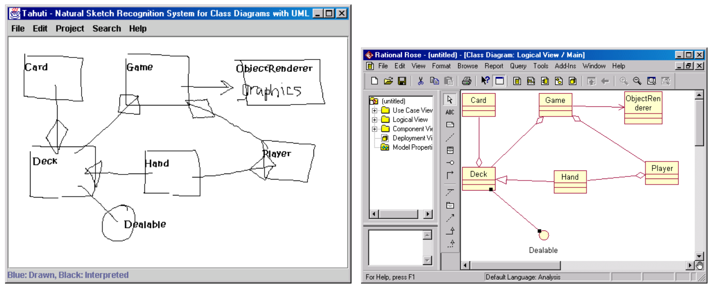


(b) Kara and Stahovich's Sim-U-Sketch system that connects to Matlab's SimuLink software [119].

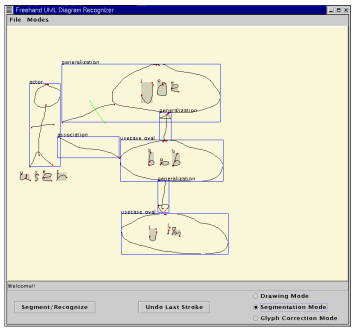


(c) Gennari et al.'s AC-SPARC system that allows users to sketch diagrams that can be simulated in SPICE [148].

Fig. 25. Examples of electrical engineering applications that utilize sketch recognition technology.


Tahuti - Natural Sketch Recognition System for Class Diagrams with UML
File Edit Project Search Help
Card
Game
ObjectRenderer
Graphics
Deck
Hand
Player
Dealable
Blue: Drawn, Black: Interpreted
Rational Rose - [untitled] - [Class Diagram: Logical View / Main]
Card
Game
ObjectRenderer
Deck
Hand
Player
Dealable
For Help, press F1
Default Language: Analysis

(a) The Tahuti system, which translates hand-sketched UML class diagrams into Rationale Rose models [29].



(b) Lank et al.'s system for recognizing various UML diagrams [149].

Fig. 26. Examples of UML sketch systems.

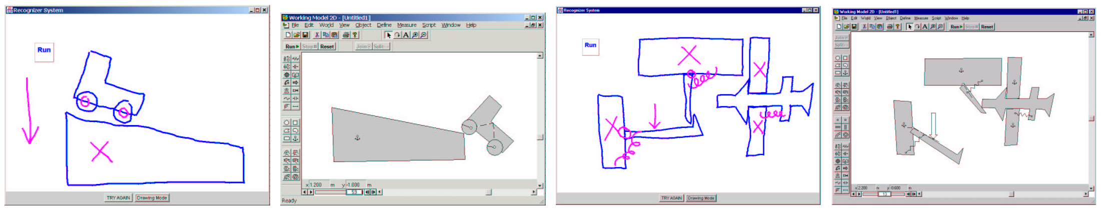


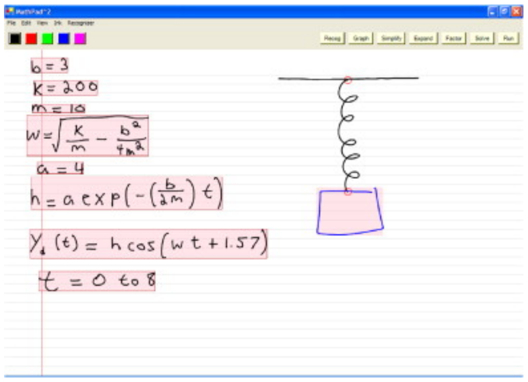
Fig. 27. Examples of mechanical sketches being recognized and translated into simulator models through the ASSIST application [18].

Mechanical engineering applications can also apply sketch recognition to interpret schematic sketches of physical devices [31]. Systems, such as ASSIST (Figure 27), allowed users to draw simple mechanical diagrams that could be brought to life through simulation [18]. With the extension system, ASSISTANCE, users could also include speech recognition to help describe the behavior of the sketched devices [32, 154].

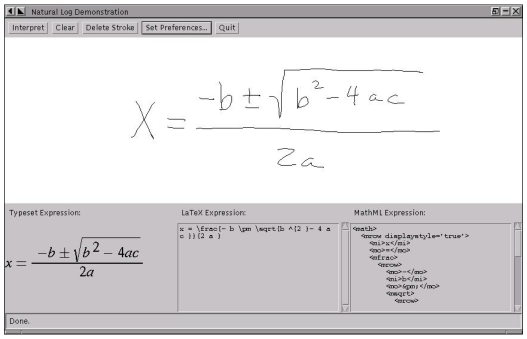
b. Sketching in Math & Science

Applications like MathPad^2, which used an online gesture-based character recognizer [107], have allowed users to sketch out mathematical expressions that can be solved, graphed, or associated with free-form illustrations [34]. AlgoSketch, also known as MathPaper, attempted to combine sketched mathematical expressions with hand-drawn flowcharts to allow pen-based algorithm sketching [156, 157, 158]. Matsakis's recognizer translated hand-drawn equations into their corresponding MathML or LaTeX expressions [159]. Other complex mathematical constructs like constraint satisfaction diagrams [7] and finite state machines [160] have also be supported by many sketch recognizers. Figure 28 shows some of these examples.

Science fields, such as chemistry and biology, also benefit from sketch recognition. For example, Ouyang described a system capable of recognizing hand-sketched chemical structures [36, 161]. In Tenneson's ChemPad, recognized sketches could be



(a) The MathPad^2 system [34].

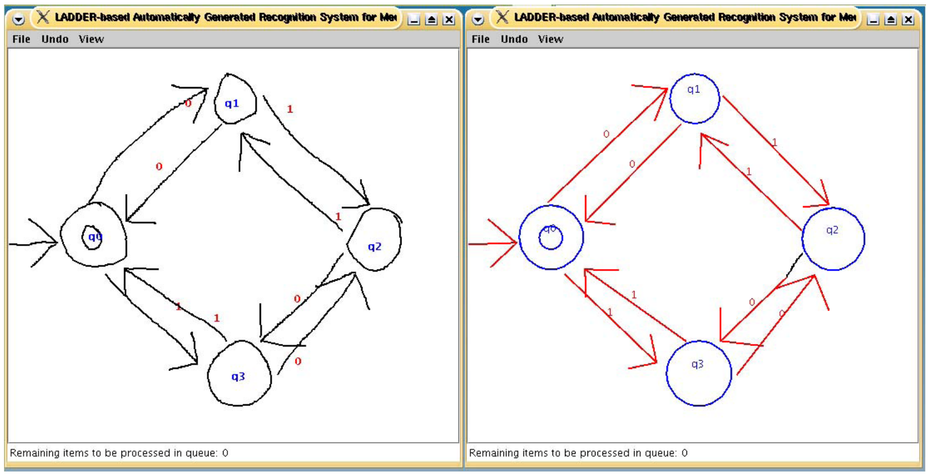


```
x = \frac{- b \pm \sqrt{b ^{2 }- 4 a
c }}{2 a }
```

```
<math>
  <mrow displaystyle='true'>
    <mi>x</mi>
    <mo>=</mo>
    <mfrac>
      <mrow>
        <mo>-</mo>
        <mi>b</mi>
        <mo>&pm;</mo>
        <msqrt>
          <mrow>
```

Done.

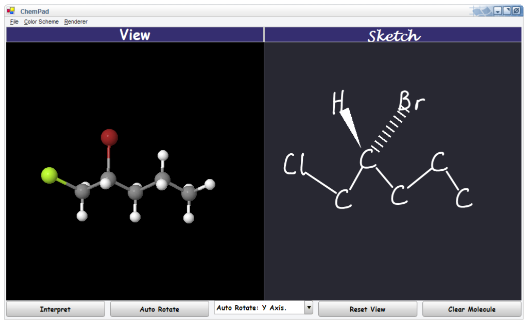
(b) Matsakis's system for translating hand-sketched math expressions into their MathML and LaTeX counterparts [159].


LADDER-based Automatically Generated Recognition System for Me
File
Undo
View
q0
q1
q2
q3
0
1

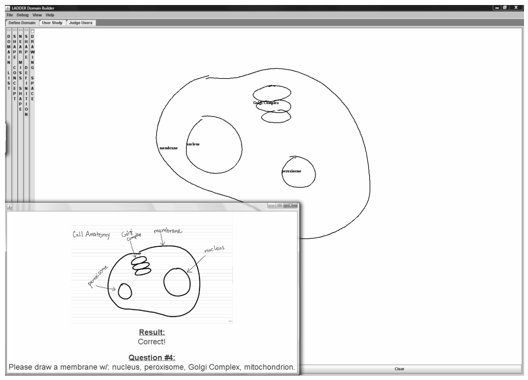
Remaining items to be processed in queue: 0

(c) Hammond's LADDER system recognizing a hand-drawn finite state machine [160].

Fig. 28. Examples of sketch-based, math applications.



(a) The ChemPad system [35, 162].



Result:
Correct!

Question #4:
Please draw a membrane w/: nucleus, peroxisome, Golgi Complex, mitochondrion.

(b) Taele et al.'s hand-sketched, biological cell recognition application [39].

Fig. 29. Examples of chemistry and biology applications.

displayed as 3D molecules [35, 162]. And in biology, sketches could be used to describe concepts like plant and animal cells [39], as seen in Figure 29.

c. Sketching in Language & Fine Arts

In addition to the STEM (science, technology, engineering, and mathematics) areas of learning, sketch recognition can also be used for language and fine art domains. For example, it can be used in grammar classes to perform sentence diagramming [20]. It can also be utilized in computer-assisted language instruction tools that teach beginners the basics of Chinese and Japanese Kanji [37, 38]. Shahzad et al. created similar tools to recognize characters from the Urdu language [163].

Fine arts can also be aided with sketch recognition. Forsberg et al. created the Music Notepad, which allowed hand-sketched sheet music to be composed through the recognition of single-stroke gestures that represented individual notes (see Figure 30) [19]. Art programs, which can range from the creation of simple stick figures [164] to the creation of entire animated sequences [165, 166], have also benefited from sketch recognition. Davis's K-Sketch system allowed novice animators, such as teachers,

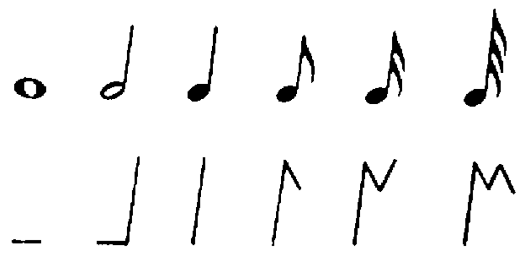
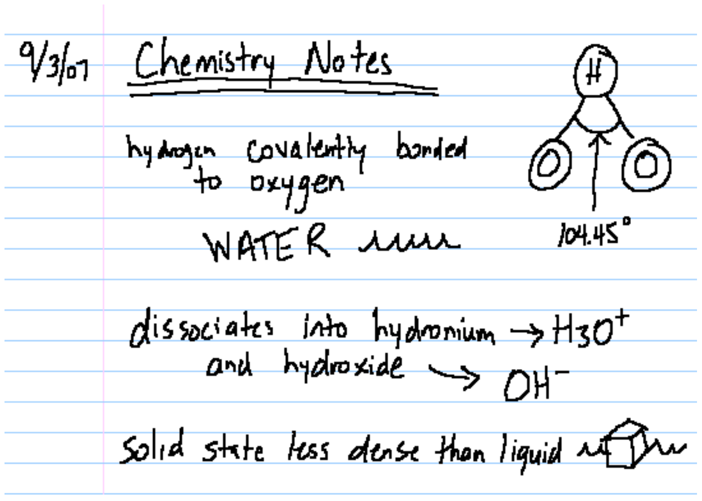


Fig. 30. Gesture set used by the Music Notepad system for composing sheet music [19].

to quickly and easily create informal animations by allowing motion paths to be specified with pen gestures [166, 167]. Sketch recognition has also been employed in conjunction with face recognition technology to create an art instruction program that taught users how to draw the human face [168].

3. Search by Sketch

Finally, sketch recognition has also been used for search by sketch applications. In most cases, sketches are used as queries to search for previously drawn sketches [16, 169], however, some researchers have begun to look toward using sketches as a means to search for real-world images as well [170]. Systems like XLibris allowed for free-form ink annotation while reading [171], and, as mentioned by the authors of MARQS, the end goal of a smart notebook that would allow users to retrieve class notes based on either textual or sketch queries would be significant (e.g., Figure 31) [16].



9/3/07 Chemistry Notes

hydrogen covalently bonded to oxygen

WATER

dissociates into hydronium $\rightarrow H_3O^+$

and hydroxide $\rightarrow OH^-$

solid state less dense than liquid

Fig. 31. Example scenario of a smart notebook where both textual and sketch queries would be beneficial.

CHAPTER III

PALEOSKETCH

A. Introduction

Recall from the previous chapter that multi-domain sketch recognizers follow a framework similar to that in Figure 10. Users make single markings to the screen. The sampling of 3-tuple points (x, y, $time$) sampled between pen-down and pen-up events make up this *stroke*. The stroke is then interpreted as being one of a set of *primitive* shapes by a *low-level recognizer*. The low-level recognizer then passes these shape interpretations into a *high-level recognizer*, which contains a list of *shape definitions* that describe the symbols of the domain. These definitions typically include a list of primitive shapes that compose each symbol, along with a set of spatial and geometric constraints between each component. The job of the high-level recognizer then becomes a problem of grouping a set of recognized primitives into a specific shape definition in order to recognize entire symbols.

PaleoSketch is designed to recognize low-level primitive shapes that are used as the building block vocabulary in these types of high-level sketch grammars. There are many different opinions as to what constitutes a "primitive" shape. Because these primitive shapes are ultimately going to be used in a humanly-descriptive sketch language, our definition of "primitive" is a shape that is hard to describe (for humans) in terms of other geometric shapes. For example, it would be difficult for a person to describe a circle in terms of a set of arcs or curves. However, a person can easily describe shapes like rectangles as a series of lines.

The primary goals of PaleoSketch include:

1. *Accuracy.* Because recognition is hierarchical, errors in the low-level system per-

colate up into high-level recognizers (e.g., a high-level recognizer cannot combine four lines into a rectangle if the lines were initially misclassified). Therefore, the low-level recognizer must be as accurate as possible.

2. *Supporting a large number of shapes.* The more shapes that are supported by a low-level recognizer, the more expressive a sketch grammar becomes. For example, many low-level recognizers do not handle helixes, yet these are used throughout the mechanical engineering domain to represent springs. Without recognizing this primitive, previous systems either relied on using a different symbol to represent springs, which may be unintuitive to end-users, or the domain was simply not supported.

3. *Placing no constraints on how users must draw primitives.* Many recognizers, such as the motion-based recognizers mentioned in Chapter II, place restrictions on how users must input shapes. For example, a circle may be specified with a clockwise path, which means any circle drawn with a counter-clockwise path would not be recognized properly.

4. *Domain-ignorance.* Low-level recognizers should not include domain-specific context when performing recognition. This is the job of the high-level recognizer. Recognition of primitive shapes should be the same, regardless of the domain the user is working in.

5. *Multiple interpretations.* Recognition is not perfect, even with human recognizers, and low-level recognizers will make mistakes. Therefore, it is important to return multiple (and valid) interpretations. Preferably, these interpretations should include confidence values, so that high-level recognition systems can correct low-level errors with the help of domain context [96], as seen in Figure 32.

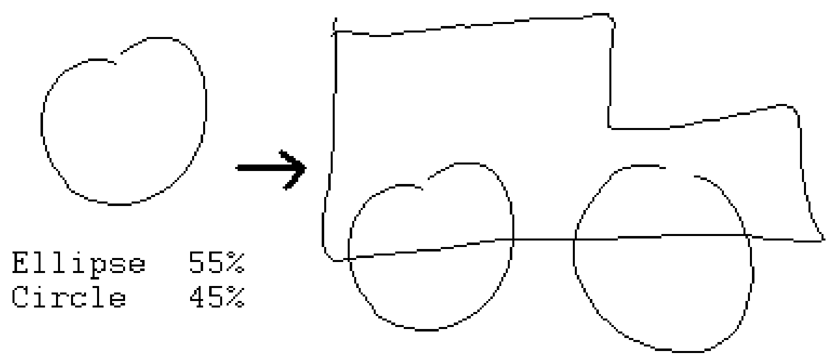


Fig. 32. Example scenario in which higher-level context can disambiguate a lower-level interpretation. In this example, the high-level recognizer may realize that the ambiguous stroke is more likely to be a circle (which represents a wheel in this domain) rather than an ellipse because of context.

6. *Beautification.* During recognition, PaleoSketch also computes the parameters necessary to provide local beautification of recognized shapes. When to perform beautification is dependent upon the individual sketch application, and is not the focus of this work. We refer readers to previous works for a full discussion on the advantages and disadvantages of various beautification times [77, 78].

In addition to the natural ambiguity that occurs with sketching, low-level recognition is a difficult problem, because many of the goals stated above are in conflict with one another. When increasing the number of primitives supported, there occurs a greater likelihood for recognition confusion, and thus, lower accuracy. Likewise, many of the drawing constraints placed on users by previous systems exist solely to improve classification accuracy. Domain context in the lower stages of recognition is also utilized by some systems so that overall accuracy can increase.

1. First Version of PaleoSketch

The initial version of the PaleoSketch recognizer used a heuristically-tuned, rule-based classifier to order and rank shape interpretations [6]. This classifier has the advantage of making it easier to diagnose misrecognitions, which can be advantageous for tutoring systems such as [38, 168] that need to describe why a stroke was drawn incorrectly (i.e., "this stroke is too curvy to be a polyline"). It also "fails gracefully," because the rule-based scheme used to perform classification is modeled after the decisions that would be made by a human recognizer. Thus, most users perceive its errors as being more acceptable.

However, the major disadvantages of the rule-based classifier are its inability to produce normalized confidence values [172] and its lack of extensibility. In order to add new shapes or features to the system, the entire rule-based algorithm has to be modified. This is one of the reasons why PaleoSketch is now being implemented using a neural network. The remainder of this dissertation utilizes the neural network version of PaleoSketch. For those interested in the rule-based version of PaleoSketch, please see [6].

2. Determining What Primitives to Support

One important question that should be addressed prior to this work is, "what primitives should be supported by a low-level recognizer?" Fortunately, earlier research into how people draw can give some insight into the shapes most commonly used by people when drawing or creating diagrams. For example, Hendry performed an experiment in which he asked users to sketch out how a search engine works [129]. Although he found that all of the sketches were widely different, he noted that there was a basic shape vocabulary used by most users. Similar observations were made by

Tversky and Lee, when they asked users to sketch out route maps [128]. Shapes like ellipses and rectangles were used to denote buildings, while lines and arcs represented roads and paths. Both of these studies are significant, because they show that, even in arbitrary domains, people maintain the simplicity of using basic geometric shapes to describe objects.

Even before these later studies, Peter van Sommers looked into the physical mechanics of how people draw [127]. When he asked a group of untrained adults and children to draw specific objects, he noted that, "the vast majority of their strokes are simple lines, arcs, circles, or dots. Relatively few contours are used - that is, lines whose shapes is steered and modulated to present shape." He later went on to describe a set of geometric primitives that he often saw in drawings. These included: lines, circles, dots, zig zags (polylines), spirals, coils (helixes), opaque areas (hatching/filled), polygons, arcs, and ellipses (rotatable). These observations, along with our own analysis of the shapes that occur in many sketch domains, led us to the following list of primitives that we aim to support in PaleoSketch:

- Line: a stroke with a relatively constant slope between all sample points.
- Polyline: a stroke consisting of multiple, connected lines.
- Circle: a stroke that has a total direction close to 2π, a relatively constant radius between the center point and each stroke point, and whose major and minor axes are close in size.
- Ellipse: a stroke with similar properties of a circle, but whose major and minor axes are not similar.
- Arc: a segment of an incomplete circle.

- Curve: a stroke whose points can be fit smoothly up to a fifth degree Bézier curve.
- Spiral: a stroke that is composed of a series of circles with continuously descending (or ascending) radii but a constant center.
- Helix: a stroke that is composed a series of circles with similar radii but with moving centers. We also assume that helixes are drawn linearly (i.e., the center moves in a single, constant direction along a straight line).
- Complex: a single stroke that is composed of a combination of multiple primitives mentioned above.

A visual depiction of each of these shapes can be seen in Figure 33. Please note that the polyline primitive is a catch-all for not only zig zags, but also for rectangles, diamonds, triangles and other polygons. When encountered with a polyline, the low-level recognizer will break the shape down into its corresponding line components and pass each one of them into the high-level recognizer. Similarly, when a complex shape is found, its sub-shapes are sent into the high-level recognizer.

3. Symbol Description Experiment

To further motivate our choice of primitives, a symbol description study was performed in order to determine what primitive shapes were typically used to describe symbols in an unfamiliar domain, as well as, the vocabulary used to describe geometric constraints [160]. The study was composed of 35 users, ranging in age from 18 to 71, with varying professional backgrounds (professors, students, dancers, teachers, business people, etc.). The users were asked to describe, both verbally and textually, symbols from a domain that is unfamiliar to most: military course of action symbols.

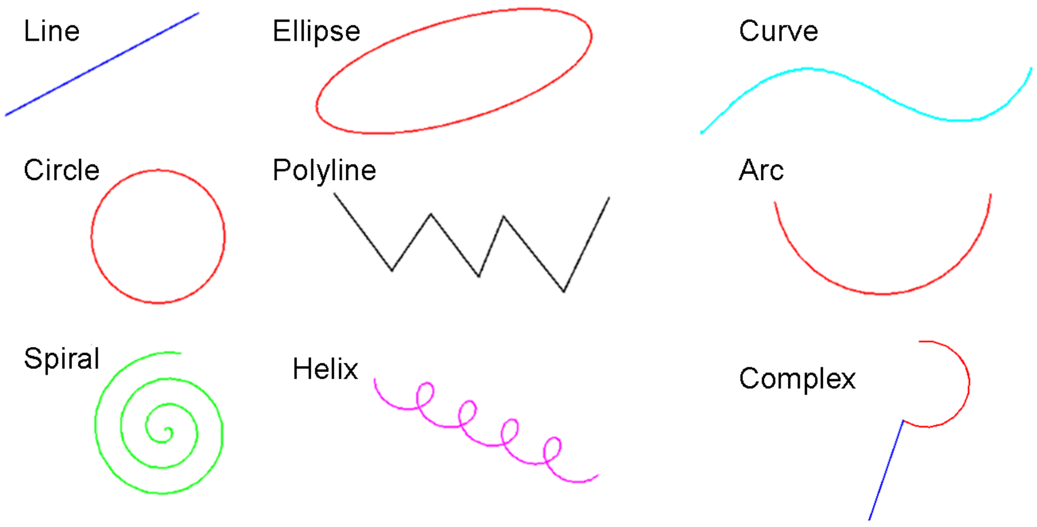


Fig. 33. Primitives supported by PaleoSketch.

The users were also asked to draw examples of these symbols based on the textual descriptions given by other users.

Through this study, we found that our primitive set was used in 85% of the shape vocabulary used to describe the symbols in this domain. This 85% includes polyline primitives that were typically described by their more formal definitions, such as rectangles, triangles, and "boxes" (also used to describe rectangles). Although these shapes are not explicit primitive shapes supported by our recognizer, they can still be described as being a polyline meeting particular high-level geometric constraints. The remaining 15% of the shape vocabulary consisted of descriptions such as "dot" or "point" (8% of total vocabulary), "shape" (3% of total vocabulary), "this" (2.5% of total vocabulary), "rounded rectangle," "segment," "figure," "text," etc.

B. Data Set

In our study, we collected two sets of data. The first set of data consisted of 900 shapes collected from 10 users. Each user drew 10 examples of each primitive shape, along with 10 examples of a complex shape. We asked users to draw a complex shape consisting of one line and one arc, an example that some recognizers have difficulty in interpreting [12]. All users were told to draw primitives in any way that was natural to them. No recognition or beautification took place during data collection. This first data set was used for training.

We then collected a second data set of the same size, with the same number of users, and used this set for testing. By collecting data in this manner, we can present the average recognition results across 10 new users to a system that has been trained offline (i.e., new users do not have to train the recognizer before using it).

Before running the test set through our recognizer, we first wanted to test it against current sketch systems in order to determine baseline recognition results of existing algorithms. The first recognizer, CALI, computes a set of bounding polygons (triangle, quadrilateral, and convex hull), and uses the ratios between the areas and perimeters of the polygons as features in a fuzzy-logic classifier. It recognizes primitives such as lines, ellipses, rectangles, and triangles [14, 15]. Because the recognizer is non-trainable, we could only test it on the shapes it supports.

The second recognizer we tested is the trainable, appearance-based HHReco recognizer, which utilizes Zernike moments as features in a support vector machine (SVM) [44].

The third, Rubine, is arguably the most widely used sketch recognizer in pen-based applications [43]. This is a trainable, motion-based recognizer that is highly accurate when shapes are drawn in the same manner across all users.

Table I. Accuracy results of existing recognizers on our collected set of primitive data.

	CALI	HHReco	Rubine	Sezgin	$1
Arc	0.97	0.96	0.51	-	0.98
Circle	0.97	0.89	0.37	0.83	0.97
Complex	-	0.79	0.19	0.99	0.93
Curve	0.61	0.77	0.47	-	0.85
Ellipse	1.0	0.44	0.61	0.56	0.95
Helix	-	0.92	0.75	-	0.96
Line	1.0	0.98	0.74	0.99	0.69
Polyline	-	0.67	0.44	0.92	0.56
Spiral	-	0.99	0.80	-	1.0
Average	91.0%	82.3%	54.22%	85.8%	87.7%

Next, we tested on a non-trainable, modified version of the geometric recognizer created by Sezgin et al. [41]. Since the Sezgin recognizer does not distinguish between ellipses and circles, we count circles as being correct if it returns an ellipse interpretation. Also, because the Sezgin recognizer does not handle spirals, helixes, or individual arcs and curves, we have omitted those shapes from testing.

The final recognizer is the trainable, vision-based $1 recognizer created by Wobbrock et al. [42]. This recognizer is essentially a 1-nearest neighbor, template matching algorithm that performs additional steps that attempt to scale and rotate strokes according to their "indicative angle." Table I shows the initial results of these recognizers on our data set.

According to LaLomia's study of handwriting recognition results in a "Wizard of Oz" study, an error rate of 5-10% was considered "very poor", with only a 3% error rate being considered "acceptable" by users [173]. If one assumes that users expect sketch recognition results to be on par with handwriting recognition results, then all of the recognizers we have initially tested would be considered "very poor." The best performing recognizer was the CALI recognizer. Although its accuracy exceeded 90%,

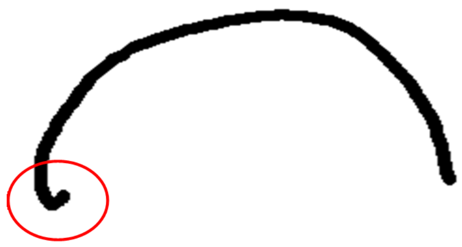


Fig. 34. An example of a drawn stroke that has a "tail."

it only supported five of the primitive shapes that we wish to recognize. Similarly, the Sezgin recognizer only supports a small subset of our primitive shapes. The appearance-based \$1 recognizer was the second best recognizer, but performed poorly on shapes that have arbitrary variations like lines, polylines, and curves. Likewise, the appearance-based HHReco recognizer performed poorly on high-variability shapes. Furthermore, the HHReco recognizer had trouble distinguishing circles and ellipses because these shapes are scaled down into similar templates. Finally, the widely-used, trainable Rubine recognizer performed the worst on our naturally-drawn sketch primitives. This is because the Rubine algorithm is gesture-based and relies on every stroke of the same shape being drawn with the exact same motion. These initial results provide extra motivation that further work is required.

C. The First Part of PaleoSketch: Pre-recognition

Before performing recognition of a stroke, it goes through a pre-recognition stage. During this stage, the stroke is "cleaned," and an initial set of statistics and graphs are computed. We begin pre-recognition by first removing consecutive, duplicate points from the stroke. These points can occur in systems with a high sampling rate. If two consecutive points either have the same x and y values, or if they have the same time value, then the second point is removed.

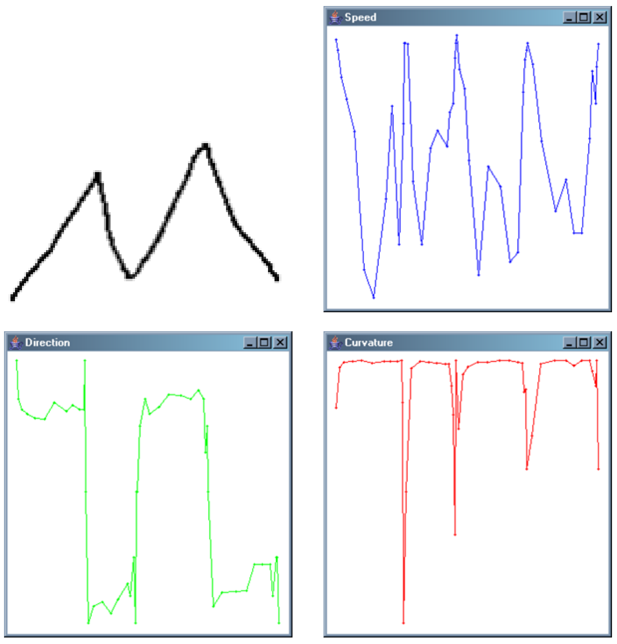


Fig. 35. Example stroke along with its corresponding speed graph (top right), direction graph (bottom left), and curvature graph (bottom right).

"Tails" at the endpoints of strokes can be significant problems for recognition (Figure 34). To allow for easier recognition, tails are removed before calculating the features of the stroke. To determine if a tail is present, we analyze the first and last 20% of the stroke and find the point within each section that has the highest curvature. If that curvature is higher than a threshold (0.5), then we break the stroke at that point and remove the tail. We do not perform tail removal on strokes with too few points (5) or with too small a stroke length (70.0). These thresholds have been determined empirically through multiple interations of the PaleoSketch recognizer.

After the stroke is cleaned, a series of graphs are computed for the stroke, including a direction graph, speed graph, and curvature graph (Figure 35). The graphs are computed using methods from [12, 41]. The direction graph measures the arctangent angle of the change in y over the change in x (i.e., the first derivative) between

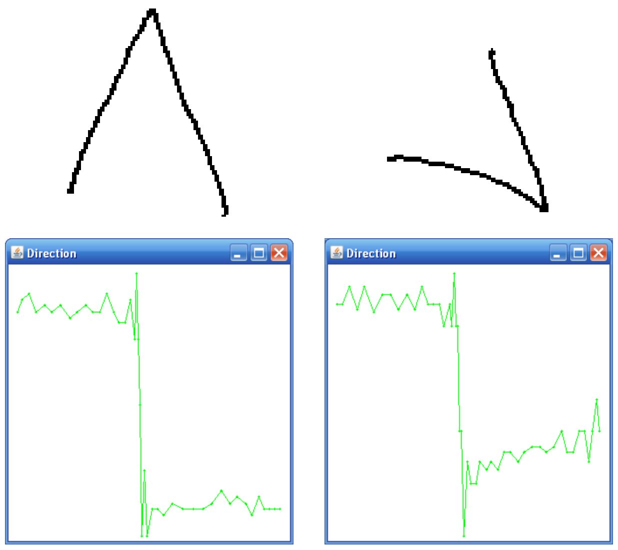


Fig. 36. Two similar polylines, drawn to different rotations, but with similar direction graphs.

consecutive points in the stroke. As seen in Figure 36, this graph is invariant to rotation. Because this angle will loop back to zero after the stroke makes a complete 2π revolution, we add (or subtract) 2π for each additional revolution from the direction value in order to make the graph continuous. The curvature graph is the second derivative of the arctangent angle over time, and the speed graph measures the change in distance between consecutive points over time.

Corner finding is one of the most important sub-processes in primitive recognition. In fact, an entire sub-field of sketch recognition has been devoted to the creation of accurate segmenters. The corner finder used by the initial version of PaleoSketch was a simplistic algorithm that attempted to produce the best polyline interpretation of a stroke [6], but was later shown to generate false positive corners [174]. The corner finder now used by PaleoSketch achieves 98% “all-or-nothing” accuracy, which penalizes for false positive corners, by combining the segmentation interpretations of five different corner finders [174, 175], including: the original PaleoSketch corner finder

[6], the Douglas-Peucker corner finder [10], the Sezgin corner finder [41], the Kim & Kim corner finder [122], and the ShortStraw corner finder [11].

In addition to corners and the various graphs of Figure 35, additional information such as the stroke's bounding box and total stroke length are also computed. By "stroke length", we refer to the sum of the distances between each pair of consecutive points. After all graphs and other information has been computed, we then calculate the features that will be used for classification.

D. Features

In addition to specifying our own set of geometric features, we also want to analyze the usefulness of existing feature sets. We will compare our feature set to the those of existing sketch recognizers. Unlike our initial experiments in Section B, which utilized the built-in classifiers for each of the recognizers, we will use the same classifier for each set of features to determine which sets are most effective for recognizing the primitives we aim to support. The feature sets we experimented on included:

- The CALI features that are based on ratios between sets of bounding polygons (24 total) [14, 15].
- The Zernike moment features of the HHReco recognizer (23 total features for order 8 Zernike moments) [44].
- The motion-based features of Rubine (13 total) [43].
- The extended Rubine features proposed by Long (22 total) [111].

1. New Geometric Features

Most of our geometric-based features are calculated using the errors between strokes and their ideal (beautified) shape interpretations. In addition, we also include features that are derived from the direction and curvature graphs computed during pre-recognition. First, we will describe a general set of features that are useful for recognizing multiple primitive shapes. Then, we will describe shape-specific features that aid in the recognition of individual primitives. These geometric features are novel because, at their core, they are based on the geometrical description of the individual shapes. This is different than many other feature sets that focus either on the motion of the stroke or provide a set of generalized features that are meant to describe any arbitrary shape.

a. General Features

The first set of general features are related to the length of the stroke, and the stroke's *major axis*, which is the line formed between the two furthest points of the stroke. These features include:

1. Total stroke length.
2. The distance between a stroke's endpoints divided by the total stroke length.
3. Major axis length.
4. Major axis angle.

The next set of features are computed with the aid of the stroke's bounding box. These features are meant to uncover the distances between each corner of the bounding box and the stroke itself. For example, each corner of a sketched rectangle's

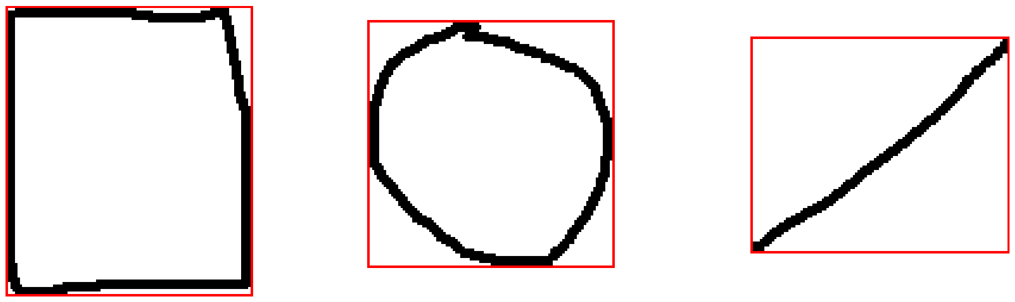


Fig. 37. Sketched primitives and their corresponding bounding boxes. Notice that all four corners of the rectangle's bounding box are near the drawn stroke, while none are close for the ellipse, and only two are near for the line.

bounding box should be close to the actual stroke, whereas with a line, only two corners of the bounding box will be near the actual stroke, and with a circle, no bounding box corners should be near the stroke. An example of this can be seen in Figure 37. The following features are all normalized by the average width and height of the bounding box in order to make them scale-independent:

5. The distance between the furthest bounding box corner and the stroke.

6. The distance between the nearest bounding box corner and the stroke.

7. The average distance between the four corners of the bounding box and the stroke.

8. The standard deviation of the four corner distances.

The final set of general features are computed using the direction and curvature graphs. The broad significance of these graphs is that they indicate "smooth" strokes, such as those representing arcs and ellipses, and "jagged" strokes, like polylines. They can also be used to determine the total rotation of a stroke [43]. These features include:

9. The number of revolutions that a stroke makes (total rotation divided by 2π).
10. The average curvature of the stroke.
11. The maximum curvature value found in the stroke.
12. The ratio between the maximum curvature value and the average curvature value.
13. The slope of the line formed by the endpoints of the direction graph.
14. The least squares error between the direction graph and the best fit (least squares) line of the direction graph.
15. The percentage of the time that the values of the direction graph are consecutively increasing, or decreasing, depending on the slope of the graph.
16. The normalized distance between direction extremes (NDDE).
17. The direction change ratio (DCR).

The last two features of this set require further explanation. The first is *normalized distance between direction extremes* (NDDE). To calculate this feature, we first take the point with the highest direction value and the point with the lowest direction value and compute the stroke length between these two points. In the event that two or more points both have the same direction value (and that value is a maximum or minimum), we use the point that occurs first in the stroke. This length is then divided by the length of the entire stroke, essentially giving us the percentage of the stroke that occurs between the two direction extremes.

For curved shapes, such as arcs, the highest and lowest directional values will typically be near the endpoints of the stroke, and thus, yield high NDDE values.

Polylines, on the other hand, normally have one or more spikes in their direction graphs. These spikes often cause the point of highest or lowest directional value to no longer be near the endpoints of the stroke. Therefore, polylines typically will have NDDE values that are lower than curved strokes (see Figure 38 for example).

The second feature also aids in determining polylines from curved strokes. We call this *direction change ratio* (DCR). Like NDDE, DCR is also meant to gauge whether or not spikes are present in the direction graph. This value is computed as the maximum change in direction divided by the average change in direction. As seen in Figure 38, the polyline has a large downward spike in its direction graph, whereas the arc has relatively little change between consecutive direction values. Therefore, polylines will typically have higher DCR values than curved strokes.

b. Line Features

Lines are defined as being a set of points that maintain a consistent slope between endpoints. For lines, we first fit a least-squares line to the stroke points. This least-squares line is used solely for computing line features; it is not used for beautification. To beautify a line, we simply connect the endpoints of the stroke. We maintain the endpoints, rather than using a best fit line, because they are significant in sketching, particularly in diagramming domains where lines are meant to act as connectors. This endpoint-significance theme continues in the beautification of the other low-level shapes as well. The line-based features that we compute are normalized by the stroke's length and include:

18. The least squares error between the best fit line and the actual stroke points [41].

19. The feature area error between the best fit line and the stroke points [12].

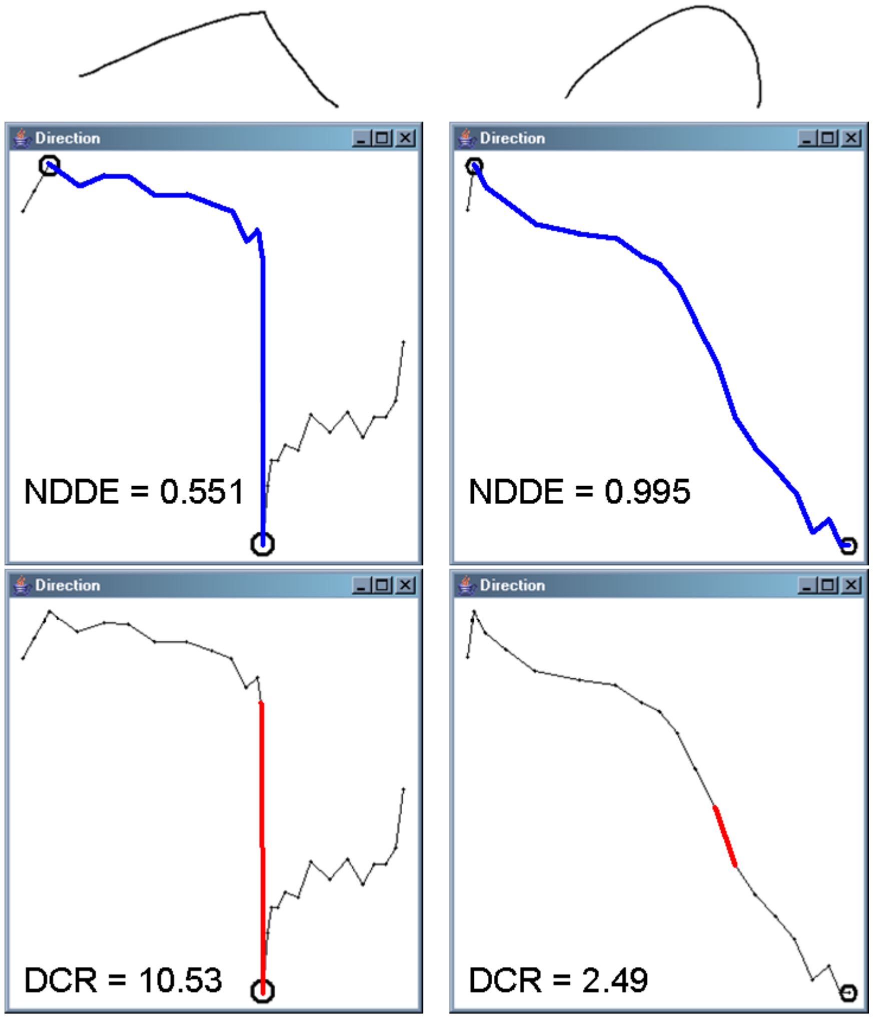


Fig. 38. Direction graphs for a polyline (left) and arc (right). The points of highest and lowest direction value are circled (middle). NDDE is calculated as the length of the stroke between direction extremes (bolded in blue), divided by the entire stroke length. DCR (bottom) is the maximum change in direction (bolded in red) divided by the average change in direction across the stroke. In this example, the polyline has a lower NDDE value because its minimum direction value is in the middle of the stroke, whereas the arc has its direction extremes closer to the endpoints. Furthermore, the polyline has a large spike in its direction graph, causing it to have a much higher DCR value than the arc, which maintains a fairly smooth continuity throughout.

c. Ellipse Features

Our ellipse recognition can account for both rotated and overtraced versions of ellipses. To estimate an ideal ellipse, we first need to calculate an a major axis, minor axis, and center point. The major axis should have already been calculated in previous steps, and the center point can be found by simply averaging the x and y values of the stroke points. The minor axis is calculated by finding the perpendicular bisector of the major axis at the center point; this line is extended and clipped where it meets the stroke points, which may be at interpolated values. Once these values are computed, we then have the information needed to generate a beautified ellipse. Rotated ellipses can be beautified by rotating around the center point, based on the angle of the major axis. The only feature we compute for ellipses is:

20. The feature area error between the ideal ellipse and the stroke points [12], normalized by the area of the ideal ellipse, which can be computed using the lengths of the major and minor axes.

d. Circle Features

Circles are simply special cases of ellipses in which the lengths of the major and minor axes are close to identical. To compute a beautified circle, we simply need a center point and a radius. We use the same center point that we calculated for an ellipse, and estimate the radius as the average distance between each stroke point and the center point. We then calculate:

21. The feature area error of the circle [12], normalized by the area of the ideal circle (πr^2).

22. The ratio between the minor axis length and major axis length.

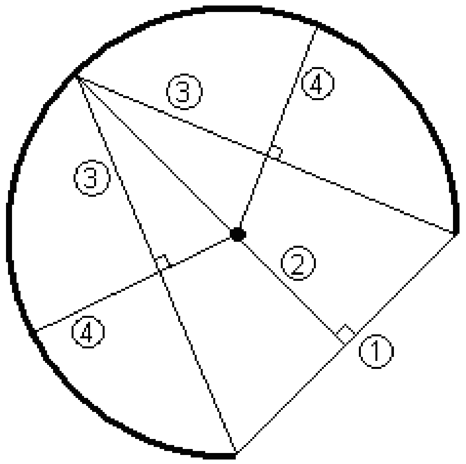


Fig. 39. Estimating the center of an arc. 1) Connect the endpoints of the stroke. 2) Take the perpendicular bisector of 1. 3) Connect the endpoints of the stroke to the point where 2 intersects the stroke. 4) Take the perpendicular bisectors of the two lines from 3 and find their intersection.

e. Arc Features

With our recognizer, we consider arcs to be segments of circles; therefore, in order to determine the best fit arc, we need to determine the best fit circle that the arc is a part of. To do this, we need to calculate the ideal center point of the arc using a series of perpendicular bisectors (see Figure 39). First, we connect the endpoints of the stroke and find the perpendicular bisector at the midpoint of that line. We determine where that bisector intersects the stroke (through interpolation between stroke points) and then connect two more lines from that point to the endpoints. We take two more bisectors at the midpoints of those two lines and find where they intersect each other. That intersection point is the center of our arc.

We then calculate the ideal radius of the arc by taking the average distance between the stroke points and the center point. A beautified arc is constructed from the ideal center, radius, and angles between the center point and endpoints. As with lines, we want to make sure that endpoint consistency is maintained. Our arc features include:

23. The radius of the ideal arc.

24. The feature area error of the ideal arc to its center point [12], normalized by the area of the arc (πr^2 * the number of revolutions of the stroke, which should be between 0.0 and 1.0).

25. The ratio between the areas of the bounding box of the ideal arc and the bounding box of the actual stroke.

f. Curve Features

We could allow for any degree curve; however, we need to limit this degree not only to keep recognition to a practical running time, but also because many shapes can easily be represented by some arbitrary n-degree curve. For our system, we estimate using fourth and fifth degree curves, choosing the degree that best fits the stroke points best.

To generate an ideal curve, we use the Bézier curve formula. In order to use this formula, we must first calculate *d+1* control points, where d is the degree of the Bézier curve. Currently, we use a naïve approach to approximate these points.

First, we find the parametric value of each stroke point. We determine a point's parametric value by dividing the length of the stroke up to that point by the length of the entire stroke. Once these values are computed, we take the endpoints (whose parametric values are 0 and 1), as well as *d-1* other points which are spread evenly across the stroke. For example, for a fourth degree curve, we would take the endpoints, along with the points whose parametric values were close to 0.25, 0.5, and 0.75. This will ensure that the points that occur 1/4, 1/2, and 3/4 of the distance along the stroke are located in the same position in the drawn stroke as they are in the beautified curve.

We then estimate the control points by solving a system of equations using these selected points and their parametric values. Once we have the estimated control points, we can generate the ideal curve according to the Bézier curve formula, where n is the degree of the curve and P_i is the set of computed control points:

$$B(t) = \sum_{i=0}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) P_i (1-t)^{n-i} t^i \tag{3.1}$$

From this ideal curve, we can compute the following:

26. The least squares error between the ideal curve and the actual stroke, normalized by stroke length

g. Polyline Features

Polylines are simply a series of connected lines drawn within a single stroke. The ideal polyline interpretation is produced by the corner finder. Therefore, all that is left to do is compute errors related to the produced segmentation. The features of polylines that we compute are:

27. The number of lines produced by the corner finder.

28. The sum of the feature area errors of each line segment [12], normalized by stroke length.

29. The sum of the least squares errors of each line segment [41], normalized by stroke length.

30. The percentage of the sub-strokes that pass a line test (as defined in [6]).

h. Spiral & Helix Features

Spirals and helixes are similar, because they can both be thought of as a sequence of circles. For spirals, we have a sequence of circles that have continuously decreasing (or increasing) radii but a constant center. With helixes, we have a sequence of circles with a constant radius but moving center. Unlike our other shapes where we generate a beautified version and compare its error to the original stroke, we found that a similar approach did not work as well with spirals and helixes, because users inherently draw these shapes with relatively more error than other primitives. In other words, it is much easier for users to draw a perfect line or circle than it is for them to draw a perfect spiral or helix. However, we can still use the geometric description of these shapes, being sequences of circles, to help recognition.

To recognize these shapes, we first break the stroke up at every 2π interval in the direction graph. Essentially, each one of these sub-strokes can be thought of as, and fit to, individual circles. For spirals, all sub-strokes should have a similar center point, but the radius of each sub-stroke should get continuously larger (or smaller). The ideal center of the spiral is taken to be the center of the bounding box for the entire stroke. An average radius is calculated as it was for circles. The features used to help distinguish spirals and helixes include:

31. The sum of the distances between the centers of each consecutive sub-stroke, normalized by the average radius times the number of revolutions that the stroke makes. This helps determine if the center is moving (helix), or is stationary (spiral).

32. The percentage of the time that the radii of consecutive sub-strokes are ascending (or descending). For spirals, radii should continuously increase or decrease, but this trend is typically not consistent with helixes.

33. The ratio between the average radius and the average width/height of the bounding box of the stroke. This feature is used to help distinguish spirals from other overtraced shapes (like circles). For overtraced circles, the bounding box radius will be close to the average radius, whereas with spirals the average radius will typically be smaller because each consecutive sub-stroke should get closer and closer to the center point of the spiral.

34. The distance between the sub-stroke center that is furthest away from the ideal center, normalized by the average radius. Again, this helps determine if the center is moving or stationary.

To beautify a spiral, we generate an Archimedes spiral that starts at the center point and has the polar equation, $r = a\theta$. In this equation the radius, r, changes as θ changes. The a value represents the "tightness" of the spiral. We set this value to be the bounding box radius divided by the total rotation of the stroke.

To generate the spiral, we continuously increment (or decrement) the θ value until the radius value reaches the bounding box radius. We decide whether to increment or decrement θ based on whether the spiral was drawn clockwise or counter-clockwise, which can be determined by looking at the slope of the direction graph. In order to preserve the outer-most endpoint of the spiral, we can shift the θ value by the angle formed by the endpoint and the starting center point. Once we find the r value that corresponds to the current θ, we simply substitute the value into the polar equation for a circle to generate x and y coordinates.

Creating a beautified helix is more complex than a spiral, since we want to maintain both of the endpoints of the stroke. Essentially, we want to use the polar equation of a circle like we did for spirals; but, instead of iteratively changing the radius, we want to change the position of the center point and maintain a constant radius. This constant radius is computed as the average distance between the stroke points and the major axis of the stroke (as done for ellipses). The starting and ending center points can be calculated by finding the points on the major axis that are a length equal to the radius away from the endpoints of the stroke. Once these two points are specified, we parametrically find the center point for the current iteration. The parametric value used is the absolute θ value, divided by the total rotation of the stroke. We continue to increase or decrease θ until its absolute value becomes greater than the total direction. At this point, the beautified helix will be generated.

E. Classifier

For classification, we use a multi-layer perceptron (MLP). MLPs are feed-forward, artificial neural networks that contain at least three layers of neurons (or nodes): an input layer, one or more hidden layers, and an output layer. Every node in a layer has a weighted connection to each node in the next layer. Supervised learning is used to train these connection weights. Activation functions are used at each node to determine the likelihood of that node "firing" or not.

We chose to use MLPs for a couple of reasons: they can produce non-linear decision surfaces, they are robust to noisy features, and they are adaptable [176]. Due to MLPs robustness, we avoid (to some degree) the need to perform dimensionality reduction of larger feature sets, as was done in a previous version of this work [172]. MLPs are also capable of producing normalized confidence values, which are beneficial to many high-level recognizers that can correct low-level errors using domain context [96]. Furthermore, MLPs can be universal approximators when given a sufficient number of hidden units [177].

In our implementation, we utilized the WEKA toolkit to produce our MLPs

[178]. The number of input nodes for each network is equivalent to the number of features, and the number of output nodes represents the number of primitive classes that we are supporting (9). We chose to have a single hidden layer consisting of 43 nodes, the sum of the number of input and output nodes for the largest feature set we tested (the PaleoSketch features). We chose to use the same static number of nodes for each feature set so that our comparison could be more fair, and the accuracy differences could be blamed less on the topology of the network.

WEKA's MLP implementation uses sigmoid activation functions at each node, and weights are learned through backpropagation. We specified a learning rate of 0.3, with a momentum value of 0.2, and performed 500 training epochs for each network. Feature values are also normalized before entering each network.

F. Handling Complex Shapes

When generating complex fits, we want to bias toward a complex interpretation that contains the fewest number of primitive shapes, while maintaining a low fit error. In the previous, decision tree version of PaleoSketch, we would start by breaking a stroke up into two sub-strokes at the point of highest curvature [6]. Each of these strokes is then recursively sent back into the recognizer until we get all non-polyline primitives. However, we cannot use such a simple approach with our neural network version of PaleoSketch, because we now have confidences associated with recognized shapes. Furthermore, the neural network only detects whether or not a complex shape is found; it does not know what sub-shapes compose the complex fit.

As with the original version, we generate a complex fit by first segmenting the stroke into two sub-strokes at the point of highest curvature. Next, we classify each sub-stroke using our MLP; complex and polyline interpretations are removed. If the interpretation of a sub-stroke has a confidence lower than 50%, then it, too, is broken at its point of highest curvature. The recursive process continues until all sub-interpretations have a confidence greater than 50%.

Next, we perform an additional step in which each consecutive pair of sub-strokes are combined and re-classified to see if they can be merged. Again, if the confidence of the shape is greater than 50% and the interpretation itself is not complex, then the sub-strokes are merged. This step is used primarily to remove sub-stroke tails that may occur. Typically, when tails occur, they often are at the spot of highest curvature and are separated from the stroke during the first complex break. In order to absorb the tail, it must be merged with the first sub-stroke of the second half of the break.

1. Confidence Modification

Initially, we assigned the confidence of a complex interpretation to be the value generated by the MLP. For the complex shapes we trained on (one line, one arc), the MLP performed quite well and gave accurate confidence values. However, when encountered with a complex combination that the neural network had not seen before (e.g., one line, one circle), the confidence became low. Therefore, we cannot trust the confidence given by the initial MLP results for the complex fit, because it is impossible to train the network with every possible complex combination.

In the event that the complex interpretation is not ranked first among interpretations, we perform the following steps to determine a more appropriate confidence. First, we average the confidences of all of the sub-interpretations of the complex fit. At this point, we are guaranteed that the complex interpretation has a confidence that is greater than 50%.

More than likely, however, a complex interpretation has a confidence above 90%, because most confident MLP interpretations are typically around 98-99%. This means that in most cases, a complex interpretation may now be incorrectly more confident than the original, best interpretation. For example, a shape that has a slight tail on it may produce a complex interpretation consisting of that shape plus one additional line. This may technically be the better fit for the stroke, but it is not necessarily what the user intended.

We want to maintain the simplest shape interpretation possible. Therefore, if we chose the confidence of the complex interpretation to be the average confidence of its sub-interpretations, then we augment this confidence by subtracting 25% for every additional sub-shape in the complex interpretation. If the best interpretation is a polyline then we compare the difference between the number of complex shapes and the number of lines in the polyline interpretation to determine what percentage should be subtracted from the complex confidence.

Using this approach, we are essentially assuming that if the network encounters a complex combination of shapes that it has not previously seen (e.g., a line followed by a circle), then the initial confidence of the best shape interpretation should be lower than 75%, which is typically the case. As we will see in the next section, this modification significantly boosted recognition of complex fits. The following scenarios illustrate how confidence modification works:

1. Suppose the user draws a line, followed by a circle, in the same stroke. The MLP returns the following results: "Curve" (54%), "Polyline" (21%), "Complex" (8%), etc. Since the complex interpretation is not the best one, we perform confidence modification. The stroke is broken at its point of highest curvature into sub-stroke A and sub-stroke B. Sub-stroke A is recognized as a line with 99% confidence, while sub-stroke B is recognized as a circle with 97% confidence.

The new confidence of the complex interpretation becomes 98% (the average). However, the complex interpretation contains two shapes while the best interpretation (curve) is only a single shape. Therefore, we subtract a 25% penalty from the complex confidence, giving it 73%. The final results are: "Complex" (73%), "Curve" (54%), "Polyline" (21%), etc. Note that these confidences can be re-normalized to ensure that they sum to 1.

2. Suppose the user draws an arc to the screen. The MLP returns the following results: "Arc" (92%), "Polyline" (3%), "Complex" (2%), etc. The recognizer breaks the stroke into two sub-strokes at the point of highest curvature. Sub-stroke A is recognized as an arc with 99% confidence, while sub-stroke B is recognized as a line with 45% confidence. Because sub-stroke B is below the 50% threshold, the recognizer breaks sub-stroke B at its point of highest curvature into sub-strokes C and D. Both of these strokes are recognized as lines with 97% and 98% confidence, respectively. The final complex interpretation is "Complex (Arc, Line, Line)," with a confidence of 98%. At this point, the complex interpretation would incorrectly be ranked above the correct arc interpretation. However, because the complex interpretation contains two additional sub-shapes, 50% is subtracted from its confidence. The final results would be: "Arc" (92%), "Complex" (48%), "Polyline" (3%), etc.

G. Experiment & Results

In our experiment, each feature set was tested against our dataset using the same N-43-9 MLP, where N is the number of features in the set. We also combined the features of CALI, HHReco, and Long into one common set and tested it as well (*Combined*). We then added the PaleoSketch features to this set to create the *All*

Table II. Accuracy results of the different feature sets using a multi-layer perceptron classifier. The first five columns represent the accuracies of the five individual feature sets. *Combined* refers to the combined feature set of CALI, HHReco, and Long. *All* refers to the combination of the *Combined* feature set, plus the Paleo features. *Modified* uses the same feature set as *All*, but also utilizes the complex confidence modification algorithm.

	CALI	HHReco	Rubine	Long	Paleo	Combined	All	Modified
Arc	0.99	0.95	0.48	0.65	0.99	0.89	0.95	0.95
Circle	0.93	0.71	0.76	0.74	0.90	0.94	0.95	0.95
Complex	0.80	0.81	0.47	0.38	0.84	0.73	0.90	0.97
Curve	0.91	0.83	0.69	0.78	0.94	0.98	0.97	0.95
Ellipse	0.99	0.46	0.77	0.94	0.99	1.0	1.0	1.0
Helix	0.97	0.93	0.90	0.95	1.0	0.97	0.99	0.99
Line	1.0	0.99	0.95	0.94	1.0	1.0	1.0	1.0
Polyline	0.85	0.54	0.52	0.62	0.97	0.75	0.96	0.99
Spiral	1.0	0.98	0.94	1.0	1.0	1.0	1.0	1.0
Average	93.8%	80.0%	72.0%	77.8%	95.9%	91.8%	96.9%	97.8%

feature set. Note that the Rubine features are a subset of the Long features, and thus included by default. We tested the *Combined* feature set using a 69-78-9 MLP, and the *All* feature set using a 103-112-9 MLP.

The number of hidden nodes was increased for these sets due to them having significantly more features than the individual feature sets. These sets are also compared against a version of the recognizer that uses the best feature set, *All*, along with the complex confidence modification algorithm (*Modified*). The accuracy results of each feature set is shown in Table II.

Overall, the individual feature set that achieved the best accuracy was the PaleoSketch feature set, followed by the CALI feature set. The difference between the accuracy of the PaleoSketch feature set and the CALI feature set is slightly statis-

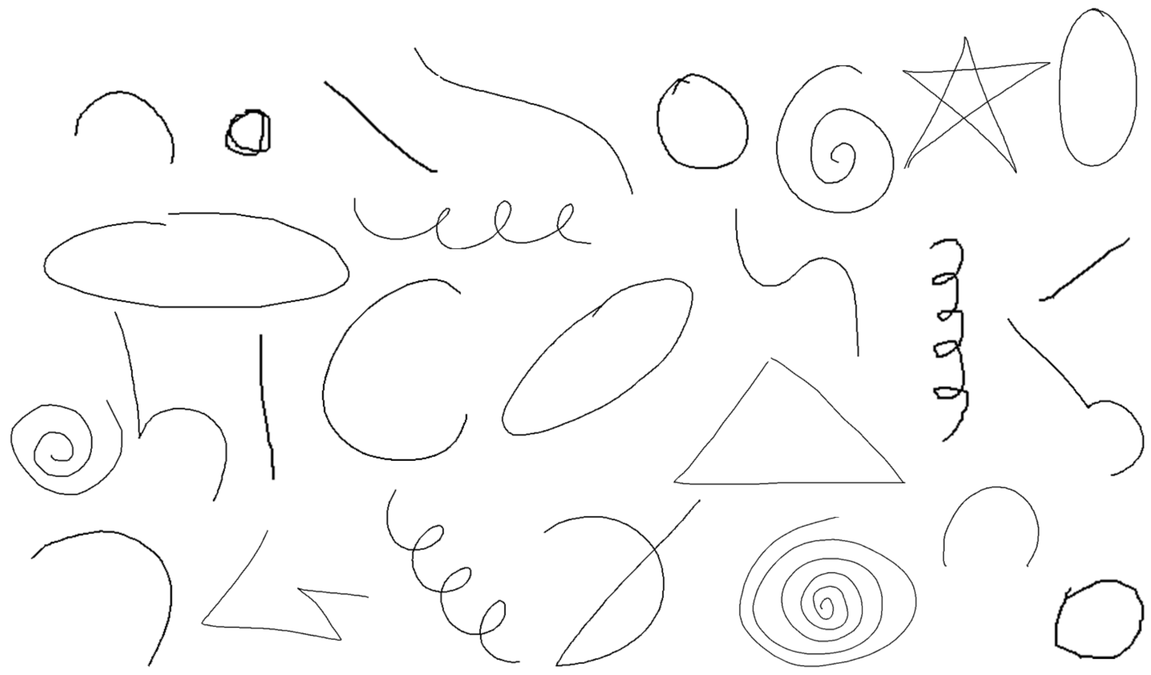


Fig. 40. Examples of correctly classified shapes. This is just a subset of the 900 sketched shapes used for testing.

tically significant ($t = 2.0244, p = 0.9569$). The best accuracy (97.8%[1]) came from combining all of the feature sets together and using the complex confidence modification scheme. This optimal version of PaleoSketch executed recognition in real-time, averaging 166 milliseconds per primitive. Examples of correctly classified shapes can be seen in Figure 40.

There are a couple of interesting observations from our findings. The first deals with the shapes that certain feature sets performed poorly on. For example, the CALI feature set had the most trouble with classifying shapes of arbitrary configuration, such as polylines, curves, and complex shapes. The makes sense because CALI relies solely on information related to the shape and sizes of bounding polygons and convex

[1]The difference between these results and the results of the original version of PaleoSketch (98.6%) [6] are not statistically significant ($t = 1.2339, p = 0.7826$)

hulls and was not necessarily designed to work with these types of shapes. Likewise, the appearance-based, HHReco features also had problems with these shapes as well. The HHReco feature set also had problems distinguishing circles and ellipses. This is due to the affine transformations that most appearance-based methods perform in order to scale shapes to a common template. The motion-based Long and Rubine feature sets performed well on shapes like lines, helixes, and spirals due to their rotation-related features, but performed poorly on all other shapes.

We can also see from these results that the choice of classifier makes a difference. Both the CALI and Rubine features performed better when used in the MLP classifier, while the HHReco features performed worse (recall the original classification rates in Table I). Another important observation is that the complex confidence modification algorithm significantly boosted the accuracy results of complex shapes.

1. Accuracy of Complex Fits

As mentioned previously, the results of Table II simply show whether or not a complex shape was recognized as such. It does not necessarily guarantee that the sub-interpretations of the complex fit are correct. We chose to have users draw complex examples consisting of one line and one arc, a notoriously hard example [12]. For our recognizer, a “one line, one arc” interpretation was correctly returned 94% of the time.

As with the original version of PaleoSketch, the primary culprit of mis-interpreted complex fits was shapes whose arc portion was drawn more elliptical than circular. This caused a “one line, one curve” interpretation to be returned instead of “one line, one arc”, as seen in Figure 41.

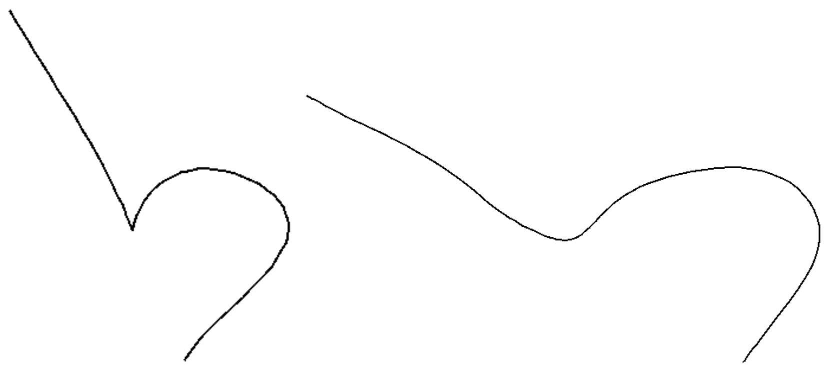


Fig. 41. Examples of complex shapes drawn with elliptical arcs, thus causing the recognizer to return a “one line, one curve” interpretation rather than “one line, one arc” (the users’ intentions).

H. Discussion

Figure 42 shows the confusion matrix for our experiments, while Figure 43 shows many of the mis-classifications that we encountered with our testing. Arcs drawn more elliptical than circular were often classified as curves (all of these were drawn by the same user). Some circles were confused with their superclass, ellipse. Complex shapes that had too smooth of a transition were recognized as curves. Curves with tails that were unsuccessfully removed during pre-recognition were broken up into complex shapes. There was also a single instance of a helix that was classified as a spiral, likely because its radii continuously decreased (in addition to its center point moving). And finally, there was a polyline whose corners were not well-defined, and thus, was broken up into a complex combination of lines and a curve.

1. Complex Fits

Successful integration of the original version of PaleoSketch allowed for initial testing of complex shapes that go beyond simple line/arc combinations (Figure 44). With

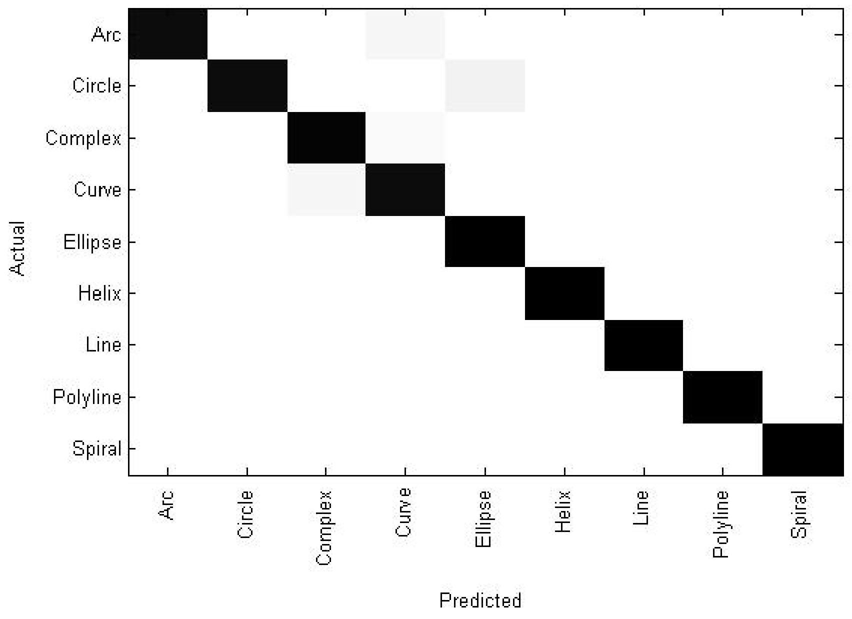
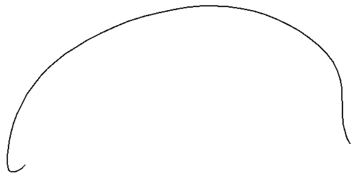
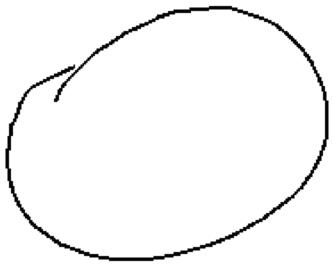
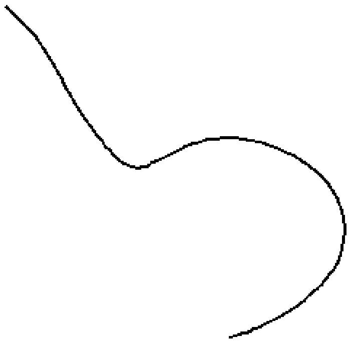
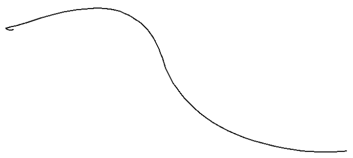


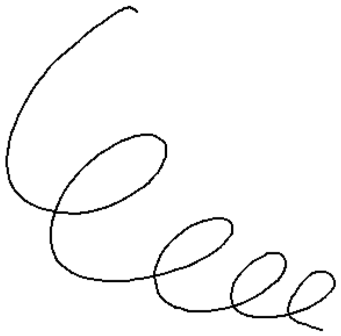
Fig. 42. Confusion matrix for our experiments using the optimal version of PaleoSketch. Dark areas indicate high confusion, while white areas indicate little or no confusion.


(a) Arcs mis-recognized as curves.


(b) Circles mis-recognized as ellipses.


(c) Complex shapes mis-recognized as curves.


(d) Curves mis-recognized as complex shapes (because of tail).


(e) Helix mis-recognized as a spiral.

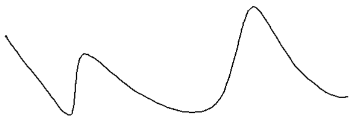

(f) Polyline mis-recognized as complex shape.

Fig. 43. Examples of mis-classifications in the PaleoSketch system.

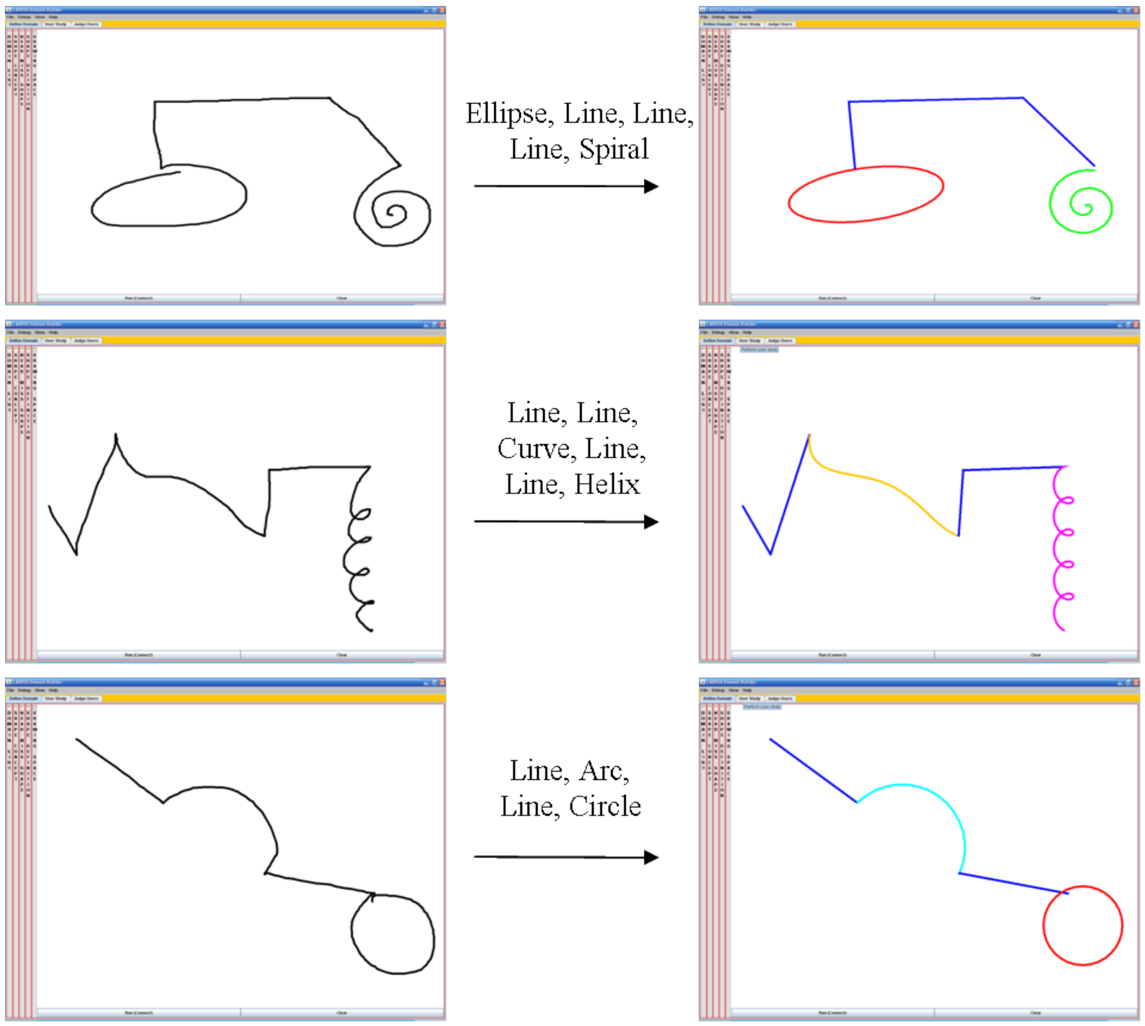


Fig. 44. Examples of higher degree complex fits achieved through the integration of PaleoSketch into LADDER.

the original version of PaleoSketch, high degree complex shapes were still recognizable. To ensure that similar results are possible with the new version of PaleoSketch, we performed brief testing in which more complicated complex shapes were sketched and recognized using the MLP that was trained solely on “one line, one arc” complexes. With the help of the complex confidence modification algorithm, many complex shapes can still be recognized, as seen in Figure 45. In Chapter V of this dissertation, we will present additional results for complex interpretations.

2. Using PaleoSketch in a Real-world Setting

PaleoSketch has been integrated into the high-level sketching language, LADDER [83]. This integration has allowed us to utilize PaleoSketch in real-world sketch applications (e.g., Figure 46). The next chapter covers the use of PaleoSketch in a large sketch domain: military course of action diagrams. In addition to the applications provided by LADDER, the PaleoSketch recognizer has been used in projects that improve corner finding [9], promote learning in children [20], teach people how to draw [168], recognize sketched Urdu characters [163], combine multiple approaches to improve recognition [136], recognize biology cell diagrams [39], and teach users how to draw Mandarin symbols [179].

3. Limitations of PaleoSketch

PaleoSketch is designed to work in conjunction with high-level sketch grammars that describe more complex symbols within a domain. It is meant solely to recognize simple geometric shapes, as it has been shown by previous research that users tend to draw object using small, basic shape vocabularies [127, 128, 129].

However, some domains may contain shapes and symbols that are hard to describe geometrically [16, 136]. Domains containing blobs, shading, and other 3-D

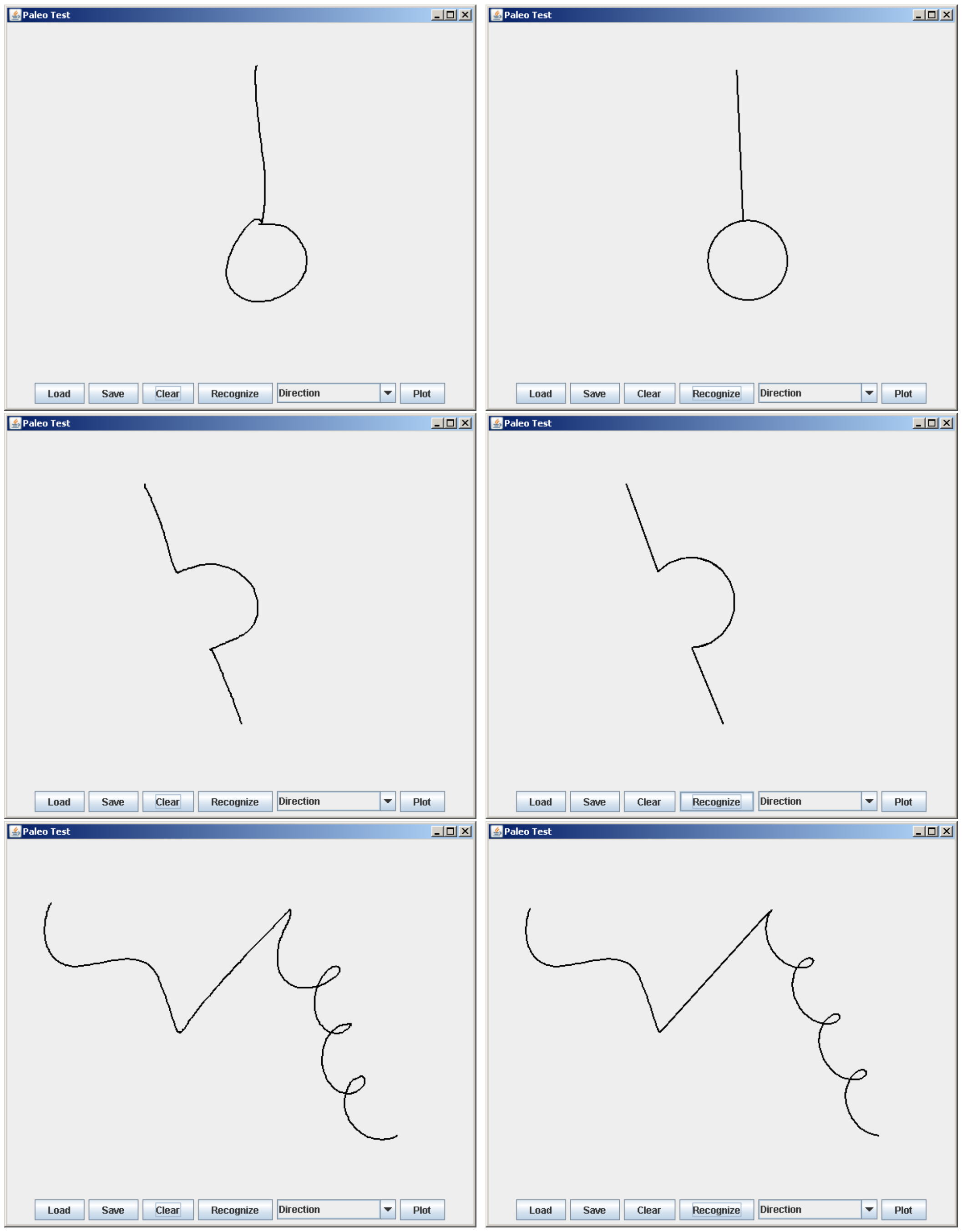


Fig. 45. Examples of complex fits recognized using the MLP version of PaleoSketch that was trained only on “one line, one arc” samples.

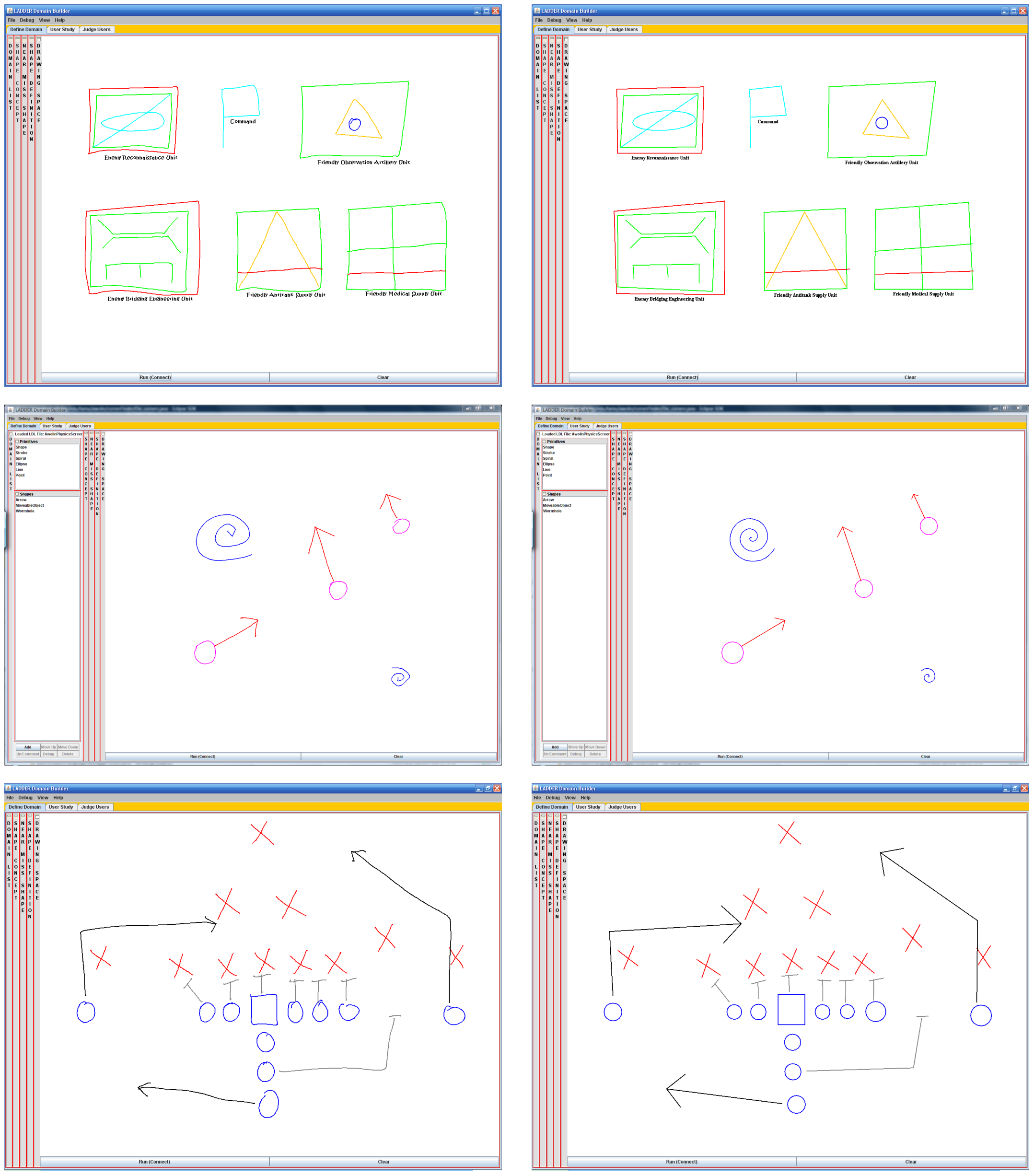

LADDER Domain Builder
File Debug View Help
Define Domain
User Study
Judge Users
Enemy Reconnaissance Unit
Command
Friendly Observation Artillery Unit
Enemy Bridging Engineering Unit
Friendly Antitank Supply Unit
Friendly Medical Supply Unit
Run (Connect)
Clear

Fig. 46. Example LADDER domains (top: military course of action, middle: planetary physics simulation [20], bottom: football play diagram) using the PaleoSketch primitive recognizer.

free-form drawing are not likely to be appropriate for a geometric-based recognizer like PaleoSketch. Handwriting is also hard to describe geometrically and should be recognized using specialized algorithms. Many researchers are currently working on groupers that can segment text strokes from shapes strokes, so that they can be given to the appropriate recognizers [131, 132, 133, 134].

As mentioned before, our goal is to produce sketch recognizers that place little to no constraints on a user's drawing style. Currently, we have made the assumption that all primitive shapes are drawn with a single stroke. As we will see in Chapter V, this assumption does not always hold. Later, we will present a multi-stroke version of PaleoSketch and discuss the challenges with recognizing these types of primitives.

I. Chapter Summary

In this chapter, we presented PaleoSketch, our algorithm for recognizing hand-sketched primitive shapes. Through our experiments, we have shown that the accuracy increase between PaleoSketch and the next best approach proved to be statistically significant. Our method maintains a high accuracy while supporting more primitives than previous algorithms. In addition, we have presented a technique for interpreting complex shapes, those composed of multiple primitives within the same stroke. PaleoSketch has been integrated into high-level recognition systems, and has been used in projects within the domains of engineering, education, sports, language studies, art, and science.

CHAPTER IV

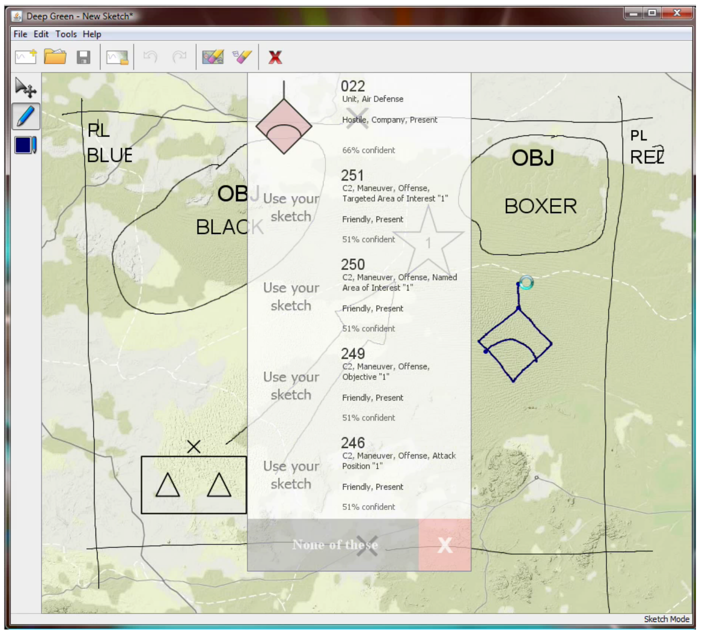
MILITARY COURSE OF ACTION

A. Introduction

Course of action diagrams (COAs) depict battle plans by showing the appropriate actions of military units. Traditional COAs are sketched on maps using colored markers or pencils. In most cases, transparent overlays are sketched on instead of the actual map in order to show different scenarios. As COAs became computerized, developers moved away from a sketching interface and replaced it with an easier-to-implement toolbar-based interface. According to military personnel, a major problem with this interface is "the awkwardness of mice and menus for what is more naturally done by sketching" [180].

However, recognizing the sketched symbols of the COA domain can be a daunting task, because there exist thousands of unique symbols. These symbols can be broken up into different categories. Unit symbols are used to denote military troops or vehicles. Typically, friendly units are drawn as (blue) rectangles, while hostile units are drawn as (red) diamonds. Modifiers are used to specify the unit size (e.g., company, brigade, platoon), as well as the unit type (e.g., armor, reconnaissance, infantry). Size modifiers are drawn outside of the unit symbol, while type modifiers are typically drawn inside the unit symbol. Other symbols include action arrows, obstacles, areas of interest, decision points, boundary and phase lines, and headquarters. Figure 47 shows a screenshot of a COA system that uses our low-level recognizer [21].

Because of the hierarchical nature of the COA domain, it lends itself well to multi-tiered sketch recognizers, as seen in Figure 48. This hierarchical approach has been used before to recognize COAs [87]. However, there were still many symbols in



022
Unit, Air Defense
Hostile, Company, Present
66% confident

251
C2, Maneuver, Offense, Targeted Area of Interest "1"
Friendly, Present
51% confident

250
C2, Maneuver, Offense, Named Area of Interest "1"
Friendly, Present
51% confident

249
C2, Maneuver, Offense, Objective "1"
Friendly, Present
51% confident

246
C2, Maneuver, Offense, Attack Position "1"
Friendly, Present
51% confident

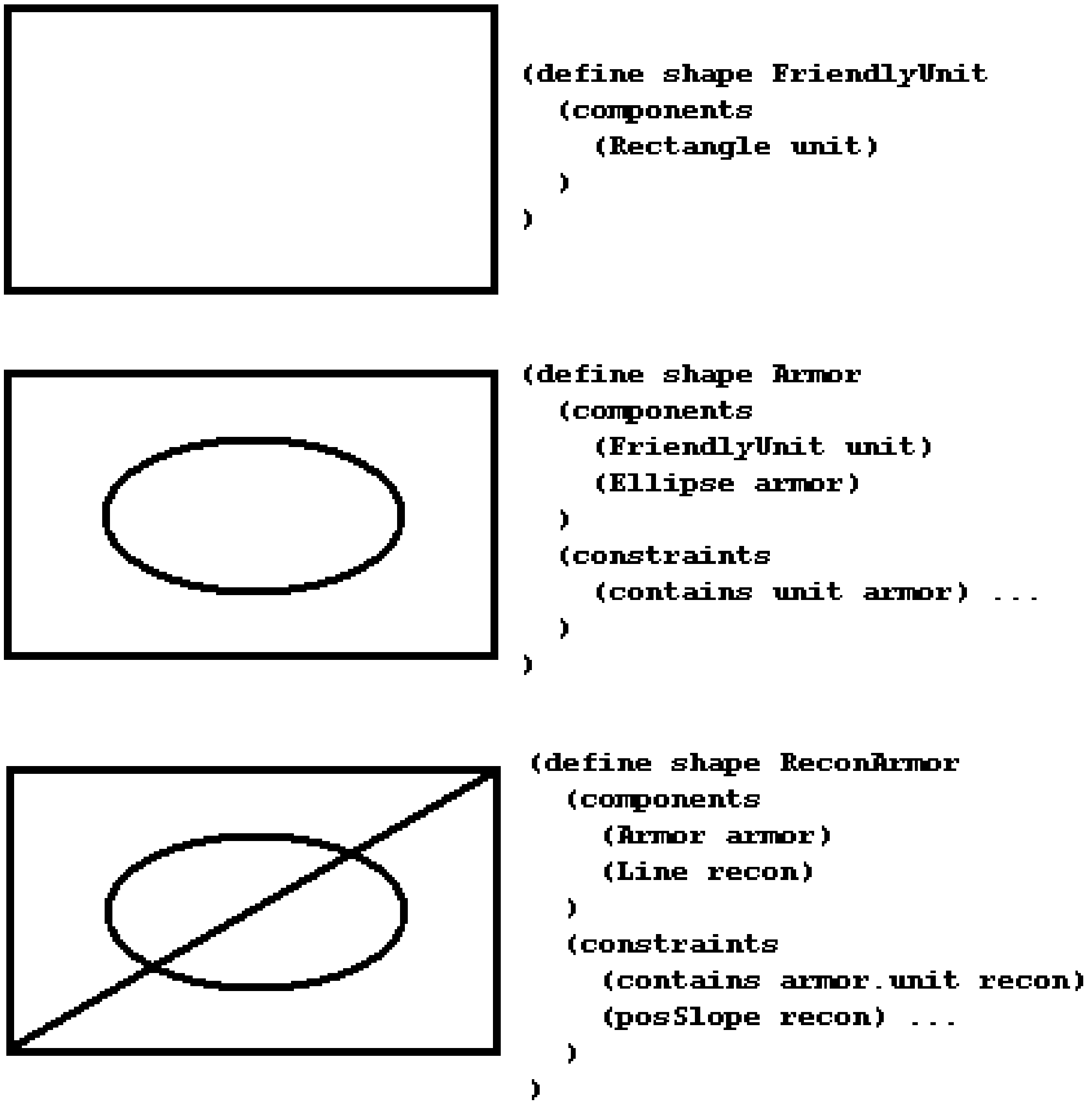
Fig. 47. Screenshot of a military course of action recognition system that utilizes our PaleoSketch primitive recognizer [21].

the COA domain that could not be handled because various primitive shapes were unsupported.

In this chapter, our existing primitive list is expanded to include shapes like waves, gulls (double arcs), lemniscates (infinity signs), and filled-in dots which are needed to fully describe many of the symbols in the COA domain. Additional features are proposed which aid in the recognition of these new primitives, and a multi-layer perceptron is used to perform classification.

B. Previous Work in COA Systems

Other researchers have looked into creating more efficient course of action systems. For example, QuickSet allows for COAs to be drawn using a sketch-based interface [181, 86]. Due to the arbitrariness and difficulty in recognizing some symbols, speech



```
(define shape FriendlyUnit
  (components
    (Rectangle unit)
  )
)
```

```
(define shape Armor
  (components
    (FriendlyUnit unit)
    (Ellipse armor)
  )
  (constraints
    (contains unit armor) ...
  )
)
```

```
(define shape ReconArmor
  (components
    (Armor armor)
    (Line recon)
  )
  (constraints
    (contains armor.unit recon)
    (posSlope recon) ...
  )
)
```

Fig. 48. Example of how COA symbols can be described hierarchically using a sketch grammar.

is also utilized to help disambiguate sketched shapes. The system only supports 21 total COA symbols and uses a gesture-based recognizer that constrains the manner in which users must sketch.

The nuSketch Battlespace (nSB) system also allows for sketch-based input [180]. However, sketched glyphs are only used for spatial reasoning purposes. Users must still select the appropriate military unit that the glyph represents by using a toolbar. Despite this, the developers of nSB found that military personnel could create COAs three to five times faster than by hand. This is likely due to the layers utilized by the authors to handle the hierarchical nature of COA symbols.

Stolt employed a hierarchical framework for recognizing COA diagrams [87]. In this work, he employed the LADDER [83] sketch grammar to perform recognition. His approach was capable of recognizing 327 distinct symbols. However, the only symbols supported were those composed of lines, ellipses, single points, polylines, or scribbles. Those consisting of the newer primitives we aim to recognize were not handled.

C. Methodology

The single-stroke primitives we aim to support for the COA domain include line, arc, ellipse, polyline, rectangle, diamond, polygon, arrow, dot (filled-in), wave, gull (double arc), lemniscate (infinity), and the symbol for nuclear/biological/chemical (NBC). Examples of each of these primitives can be seen in Figure 49.

In some hierarchical sketch systems, shapes like rectangles, diamonds, polygons, and arrows are typically handled by the high-level recognizer. When drawn in a single stroke, it is redundant for the primitive recognizer to break the stroke down into a set of lines, pass those lines to a high-level recognizer, only to have it recombine the lines

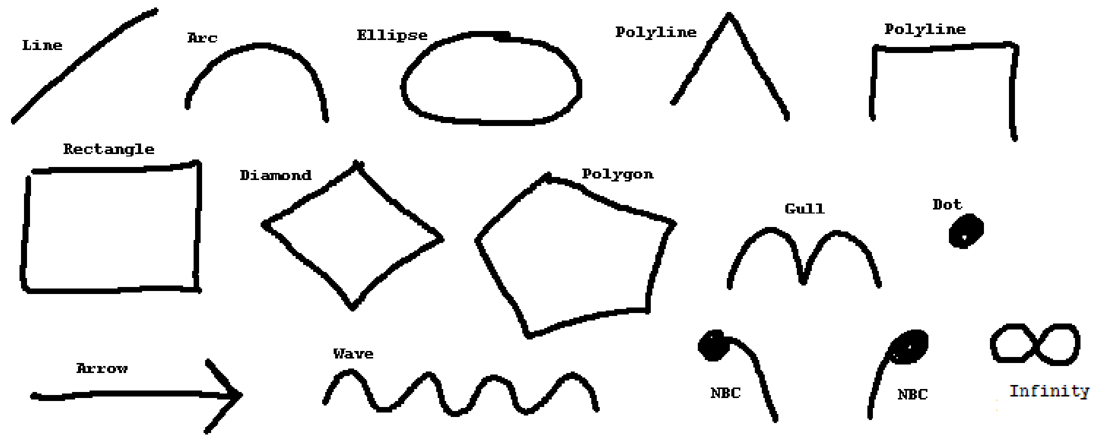


Fig. 49. Example of each of the sketched COA primitives.

back into the original shape. Furthermore, rectangles and diamonds are perceptually important shapes for humans, so the corners are drawn less exact than they are for other polygons. Therefore, we have added these "mid-level" primitive shapes in order to avoid redundant computation. However, to clarify, multi-stroke versions of these primitives (e.g., a rectangle drawn with four individual lines) can still be defined using high-level shape grammars, like LADDER [83].

1. Features

The features of the COA primitive recognizer include many of the same features from the previous chapter, including the CALI features [14, 15], Zernike moments [44], and the expanded Rubine features [111]. A subset of the geometric-based features proposed in the previous chapter are also included. The features related to primitives that are not included in the domain, such as helix, spiral, and curve were removed. In addition to these, features that help to describe the newer primitives in the domain are also included. The only primitive shape that does not require additional features to be sufficiently recognized is the lemniscate (infinity sign).

a. Rectangle & Diamond Features

Rectangles and diamonds are primarily used in the COA domain for representing friendly and hostile units, respectively. For an ideal rectangle interpretation, we use a stroke's bounding box. To determine how well the interpretation matches the original stroke, we utilize a feature area error metric [12]. To find the feature area for a rectangle, each stroke point is first assigned to one of the four sides of the bounding box, whichever side is closest. We then sum the feature areas errors of each of the four sides to their corresponding lines.

In addition to feature area, we also compute the ratio between the total stroke length and the perimeter of the bounding box, as well as the ratio between the lengths of the stroke's major axis (distance between the two furthest points on the stroke) and the diagonal of the bounding box. In an ideal case, both of these ratios should be close to 1.0. To account for diamonds, we simply rotate the stroke 45 degrees and re-compute the same features on the rotated stroke.

b. Arrow Features

Arrows are used throughout the COA domain to represent various attack or movement paths, and are even used to distinguish mortar units. After performing corner finding on a single stroke arrow, we should have at least four segments, as seen in Figure 50. The last two segments make up the arrow head (shown as A and B in Figure 50), the third to last makes up the overtraced part of the arrow head (C), and remaining segments ($\{D\}$) at the beginning of the stroke make up the arrow's path (non-linear paths are allowed). By "segment," we are referring to the red lines produced in Figure 50. After performing corner finding we compute the following features:

- The total number of segments produced from corner finding.

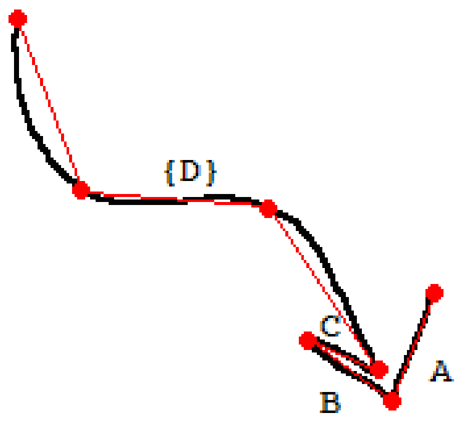


Fig. 50. Example corner finding results on a single-stroke arrow. A and B represent the sub-segments that make up the arrow head, C represents the sub-segment that makes up the overtraced part of the head, and {D} represents the remaining sub-segments that make up the path of the arrow.

- The size ratio between segments A and B, because the segments that make up the arrow head should be similar in size.
- The distance between the first point of segment A and the first point of segment C (normalized by stroke length). This verifies that the two points that make up the tip of the arrow are near.
- The number of times the sub-segments of {D} intersect the line formed between the first point of segment B and the last point of segment A.

c. Dot Features

Filled-in dots are used within the domain to denote artillery units. They are also used as platoon modifiers. For filled-in dots we compute the stroke's density, which is defined as the total stroke length divided by the area of the stroke's bounding box. We also compute the ratio between the stroke's width and height in order to determine if it is close to circular. Another feature that is beneficial to recognizing

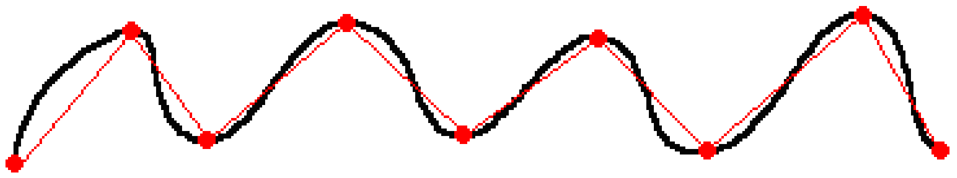


Fig. 51. Example results produced by the alternative segmenter on a sketched wave.

dots is total rotation. However, we do not add this feature, because it has already been calculated in the existing set of features (Rubine).

d. Wave Features

Waves denote amphibious units. To calculate features, we first compute an alternative segmentation that finds points along the stroke where the direction of the stroke changes from positive to negative (or vice versa), as seen in Figure 51. This essentially finds the "peaks" and "valleys" that occur along the stroke. Once this segmentation has been produced, we can calculate the following features:

- The total number of segments produced by the alternative segmenter.
- The ratio between the length of the shortest segment and the length of the largest segment. In an ideal case, all segments should be the same size.
- The percentage of the time that the x value of the first point of each consecutive segment is increasing (if drawn left to right) or decreasing (if drawn from right to left).

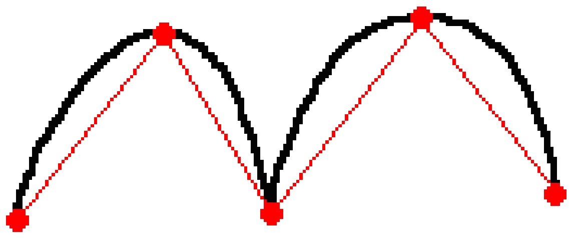


Fig. 52. Example results produced by the alternative segmenter on a sketched gull.

e. Gull Features

Gulls designate airborne units, and like waves, we utilize the results of the alternative segmentation produced in Figure 52. Ideally, the results of this segmentation should produce four segments. From the segmentation, we compute:

- The ratio between the shortest segment and the sum of the lengths of all segments.
- The angle between the second and third segments. If there are not enough segments, then this angle is defaulted to zero, which may be the case in non-gull shapes.
- The average slope of the last two segments, which should cancel out and be close to zero.
- The percentage of the time that the first segment has a positive slope, the second a negative, the third a positive, and the fourth a negative. This feature helps prevent upside-down gulls from being recognized as gulls.

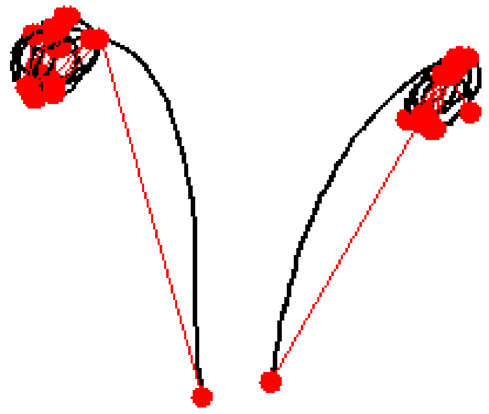


Fig. 53. Example results produced by the alternative segmenter on sketched NBC symbols.

f. NBC Features

NBC stands for “nuclear, biological, chemical”, and its symbol is drawn as a filled-in dot that has an extended tail. To recognize these symbols, we use the same segmentation that we did with waves and gulls, as seen in Figure 53. Ideally, the longest of the end segments (either the first or last) will represent the tail, while the remaining segments make up the filled-in dot portion of the symbol. The features we use to help recognize NBC symbols include:

- The length ratio between the shortest of the first and last segment, and the longest of the first and last segment. Ideally, this ratio should be large because the tail portion of the symbol should be much longer than the dot portion.
- The density of the sub-stroke that makes up the dot portion of the symbol.
- The number of revolutions (total rotation) of the sub-stroke that makes up the dot portion of the symbol.

2. Classifier

For classification, we used a multi-layer perceptron. As with the MLP in the previous chapter, we utilized the WEKA toolkit [178] and used the exact same parameters. The number of input nodes was equivalent to the number of features we computed (125), and the number of output nodes represented the number of primitive classes that we supported for this domain (13). The single hidden layer consisted of 138 nodes, again, because this value equaled the sum of the number of input and output nodes. The MLP used a sigmoid activation function at each node, and weights were learned through backpropagation. We specified a learning rate of 0.3, with a momentum value of 0.2, and performed 500 training epochs. Feature values were also normalized before entering the network.

3. Data

For our experiments, we collected two independent sets of COA data. When collecting the first set of data (A), we asked 10 users to draw two examples of each of the COA symbols shown in Figure 54. This specific subset of COA shapes were chosen in order to collect as much data for each individual primitive as possible, without burdening the user. Because we are concerned with only recognizing shapes and symbols, strokes used to represent text were ignored. The problem of clustering text strokes and shapes strokes is an ongoing area of research in sketch recognition [131, 133]. The second set of data (B) was collected from a single user. The goal of this data set was to test the robustness of the system with a large number of COA symbols. This user drew between 10 and 20 examples of 379 unique symbols in the domain. Table III shows the number of each primitive collected in each data set.

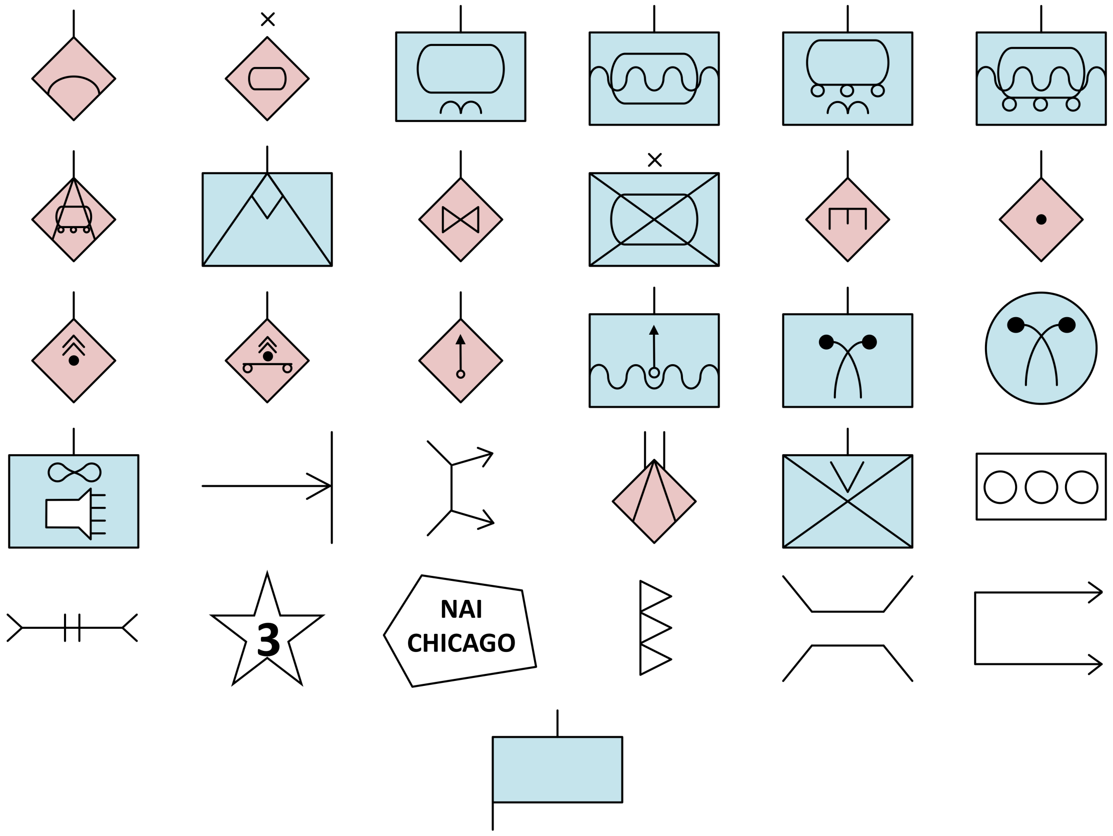


Fig. 54. Subset of COA symbols used for data collection (data set A).

D. Results

We performed three experiments in total. The first two involved performing 10-fold cross validation on each of the data sets. In the third experiment, we used data set A for training and data set B for testing. This shows the results of a classifier trained with only a subset of COA symbols, but tested using a more expansive set of symbols drawn by a user that provided no training data himself.

Table III shows the results of each of the three experiments on a primitive-by-primitive basis. Data set A, which contained data from 10 different users had a total weighted accuracy of 98.5%. The data containing a larger number of COA symbols, but drawn by a single user, had a total weighted accuracy of 99.8%. In our third experiment where data set A was used for training, and set B for testing, we achieved a weighted accuracy of 99.4%. Overall, recognition took approximately 11 milliseconds[1] per primitive, even with the computation of all 125 features.

E. Discussion

One point that we should make about the data collected in set B is that it contains a larger majority of lines than that of set A. The reason for this is because set B contains "anticipated" unit symbols, which are drawn using dashed boundaries. Lines are arguably the easiest primitive to recognize, but they are also the most common. One may argue that we achieve high accuracy rates on data set B simply because it contains a large number of lines. However, we would like to point out that if we ignore lines altogether, data set B still achieves over 99% weighted accuracy on the remaining primitives during cross-validation.

[1]The reason for the speed-up versus the results in the previous chapter are due to the recursive process of complex shape interpretation not being used.

Table III. Number of occurrences of each primitive shape in our data sets as well as the accuracy of 10-fold cross-validation on each set. The final column shows the accuracy results when we train with dataset A and test with dataset B. “Total” accuracy represents the flat average of all shape accuracies, while “weighted” average shows the average accuracy of each shape weighted by its number of occurrences. *Dataset B consisted of more lines because it included “anticipated” unit symbols, which are drawn as dashed rectangles or diamonds.

Primitive	Number (A)	Accuracy (A)	Number (B)	Accuracy (B)	Accuracy (A/B)
Line	837	0.996	32075*	0.999	0.999
Arc	20	1.0	30	1.0	1.0
Ellipse	374	0.987	2266	0.994	0.996
Polyline	376	0.979	1036	0.986	0.973
Rectangle	181	0.989	1419	0.996	0.937
Diamond	202	0.990	667	1.0	0.993
Polygon	61	0.934	265	0.986	0.830
Arrow	82	0.939	98	1.0	0.949
Dot	62	0.984	325	0.972	0.946
Wave	19	0.947	158	1.0	0.981
Gull	41	0.976	67	0.970	0.896
Infinity	20	0.900	11	0.818	0.727
NBC	61	1.0	42	1.0	1.0
Total/Avg.	2336	97.1%	38459	97.9%	94.1%
Weighted		98.5%		99.8%	99.4%

The major limitation of the existing work is that primitive shapes must be drawn using a single stroke. Although multi-stroke primitives, such as 4-line rectangles or 3-line arrows, can still be defined and recognized through a high-level shape grammar, recognition still suffers from the existence of continuation and touch-up strokes [182]. Furthermore, recognizing multi-stroke primitives at the low-level will relieve some of the complexity from the higher-level recognition system. The biggest obstacle to achieving this is the development of adequate clustering algorithms for detecting groups of strokes that make up a single primitive that remain domain-ignorant. The next chapter of this dissertation discusses our first attempt at solving this problem.

F. Chapter Summary

We have presented a method for recognizing a large number of primitive shapes that can be used to describe sketched symbols in the military course of action domain. Many of these primitive shapes, such as rectangles, diamonds, waves, and infinity signs may also be present in additional sketch domains. Our approach involves combining the features of existing recognizers with those developed to help recognize the newer shapes of the domain. A multi-layer perceptron was used to perform classification, and results indicate recognition rates that exceed 98.5% on a data set gathered from 10 different users. The system has further been tested on data representing 379 unique COA symbols collected from a single user, with weighted accuracy rates above 99%.

CHAPTER V

MULTI-STROKE PRIMITIVES

A. Introduction

Thus far, we have assumed that all primitive shapes will be drawn with a single stroke. Although this assumption typically holds true in most cases, there are instances in which a user may choose to draw a primitive shape with more than one stroke. We call these additional strokes *continuation strokes* (Figure 55).

The reason continuation strokes are problematic is because high-level recognizers assume that every primitive has meaning. For example, if a user draws a line to the screen and decides later in the sketching process to make it longer, then the low-level recognizer will find and return two individual lines to the high-level recognizer. At this point, the high-level recognizer will attempt to find shape definitions that consist of two lines rather than one.

As another example, imagine that a user draws a rectangle to the screen, but does not fully close the space between the endpoints (as seen in Figure 55). Mistakenly, the user then attempts to "help" the recognizer by adding an additional stroke to close

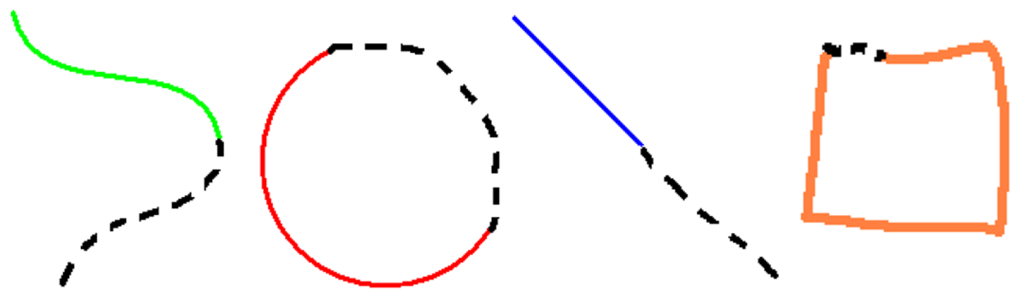


Fig. 55. Examples of continuation strokes.

the gap. We call these *touch-up strokes*. Most likely, the primitive recognizer will still be able to classify the first stroke as a rectangle, but will classify the second stroke as a line. Now, the high-level recognizer has an additional line that it is incorrectly attempting to place into a shape definition.

The ultimate goal of our work is to place no constraints on how users must draw primitive shapes; this includes the single-stroke assumption. We believe it is the job of the low-level recognizer to handle multi-stroke primitive recognition, because it is the part of the multi-tiered sketch process that has knowledge of basic shapes. The high-level recognizer only has knowledge of spatial and geometric relationships between recognized primitives.

The biggest obstacle to achieving multi-stroke primitive recognition at the low-level stage of recognition is developing adequate grouping algorithms for detecting strokes that make up a single primitive, while remaining domain-ignorant. By itself, grouping strokes is an exponentially hard problem because every stroke must be compared with every other stroke on the screen. Some researchers have looked into ways of speeding up the process, but it still remains a $O(n^2)$ problem [183].

One way that existing researchers have attempted to solve the multi-stroke problem is by placing constraints on how users draw shapes with multiple strokes. For example, some algorithms use temporal constraints, or timeouts, to denote shape completion [13, 14]. With these systems, users must either draw an entire shape within a specified time, or make continuation strokes within a given time frame to be recognized. Other systems utilize simple button clicks to designate when a shape is finished and ready to be recognized [90]. Another constraint used by some systems is to disallow stroke interspersing [184]. That is, if the user draws shape A and then draws shape B, she is not allowed to make additional changes to shape A. Shape A must be fully completed before the user can continue drawing any other shape. In

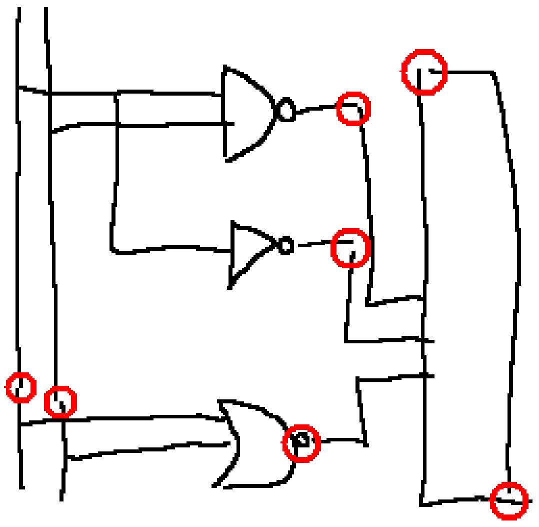


Fig. 56. Examples of continuation strokes in a hand-sketched circuit diagram.

domains like circuit diagrams, lines (wires) are often extended after other shapes have already been drawn (Figure 56). Finally, some algorithms compare the slopes and spatial distances between strokes, but only work for shapes composed of lines. For example, the Tahuti recognizer only supports multi-stroke rectangles and arrows [29]. The goal of this chapter is to explore a potential means for recognizing multi-stroke primitives, linear and curvilinear, without requiring these special constraints.

B. Implementation

In order for the low-level recognizer to classify multi-stroke primitives, it must have more communication with high-level recognizers which have access to every stroke on the screen. Traditionally, primitive recognizers are treated as black boxes where an individual stroke comes in and a recognized primitive goes out. In order to recognize multi-stroke primitives, the low-level recognizer must also have access to every stroke on the screen.

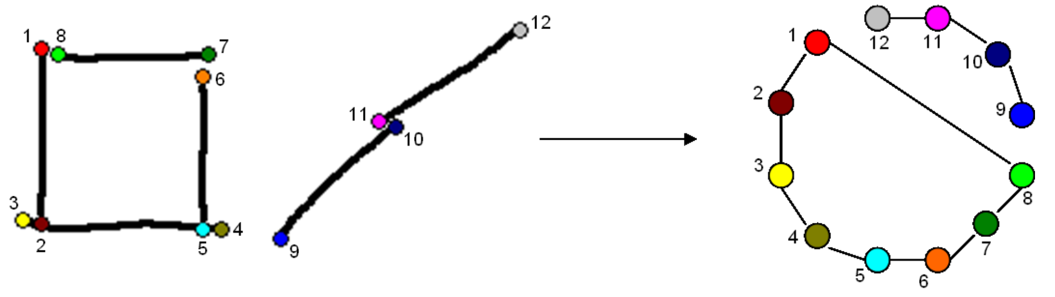


Fig. 57. Example graph built from a set of six strokes representing two primitive shapes.

In the current implementation of our multi-stroke algorithm, we require that the high-level recognizer gives a set of strokes as input to the low-level recognizer. These can be all of the strokes on the screen, or just a subset of strokes if the high-level recognizer employs its own form of stroke grouping [183]. The low-level algorithm returns, as output, an optimal set of recognized primitives. The low-level recognizer maintains a hash table history, keyed to the unique ID of each stroke, to avoid redundant recognition of individual strokes. Once all strokes have been classified using PaleoSketch, they are sent to the multi-stroke algorithm. Strokes that are classified as complex shapes are not considered for multi-stroke combination. The multi-stroke algorithm has five parts: graph building, graph searching, stroke combination, false positive removal, and arrow detection.

1. Graph Building

The first step of our multi-stroke algorithm involves constructing a graph of spatially close strokes (Figure 57). We are make the assumption that candidate strokes for multi-stroke primitives will likely have endpoints that are near one another. In our graph, nodes represent the endpoints of strokes and edges represent endpoints that are spatially near or lie on the same stroke. To determine if the endpoints of two different strokes are near, the following conditions must be met:

1. The Euclidean distance between the endpoints, divided by the average stroke length of the two strokes, must be within some threshold (we used 0.1^1). Another way to think about this is that the gap between the two strokes must be less than 10% of the average stroke length of the two strokes. A similar metric was used in the original version of PaleoSketch to determine if a stroke represented a closed shape [6].

2. The Euclidean distance between the endpoints, divided by the average width (or height, whichever is largest) of the bounding boxes of the two strokes, must be within some threshold (we used 0.15).

In both of these distance metrics, we divide by measures of stroke size. We do this so that larger strokes are allowed to have larger gaps between themselves, while smaller strokes should have smaller gaps. However, if the distance between the two endpoints is smaller than 10 pixels, then the endpoints are automatically considered near. In these cases, we use this static pixel threshold, rather than the tests based on stroke size, to allow for small strokes to be considered connected as well. In particular, we found multiple instances of small strokes being present at the beginning of much larger strokes. This typically occurs by accident when the user to about to begin sketching. With this static threshold, however, these errant strokes can be absorbed into the larger strokes.

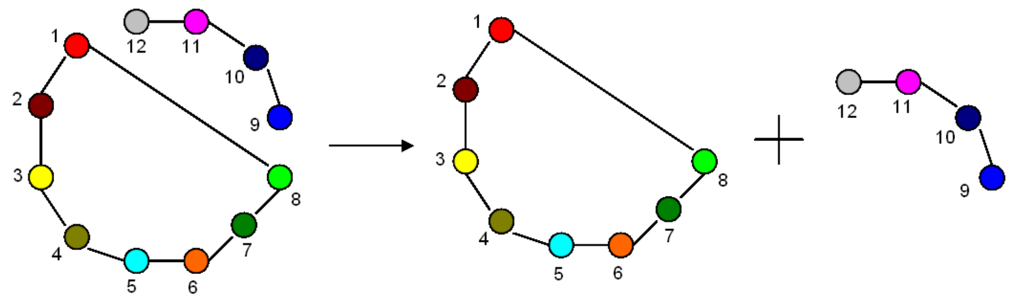


Fig. 58. We use a searching algorithm to find the strongly connected components of the graph. These correspond to the strokes that are likely candidates to be merged.

2. Graph Searching

Once our graph has been generated, we search it for *strongly connected components* (Figure 58). Strongly connected components are the maximal, strongly connected subgraphs within a graph. By "strongly connected," we mean that from every node in the graph there exists some path to every other node in the graph. Typically, these strongly connected subgraphs will indicate candidate strokes that need to be combined into multi-stroke primitives.

To search for strongly connected components, we utilize Tarjan's algorithm [185]. Tarjan's algorithm is a depth-first search algorithm and runs in linear time in the number of edges present in the graph. Once all possible sub-graph combinations have been discovered, we rank the combinations from those containing the most nodes to those containing the least nodes.

At this point, we have a list of multiple stroke *combinations*. Each combination in the list likely has overlapping strokes with the other combinations in the list. The example in Figure 58 shows only the top-two connected components that would be

[1]Thresholds determined empirically with an original set of author-drawn data.

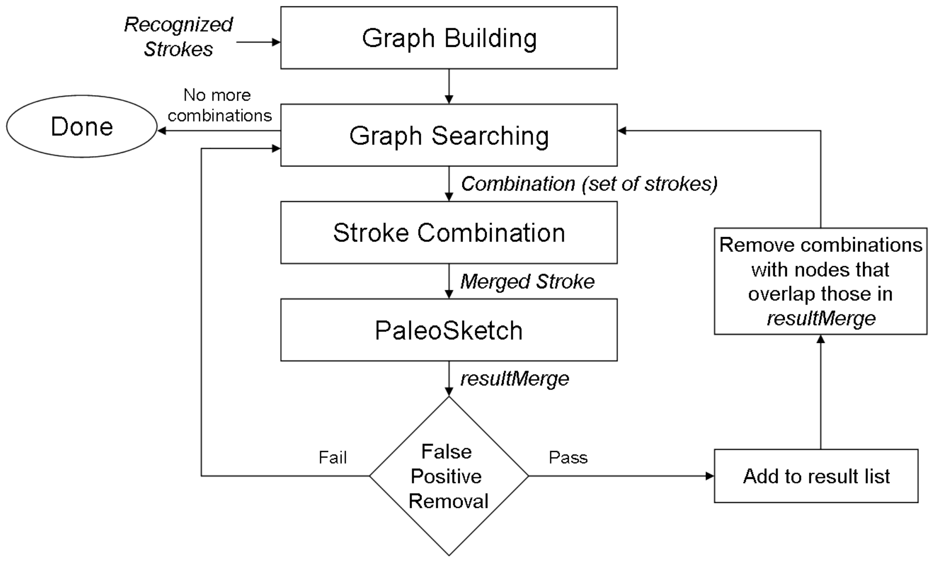


Fig. 59. Multi-stroke recognition flowchart.

present in the list of combinations. All possible sub-graphs of these two components will also be present in the list. This is why the list is first ranked based on the number of nodes.

Next, we greedily iterate through the list of combinations. The strokes of each combination are merged together using the algorithm in the next step ("Stroke Combination"). This merged stroke is then classified using PaleoSketch. The stroke is then sent to the "False Positive Removal" stage, which determines if the stroke should remained combined or not. If the stroke passes this stage, we have successfully recognized a multi-stroke primitive. All remaining combinations in the list that contain any of the nodes that were used for this primitive are removed. If the stroke fails the "False Positive Removal" stage, then the combination fails and the algorithm continues with the next combination in the list. Figure 59 shows a flowchart of this process.

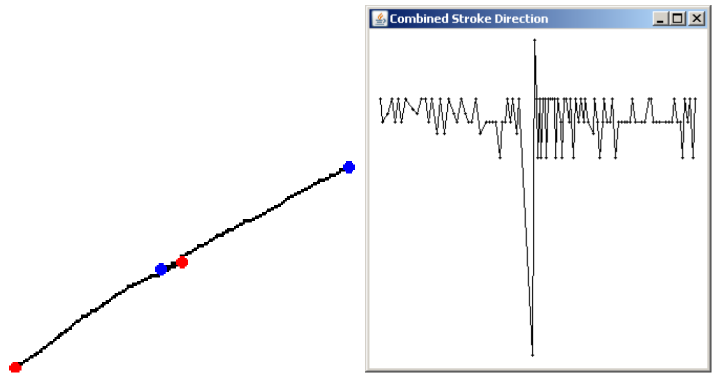


Fig. 60. Example of two lines that are merged by simply connecting the endpoints. In this case, the direction graph contains a discontinuity even though lines typically have a smooth direction graph.

3. Stroke Combination

Initially, stroke combination seems like a trivial problem, but some consideration must be taken before merging strokes together. First, we must ensure that the strokes are in a logical order. This order may not necessarily be the order in which they were drawn. For example, imagine a user draws a rectangle with four lines. The first line drawn is the left side of the rectangle (L), followed by the bottom (B), then the top (T), and finally the right side (R). In this case, it is not logical to order them based on time ($LBTR$), because we would make an incorrect connection between the endpoints of the top and bottom strokes. Instead, we would want a logical combination, such as $LBRT$ or $LTRB$.

Another issue we must consider when combining strokes is that connecting the endpoints directly may not be the best option when merging two strokes together. Algorithms such as PaleoSketch rely heavily on the consistency of direction and curvature graphs. If we simply connect the endpoints of two strokes that slightly overlap, then we will have a discontinuity in the merged stroke's direction graph (Figure 60).

Therefore, if we are connecting the last endpoint of stroke A with the first endpoint of stroke B, we start by searching for the two points (within the last 10% of stroke A and first 10% of stroke B) that are closest to one another. We then connect the strokes at these two points, clipping the trailing/leading portions of the two strokes at the ends. If the strokes overlap, then this clipped portion will coincide with the overlapping sections. If the two strokes have a gap between them, then the two points that are closest to each other would naturally be the endpoints. In this case, it is safe to simply connect them.

4. False Positive Removal

Once a stroke has been generated from the combination of other strokes, we want to determine if the resulting classification of that stroke is better than the result of simply leaving the strokes unmerged. Essentially, these are rules of when not to combine strokes together. Recall that we not only have the interpretation of the merged stroke, but we also have the original shape interpretations of the strokes that were combined.

We have established the following principles that specify conditions in which a set of strokes should **not** be merged. These principles have been determined empirically through the observation of merging problems with an original set of author-drawn data.

1. *Avoid complexity when possible.* For example, if the result of merging two shapes together is a complex shape, then merging should not be performed. Obviously, combining two shapes, only to divide them later in the recognition process, is counter-intuitive and unnecessary.

2. *Shapes should maintain constant (curvi)linearity.* If the result of merging is

a linear shape (e.g., line, polyline, polygon), then all of the strokes that were merged should have originally been linear as well. For example, two arcs should not form a polyline. Furthermore, when combining shapes into a polyline, the degree of that polyline should equal the sum of the degrees of the sub-shapes (e.g. four lines should not be combined into a "Polyline (5)" or a "Polyline (3)"). Likewise, if the result of merging is a curvilinear shape (e.g., arc, ellipse, wave), then at least one of the original shapes should have also have been curvilinear. For example, three lines should not be merged into an arc.

3. *Touch-up strokes should have a minimal effect on existing recognized shapes.* Touch-up strokes occur when users are "cleaning up" existing recognized, completed shapes. Because of this, they should not change the recognition of the previously recognized stroke. For example, if the previously recognized shape is a closed shape (e.g. rectangle, ellipse), then merging an additional stroke with this shape should not change the recognition result. If it does, then we should not merge. Touch-up strokes are also often small compared to the original shape. Thus, we also prohibit merging if the stroke alters an original closed shape's bounding box by more than 10%.

4. *Continuation strokes should maintain consistency.* Continuation strokes occur when a user attempts to make an existing shape longer. When merging two lines into one line, it makes sense that the path should be continuous. When extending the shaft of a previously recognized arrow, the result of the merge should remain an arrow. Shapes like waves should also maintain a consistent path. To check this, we verify that the new wave's height does not exceed 50% of the original wave's height.

5. *Impose confidence constraints on shapes when needed.* In some cases, particular

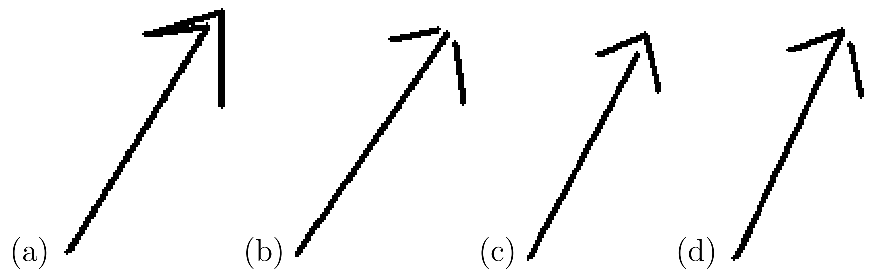


Fig. 61. Examples of the different ways that users draw arrows: with one stroke (a), with three strokes (b), or two different variations of two strokes (c,d).

shapes may lend themselves to being the frequently defaulted-to option when existing shapes are errantly merged. In the case of our initial set of author-drawn data, this tended to be complex shapes and arrows. Because complex shapes are avoided (principle 1), we only had to worry about arrows. From our observation, most true positive arrows will have a high confidence, while defaulted-to arrows have a low confidence. By requiring multi-stroke arrows to have a confidence of at least 75%, we were able to continue to recognize true positive arrows while reducing the number of false positive merges.

5. Arrow Detection

Arrows are typically drawn in one of four ways, as seen in Figure 61. Some users draw the shaft and the head all in a single stroke (Figure 61(a)). This case is handled by the single-stroke primitive recognizer and occurred 65.4% of the time in our data set. Other users will draw the shaft in one stroke and each of the two heads with two additional strokes (Figure 61(b)). In this case, the endpoints of all three strokes should be near and can be handled by the existing multi-stroke algorithm. This case occurred in only 3.7% of the arrows drawn in our data set. However, there are two cases that cannot be handled by the existing multi-stroke algorithm, because the endpoints of the strokes that need to be combined will not be near. To support these cases, we have developed a set of rules to search for these specific types of arrows.

1. The first case involves the user drawing a shaft in one stroke, and the arrow head in a second stroke (Figure 61(c)). This occurred 20.6% of the time in our data set. In this case, the head will likely be drawn as a two-lined polyline. Therefore, as we loop through the strokes we are processing, if we encounter a "Polyline (2)" interpretation, we perform the following. First, we find the the segmentation point of the polyline. This is the tip of the arrow head. Next, we loop through the remaining non-closed strokes and determine if any of their endpoints are near the tip of the arrow head (using a similar distance check to that in "Graph Building"). If we find a candidate, then we combine the strokes together and classify using PaleoSketch. If the interpretation returned is an arrow with greater than 50% confidence, then we return the arrow interpretation.

2. The second case occurs when the user draws the shaft and one half of the arrow head in a single stroke, and the second half of the arrow head with a single line (Figure 61(d)). This happened 10.3% of the time in our data set. Remember that the shaft does not necessarily have to be a simple line. If we encounter a "Polyline (2)" or a complex interpretation containing one line and one non-closed shape, we do the following. First, we loop through the remaining line-only strokes and perform a distance check between its endpoints and the segmentation point of the polyline or complex. If they are near, we combine them and re-classify. Again, if an arrow interpretation is returned with greater than 50% confidence, then we keep the strokes merged and return the interpretation.

C. Experiment

We asked 10 users to draw 2 examples of each of the symbols and small-scale diagrams shown in Figure 62. These samples come from various sketch domains including military course of action, circuit diagrams, chemistry, physics, and mechanical engineering. We chose to have users draw these symbols rather than ask them to artificially draw examples of multi-stroke primitives. This not only allows us to test our accuracy, but to determine how frequently multi-stroke primitives occur in naturally sketched data.

The symbols in this study are composed of a number of primitives from the previous two chapters. Specifically, the primitives involved in this study are: lines, arcs, curves, ellipses, diamonds, rectangles, polylines, polygons, arrows, dots, waves, helixes, spirals, infinity signs, and complexes.

At no point in the study were users asked to explicitly state what the intention was of each of their drawn strokes. In order to determine the correctness of our recognizer, we use the interpretation of the shape as it would be defined in a high-level grammar's shape definition.

To test our algorithm, we performed 10-fold cross validation over the entire set of data. With this, one user's data is saved for testing while the remaining nine are used for training. This is done for all users and the results are averaged. When training our MLP version of PaleoSketch, we trained only on single strokes (i.e., we did not train on labeled multi-stroke primitives). We computed accuracies for both single-stroke and multi-stroke primitives. In order for a multi-stroke primitive to be classified as correct, it must have been properly merged as well as recognized, with no extra or missing strokes in the interpretation.

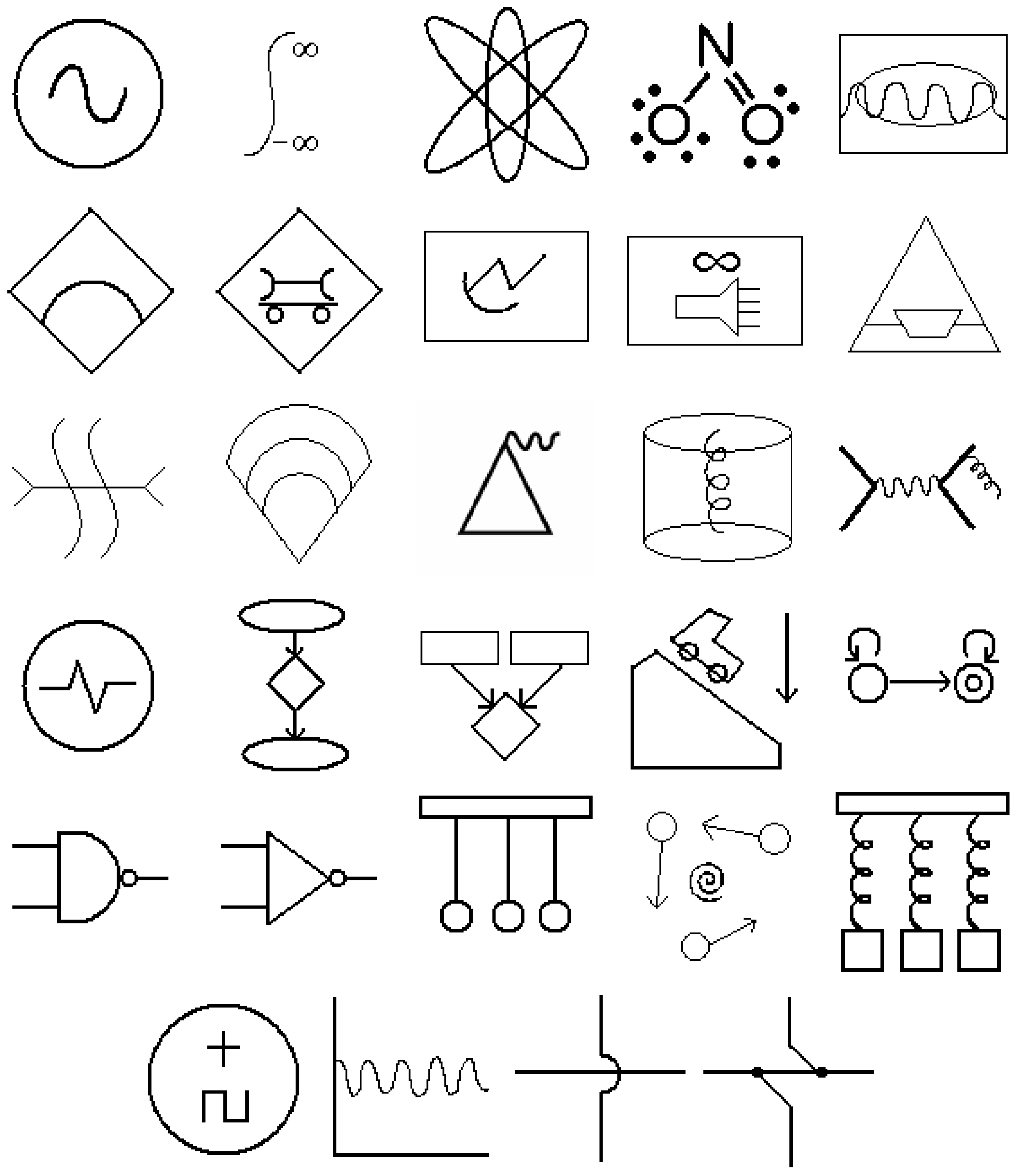


Fig. 62. Symbols from multiple domain that were used for data collection.

D. Results

Table IV shows the percentage of the time that we encountered multi-stroke primitives in our data set. In all, multi-stroke primitives made up only 9% of all of the primitives in the data set. The most commonly drawn primitives using multiple strokes were polygons (39.3%), arrows (35.2%), diamonds (30%), and rectangles (18.8%). All other primitives were drawn with multiple stroke less than 10% of the time. Interestingly, our results coincide with those of van Sommers when he explained that the more sides there are in a figure, the less threading (drawing multiple lines within a single stroke) there is per corner [127].

1. Single-stroke

Table V contains the accuracies of both single-stroke and multi-stroke primitives. In total, we achieved 96% recognition of all primitives in the data set. With single-stroke primitives, our weighted accuracy was 96.6%. This is slightly more than a full percentage point lower than the results in Chapter III. However, much of this error is due to some specific shapes.

For example, single-stroke spirals performed poorly with only 78.9% recognition. In Chapter III, we had a perfect 100% recognition of spirals. We believe the primary reason for this decrease is the lack of spiral training data in the set of symbols we collected. In total, there were only 20 spirals drawn, which is less than 1% of the entire data.

Another troublesome primitive was the dot (88.6%). In this instance, most of the incorrect dots were all drawn by a single user. Unlike other users who drew dots as small, filled-in circles, this user simply made quick, small strokes to the screen to designate dots. In most cases, these were classified as lines (Figure 63). In some

Table IV. Number of each type of sketched primitive present in the collected data. Complex and polyline shapes cannot be drawn with multiple strokes.

	Total	Single-stroke	Multi-stroke	% Multi-stroke
Arc	105	101	4	3.8%
Arrow	216	140	76	35.2%
Complex	120	120	-	-
Curve	78	78	0	0%
Diamond	60	42	18	30.0%
Dot	219	202	17	7.8%
Ellipse	478	462	16	3.3%
Helix	84	82	2	2.4%
Infinity	59	59	0	0%
Line	485	475	10	2.1%
Polygon	117	71	46	39.3%
Polyline	214	214	-	-
Rectangle	149	121	28	18.8%
Spiral	20	19	1	5.0%
Wave	67	62	5	7.5%
Overall	2471	2248	223	9.0%

Table V. Accuracy results after 10-fold cross-validation. "Average" represents flat averages, while "weighted averages" represent averages that are weighted by the number of occurrences of each shape type.

	Single-stroke	Multi-stroke	Weighted Avg.
Arc	0.980	0.0	0.943
Arrow	0.979	0.934	0.963
Complex	0.908	-	0.908
Curve	0.962	-	0.962
Diamond	0.976	0.944	0.967
Dot	0.886	1.0	0.895
Ellipse	0.987	0.688	0.977
Helix	0.963	1.0	0.964
Infinity	1.0	-	1.0
Line	0.992	0.900	0.990
Polygon	0.930	0.913	0.923
Polyline	0.958	-	0.958
Rectangle	0.983	0.929	0.973
Spiral	0.789	0.0	0.750
Wave	1.0	1.0	1.0
Average	95.5%	75.5%	94.5%
Weighted Avg.	96.6%	89.7%	96.0%

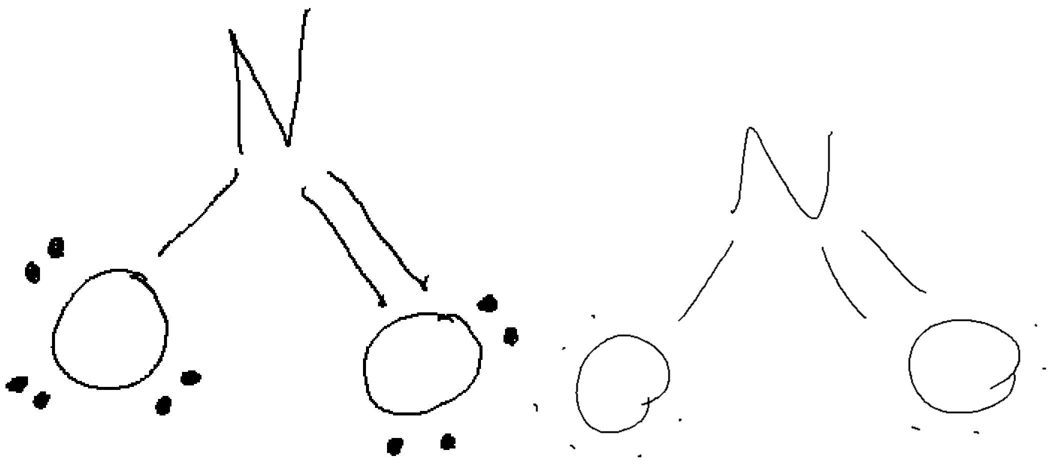


Fig. 63. Examples showing the way that most users drew dots (left) versus the single user that drew them with small strokes (right).

systems, any small markings relative to the screen size are automatically classified as dots [160]. If we were to make a similar assumption, then these dots could easily be classified correctly.

Other shapes that had lower single-stroke accuracies were complex shapes (90.8%) and polygons (93%). The biggest problem for polygons was distinguishing non-rectangular quadrilaterals from rectangles. There was also much confusion between polygons and complexes. Issues with complex shapes will be discussed in more detail in a later section.

2. Multi-stroke

Overall, we achieved 89.7% weighted accuracy for multi-stroke primitives. This means that the primitive was both combined and recognized correctly. Examples of correctly classified multi-stroke primitives can be seen in Figure 64. Primitives that did not combine, but should have, are considered *false negatives.* Primitives that were not meant to be combined, but were, are considered *false positives.*

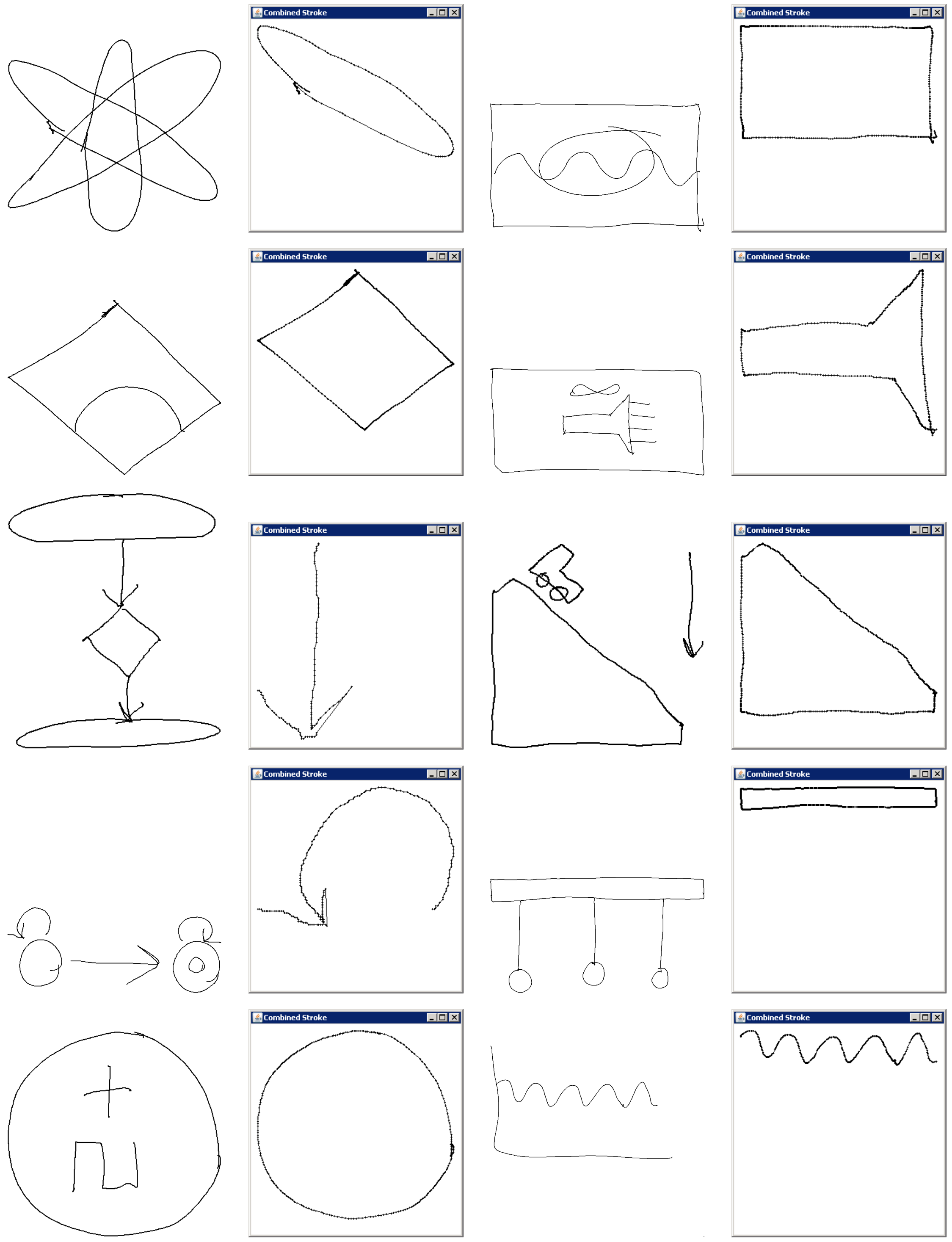


Fig. 64. Examples of correctly merged and classified multi-stroke primitives (original sketch and combined primitive).

Most of the errors we encountered were due to strokes not being merged correctly, rather than being mis-classified (false negatives). One limitation of our multi-stroke approach is that it relies heavily on single-stroke interpretations being correct. For example, some strokes that should have merged to form a multi-stroke primitive were incorrectly classified as a complex shape. Because we disallow complex shapes from being merged, the strokes were never combined.

Another reason some strokes were not combined is because the distance between their endpoints was too great. Examples of this can be seen in Figure 65. In these examples, the endpoint of the larger stroke is closer to the midpoint of the smaller stroke than it is to its endpoint. Therefore, the distance between the endpoints was not within the thresholds that we specified in multi-stroke algorithm, which were based on the size of the strokes to combine.

Additional issues that we encountered were examples of users attempting to correct poor sampling or users performing overtracing. Figure 66 shows an example of a user who, while drawing an ellipse, encountered a lag during sampling. This resulted in many points being lost, which the user then attempted to supplement with an additional stroke. Likewise, in Figure 67, one user attempted to clean up some of his shapes by adding additional, overtraced strokes. In both of these cases, strokes were never merged because the endpoints were too far apart.

When computing accuracy, we did not account for false positive merges. However, in all 2,471 primitives that we tested, we only encountered 6 false positive combinations. Furthermore, most of these false positives consequently caused false negatives to occur.

For example, in Figure 68 one user drew both of his NAND gates using 3 lines and an arc, rather than one line and an arc. Since the shape definition of a NAND gate would only consist of one line and one arc, we consider anything containing more

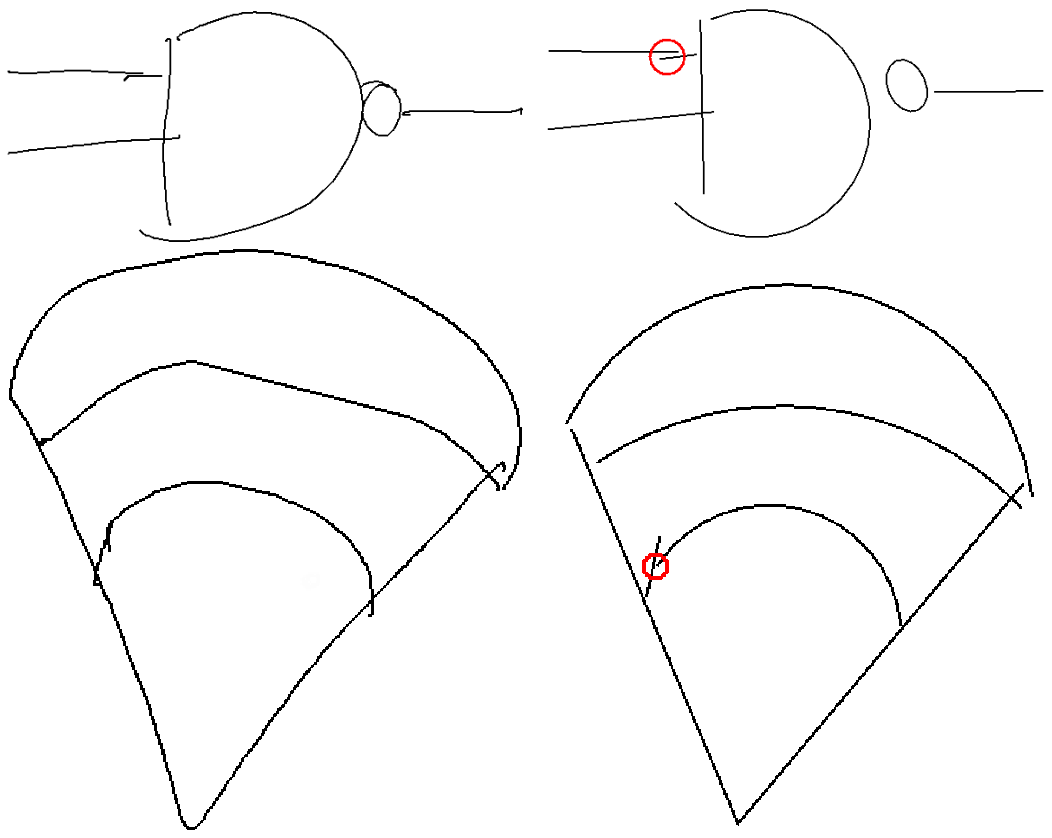


Fig. 65. Examples of multi-stroke primitives that were unsuccessfully merged due to endpoint distances being too great.

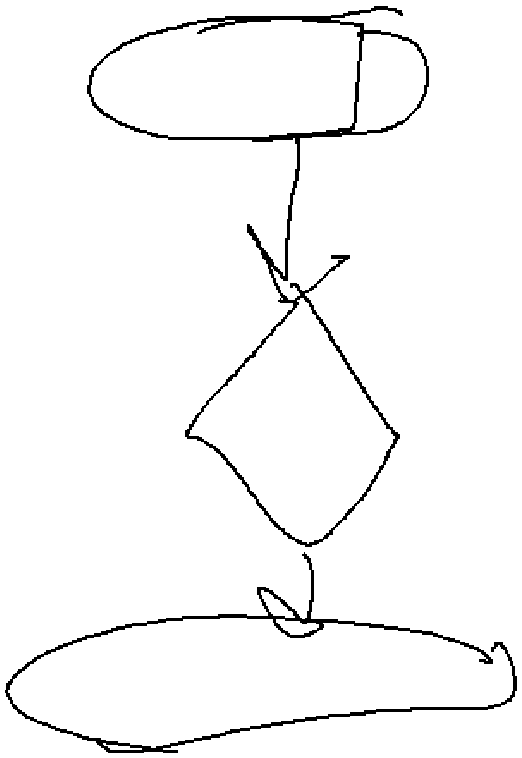


Fig. 66. Example of a user attempting to correct an error in sampling (top ellipse).

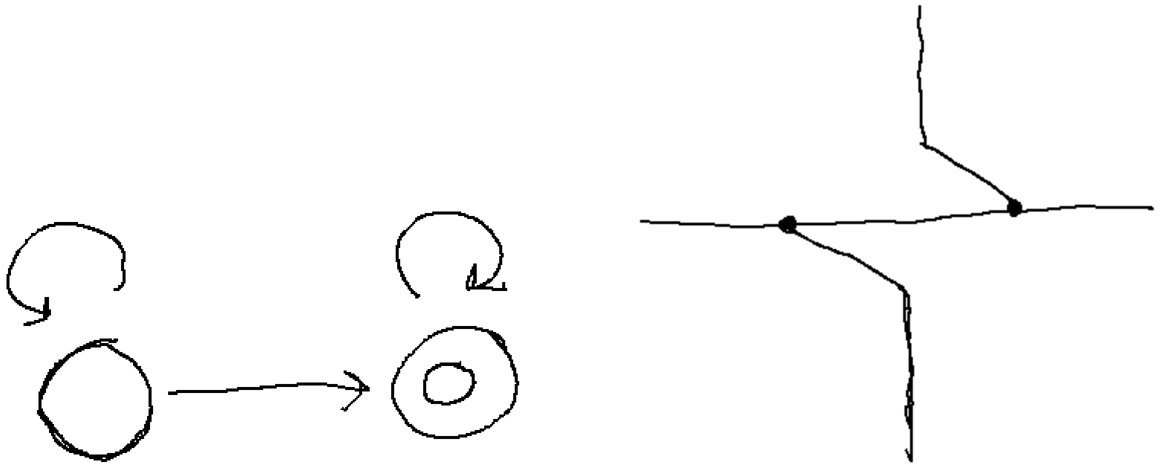


Fig. 67. Examples of multi-stroke overtracing performed by one user.

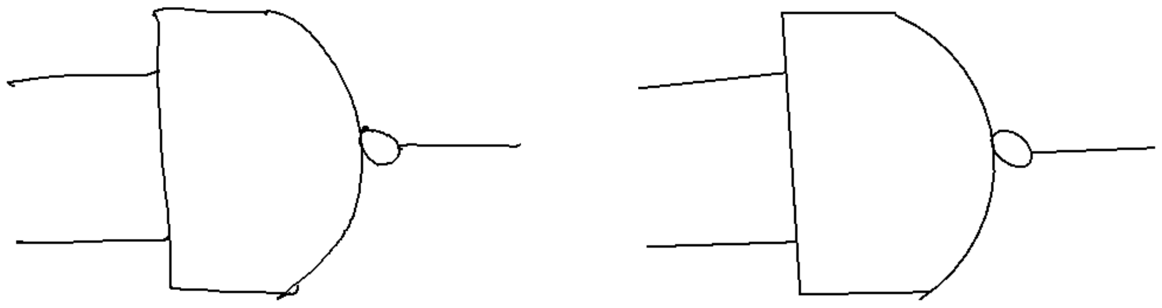


Fig. 68. Example of a NAND gate drawn by one particular user. This resulted in both a false positive and a false negative combination.

components to be incorrect. In this case, these extra lines were merged with the vertical line to produce a three-line polyline. This was both a false positive because the lines were merged with the vertical line, as well as a false negative because they were not merged with the arc.

As another example, see Figure 69. In this example, a false positive (and negative) was generated due to the greedy nature of our algorithm. The user drew both vertical lines first, followed by two strokes to make up the bottom ellipse. The first stroke that makes up the ellipse was incorrectly merged as a polyline with the two vertical strokes. This, consequently, kept it from correctly merging with the second half of the ellipse, which caused a false negative.

In some few cases, the algorithm did correctly merge the strokes, but they were mis-classified. Figure 70 shows an example of two strokes that were correctly merged, but then mis-classified as a rectangle instead of a polygon.

3. Complex Shapes

As with our experiments in Chapter III, we also wanted to determine how well our algorithm performed on complex interpretations. Unlike our previous data set that

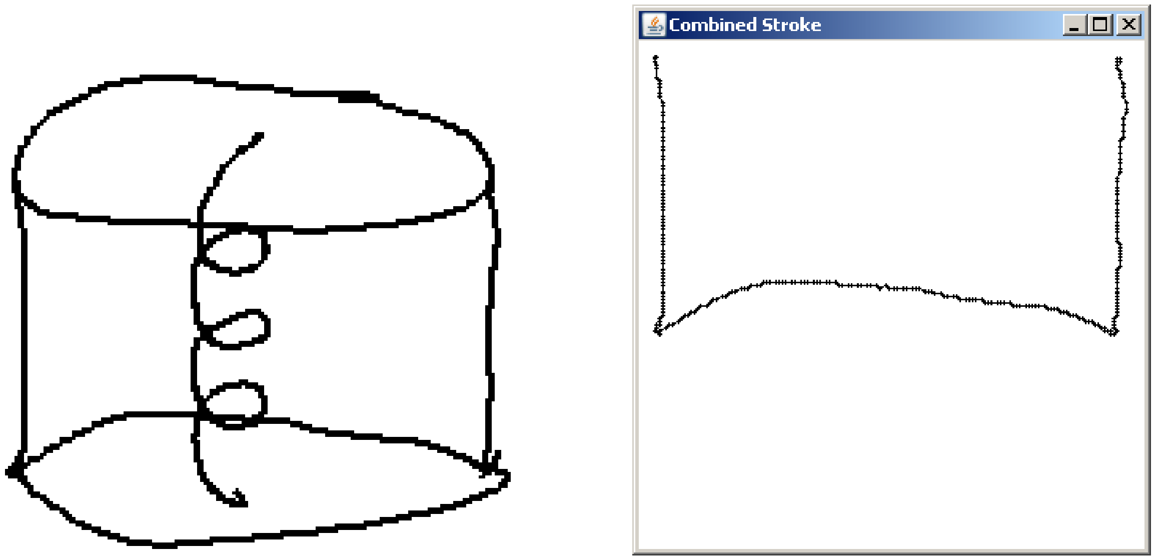


Fig. 69. Example of a false positive generated due to the greedy nature of the algorithm.

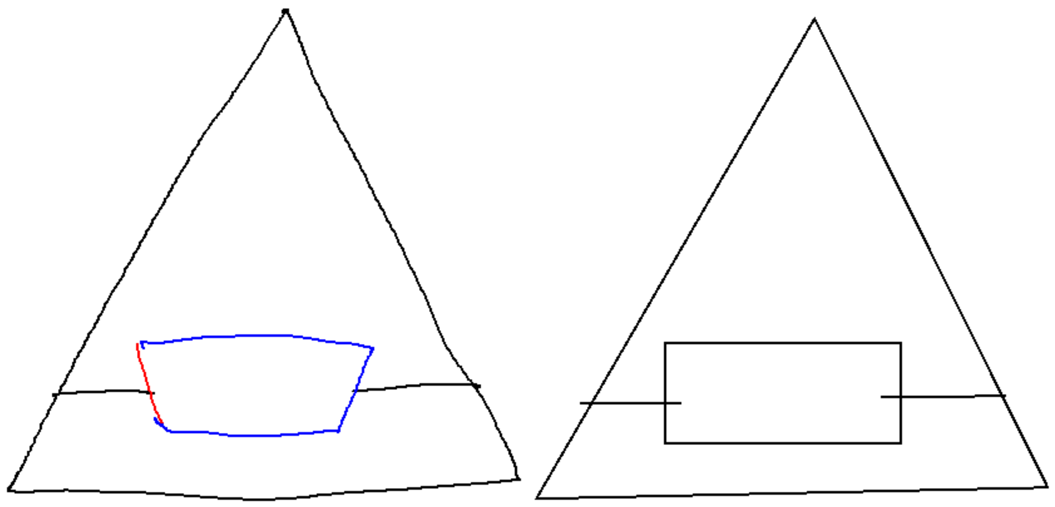


Fig. 70. Example of multi-stroke primitives that was merged correctly, but classified wrong.

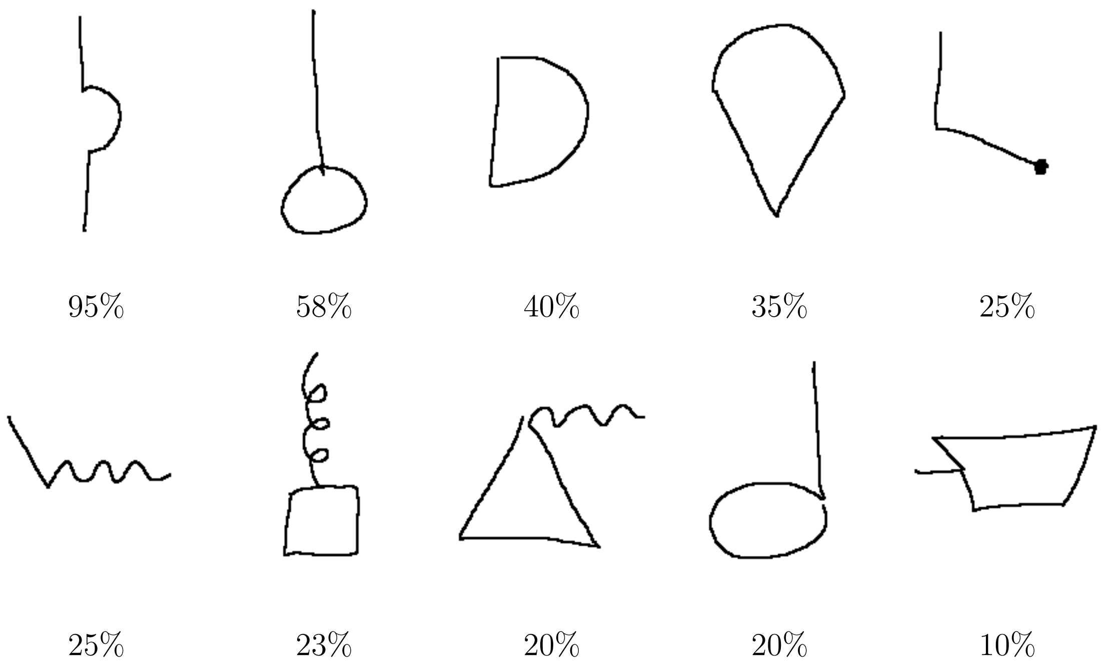


Fig. 71. Examples of the complex shapes encountered in the data set, along with the percentage of the time that each was drawn in a single-stroke rather than multiple strokes.

only contained a complex shape of one line and one arc, this data set contained many more complex combinations. Figure 71 shows the multi-primitive shapes from our data set that were drawn using a single stroke. Underneath each shape is the percentage of the time that it was drawn in a single stroke, rather than multiple strokes.

Complex shapes made up less than 5% of all of the primitive shapes in the data set. This means there were more instances of multi-stroke primitives than there were of single-stroke complex primitives. The MLP correctly classified instances of complex shapes 90.8% of the time. We returned the correct interpretation for complex shapes 78% of the time. More specifically, the correct interpretation was returned 88.8% of the time if the complex shape consisted of only two sub-shapes (73.4% of all complex shapes). However, if the interpretation contained three sub-shapes the accuracy was only 50% (26.6% of all complex shapes). We did not encounter any complex interpretations that contained more than 3 shapes.

The majority of mis-interpreted complex shapes came from the most commonly drawn complex shape: the line, arc, line shape seen in Figure 73. In this case, the stroke was undersegmented. By *undersegmented*, we mean that the interpretation returned too few shapes. For example, most line, arc, line combinations were returned as line, curve interpretations. This is because the complex interpreter accepts any sub-shape that has a confidence of at least 50% (as described in Chapter III). In this case, the curve had a high enough confidence that it did not require further segmentation. In fact, most of these curves had confidences above 98%. Undersegmented complex shapes made up 83.3% of the total error.

The remaining error came in the form of oversegmented complex interpretations, and a few cases of complex overtracing. Most oversegmentations were a result of how the user drew the shape. Either there was a large tail on the stroke, or the user purposefully added additional shapes to the stroke, as seen in Figure 74 when the user added additional lines to the tops of his helixes.

Occasionally we also encountered examples of overtracing, also seen in Figure 75. These examples came from the same user that drew multi-stroke, overtraced shapes. In this case, the additional stroke kept the shape from being properly interpreted. Because of the rarity of overtraced shapes in our data set, we leave this as a problem for future work.

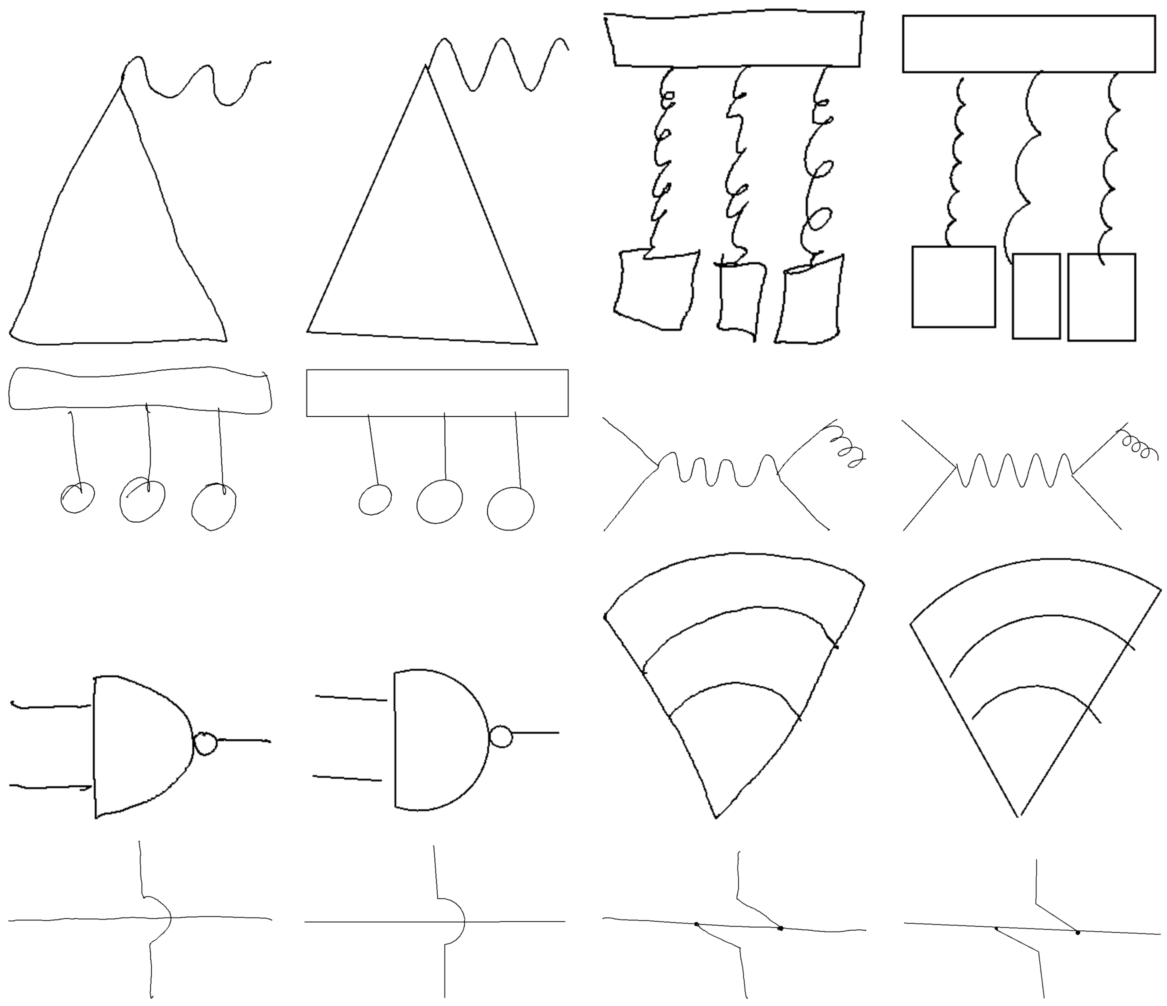


Fig. 72. Examples of correctly interpreted complex shapes (original strokes and recognized shapes).

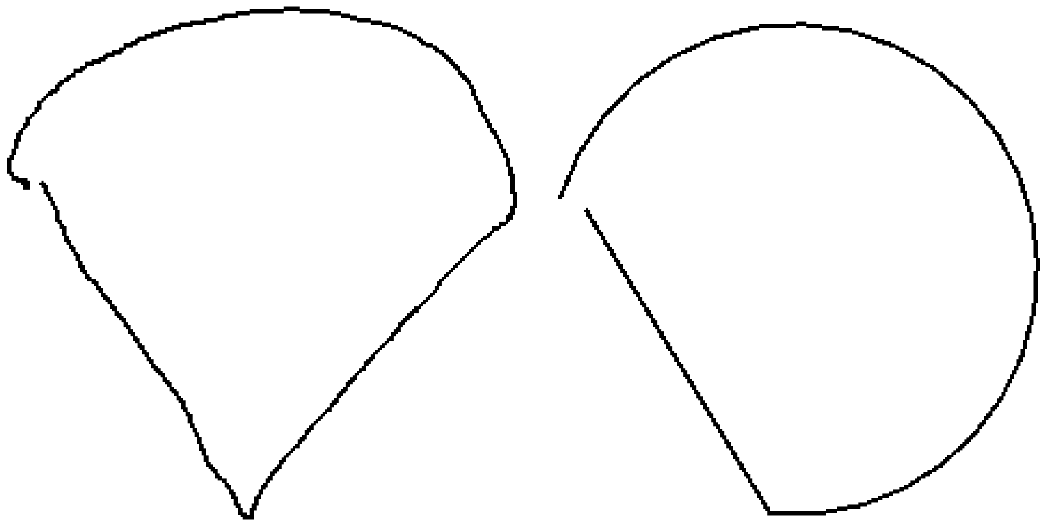


Fig. 73. Example of a complex undersegmentation.

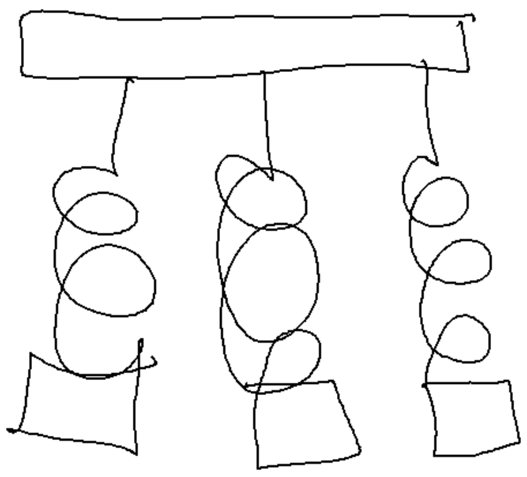


Fig. 74. Example of complex oversegmentation. Additional lines at the tops of the helixes led to oversegmentations.

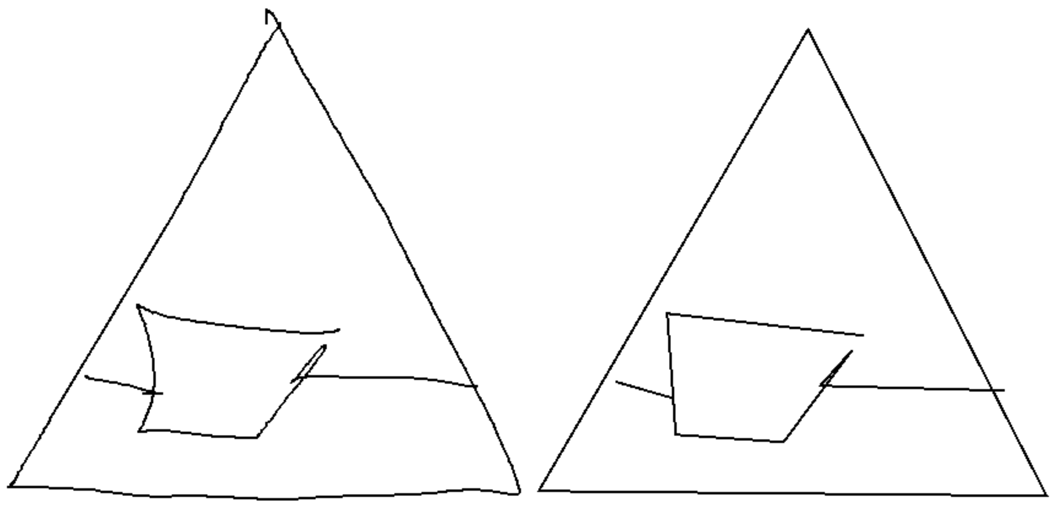


Fig. 75. Overtraced lines within complex shapes kept them from being properly interpreted (left).

E. Discussion

In this chapter, we have introduced an algorithm for recognizing multi-stroke primitives that, unlike previous approaches, places no drawing constraints on the users. Although the algorithm achieved successful results, there is still much room for improvement.

1. Comparison to Other Methods

Comparing our algorithm to other multi-stroke recognizers is difficult because of the constraints used by many of these approaches. Most have been tested solely on isolated primitive shapes and rely on special button presses [44, 90] or unspecified timeouts [14] to group the strokes that belong to a single primitive. These types of approaches completely disallow stroke interspersing. Furthermore, many of these recognizers do not support the full range of primitives that we have supported through PaleoSketch.

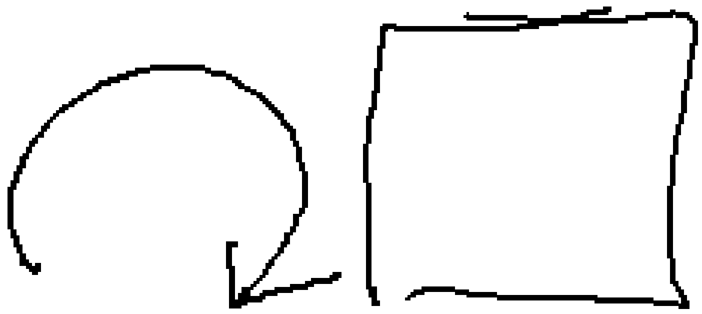


Fig. 76. Examples of curved arrows and rectangles with overlapping sides that Tahuti has trouble recognizing.

The most comparable recognizer to what we have done is the Tahuti recognizer [29]. Unlike the aforementioned recognizers, Tahuti does allow for stroke interspersing, up to a certain number of strokes. However, one notable downside of the Tahuti recognizer is that it relies on a combination of endpoint distances and slope continuation to perform stroke grouping. Our recognizer relies solely on endpoint distances. Because of the added constraint of slope continuation, the Tahuti recognizer only handles multi-stroke rectangles and linear arrows. It does not handle any form of multi-stroke, curvilinear shapes, such as multi-stroke ellipses. We tested our arrow and rectangle data using the Tahuti recognizer and found that it achieved 53.2% accuracy on arrows and only 33.6% accuracy on rectangles, even though our recognizer achieved over 96% on both of these shapes. The primary reason for this is because Tahuti does not support arrows with curved shafts or rectangles that have overlapping sides (Figure 76).

Some may also argue that multi-stroke primitives can be handled by higher-level recognition systems, such as LADDER [160]. For example, a rectangle composed of four strokes can be described as four lines that have perpendicular angles and congruent sides. A multi-stroke line is simply two lines that have near endpoints and a similar slope. Although this is true, the possible combinations of shapes that can form multi-stroke primitives would soon become unmanageable. A multi-stroke circle could be formed from two arcs, or from an arc and a line, or from two arcs and a line, or from three arcs, or from a curve and line, etc. The possibilities are endless, and each would require a specific shape definition in the high-level system. This is the primary reason why we believe multi-stroke primitive recognition should be the task of the low-level recognizer.

2. Multiple Interpretations

One of the downfalls of our approach is that many aspects of the algorithm are greedy. For instance, when determining if strokes should be combined, the algorithm looks solely at the best primitive interpretation of individual strokes rather than considering the confidences of every possible primitive. Furthermore, the algorithm is greedy when it comes to the order in which to combine strokes (recall the problems in Figure 69). One possible improvement to the algorithm would be for it to consider and rank multiple, multi-stroke interpretations. Each possible combination of strokes could be considered, and confidences could be used to determine the best interpretation possible.

This would be a difficult task, however, because of speed requirements. With our multi-stroke algorithm, it took, on average, 372 milliseconds to classify each primitive. This is almost two and half times slower than our single-stroke recognizer. Adding the additional complexity of searching through all possible multi-stroke interpretations for a given set of strokes would further slow down recognition.

3. Improving Complex Interpretations

We have seen that complex interpretation accuracy drops with a larger number of complex combinations. There is a delicate balance between creating complex interpretations that are not undersegmented or oversegmented. We attempted, to some degree, to experiment with the threshold for determining if a sub-interpretation should continue to be segmented further. Anything between 50% and 90% seemed to make little difference. When the threshold was raised to above 90%, many existing, correct complex interpretations started to become oversegmented. One way to possibly improve complex interpretations would be to train on the sub-shapes within complex interpretations. This would allow us to raise the our threshold without producing oversegmentations, because sub-shapes would now become more confident. Multiple complex interpretations would also be beneficial, but we would also need to consider the performance decrease with this added complexity.

4. Improving Grouping

Finally, another area of our algorithm that could be improved is grouping. Currently, we use a simple spatial distance between endpoints to determine if strokes should be grouped. Although this assumption works for over 93% of the cases[2], it still is not valid for some examples (e.g., overtraced shapes). Using a combination of temporal, plus spatial, information for clustering may improve the overall accuracy of our approach. One must be careful, though, not to rely solely on temporal information, as this may constrain the manner in which a user must draw. For overtraced shapes, comparing the overlap of bounding boxes may also be a possible indicator for combination.

[2]This was determined by testing multi-stroke accuracy when we assumed perfect single-stroke accuracy.

5. Additional Improvements

One thing to remember about our experiments is that we tested on a full set of primitives that represented many different domains. In a real world application, it is unneccesary to have every primitive turned on for every domain. Specific primitive shapes should be capable of being turned on and off. In most cases, we would expect that real world accuracies could actually be better than our tested accuracies because fewer primitives would need to be turned on for each domain.

Another thing to keep in mind it that we performed our experiments using cross-fold validation, based on user. This means that each testing user provided no training data to the neural network. As we saw in some examples, like the user who drew dots with small marks, user-specific styles still exist and may cause problems with recognition. One benefit of our MLP classifier, however, is that it is adaptable and could be made to learn over time by modifying its weights based on the correct/incorrect classification of user-specific examples. A similar approach was used in our MARQS system and proved to be beneficial over time [16].

F. Chapter Summary

In this chapter, we introduced a recognition algorithm that achieves close to 90% weighted accuracy on multi-stroke primitives. The algorithm is capable of recognizing these primitives without requiring special drawing constraints, such as timeouts, button presses, or prohibiting interspersing. Although the approach produces acceptable results, much improvement can still be made. Areas of possible improvement include returning multiple interpretations, improving complex fits, and discovering better methods of stroke clustering.

CHAPTER VI

SOUSA

A. Introduction

Although existing domain-specific datasets are readily available, most sketch recognition researchers are forced to collect new data for their particular domain. Creating tools to collect and label sketched data can take time, and, if every researcher creates their own toolset, much time is wasted that could be better suited toward advanced research. Therefore, we have designed and built a general-purpose sketch collection and verification tool that allows researchers to design custom user studies through an online applet residing on our group's web page. By hosting such a tool through our site, we hope to provide researchers with a quick and easy way of collecting data while serving a secondary purpose of creating a universal repository of sketch data that can be made readily available to other sketch recognition researchers. The tool is called SOUSA (sketch-based online user study applets), and has gone through many revisions to date. It can be accessed through our group's webpage at `http://srlweb.cs.tamu.edu/srlng/sousa/`. To date, SOUSA hosts over 150 data collection studies, representing over 18,000 sketched data files.

B. Previous Efforts

Researchers in the sketch and visual fields of computer science have been working toward large, standardized datasets that can be used in many domains. A standardized dataset has benefits including the lack of data collection for researchers and uniform comparison results between systems.

One large corpus of sketch recognition data is the ETCHA Sketches data referred to in [186]. This dataset contains hand-sketched drawings of family trees, circuit diagrams, floor plans, and geometric objects. Two image processing datasets include Caltech's 256 Google and Picsearch collection [187] and the CBCL StreetScenes image database [188]. The Caltech dataset contains over 30,000 object images in 256 categories, each category ranging in size. The CBCL StreetScenes database has labeled components of images taken on streets, such as cars, people, and roads.

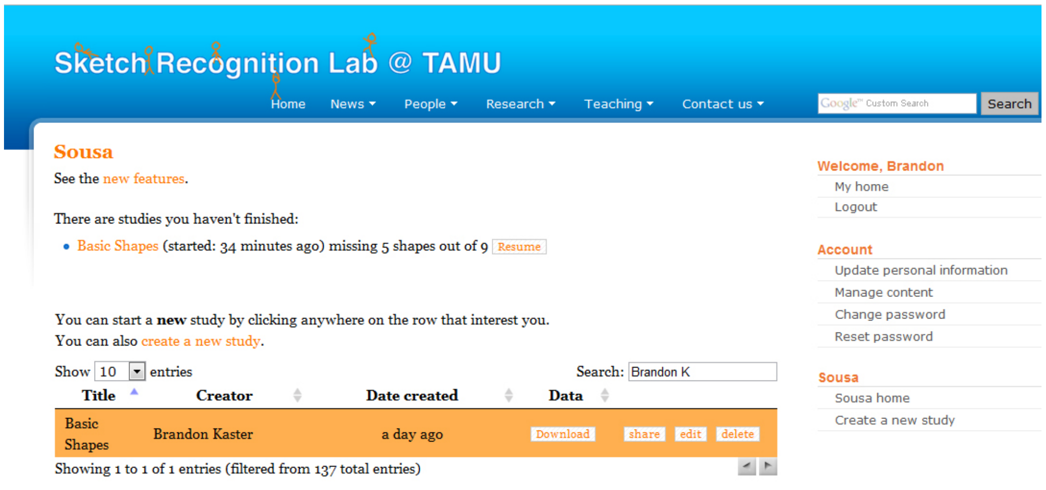
Unfortunately, an abundance of general data is not always applicable to specialized domains. Creating a sketch recognition system to recognize sketched Kanji symbols, for example, forces the developer to perform their own data collection and labeling as no standardized Kanji dataset exists. The two main ways to collect this necessary data are by gathering and labeling real-world data or by having users interact with a specialized program that handles the data collection and labeling process.

Labeling real-world data has been an issue in visual processing. For static images, labeling programs have attempted to make the labeling process fun and entertaining in order to have the public label large datasets for the researchers [189, 190, 191, 192]. Sketch recognition labeling programs have not yet embraced these techniques, but the labeling program in [193] attempts to make labeling quick and efficient.

C. The SOUSA System

SOUSA was originally implemented as a set of Java applets and provided no security for ensuring that the original creators of studies were the only ones allowed to perform parameter modification [194]. Furthermore, the original system provided no mechanism to allow researchers to search for the data of others.

The newest version of SOUSA utilizes a MySQL database and Python server with Javascript client software to resolve many of the issues found in the initial version



Sousa

See the new features.

There are studies you haven't finished:

- Basic Shapes (started: 34 minutes ago) missing 5 shapes out of 9

You can start a **new** study by clicking anywhere on the row that interest you.
You can also create a new study.

Show 10 entries Search: Brandon K

Title	Creator	Date created	Data
Basic Shapes	Brandon Kaster	a day ago	Download share edit delete

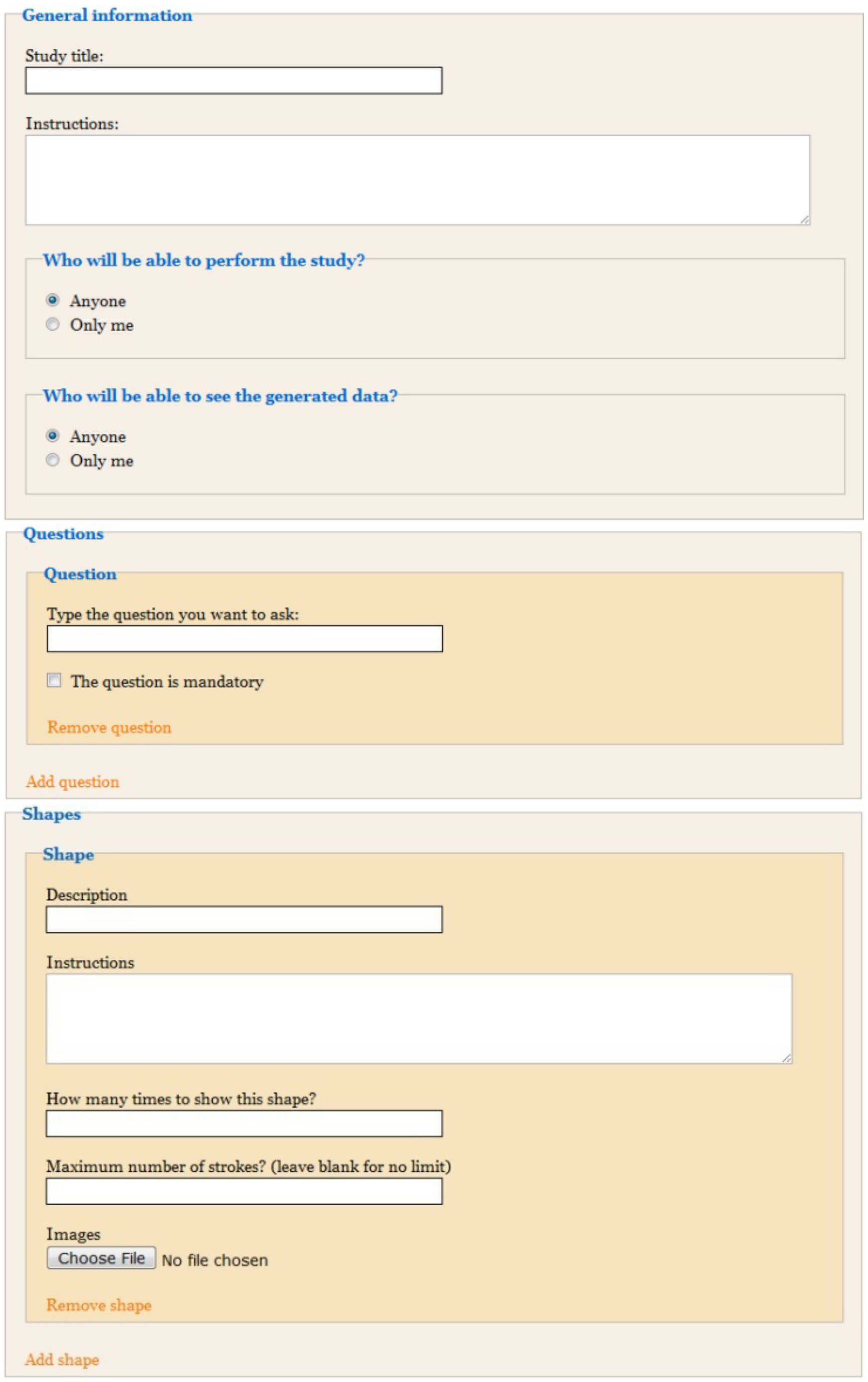
Showing 1 to 1 of 1 entries (filtered from 137 total entries)

Fig. 77. Screenshot of the home page for the current version of SOUSA.

[195]. Javascript was chosen over the existing Java applets because it allowed for a faster sampling rate (from 80 points/second to 120 points/second). Data is collected and saved in an easy-to-parse XML format.

Once a user signs up on our server, they are presented with a home screen (Figure 77) where they can browse and search for existing sketch data, create and manage their own data collection study, and perform collection studies for other users.

SOUSA allows users to create and perform two different forms of user studies. Collections studies allow researchers to ask users to draw specific shapes or symbols from a domain. These inquiries can be presented with or without accompanying images. Verification studies can be created after data collection and allows researchers to get classification opinions of numerous “human recognizers.” This can be helpful to determine ambiguous examples, and also allows alternative interpretations to be attached to sketched samples.


General information

Study title:

Instructions:

Who will be able to perform the study?

[✓ Anyone] Only me

Who will be able to see the generated data?

[✓ Anyone] Only me

Questions

Question

Type the question you want to ask:

☐ The question is mandatory

Remove question

Add question

Shapes

Shape

Description

Instructions

How many times to show this shape?

Maximum number of strokes? (leave blank for no limit)

Images

Choose File No file chosen

Remove shape

Add shape

Fig. 78. Screenshot of the SOUSA form for creating a collection study.

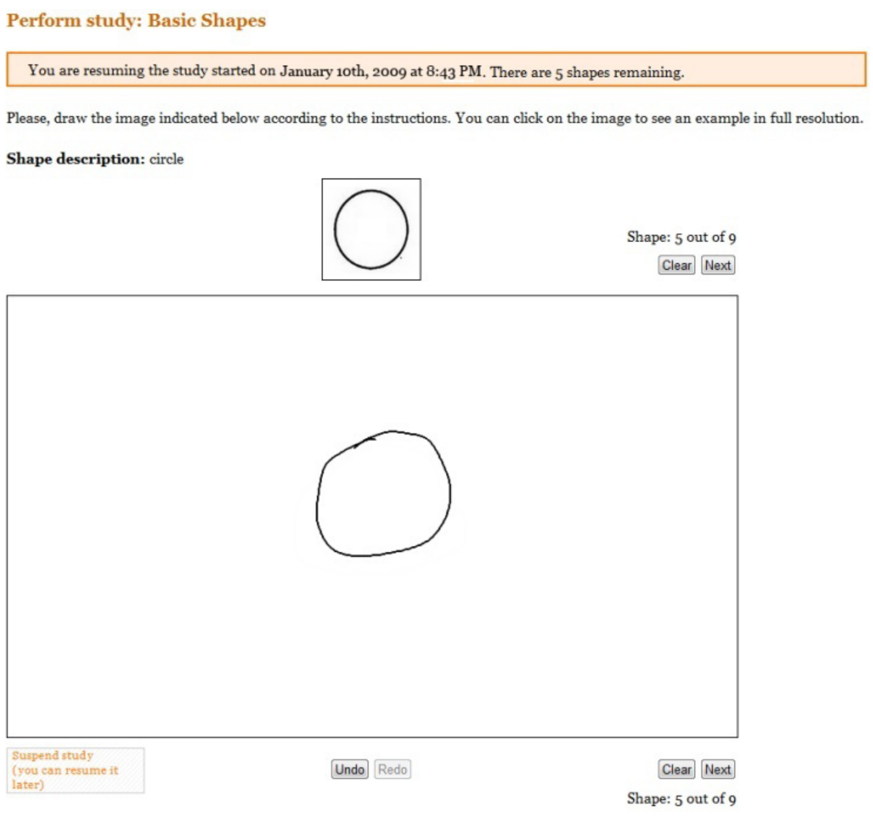
1. Collection Studies

When creating a collection study, users are presented with the form shown in Figure 78. Researchers can control aspects of the study including who can perform the study and who can download the data from the study. They can also create a questionnaire in order to gain information about the users who perform the studies (e.g., age, sex, occupation, input device). When specifying the shapes and symbols to collect, users simply provide a description, number of times each symbol is to be drawn, and can even specify the maximum amount of strokes that are allowable. This is mainly beneficial to researchers that are interested in gesture recognition, or single-stroke primitive recognition. In addition, researchers also have the option of uploading an image that can be shown in the event that the user is not sure how to draw a particular symbol. This allows researchers to collect data from users, even if they are not experts in the domain.

Once the form has been filled out by the researcher, a unique hyperlink is generated that the researcher can use to point to the generated study. When users visit the link, they can perform the study, as seen in Figure 79. Users can also perform the study by browsing a list of publicly available studies on our website, as long as the researcher did not designate the study to be private. When performing a study, users can see their progress and have the capabilities to navigate back and forth between sketches. They are also allowed to clear the screen or undo individual strokes when drawing. They even have the ability to suspend and resume the study at a later time.

2. Verification Studies

Verification studies allow researchers to get the opinion of “human recognizers” on the sketched data they have collected in previous studies. This not only allows researchers

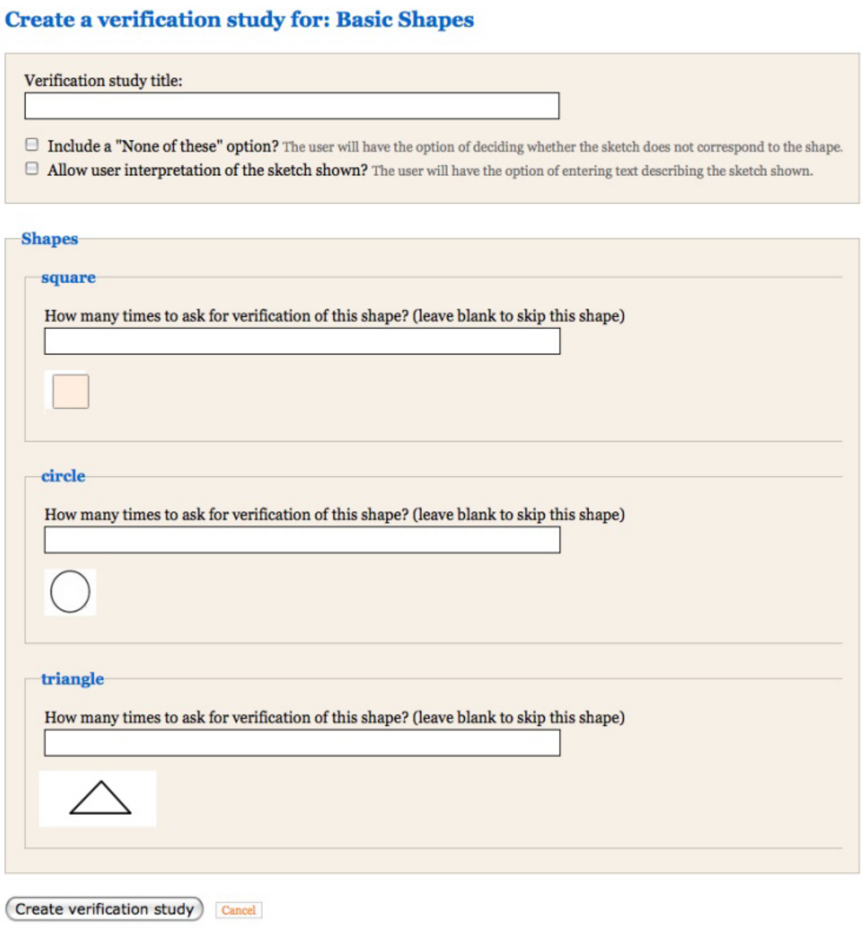

Perform study: Basic Shapes

You are resuming the study started on January 10th, 2009 at 8:43 PM. There are 5 shapes remaining.

Please, draw the image indicated below according to the instructions. You can click on the image to see an example in full resolution.

Shape description: circle

Fig. 79. Screenshot of the SOUSA data collection applet.



Create a verification study for: Basic Shapes

Verification study title:

☐ **Include a "None of these" option?** The user will have the option of deciding whether the sketch does not correspond to the shape.

☐ **Allow user interpretation of the sketch shown?** The user will have the option of entering text describing the sketch shown.

Shapes

square

How many times to ask for verification of this shape? (leave blank to skip this shape)

circle

How many times to ask for verification of this shape? (leave blank to skip this shape)

triangle

How many times to ask for verification of this shape? (leave blank to skip this shape)

Fig. 80. Screenshot of the form used to create a verification study.

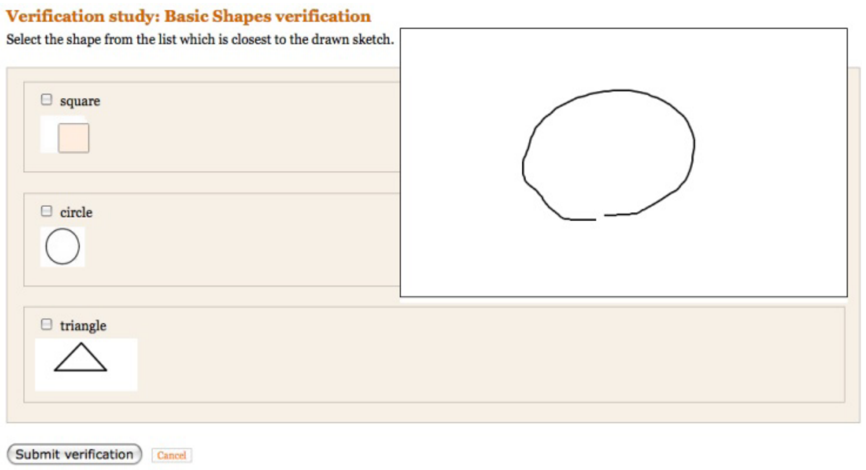

Verification study: Basic Shapes verification
Select the shape from the list which is closest to the drawn sketch.
square
circle
triangle
Submit verification
Cancel

Fig. 81. Screenshot of a verification study.

to quickly find incorrectly drawn samples, but also allows them to identify ambiguous cases, which could be helpful during the training phase of recognition. When creating a verification study, researchers are shown the form seen in Figure 80.

When performing a verification study, users have the option of attaching multiple labels to drawn shape, as seen in Figure 81. If the creator of the study chooses, a “none of these” option may also be present.

D. Evaluation

The primary beneficiaries of our system are sketch recognition researchers. To evaluate our system, 10 students from a sketch recognition class, as well as 11 students from a computer-human interaction class, used our the initial version of the SOUSA system to collect data and evaluate perceptual thresholds with verification studies [194]. Data was collected for various domains including Kanji, a physics simulator, and memory games, just to name a few. Verification studies were also created by some students to help determine perceptual thresholds for sketched strokes, such as whether two lines are parallel, touching, intersecting, near, far, etc. Our system has also been used by various researchers from our group to collect data for a number of projects.

We asked users to give us comments and feedback afterward to help evaluate the usefulness of our system. Overall, students and researchers felt the system was very helpful and quick and easy to use. According to one user, "after you use it once and get it working, it is very easy and quick to create new user studies to get data for different sketching domains." The most time consuming part of creating studies for most users was creating the images to be shown for each shape.

One suggestion for the system was the ability to ask questions of the user after a study in addition to before the study in a pre-questionnaire/post-questionnaire form. Users also noted occasional lags and delays during peak times when performing studies. Most of these lag delays were due to other resource-intensive processes being run on the web server, unbeknownst to us at the time. As of today, most of the lag problems have been remedied. Many other issues, such as those related to the security of the system, have been resolved with the newer versions of SOUSA [195].

E. Future Work

SOUSA is an ongoing project being maintained by members of the Sketch Recognition Lab. While the existing system is stable, feedback from users has identified a number of issues that we hope to resolve in future versions, including:

- The ability to "close" a study, and disallow further collection, without having to delete it completely.
- The ability to maintain dataset revisions. For example, if a paper is published with a set of data, but additional data is added after publication, we want to

allow users to download specific revisions so they can compare algorithms on the exact same samples.

- The ability to allow data collection to take place offline, while not connected to the Internet.
- The ability to upload previously collected data, or data collected through some other tool.
- The ability of performing verification on data that was not collected through SOUSA.
- The ability to perform studies on a mobile platform.
- The ability for researchers to include SOUSA in their existing software. We are already working on an API that will hopefully allow this.

F. Chapter Summary

SOUSA is a sketch-based, online user study application developed to aid in the creation of a universal, standardized set of sketch data. The goal of SOUSA is to make sketch-based data collection more efficient and practical for researchers, allowing them to focus on higher-level tasks. The primary benefits of SOUSA include:

- The ability to create and manage collection studies.
- The ability to create and manage verification studies.
- The ability to dynamically upload images that are associated with symbols to be collected.
- Secure user accounts.

- Public and private access options for studies and data.
- The ability to browse and search for specific sketch data.
- The ability to download previously collected data.
- The ability to suspend and resume studies.
- The ability to clear and undo strokes.
- The ability to ask questions of users before they perform a study.

CHAPTER VII

OFFICE ACTIVITY RECOGNITION

A. Preface

This chapter discusses a project that is independent of sketch recognition, but shows that many of the issues related to *free-sketch recognition* [45] apply to other domains as well. One of the primary goals of our low-level sketch methods in previous chapters is to allow users to interact freely with our system. Users should not be forced to learn how to use a system before interacting with it; the system should be natural and intuitive from the beginning. This means that the system needs to be smart enough to handle the variations that occur across multiple users.

In this chapter, we introduce a project that centers around activity recognition in an office setting. The goal is to understand if we can determine the objects that a user interacts with, simply by monitoring his or her hand posture. One of the key findings of this work directly relates to the concepts learned through our work in sketch recognition; if we allow users to interact with objects naturally, some users will do things differently from other users. In this specific work, we have found that individual participants will use similar hand postures each time he interacts with an object. However, those hand postures may be significantly different than that of a previous user.

B. Introduction

As the future of computing heads toward ubiquity, wearable computing, coupled with context-aware applications, will become more prevalent. Context-aware applications are those which adapt one's computing environment based on "where you are, who

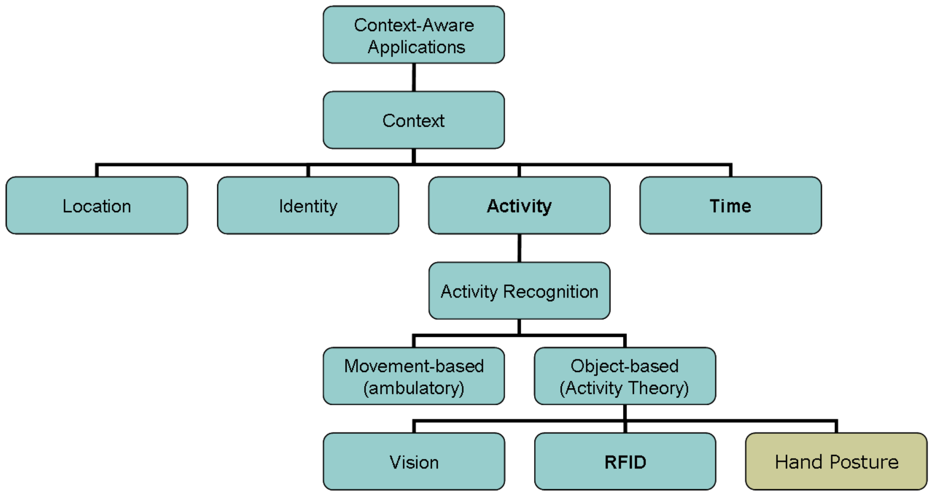

Context-Aware Applications
Context
Location
Identity
Activity
Time
Activity Recognition
Movement-based (ambulatory)
Object-based (Activity Theory)
Vision
RFID
Hand Posture

Fig. 82. Framework indicating where hand posture fits into the overall scheme of context-aware computing.

you are with, and what resources are nearby" [196]. According to Dey and Abowd, the most important types of context are location, identity, activity, and time [197]. Of these forms of context, activity is one of the hardest to capture and is used less frequently by many context-aware applications [198]. However, we believe activity-based context can play a significant role in applications, particularly those involving wearable and pervasive computers.

As a simple example, imagine a scenario in which a user is at her place of employment and engaged in conversation over her office telephone. Someone else, meanwhile, attempts to call the user on her mobile phone. If the mobile phone has access to the user's activity context (e.g., the user is currently on her office telephone), then the mobile phone could respond by informing the caller that the user is currently engaged in another conversation on her office phone. Another example could be a coffee machine that can determine when to start a new brew based on the number of times a user has picked up his mug to take a drink. These scenarios are just a few possibilities that show the potential benefit of knowing activity-related context.

The goal of activity recognition is to aid context-aware applications by providing information to help explain what a user is currently doing (i.e., what activity the user is engaged in). An issue activity recognition researchers face, however, is how to define what an activity is and how to determine when it is taking place. One answer may lie in *Activity Theory* [199, 200]. According to this theory, activities have objectives and are accomplished via tools and objects. Therefore, one can assume that if we can determine the object that a user is interacting with, then we may be able to imply something regarding the activity that the user is currently engaged in. Some frameworks have been created to model activities in this manner, but were implemented in virtual environments in which interaction with objects is assumed to be given by some form of sensor values [201, 202]. When applying such frameworks to a real-world domain, we still face the issue of determining when an appropriate interaction is taking place. In order to achieve full contextual-awareness, one must address the category of contextual sensing as it is the lowest, most basic part of context-aware applications [203].

It can be argued a person's hands are his primary means of interacting with tangible objects. They also can serve as a secondary source of communication through gesturing. Because of this, we focus our attention solely on haptic input experienced via the hands. Figure 82 gives an idea of where hand posture can be beneficial in the overall scheme of context-aware computing. By "hand posture" we refer to the static positioning and orientation of the fingers and palm. This differs from dynamic "hand gestures" which refer not only to posture of the hand, but also the orientation of the hand in three-dimensional Euclidean space.

The two most common approaches to tracking haptic activity and interaction in a real world setting are through cameras or wearable sensors. Vision-based techniques require cameras that are either placed within a room [204], or that are wearable [205].

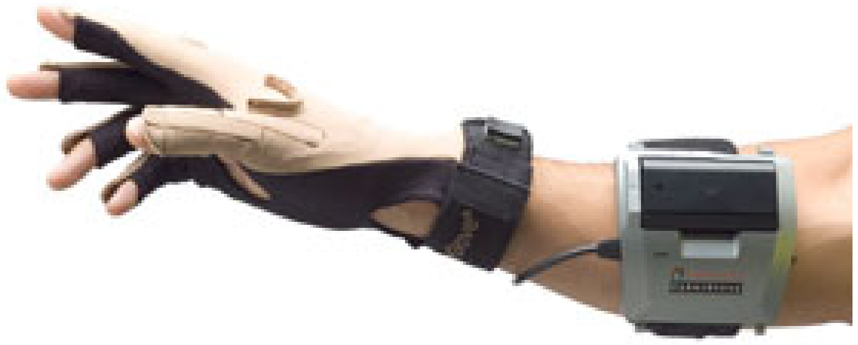


Fig. 83. The 22-sensor CyberGlove II by Immersion Corporation [22].

Stationary cameras placed in a room have the advantage of being less obtrusive to the user, but also makes the context-capturing system static to that one location. Wearable cameras allow for context-capturing systems to become mobile, but still have the problem which most vision-based approaches experience when dealing with the interaction of objects: occlusions (which typically occur because of the object itself).

Because our interests lay more with interaction and less with writing vision-based algorithms to handle occlusions, we decided to use glove-based sensor input provided by Immersion's 22-sensor CyberGlove II (Figure 83). The primary goal of our work is to determine if hand posture can be used as a cue to help determine the objects a user interacts with, thus providing some form of activity-related context. To give some real-world practicality to our problem, we chose to perform our experiments in an office domain, a setting we believe could benefit from context-aware applications.

A secondary goal of our work is to determine the variability of hand postures between different users who interact with the same objects or are asked to perform the same gestures. In other words, when users are allowed to interact with objects or perform gestures as they would naturally, will they use similar or dissimilar hand postures? This motivation of creating systems which allow natural interaction rather than forcing the user to learn specified behaviors is shared with our previous work in sketch recognition [45].

C. Related Work in Activity Recognition

The term *activity recognition* does not solely include the recognition of activity through objects, as defined by Activity Theory, but also includes the recognition of activities a user performs with his own body. These *ambulatory* activities typically include standing, walking, running, sitting, ascending/descending stairs, and other common body-related movements. Previous works have attempted to recognize movement-related activities using vision-based approaches [206] or wearable accelerometers [207, 208, 209, 210, 211]. Some approaches include other inputs like ambient light, barometric pressure, humidity, and temperature [212]. Minnen et al. used accelerometers and audio in order to capture "interesting" behavior in a journal, which could then be used to help treat people with behavioral syndromes like autism [213]. Ward et al. also used a combination of accelerometers and sound in order to determine activity in a workshop setting [214]. Our work shares a similar view of activity recognition through tools and objects, but uses hand posture, rather than sound and motion, in order to determine interaction.

Other works have also shared a common motivation of activity recognition through objects; however, their approaches have been different from ours. In particular, objects are tagged with radio-frequency identification (RFID) sensors [215, 216]. The user then wears a glove with a built-in RFID tag reader which enables the system to determine the objects being interacted with. While this approach is not prone to much error, it constrains the user to only interacting with the tagged objects in his environment.

Some works have shared a similar domain as ours (i.e., an office setting). Ayers and Shah presented a vision-based system that required prior knowledge about the layout of the room [217]. This system was mainly used for security purposes, in order to determine if unauthorized peoples were performing unauthorized activities. Oliver et al. proposed a system that included audio, video, and computer interaction input [218]. However, their work focused primarily on providing an environmental context (e.g., determining if a conversation is taking place or a presentation is being given), rather than determining the objects a user is interacting with. Finally, the PARCTab system introduced palm-sized computers that could be used to create a ubiquitous computing environment in an office setting [219].

The idea of using hand posture to recognize types of grasps has been proposed in previous works using both vision-based optical markers [220] and glove-based input [221]. While these works are similar to what we have done, we should make the distinction that our goal is to recognize objects rather than grasp types. Many times objects are interacted with using similar grasp types. Our goal is to determine if hand posture can yield a fine enough resolution to determine the object a user is currently interacting with, even if that object is grasped with the same grasp type as another object in the domain.

D. Experiment

For our experiments, we utilized a single, right-handed, wireless, 22-sensor Cyber-Glove II device developed by Immersion Corporation [22]. The glove provides three flexion sensors per finger, four abduction (finger spread) sensors, a palm-arch sensor, and sensors that measure wrist flexion and abduction. Although the glove allows for faster sampling rates (up to 90 readings per second), we sampled the glove at a lower rate (10 readings per second) in order to reduce the amount of data we collected.

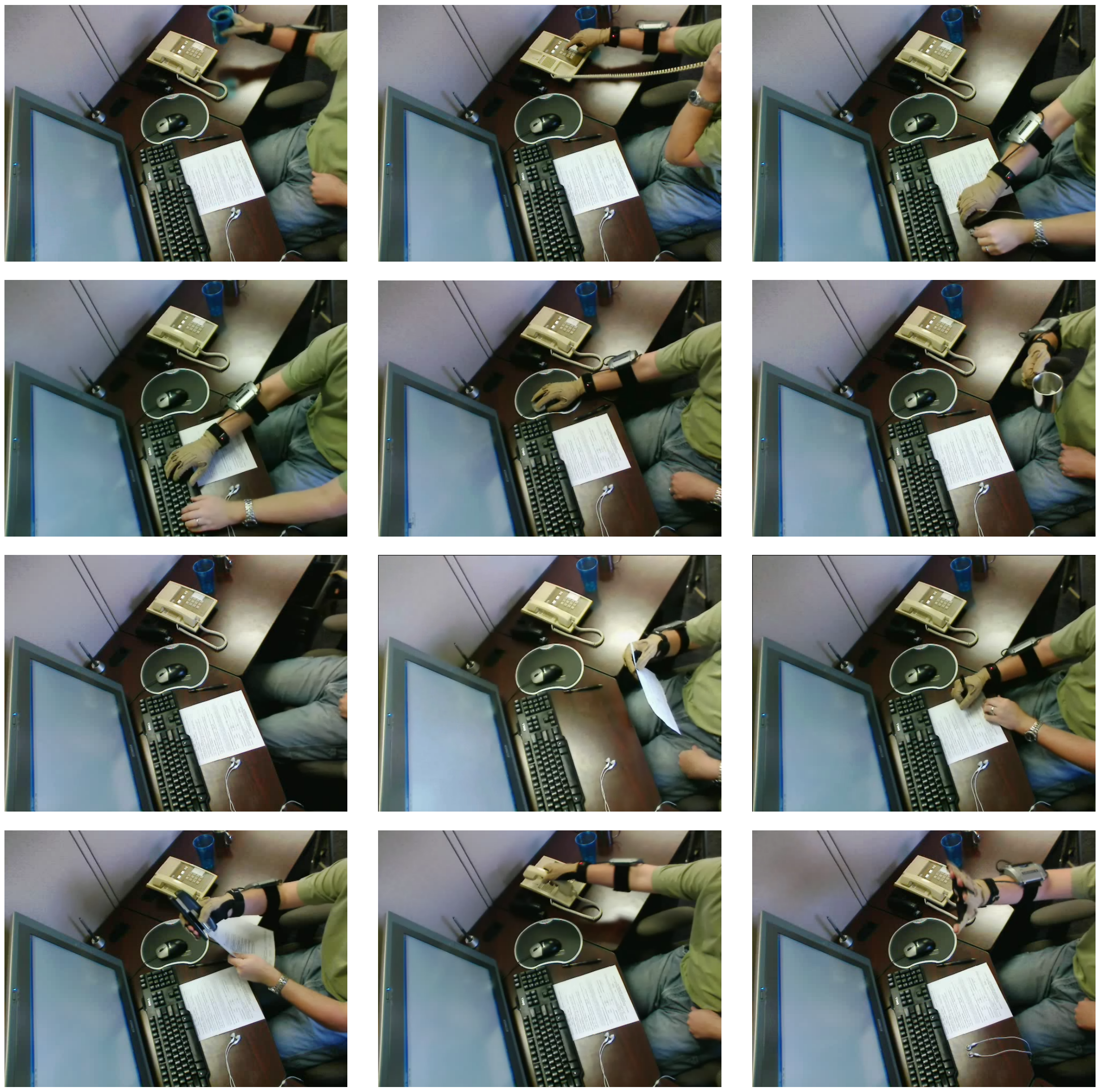
A total of 8 users (7 male, 1 female) participated in our data collection study. All of the users were graduate students, and about half had previous experience interacting with the CyberGlove, mainly for sign language recognition tasks. We asked users to interact with objects that may be typically found around the desk in an office. Table VI lists the 12 types of interactions we collected. Each user performed each interaction 5 different times, during which the user's hand posture was capture through the CyberGlove. For each interaction, we averaged the values of each sensor to create an input vector for our classifiers.

1. Classifiers

For our tests, we experimented with a number of different classifiers. Because we are focused primarily on determining how much information hand postures provide, as well as determining whether or not this information is common across all users, we decided to test simple classifiers that could yield decent baseline results. In particular, we experimented with a basic weighted linear classifier [43] and a k-nearest neighbor (KNN) classifier with varying neighborhood sizes [222]. We used a basic Euclidean distance metric for our KNN classifiers.

We also attempted to use a quadratic classifier, but due to the high level of noise in the glove data and the high correlation between individual sensors, we experienced many singularities in the covariance matrix. Even after performing ridge regression [222, 223], the covariance matrices of the individual interaction classes were still badly scaled. Because of these numerical instabilities, we achieved poor results using the quadratic classifier and thus omitted the results from this chapter.

Table VI. The types of interactions we collected for our experiment. From left to right, top to bottom: drinking from a cup, dialing a telephone, picking up earphones, typing on a keyboard, using a mouse, drinking from a mug, opening a drawer, reading a piece of paper, writing with a pen, stapling papers, answering the telephone, and waving.



2. Feature Spaces

In addition to experimenting with different classifiers, we also tried a number of different feature spaces. We chose to look into other feature spaces because of the high degree of noise and correlation we expected between the 22 sensors of the CyberGlove. The first space we tested was the raw sensor values as a 22-dimensional feature vector.

The next two spaces are projections based on Principal Components Analysis (PCA) [222, 224, 225, 226] and Linear (Fisher's) Discriminant Analysis (LDA) [222, 227]. Both of these two techniques have typically been used to perform dimensionality reduction. PCA attempts to project the data in the direction of maximum variance, while LDA attempts to project data in the direction that maximizes separability between classes. Although some principal components are typically removed during PCA for dimensionality reduction purposes, we maintained all 22 components when performing our tests in order to ensure no information was lost due to the projection. Likewise, with LDA, we only removed eigenvectors which had imaginary eigenvalues.

E. Results

We performed a set of tests to determine how well hand postures could be used for recognition on both a user-independent and user-dependent system. User-independent systems are those which are trained using the data from a variety of different users, in the hopes that enough training will yield a system that will generalize well to new users. These systems do not require training data from new users, and work "right out of the box." User-dependent systems are trained using only the data of the end-user. Because these systems are trained specifically for one user, they tend to be more accurate but may not work well for secondary users. For each type of system, we experimented with different combinations of classifiers and feature spaces.

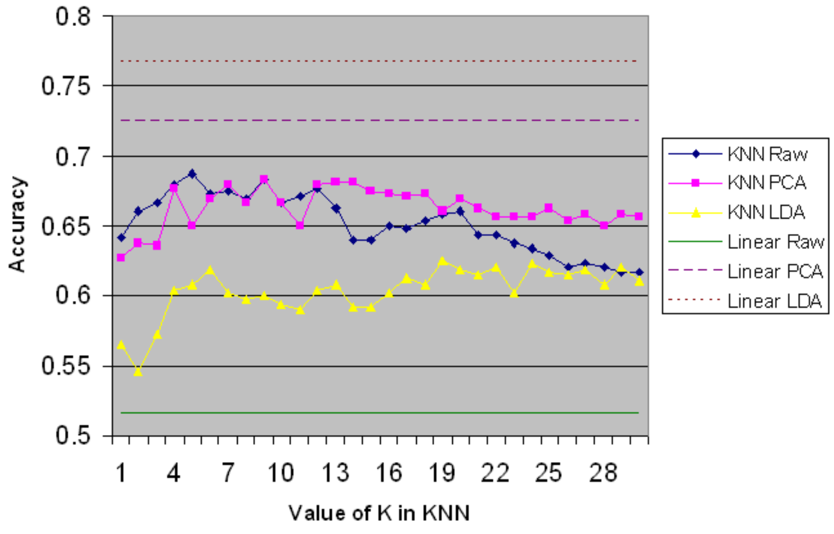


Fig. 84. Average accuracy results of a user-independent system with varying neighborhood sizes (k) in the KNN classifier. Results for the linear classifiers are shown as lines since they are constant and do not require a varying neighborhood.

1. User-Independent System

To determine the accuracy of a user-independent system, we performed leave-one-out cross-validation across all 8 of our users. With this form of validation, the data from 7 users are used for training while the data from the remaining user is used for testing; this is done for each individual user and then averaged. This form of testing mimics a system which is trained offline and then used by a brand new user. Figure 84 shows the results for the user-independent system.

The user-independent system achieved its highest average accuracy, 76.7%, using the linear classifier and LDA-projected feature space. The minimum accuracy for one user was 66.7%, while the maximum accuracy for another user was 91.7% using this classifier/feature space combination. The system's lowest average accuracy, 51.7%,

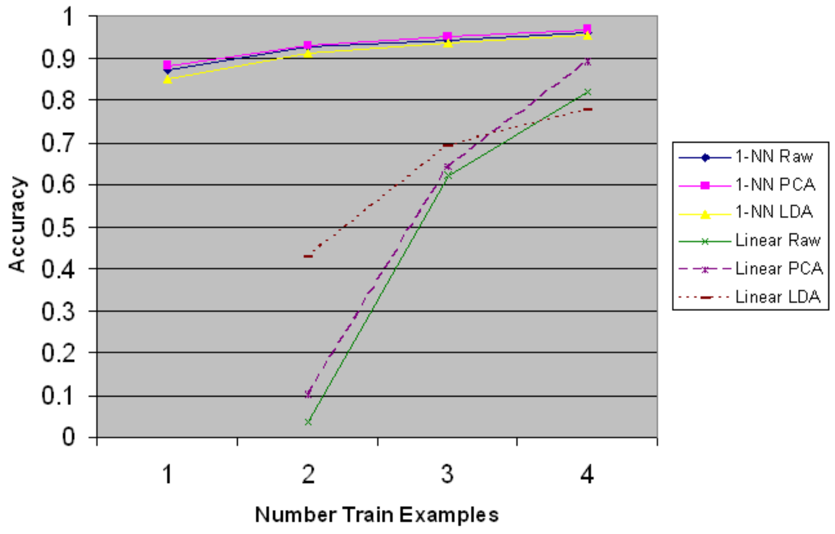


Fig. 85. Accuracy results of a user-dependent system with varying number of training samples per interaction type.

came from the same linear classifier using the raw feature space. However, the fact that we achieved over 90% for a single user using a user-independent system gives some optimism that with enough training data (more than what we tested with), a user-independent system could potentially be achievable.

2. User-Dependent System

To simulate the effect of a user-dependent system, we trained the classifier using only data from a given test user. We tried different combinations of training and testing with an increasing amount of training examples, as explained below. Figure 85 shows the average results of a user-dependent system (across all users) with various numbers of training examples per interaction class.

In the first user-dependent test, a single, random example of each interaction was used for training while the remaining examples were used for testing. In the next experiment, two samples of each interaction were randomly selected to use for training and three remaining samples were used to test. The results of each experiment were averaged across 100 folds. In this test, we fixed k=1 for the KNN classifier since some tests would inherently fail due to the lack of sufficient training examples (i.e., our first test only requires the user to give a single training sample, in which case a neighborhood size of two would not make sense). Also, results were not computable for a linear classifier given a single example per class, because the classifier requires at minimum two training samples in order to calculate covariance between features.

Obviously, as more examples are used for training, the accuracy of the system improves for all classifier/feature space combinations. We see in Figure 85 that for this type of system, the 1-NN classifier outperformed the linear classifier. However, we also notice that the linear classifier starts out poorly (because of the lack of sufficient training data), but rapidly increases in accuracy as more training samples become available. Thus, it is possible that as more training samples are provided, the accuracy of the linear classifier could surpass that of the KNN classifier. If this is the case, then an interactive learning scheme used to combine the two types of classifiers could be used to improve accuracy over time [16].

Overall, the highest accuracy for the user-dependent system came from the 1-NN classifier using PCA feature space. We can see that given a single training sample, the accuracy of the user dependent system using this combination (88.2%) is already higher than the average accuracy of the best user-independent system. As the number of training examples increases to four, the same classifier/feature space combination reached a maximum average accuracy of 96.9%. The fact that the PCA feature space ultimately outperformed the other feature spaces for both the 1-NN and linear classifier indicates that there is a lot of information in the variance of the data for individual users.

F. Discussion

Overall, it can be concluded that when users are allowed to interact with objects in a natural, unconstrained manner, a user-dependent system will likely be more suitable for recognition rather than a previously trained user-independent system, unless sufficient training data is available. The reason for such a poorly performing user-independent system was due to a high degree of variation in the way users interacted with the same objects. This was also the likely reason that the linear classifier performed so poorly on the raw feature space of the user-independent system (because interaction classes were not linearly separable). For example, when asked to staple papers, some users would pick the stapler up to use it (thus interacting with it using a circular, or "C"-style, grip) while other users would staple papers by leaving the stapler on the surface and pressing down on it using an open palm. Another example was the act of dialing, which some users performed by using a pointing posture while others used an open palm (like typing on the number pad of a keyboard). Figure 86 shows some examples of these interaction variations.

Figures 87 and 88 show the confusion matrices of a 1-NN classifier using the raw feature space for both types of systems (user-independent and user-dependent). While these are not necessarily the optimal classifier/feature space combinations for these types of systems, this classifier/feature space combination shows us the simplest template matching approach to give some insight into the overlap and confusion between interaction types.

A confusion (or matching) matrix is used in machine learning to show predicted versus actual classifications [228]. The ideal confusion matrix would be entirely white with the diagonal being completely black. This indicates perfect classification (i.e., actions are only classified as themselves). Dark areas that occur off of the diagonal

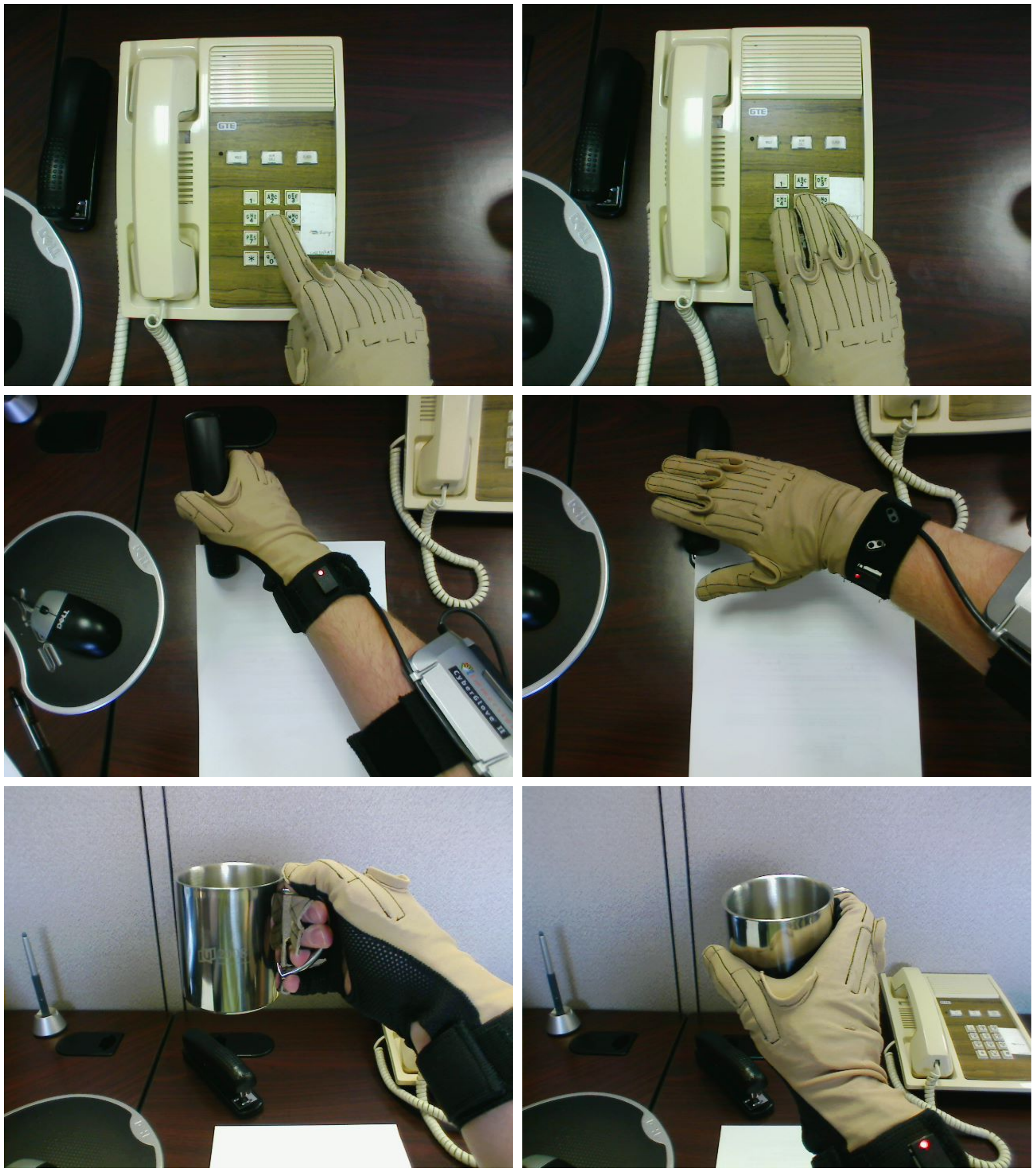


Fig. 86. Examples of variations in interaction with the same objects.

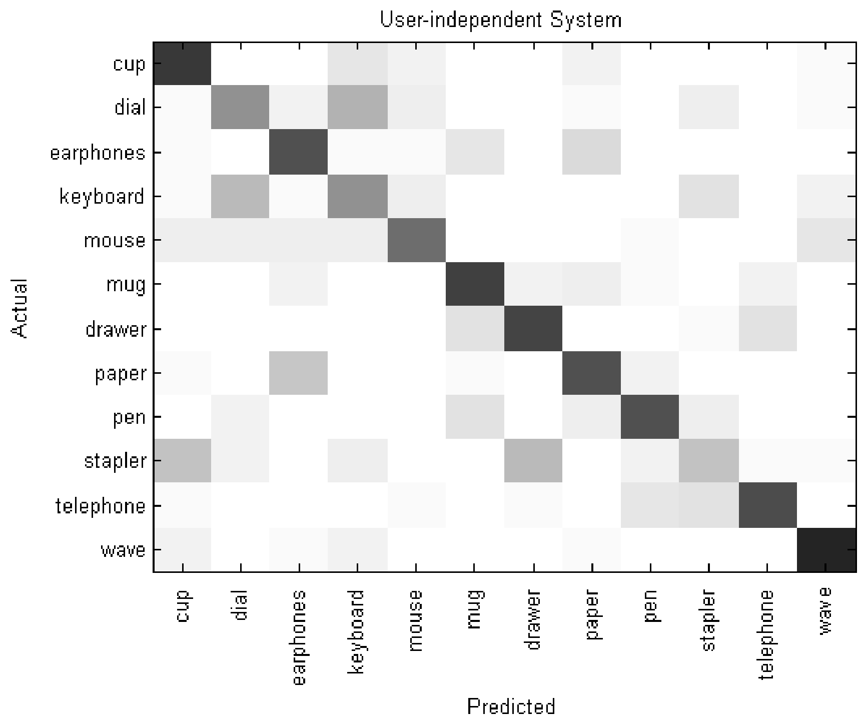


Fig. 87. Confusion matrix for the 1-NN/raw feature space, user-independent system. Dark areas indicate high confusion, while white areas indicate little or no confusion. Note that the values along the diagonal are lightest for dialing, typing, and stapling. This indicates that these actions contain the most variance across different users.

indicate that an interaction has been incorrectly classified as another interaction (i.e., an interaction is "confused" with another by the classifier). When looking at the confusion matrix for the user-independent system, it is easy to see the examples of interactions that had a high degree of variation across multiple users. Figure 87 shows the confusion matrix for the user-independent system. Dark areas indicate high confusion with lighter areas indicating little confusion.

According to this confusion matrix, the three actions that contained the most variance across all users were dialing the telephone, typing on the keyboard, and stapling papers together. This corresponds to the observations we made in Figure 86.

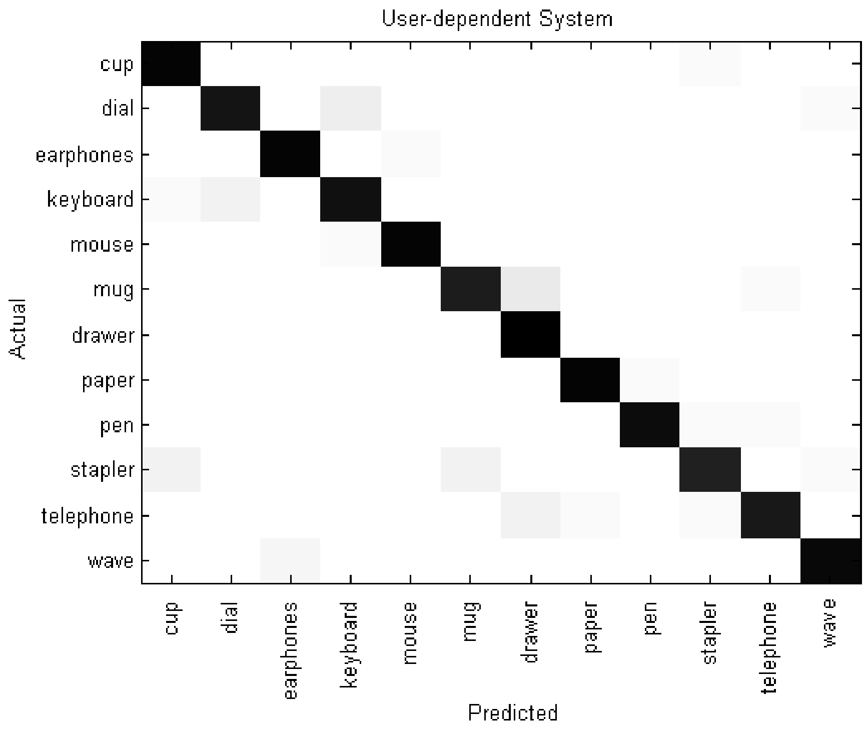


Fig. 88. Confusion matrix for the 1-NN/raw feature space, user-dependent system. Dark areas indicate high confusion, while white areas indicate little or no confusion.

We can also see that there was a lot of confusion between dialing the telephone and typing on a keyboard. We believe most of this confusion can be attributed to the users who dialed the telephone with an open palm, the posture typically used when typing. Other areas of high confusion included: stapling and drinking from a cup (when both were interacted with using a circular grip), picking up earphones and picking up a piece of paper (both of which use pinching postures), and opening a drawer and using a stapler (both of which can be done with a circular grip).

The confusion matrix for the user-dependent system gives us idea of the interactions that varied most across an individual user (see Figure 88). Obviously, this matrix contains much less confusion overall that the user-independent system; however, there are still some areas of confusion. Most notably, is the confusion that still occurs between typing on the keyboard and dialing the telephone. As with the user-independent system, there is still some confusion between interactions that can occur with objects held with a circular grip: cup and stapler, mug and stapler, drawer and telephone, and drawer and mug. There was also a small amount of confusion between picking up earphones and waving. We believe this was due to a single user who waved by bending the four non-thumb fingers towards the palm, rather than waving the hand with all fingers extended. Because of this type of wave, there were occasional examples of confusion with the pinching associated with holding a piece of paper.

One potential way that we could improve the user-independent system is by performing multi-modal training. With this form of training, we would identify the different variations of how users interact with the same object. We can then train our system by having these different variations be separate training sets that are all mapped to the same label. For example, we would have one set of training examples where users point to dial the phone and another set where users dial using an open palm. Although these would be classified as separate instances, they would both ultimately map to the same action, dialing.

G. Future Work

In this work we have seen that hand posture can be used as a cue in performing object-based activity recognition. However, we have also seen that some objects are interacted with using similar postures, even in a user-dependent system. For future work, we hope to combine hand posture with other forms of input that measure hand movement. These inputs could come in the form of accelerometers or 3-D position trackers. We believe that this extra information could be used to disambiguate interactions like using a stapler and drinking from a cup, both of which could potentially use a similar posture but would likely have different movements associated with them. This would then make our approach to activity recognition be based on hand gestures rather than hand postures.

The other obvious area for future work deals with segmentation and noise detection. In our experiments, data was recorded on an isolated interaction basis. For this approach to be beneficial to a real-world system, we would need to develop ways of designating when an interaction starts and stops. We also need to be able to detect instances when no interaction is taking place at all. The issue of segmenting hand postures and gestures is still an ongoing research effort [229, 230].

In addition to these issues, we also plan to try our experiments with more sophisticated classifiers. We have showed that reasonable results can be given with a simple 1-nearest neighbor or linear algorithm. In the near future we hope to implement and test other algorithms like neural networks, support vector machines, and hidden Markov models (HMMs). Using these more advanced classifiers will likely lead to higher accuracy, and may also provide an extra advantage of providing automatic segmentation. For example, Iba et al. were able to successfully recognize and segment hand gestures used to control mobile robots by introducing a “wait state” into their HMM [231]. It would also be beneficial to analyze additional dimensionality reduction techniques, such as kernel PCA/LDA (which are non-linear projections), as the CyberGlove contains many sensors that produce extra noise during hand posture recognition [232].

H. Chapter Summary

In this chapter, we have shown that hand posture can be used as a cue to perform object-based activity recognition, which can provide important context to context-aware applications. Furthermore, we have determined that when users are allowed to interact with objects as they would naturally, a user-dependent system is preferable over a user-independent system because of the high variation between users interacting with the same objects. We have shown that such a user-dependent system is capable of producing up to 96.9% accuracy using a simple linear classifier along with a PCA projection of the the raw sensor values from the CyberGlove II. For future work, hand posture could be combined with movement in order to yield higher recognition results. Segmentation and noise detection strategies also need to be investigated so that this approach can be used in a real-time, context-capturing system.

CHAPTER VIII

CONCLUSION

In this dissertation, we have looked into methods and algorithms for improving hand-sketched, primitive shape recognition. Our techniques have shown to not only be more accurate, but also to provide minimal constraint on how users must draw. This is a significant step in the advancement of free-sketch recognition [45]. By improving low-level recognition, we allow for the development of additional sketch-based applications that can have a broad impact on fields like engineering, education, science, mathematics, arts, and even music. In fact, our PaleoSketch recognizer has already been used in projects that improve corner finding [9], promote learning in children [20], teach people how to draw [168], recognize sketched Urdu characters [163], combine multiple approaches to improve recognition [136], recognize biology cell diagrams [39], and teach users how to draw Mandarin symbols [179]. Specifically, the main contributions of this work include:

- Showing that our geometric features provide a statistically significant difference over existing feature sets in recognizing low-level primitive shapes.
- Developing a primitive recognizer, PaleoSketch, that is capable of classifying more primitive shapes than existing recognizers. It does this without sacrificing accuracy or placing drawing constraints on the user. Our recognizer achieved close to 98% accuracy on single-stroke primitives, a result that finally reaches an “acceptable” level of recognition accuracy in pen-based interfaces [173].
- Presenting an initial technique for interpreting complex shapes, those composed of multiple primitives. This technique was shown to be quite accurate for shapes composed of two shapes, however, work may still be required in recognizing

complex shapes consisting of more than two shapes. The primary problem with interpreting complex fits of more than two shapes is undersegmentation (i.e., returning too few shapes). This could potentially be remedied with research into providing multiple complex interpretations, or by training on the sub-shapes of complex fits found in the training data.

- Showing that our low-level recognizer is capable of handling the shapes of one of the largest sketch domains in existence: military course of action diagrams. Our recognizer achieved accuracies above 98% on a primitive set consisting of 13 different primitive shapes, making up over 375 unique symbols in the domain.

- Learning more about the manner in which users draw primitive shapes. For example, we have learned that users are more likely to draw multi-stroke primitives (9% of the time) than they are to draw complex shapes (less than 5% of the time). Furthermore, we have seen that the more sides a shape has, the more likely the user is to draw it with multiple strokes. This observation coincides with those of previous researchers [127].

- Presenting one of the first multi-stroke primitive recognizers that provides both clustering and classification capabilities, without requiring the constraints that most systems place on users, such as button presses, temporal timeouts, or stroke interspersing prohibition. Overall, we achieved almost 90% recognition of multi-stroke primitives. Improved clustering techniques, as well as multiple interpretations, could provide additional aid in recognizing multi-stroke primitives.

- Proposing and implementing a web-based system, SOUSA, for collecting and sharing sketched data. SOUSA allows researchers to not only quickly collect sketch data, but also allows for data set sharing. This will allow researchers to easily compare algorithms on a common set of sketch examples, which will help advance the field of sketch recognition.

- Showing that the human-computer interaction principles of our work in sketch recognition translate into other research domains, such as office activity recognition. In this project, we showed how we can recognize the real-world objects that a user interacts with, simply by monitoring hand posture. One of the key findings of this work is that individual participants would use similar hand postures with interacting with the same object; however, these postures tended to be user-specific. These results indicate that every user is different, but recognition systems should be smart enough to handle this and should allow for unconstrained interaction.

REFERENCES

[1] J. Y. Han, “Low-cost multi-touch sensing through frustrated total internal reflection,” in *UIST ’05: Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, 2005, pp. 115–118.

[2] P. Brandl, C. Forlines, D. Wigdor, M. Haller, and C. Shen, “Combining and measuring the benefits of bimanual pen and direct-touch interaction on horizontal interfaces,” in *AVI ’08: Proceedings of the Working Conference on Advanced Visual Interfaces*, 2008, pp. 154–161.

[3] J. Lee, S. Hudson, and P. Dietz, “Hybrid infrared and visible light projection for location tracking,” in *UIST ’07: Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, 2007, pp. 57–60.

[4] J. M. Peschel, B. Paulson, and T. Hammond, “A surfaceless pen-based interface,” in *C&C ’09: Proceeding of the 7th ACM Conference on Creativity and Cognition*, 2009, pp. 433–434.

[5] T. Pavlidis and C. J. Van Wyk, “An automatic beautifier for drawings and illustrations,” *SIGGRAPH Computer Graphics*, vol. 19, no. 3, pp. 225–234, 1985.

[6] B. Paulson and T. Hammond, “Paleosketch: Accurate primitive sketch recognition and beautification,” in *IUI ’08: Proceedings of the 13th International Conference on Intelligent User Interfaces*, 2008, pp. 1–10.

[7] T. Hammond and B. O’Sullivan, “Recognizing free-form hand-sketched constraint network diagrams by combining geometry and context,” in *Proceedings of Eurographics Ireland*, 2007, pp. 67–74.

[8] J. Arvo and K. Novins, "Fluid sketches: Continuous recognition and morphing of simple hand-drawn shapes," in *UIST '00: Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, 2000, pp. 73–80.

[9] A. Wolin, B. Paulson, and T. Hammond, "Sort, merge, repeat: An algorithm for effectively finding corners in hand-sketched strokes," in *SBIM '09: Proceedings of the 6th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2009, pp. 93–99.

[10] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[11] A. Wolin, B. Eoff, and T. Hammond, "Shortstraw: A simple and effective corner finder for polylines," in *SBIM '08: Proceedings of the 5th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2008, pp. 33–40.

[12] B. Yu and S. Cai, "A domain-independent system for sketch recognition," in *GRAPHITE '03: Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, 2003, pp. 141–146.

[13] A. Apte, V. Vo, and T. D. Kimura, "Recognizing multistroke geometric shapes: An experimental evaluation," in *UIST '93: Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology*, 1993, pp. 121–128.

[14] M. J. Fonseca and J. A. Jorge, "Using fuzzy logic to recognize geometric shapes interactively," in *The 9th IEEE International Conference on Fuzzy Systems*, 2000, vol. 1, pp. 291–296.

[15] M. J. Fonseca, C. Pimentel, and J. A. Jorge, “Cali: An online scribble recognizer for calligraphic interfaces,” in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, 2002, pp. 51–58.

[16] B. Paulson and T. Hammond, “Marqs: Retrieving sketches using domain- and style-independent features learned from a single example using a dual-classifier,” *Journal of Multi-Modal User Interfaces*, vol. 2, no. 1, pp. 3–11, 2008.

[17] A. Caetano, N. Goulart, M. Fonseca, and J. Jorge, “Javasketchit: Issues in sketching the look of user interfaces,” in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, 2002, pp. 9–14.

[18] C. Alvarado and R. Davis, “Resolving ambiguities to create a natural computer-based sketching environment,” in *IJCAI ’01: Proceedings of the International Joint Conference on Artificial Intelligence*, 2001, pp. 1365–1371.

[19] A. Forsberg, M. Dieterich, and R. Zeleznik, “The music notepad,” in *UIST ’98: Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, 1998, pp. 203–210.

[20] B. Paulson, B. Eoff, A. Wolin, J. Johnston, and T. Hammond, “Sketch-based educational games: ‘Drawing’ kids away from traditional interfaces,” in *IDC ’08: Proceedings of the 7th International Conference on Interaction Design and Children*, 2008, pp. 133–136.

[21] T. Hammond, D. Logsdon, J. Peschel, J. Johnston, P. Taele, A. Wolin, and B. Paulson, “A sketch recognition interface that recognizes hundreds of shapes in course-of-action diagrams,” in *CHI ’10: Extended Abstracts on Human Factors in Computing Systems (to appear)*, 2010.

[22] Immersion, “Wireless data glove: The cyberglove ii wireless system,” February 2009, *http://www.immersion.com/3d/products/cyber_glove.php*.

[23] G. Johnson, M. D. Gross, J. Hong, and E. Y.-L. Do, “Computational support for sketching in design: A review,” *Foundations and Trends in Human-Computer Interaction*, vol. 2, no. 1, pp. 1–93, 2009.

[24] B. P. Bailey and J. A. Konstan, “Are informal tools better?: Comparing demais, pencil and paper, and authorware for early multimedia design,” in *CHI '03: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2003, pp. 313–320.

[25] V. Goel, *Sketches of Thought*, MIT Press, Cambridge, MA, 1995.

[26] A. Black, “Visible planning on paper and on screen: The impact of working medium on decision-making by novice graphic designers,” *Behaviour and Information Technology*, vol. 9, no. 4, pp. 283–296, 1990.

[27] Y. Y. Wong, “Rough and ready prototypes: Lessons from graphic design,” in *CHI '92: Posters and Short Talks of the SIGCHI Conference on Human Factors in Computing Systems*, 1992, pp. 83–84.

[28] L. Yeung, B. Plimmer, B. Lobb, and D. Elliffe, “Effect of fidelity in diagram presentation,” in *BCS-HCI '08: Proceedings of the 22nd British HCI Group Annual Conference on HCI*, 2008, pp. 35–44.

[29] T. Hammond and R. Davis, “Tahuti: A geometrical sketch recognition system for uml class diagrams,” in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, 2002, pp. 59–66.

[30] C. Alvarado and R. Davis, "Sketchread: A multi-domain sketch recognition engine," in *UIST '04: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, 2004, pp. 23–32.

[31] T. Kurtoglu and T. F. Stahovich, "Interpreting schematic sketches using physical reasoning," in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, 2002, pp. 78–85.

[32] M. Oltmans and R. Davis, "Naturally conveyed explanations of device behavior," in *PUI '01: Proceedings of the 2001 Workshop on Perceptive User Interfaces*, 2001, pp. 1–8.

[33] T. F. Stahovich, "Sketchit: A sketch interpretation tool for conceptual mechanical design," Tech. Rep. 1573, MIT Artificial Intelligence Laboratory, 1996.

[34] J. J. LaViola and R. C. Zeleznik, "Mathpad^2: A system for the creation and exploration of mathematical sketches," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 432–440, 2004.

[35] D. Tenneson and S. Becker, "Chempad: Generating 3D molecules from 2D sketches," in *SIGGRAPH '05: ACM SIGGRAPH Posters*, 2005, p. 87.

[36] T. Y. Ouyang and R. Davis, "Recognition of hand drawn chemical diagrams," in *AAAI '07: Proceedings of the 22nd Conference on Artificial Intelligence*, 2007, pp. 846–851.

[37] P. Taele and T. Hammond, "Using a geometric-based sketch recognition approach to sketch chinese radicals," in *AAAI '08: Proceedings of the 23rd Conference on Artificial Intelligence*, 2008, pp. 1832–1833.

[38] P. Taele and T. Hammond, "Hashigo: A next-generation sketch interactive system for Japanese kanji," in *IAAI '09: 21st Innovative Applications of Artificial Intelligence Conference*, 2009, pp. 153–158.

[39] P. Taele, J. Peschel, and T. Hammond, "A sketch interactive approach to computer-assisted biology instruction," in *Proceedings of the 2009 IUI Workshop on Sketch Recognition Posters*, 2009.

[40] J. Peschel and T. Hammond, "Strat: A sketched-truss recognition and analysis tool," in *International Conference on Distributed Multimedia Systems*, 2008, pp. 282–287.

[41] T. M. Sezgin, T. Stahovich, and R. Davis, "Sketch based interfaces: Early processing for sketch understanding," in *PUI '01: Proceedings of the 2001 Workshop on Perceptive User Interfaces*, 2001, pp. 1–8.

[42] J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes," in *UIST '07: Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, 2007, pp. 159–168.

[43] D. Rubine, "Specifying gestures by example," in *SIGGRAPH '91: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, 1991, pp. 329–337.

[44] H. H. Hse and A. R. Newton, "Recognition and beautification of multi-stroke symbols in digital ink," *Computers & Graphics*, vol. 29, no. 4, pp. 533–546, 2005.

[45] T. Hammond, B. Eoff, B. Paulson, A. Wolin, K. Dahmen, J. Johnston, and

P. Rajan, “Free-sketch recognition: Putting the CHI in sketching,” in *CHI ’08: Extended Abstracts on Human Factors in Computing Systems*, 2008, pp. 3027–3032.

[46] I. E. Sutherland, “Sketch pad a man-machine graphical communication system,” in *DAC ’64: Proceedings of the SHARE Design Automation Workshop*, 1964, pp. 6.329–6.346.

[47] D. C. Engelbart and W. K. English, “A research center for augmenting human intellect,” in *AFIPS ’68: Proceedings of the Fall Joint Computer Conference, Part I*, 1968, pp. 395–410.

[48] T. O. Ellis, J. F. Heafner, and W. L. Sibley, “The grail project: An experiment in man-machine communications,” RAND Memorandum RM-5999-ARPA, RAND Corporation, Santa Monica, CA, 1969.

[49] N. Negroponte, L. B. Groisser, and J. Taggert, “Hunch: An experiment in sketch recognition,” in *Environmental Design: Research and Practice, Volumes One and Two*, W. J. Mitchell, Ed., pp. 22.1.1–22.1.15. American Institute of Architects, Washington, D.C., 1972.

[50] N. Negroponte, “Recent advances in sketch recognition,” in *AFIPS ’73: Proceedings of the National Computer Conference and Exposition*, 1973, pp. 663–675.

[51] Palm Inc., “Palm USA - mobile products for consumers, professionals, and businesses,” November 2009, *http://www.palm.com.*

[52] Apple Inc., “Apple - iphone - mobile phone, ipod, and internet device.,” November 2009, *http://www.apple.com/iphone.*

[53] Hewlett-Packard Development Company L.P., "Handheld computers, windows mobile, smartphone, pda, gps device | hp," November 2009, *http://welcome.hp.com/country/us/en/prodserv/handheld.html.*

[54] Research In Motion Limited, "Smartphones, cell phones & smart phones at blackberry.com," November 2009, *http://www.blackberry.com.*

[55] Wacom, "Bamboo overview," November 2009, *http://www.wacom.com/bamboo/.*

[56] Wacom, "Cintiq - product overview," November 2009, *http://www.wacom.com/cintiq/.*

[57] Lenovo, "Lenovo - laptop computers - thinkpad," November 2009, *http://shop.lenovo.com/us/notebooks/thinkpad/.*

[58] Axiotron Corp., "Axiotron : Modbook," November 2009, *http://www.axiotron.com/index.php?id=modbook.*

[59] SMART Technologies ULC, "Smart technologies, industry leader in interactive whiteboard technology, the smart board," November 2009, *http://smarttech.com/.*

[60] Mimio, "Mimio interactive board overview," November 2009, *http://www.mimio.com/products/mimio_board/index.asp.*

[61] Anoto Group AB, "Anoto - start," November 2009, *http://www.anoto.com/.*

[62] Logitech, "io personal digital pen," November 2009, *http://www.logitech.com/index.cfm/431/963.*

[63] A. Meyer, "Pen computing: A technology overview and a vision," *SIGCHI Bulletin*, vol. 27, no. 3, pp. 46–90, 1995.

[64] C. Alvarado, "Multi-domain sketch understanding," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2004.

[65] R. C. Davis, T. S. Saponas, M. Shilman, and J. A. Landay, "Sketchwizard: Wizard of Oz prototyping of pen-based user interfaces," in *UIST '07: Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, 2007, pp. 119–128.

[66] B. Eoff and T. Hammond, "User identification by means of sketched stroke features," in *AAAI '08: Proceedings of the 23rd Conference on Artificial Intelligence*, 2008, pp. 1794–1795.

[67] E. Saund and T. P. Moran, "Perceptual organization in an interactive sketch editing application," in *IEEE International Conference on Computer Vision*, 1995, pp. 597–607.

[68] E. Saund and E. Lank, "Stylus input and editing without prior selection of mode," in *UIST '03: Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, 2003, pp. 213–216.

[69] E. Saund, J. Mahoney, D. Fleet, D. Larner, and E. Lank, "Perceptual organization as a foundation for intelligent sketch editing," in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, 2002, pp. 118–125.

[70] E. Saund, D. Fleet, D. Larner, and J. Mahoney, "Perceptually-supported image editing of text and graphics," in *UIST '03: Proceedings of the 16th Annual*

ACM Symposium on User Interface Software and Technology, 2003, pp. 183–192.

[71] Y. Zhu, J. Johnston, and T. Hammond, "Ringedit: A control point based editing approach in sketch recognition systems," in *Proceedings of the 2009 IUI Workshop on Sketch Recognition*, 2009.

[72] P. Morrel-Samuels, "Clarifying the distinction between lexical and gestural commands," *International Journal of Man-Machine Studies*, vol. 32, no. 5, pp. 581–590, 1990.

[73] K. Hinckley, P. Baudisch, G. Ramos, and F. Guimbretiere, "Design and analysis of delimiters for selection-action pen gesture phrases in scriboli," in *CHI '05: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2005, pp. 451–460.

[74] R. Zeleznik and T. Miller, "Fluid inking: Augmenting the medium of free-form inking with gestures," in *GI '06: Proceedings of Graphics Interface*, 2006, pp. 155–162.

[75] D. Goldberg and C. Richardson, "Touch-typing with a stylus," in *CHI '93: Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, 1993, pp. 80–87.

[76] T. Igarashi, S. Matsuoka, S. Kawachiya, and H. Tanaka, "Interactive beautification: A technique for rapid geometric design," in *UIST '97: Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, 1997, pp. 105–114.

[77] B. Plimmer and J. Grundy, "Beautifying sketching-based design tool content:

Issues and experiences," in *AUIC '05: Proceedings of the 6th Australasian Conference on User Interface*, 2005, pp. 31–38.

[78] P. Wais, A. Wolin, and C. Alvarado, "Designing a sketch recognition front-end: User perception of interface elements," in *SBIM '07: Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling*, 2007, pp. 99–106.

[79] A. C. Long, J. A. Landay, and L. A. Rowe, "'Those look similar!' Issues in automating gesture design advice," in *PUI '01: Proceedings of the 2001 Workshop on Perceptive User Interfaces*, 2001, pp. 1–5.

[80] J. Mankoff, S. E. Hudson, and G. D. Abowd, "Providing integrated toolkit-level support for ambiguity in recognition-based interfaces," in *CHI '00: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2000, pp. 368–375.

[81] J. Mankoff, G. D. Abowd, and S. E. Hudson, "Oops: A toolkit supporting mediation techniques for resolving ambiguity in recognition-based interfaces," *Computers & Graphics*, vol. 24, no. 6, pp. 819–834, 2000.

[82] J. I. Hong and J. A. Landay, "Satin: A toolkit for informal ink-based applications," in *UIST '00: Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, 2000, pp. 63–72.

[83] T. Hammond and R. Davis, "Ladder, a sketching language for user interface developers," *Computers & Graphics*, vol. 29, no. 4, pp. 518–532, 2005.

[84] R. L. Kullberg, "Mark your calendar!: Learning personalized annotation from integrated sketch and speech," in *CHI '95: Conference Companion on Human*

Factors in Computing Systems, 1995, pp. 302–303.

[85] A. Adler and R. Davis, "Speech and sketching: An empirical study of multimodal interaction," in *SBIM '07: Proceedings of the 4th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2007, pp. 83–90.

[86] P. R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow, "Quickset: Multimodal interaction for distributed applications," in *MULTIMEDIA '97: Proceedings of the 5th ACM International Conference on Multimedia*, 1997, pp. 31–40.

[87] K. Stolt, "Sketch recognition for course of action diagrams," M.S. thesis, Massachusetts Institute of Technology, Cambridge, MA, 2007.

[88] B. Pasternak and B. Neumann, "Adaptable drawing interpretation using object-oriented and constraint-based graphic specification," in *Proceedings of the 2nd International Conference on Document Analysis and Recognition*, 1993, pp. 359–364.

[89] R. P. Futrelle and N. Nikolakis, "Efficient analysis of complex diagrams using constraint-based parsing," *International Conference on Document Analysis and Recognition*, vol. 2, pp. 782, 1995.

[90] C. Calhoun, T. F. Stahovich, T. Kurtoglu, and L. B. Kara, "Recognizing multistroke symbols," in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, 2002, pp. 15–23.

[91] G. Costagliola, V. Deufemia, G. Polese, and M. Risi, "A parsing technique for sketch recognition systems," in *2004 IEEE Symposium on Visual Languages and Human Centric Computing*, 2004, pp. 19–26.

[92] G. Costagliola, V. Deufemia, and M. Risi, "Sketch grammars: A formalism for describing and recognizing diagrammatic sketch languages," in *ICDAR '05: Proceedings of the 8th International Conference on Document Analysis and Recognition*, 2005, pp. 1226–1231.

[93] M. D. Gross and E. Y.-L. Do, "Ambiguous intentions: A paper-like interface for creative design," in *UIST '96: Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, 1996, pp. 183–192.

[94] M. D. Gross and E. Y.-L. Do, "Demonstrating the electronic cocktail napkin: A paper-like interface for early design," in *CHI '96: Conference Companion on Human Factors in Computing Systems*, 1996, pp. 5–6.

[95] M. D. Gross, "The electronic cocktail napkin - a computational environment for working with design diagrams," *Design Studies*, vol. 17, no. 1, pp. 53–69, 1996.

[96] C. Alvarado, M. Oltmans, and R. Davis, "A framework for multi-domain sketch recognition," in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, 2002, pp. 1–8.

[97] M. Shilman, H. Pasula, S. Russell, and R. Newton, "Statistical visual language models for ink parsing," in *Sketch Understanding Papers from the 2002 AAAI Spring Symposium*, 2002, pp. 126–132.

[98] O. Veselova, "Perceptually based learning of shape descriptions," M.S. thesis, Massachusetts Institute of Technology, Cambridge, MA, 2007.

[99] T. Hammond and R. Davis, "Interactive learning of structural shape descriptions from automatically generated near-miss examples," in *IUI '06: Proceed-*

ings of the 11th International Conference on Intelligent User Interfaces, 2006, pp. 210–217.

[100] O. Veselova and R. Davis, “Perceptually based learning of shape descriptions for sketch recognition,” in *AAAI’04: Proceedings of the 19th Conference on Artificial Intelligence*, 2004, pp. 482–487.

[101] D. Sharon and M. van de Panne, “Constellation models for sketch recognition,” in *SBIM ’06: Proceedings of the 3rd Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2006, pp. 19–26.

[102] T. M. Sezgin, “Sketch interpretation using multiscale stochastic models of temporal patterns,” Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2006.

[103] Z. Sun, E. Jiang, and J. Sun, “Adaptive online multi-stroke sketch recognition based on hidden markov model,” *Lecture Notes in Artificial Intelligences*, vol. 3784, pp. 948–957, 2005.

[104] M. Szummer and Y. Qi, “Contextual recognition of hand-drawn diagrams with conditional random fields,” in *IWFHR ’04: Proceedings of the 9th International Workshop on Frontiers in Handwriting Recognition*, 2004, pp. 32–37.

[105] W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, pp. 202–209, McGraw-Hill, New York, NY, 2nd edition, 1973.

[106] C. H. Blickenstorfer, “Graffiti: Wow!,” *Pen Computing Magazine*, pp. 30–31, January 1995.

[107] X. Li and D.-Y. Yeung, “On-line handwritten alphanumeric character recognition using dominant points in strokes,” *Pattern Recognition*, vol. 30, no. 1, pp.

31–44, 1997.

[108] J. A. Landay and B. A. Myers, "Interactive sketching for the early stages of user interface design," in *CHI '95: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1995, pp. 43–50.

[109] B. A. Myers, R. G. McDaniel, R. C. Miller, A. S. Ferrency, A. Faulring, B. D. Kyle, A. Mickish, A. Klimovitski, and P. Doane, "The amulet environment: New models for effective user interface software development," *IEEE Transactions on Software Engineering*, vol. 23, no. 6, pp. 347–365, 1997.

[110] C. H. Damm, K. M. Hansen, and M. Thomsen, "Tool support for cooperative object-oriented design: Gesture based modelling on an electronic whiteboard," in *CHI '00: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2000, pp. 518–525.

[111] A. C. Long Jr., J. A. Landay, L. A. Rowe, and J. Michiels, "Visual similarity of pen gestures," in *CHI '00: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2000, pp. 360–367.

[112] B. Plimmer and I. Freeman, "A toolkit approach to sketched diagram recognition," in *BCS-HCI '07: Proceedings of the 21st British CHI Group Annual Conference on HCI*, 2007, pp. 205–213.

[113] H. Choi, B. Paulson, and T. Hammond, "Gesture recognition based on manifold learning," in *Structural, Syntactic, and Statistical Pattern Recognition*, vol. 5342 of *Lecture Notes in Computer Science*, pp. 247–256. Springer, 2008.

[114] M. Oltmans, "Envisioning sketch recognition: A local feature based approach to recognizing informal sketches," Ph.D. dissertation, Massachusetts Institute

of Technology, Cambridge, MA, 2007.

[115] S. Yang, “Symbol recognition via statistical integration of pixel-level constraint histograms,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 2, pp. 278–281, 2005.

[116] E. Learned-Miller, “Data driven image models through continuous joint alignment,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 2, pp. 236–250, 2006.

[117] L. B. Kara and T. F. Stahovich, “Hierarchical parsing and recognition of hand-sketched diagrams,” in *UIST ’04: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, 2004, pp. 13–22.

[118] L. B. Kara and T. F. Stahovich, “An image-based trainable symbol recognizer for sketch-based interfaces,” in *AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural*, 2004, pp. 99–105.

[119] L. B. Kara and T. F. Stahovich, “Sim-u-sketch: A sketch-based interface for simulink,” in *AVI ’04: Proceedings of the Working Conference on Advanced Visual Interfaces*, 2004, pp. 354–357.

[120] A. Wolin, B. Eoff, and T. Hammond, “Search your mobile sketch: Improving the ratio of interaction to information on mobile devices,” in *Proceedings of the 2009 IUI Workshop on Sketch Recognition*, 2009.

[121] T. Y. Ouyang and R. Davis, “A visual approach to sketched symbol recognition,” in *IJCAI ’09: Proceedings of the International Joint Conference on Artificial Intelligence*, 2009, pp. 1463–1468.

[122] D. H. Kim and M.-J. Kim, "A curvature estimation for pen input segmentation in sketch-based modeling," *Computer-Aided Design*, vol. 38, no. 3, pp. 238–248, 2006.

[123] H. Hse, M. Shilman, and A. R. Newton, "Robust sketched symbol fragmentation using templates," in *IUI '04: Proceedings of the 9th International Conference on Intelligent User Interfaces*, 2004, pp. 156–160.

[124] J. Hershberger and J. Snoeyink, "Speeding up the douglas-peucker line-simplification algorithm," in *Proceedings of the 5th International Symposium on Spatial Data Handling*, 1992, pp. 134–143.

[125] Y. Xiong and J. J. LaViola Jr., "Revisiting shortstraw: Improving corner finding in sketch-based interfaces," in *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, 2009, pp. 101–108.

[126] C. F. Herot, "Graphical input through machine recognition of sketches," in *SIGGRAPH '76: Proceedings of the 3rd Annual Conference on Computer Graphics and Interactive Techniques*, 1976, pp. 97–102.

[127] P. van Sommers, *Drawing and Cognition: Descriptive and Experimental Studies of Graphic Production Processes*, Cambridge University Press, Cambridge, UK, 1984.

[128] B. Tversky and P. Lee, "Pictorial and verbal tools for conveying routes," in *Spatial Information Theory: Cognitive and Computational Foundations of Geographic Information Science*, vol. 1661 of *Lecture Notes in Computer Science*, pp. 51–64. Springer, 1999.

[129] D. G. Hendry, "Sketching with conceptual metaphors to explain computational

processes," in *VLHCC '06: Proceedings of the Visual Languages and Human-Centric Computing*, 2006, pp. 95–102.

[130] J. Grundy and J. Hosking, "Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool," in *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, 2007, pp. 282–291.

[131] M. Shilman, Z. Wei, S.i Raghupathy, P. Simard, and D. Jones, "Discerning structure from freeform handwritten notes," in *ICDAR '03: Proceedings of the 7th International Conference on Document Analysis and Recognition*, 2003, pp. 60–65.

[132] C. M. Bishop, M. Svensen, and G. E. Hinton, "Distinguishing text from graphics in on-line handwritten ink," in *IWFHR '04: Proceedings of the 9th International Workshop on Frontiers in Handwriting Recognition*, 2004, pp. 142–147.

[133] R. Patel, B. Plimmer, J. Grundy, and R. Ihaka, "Ink features for diagram recognition," in *SBIM '07: Proceedings of the 4th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2007, pp. 131–138.

[134] A. Bhat and T. Hammond, "Using entropy to distinguish shape versus text in hand-drawn diagrams," in *IJCAI '09: Proceedings of the International Joint Conference on Artificial Intelligence*, 2009, pp. 1395–1400.

[135] K. Dahmen and T. Hammond, "Distinguishing between sketched scribble look alikes," in *AAAI '08: Proceedings of the 23rd Conference on Artificial Intelligence*, 2008, pp. 1790–1791.

[136] P. Corey and T. Hammond, "Gladder: Combining gesture and geometric sketch

recognition," in *AAAI '08: Proceedings of the 23rd Conference on Artificial Intelligence*, 2008, pp. 1788–1789.

[137] E. Y.-L. Do, "Design sketches and sketch design tools," *Knowledge-Based Systems*, vol. 18, no. 8, pp. 383 – 405, 2005.

[138] J. Lin, M. W. Newman, J. I. Hong, and J. A. Landay, "Denim: Finding a tighter fit between tools and practice for web site design," in *CHI '00: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2000, pp. 510–517.

[139] J. Lin, M. W. Newman, J. I. Hong, and J. A. Landay, "Denim: An informal sketch-based tool for early stage web design," in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, 2002, pp. 148–149.

[140] S. Han and G. Medioni, "3Dsketch: Modeling by digitizing with a smart 3D pen," in *MULTIMEDIA '97: Proceedings of the 5th ACM International Conference on Multimedia*, 1997, pp. 41–49.

[141] T. Igarashi, S. Matsuoka, and H. Tanaka, "Teddy: A sketching interface for 3D freeform design," in *SIGGRAPH '99: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, 1999, pp. 409–416.

[142] S.-H. Bae, R. Balakrishnan, and K. Singh, "Ilovesketch: As-natural-as-possible sketching system for creating 3D curve models," in *UIST '08: Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, 2008, pp. 151–160.

[143] O. Bimber, L. M. Encarnacao, and A. Stork, "A multi-layered architecture for

sketch-based interaction within virtual environments," *Computers & Graphics*, vol. 24, pp. 851–867, 2000.

[144] E. Y.-L. Do, "Vr sketchpad, create instant 3D worlds by sketching on a transparent window," in *Proceedings of CAAD Futures*, 2001, pp. 161–172.

[145] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes, "Sketch: An interface for sketching 3D scenes," in *SIGGRAPH '96: Proceedings of the 23rd International Conference on Computer Graphics and Interactive Techniques*, 1996, pp. 163–170.

[146] M. D. Gross and E. Y.-L. Do, "Drawing on the back of an envelope: A framework for interacting with application programs by freehand drawing," *Computers & Graphics*, vol. 24, no. 6, pp. 835 – 849, 2000.

[147] H. Lipson and M. Shpitalni, "Correlation-based reconstruction of a 3D object from a single freehand sketch," in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, 2002, pp. 99–104.

[148] L. Gennari, L. B. Kara, T. F. Stahovich, and K. Shimada, "Combining geometry and domain knowledge to interpret hand-drawn diagrams," *Computers & Graphics*, vol. 29, no. 4, pp. 547–562, 2005.

[149] E. Lank, J. S. Thorley, and S. J.-S. Chen, "An interactive system for recognizing hand drawn uml diagrams," in *CASCON '00: Proceedings of the 2000 Conference of the Centre for Advanced Studies on Collaborative Research*, 2000, pp. 7–21.

[150] B. Plimmer, J. Grundy, J. Hosking, and R. Priest, "Inking in the ide: Experiences with pen-based design and annotatio," in *VL/HCC '06: IEEE Sympo-*

sium on Visual Languages and Human-Centric Computing, 2006, pp. 111–115.

[151] L. Qiu, “Sketchuml: The design of a sketch-based tool for uml class diagrams,” in *ED-MEDIA ’07: Proceedings of World Conference on Educational Multimedia, Hypermedia & Telecommunications*, 2007, pp. 986–994.

[152] G. Tilak and K. Ananthakrishnan, “Sketchuml sketch based approach to class diagrams,” in *Proceedings of the 2009 IUI Workshop on Sketch Recognition Posters*, 2009.

[153] M. D. Muzumdar, “Icemendr: Intelligent capture environment for mechanical engineering drawing,” M.S. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

[154] M. Oltmans, “Understanding naturally conveyed explanations of device behavior,” Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2007.

[155] S. W. Zamora and E. A. Eyjolfsdottir, “Circuitboard: Sketch-based circuit design and analysis,” in *Proceedings of the 2009 IUI Workshop on Sketch Recognition*, 2009.

[156] R. Zeleznik, T. Miller, C. Li, and J. J. LaViola Jr., “Mathpaper: Mathematical sketching with fluid support for interactive computation,” in *Smart Graphics*, vol. 5166 of *Lecture Notes in Computer Science*, pp. 20–32. Springer, 2008.

[157] C. Li, T. Miller, R. Zeleznik, and J. LaViola Jr., “Algosketch: Algorithm sketching and interactive computation,” in *SBIM ’08: Proceedings of the 5th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, 2008, pp. 175–182.

[158] T. O'Connell, C. Li, T. S. Miller, R. C. Zeleznik, and J. J. LaViola Jr., "A usability evaluation of algosketch: A pen-based application for mathematics," in *SBIM '09: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, 2009, pp. 149–157.

[159] N. E. Matsakis, "Recognition of handwritten mathematical expressions," M.S. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

[160] T. Hammond, "Ladder: A perceptually-based language to simplify sketch recognition user interfaces development," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2007.

[161] T. Y. Ouyang, "Recognition of hand drawn chemical diagrams," M.S. thesis, Massachusetts Institute of Technology, Cambridge, MA, 2007.

[162] D. Tenneson, "Interpretation of molecule conformations from drawn diagrams," Ph.D. dissertation, Brown University, Providence, RI, 2008.

[163] N. Shahzad, B. Paulson, and T. Hammond, "Urdu qaeda: Recognition system for isolated urdu characters," in *Proceedings of the 2009 IUI Workshop on Sketch Recognition*, 2009.

[164] J. V. Mahoney and M. P. J. Fromherz, "Three main concerns in sketch recognition and an approach to addressing them," in *Sketch Understanding Papers from the 2002 AAAI Spring Symposium*, 2002, pp. 105–112.

[165] F. Di Fiore and F. Van Reeth, "A multilevel sketching tool for 'pencilandpaper' animation," in *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding*, 2002, pp. 32–36.

[166] R. C. Davis, B. Colwell, and J. A. Landay, “K-sketch: A ‘kinetic’ sketch pad for novice animators,” in *CHI ’08: Proceeding of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems*, 2008, pp. 413–422.

[167] R. C. Davis, “K-sketch: A ’kinetic’ sketch pad for novice animators,” Ph.D. dissertation, University of California, Berkeley, CA, 2008.

[168] D. M. Dixon, “A methodology using assistive sketch recognition for improving a persons ability to draw,” M.S. thesis, Texas A&M University, College Station, TX, 2009.

[169] T. Kato, T. Kurita, N. Otsu, and K. Hirata, “A sketch retrieval method for full color image databases - query by visual example,” in *11th IAPA International Conference on Pattern Recognition*, 1992, pp. 530–533.

[170] W. H. Leung and T. Chen, “Retrieval of sketches based on spatial relation between strokes,” in *Proceedings of the 2002 International Conference on Image Processing*, 2002, pp. I–908–I–911.

[171] B. N. Schilit, G. Golovchinsky, and M. N. Price, “Beyond paper: Supporting active reading with free form digital ink annotations,” in *CHI ’98: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1998, pp. 249–256.

[172] B. Paulson, P. Rajan, P. Davalos, R. Gutierrez-Osuna, and T. Hammond, “What!?! No rubine features?: Using geometric-based features to produce normalized confidence values for sketch recognition,” in *Proceedings of the VL/HCC Sketch Tools for Diagramming Workshop*, 2008, pp. 57–63.

[173] M. LaLomia, “User acceptance of handwritten recognition accuracy,” in *CHI*

'94: Conference Companion on Human Factors in Computing Systems, 1994, pp. 107–108.

[174] A. Wolin, "Segmenting hand-drawn strokes," M.S. thesis, Texas A&M University, College Station, TX, 2010.

[175] A. Wolin, B. Paulson, and T. Hammond, "The evolution of stroke segmentation," 2010, manuscript in preparation.

[176] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice-Hall, Upper Saddle River, NJ, 2nd edition, 1999.

[177] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[178] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten, "The weka data mining software: An update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[179] P. Taele and T. Hammond, "Lamps: A sketch recognition-based teaching tool for mandarin phonetic symbols i," *Visual Languages and Computing (to appear)*, 2010.

[180] K. D. Forbus, J. Usher, and V. Chapman, "Sketching for military courses of action diagrams," in *IUI '03: Proceedings of the 8th International Conference on Intelligent User Interfaces*, 2003, pp. 61–68.

[181] J. A. Pittman, I. A. Smith, P. R. Cohen, S. L. Oviatt, and T.-C. Yang, "Quickset: A multimodal interface for military simulation," in *CGF-BR '96: Pro-*

ceedings of the 6th Conference on Computer-Generated Forces and Behavioral Representation, 1996, pp. 217–224.

[182] B. Paulson and T. Hammond, "Towards a framework for truly natural low-level sketch recognition," in *2009 IUI Workshop on Sketch Recognition Posters*, 2009.

[183] T. A. Hammond and R. Davis, "Recognizing interspersed sketches quickly," in *GI '09: Proceedings of Graphics Interface*, 2009, pp. 157–166.

[184] R. Zhao, "Incremental recognition in gesture-based and syntax-directed diagram editors," in *CHI '93: Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, 1993, pp. 95–100.

[185] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972.

[186] M. Oltmans, C. Alvarado, and R. Davis, "Etcha sketches: Lessons learned from collecting sketch data," in *AAAI Fall Symposium: Making Pen-Based Interaction Intelligent and Natural*, 2004, pp. 134–140.

[187] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," Tech. Rep., California Institute of Technology, 2007.

[188] S. M. Bileschi, "Streetscenes: Towards scene understanding in still images," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2006.

[189] L. von Ahn and L. Dabbish, "Labeling images with a computer game," in *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2004, pp. 319–326.

[190] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, “Labelme: A database and web-based tool for image annotation,” *MIT AI Lab Memo AIM-2005-025*, vol. 1, pp. 1–10, 2005.

[191] L. von Ahn, R. Liu, and M. Blum, “Peekaboom: A game for locating objects in images,” in *CHI ’06: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2006, pp. 55–64.

[192] G. Johnson, “Picturephone: A game for sketch data capture,” in *2009 IUI Workshop on Sketch Recognition*, 2009.

[193] A. Wolin, D. Smith, and C. Alvarado, “A pen-based tool for efficient labeling of 2D sketches,” in *SBIM ’07: Proceedings of the 4th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2007, pp. 67–74.

[194] B. Paulson, A. Wolin, J. Johnston, and T. Hammond, “Sousa: Sketch-based online user study applet,” in *SBIM ’08: Proceedings of the 5th Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2008, pp. 81–88.

[195] B. L. Kaster, E. R. Jacobson, W. Moreira, B. Paulson, and T. A. Hammond, “Sousa v2.0: Automatically generating secure and searchable data collection studies,” in *VLC ’09: International Workshop on Visual Languages and Computing*, 2009, pp. 365–368.

[196] B. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in *Proceedings of the Workshop on Mobile Computing Systems and Applications*, 1994, pp. 85–90.

[197] A. K. Dey and G. D. Abowd, “Towards a better understanding of context and context-awareness,” GVU Technical Report GIT-GVU-99-22, College of

Computing, Georgia Institute of Technology, Atlanta, GA, 1999.

[198] A. K. Dey, D. Salber, G. D. Abowd, and M. Futakawa, "The conference assistant: Combining context-awareness with wearable computing," in *Proceedings of the 3rd International Symposium on Wearable Computers*, 1999, pp. 21–28.

[199] K. Kuutti, "Activity theory as a potential framework for human-computer interaction research," in *Context and Consciousness: Activity Theory and Human-Computer Interaction*, B. A. Nardi, Ed., pp. 17–44. MIT Press, Cambridge, MA, 1995.

[200] B. A. Nardi, "Studying context: A comparison of activity theory, situated action models, and distributed cognition," in *Context and Consciousness: Activity Theory and Human-Computer Interaction*, B. A. Nardi, Ed., pp. 69–102. MIT Press, Cambridge, MA, 1995.

[201] D. Surie, F. Lagriffoul, T. Pederson, and D. Sjolie, "Activity recognition based on intra and extra manipulation of everyday objects," *Ubiquitous Computing Systems*, vol. 4836, pp. 196–210, 2007.

[202] D. Surie, T. Pederson, F. Lagriffoul, L.-E. Janlert, and D. Sjolie, "Activity recognition using an egocentric perspective of everyday objects," *Ubiquitous Intelligence and Computing*, vol. 4611, pp. 246–257, 2007.

[203] J. Pascoe, "Adding generic contextual capabilities to wearable computers," in *Proceedings of the 2nd International Symposium on Wearable Computers*, 1998, pp. 92–99.

[204] D. J. Moore, I. A. Essa, and M. H. Hayes III, "Exploiting human actions and object context for recognition tasks," in *Proceedings of the 7th IEEE*

International Conference on Computer Vision, 1999, vol. 1, pp. 80–86.

[205] W. W. Mayol and D. W. Murray, “Wearable hand activity recognition for event summarization,” in *Proceedings of the 9th IEEE International Symposium on Wearable Computers*, 2005, pp. 122–129.

[206] X. Sun, C.-W. Chen, and B. S. Manjunath, “Probabilistic motion parameter models for human activity recognition,” in *Proceedings of the 16th International Conference on Pattern Recognition*, 2002, vol. 1, pp. 443–446.

[207] L. Bao and S. S. Intille, “Activity recognition from user-annotated acceleration data,” *Pervasive Computing*, vol. 3001, pp. 1–17, 2004.

[208] J. B. J. Bussmann, W. L. J. Martens, J. H. M. Tulen, F. C. Schasfoot, H. J. G. van den Berg-Emons, and H. J. Stam, “Measuring daily behavior using ambulatory accelerometry: The activity monitor,” *Behavior Research Methods, Instruments, & Computers*, vol. 33, no. 3, pp. 349–356, 2001.

[209] F. Foerster, M. Smeja, and J. Fahrenberg, “Detection of posture and motion by accelerometry: A validation study in ambulatory monitoring,” *Computers in Human Behavior*, vol. 15, no. 5, pp. 571–583, 1999.

[210] N. Kern, B. Schiele, and A. Schmidt, “Multi-sensor activity context detection for wearable computing,” *Ambient Intelligence*, vol. 2875, pp. 220–232, 2003.

[211] S.-W. Lee and K. Mase, “Activity and location recognition using wearable sensors,” *IEEE Pervasive Computing*, vol. 1, no. 3, pp. 24–32, 2002.

[212] J. Lester, T. Choudhury, N. Kern, G. Borriello, and B. Hannaford, “A hybrid discriminative/generative approach for modeling human activities,” in *International Joint Conferences on Artificial Intelligence*, 2005, pp. 766–772.

[213] D. Minnen, T. Starner, J.A. Ward, P. Lukowicz, and G. Troster, "Recognizing and discovering human actions from on-body sensor data," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, 2005, pp. 1545–1548.

[214] J. A. Ward, P. Lukowicz, G. Troster, and T. E. Starner, "Activity recognition of assembly tasks using body-worn microphones and accelerometers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1553–1567, 2006.

[215] D. J. Patterson, D. Fox, H. Kautz, and M. Philipose, "Fine-grained activity recognition by aggregating abstract object usage," in *Proceedings of the 9th IEEE International Symposium on Wearable Computers*, 2005, pp. 44–51.

[216] M. Philipose, K. P. Fishkin, M. Perkowitz, D. J. Patterson, D. Fox, H. Kautz, and D. Hahnel, "Inferring activities from interactions with objects," *IEEE Pervasive Computing*, vol. 3, no. 4, pp. 50–57, 2004.

[217] D. Ayers and M. Shah, "Monitoring human behavior from video taken in an office environment," *Image and Vision Computing*, vol. 19, no. 12, pp. 833–846, 2001.

[218] N. Oliver, A. Garg, and E. Horvitz, "Layered representations for learning and inferring office activity from multiple sensory channels," *Computer Vision Image Understanding*, vol. 96, no. 2, pp. 163–180, 2004.

[219] R. Want, B. Schilit, N. Adams, R. Gold, K. Petersen, D. Goldberg, J. Ellis, and M. Weiser, "The parctab ubiquitous computing experiment," *IEEE Personal Communications*, vol. 353, pp. 45–101, 1996.

[220] L. Y. Chang, N. S. Pollard, T. M. Mitchell, and E. P. Xing, "Feature selection for grasp recognition from optical markers," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 2944–2950.

[221] K. Bernardin, K. Ogawara, K. Ikeuchi, and R. Dillmann, "A sensor fusion approach for recognizing continuous human grasping sequences using hidden markov models," *IEEE Transactions on Robotics*, vol. 21, no. 1, pp. 47–57, 2005.

[222] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, NY, 2006.

[223] A. N. Tikhonov, "On the stability on inverse problems," *Doklady Akademii Nauk SSSR*, vol. 39, no. 5, pp. 195–198, 1943.

[224] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, New York, NY, 2nd edition, 1990.

[225] K. Pearson, "On lines and planes of closest fit to systems of points in space," *Philosophical Magazine*, vol. 2, pp. 559–572, 1901.

[226] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *Journal of Educational Psychology*, vol. 24, pp. 417–441, 1933.

[227] R. A. Fisher, "The statistical utilization of multiple measurements," *Annals of Eugenics*, vol. 8, pp. 376–386, 1938.

[228] R. Kohavi and F. Provost, "Glossary of terms," *Machine Learning*, vol. 30, no. 2, pp. 271–274, 1998.

[229] P. A. Harling and A. D. N. Edwards, "Hand tension as a gesture segmentation cue," in *Proceedings of Gesture Workshop on Progress in Gestural Interaction*, 1996, pp. 75–88.

[230] C. Li and B. Prabhakaran, "A similarity measure for motion stream segmentation and recognition," in *Proceedings of the 6th International Workshop on Multimedia Data Mining*, 2005, pp. 89–94.

[231] S. Iba, J. M. V. Weghe, C. J. J. Paredis, and P. K. Khosla, "An architecture for gesture-based control of mobile robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1999, vol. 2, pp. 851–857.

[232] B. Scholkopf, A. J. Smola, and K.-R. Muller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

VITA

Name:	Brandon Chase Paulson
Address:	Department of Computer Science & Engineering Texas A&M University 3112 TAMU College Station, TX 77843
Email Address:	bpaulson@cse.tamu.edu
Education:	B.S., Computer Science, Baylor University, 2004 Ph.D., Computer Science, Texas A&M University, 2010