UNIVERSITY OF CALIFORNIA
RIVERSIDE

Enabling Instructional Applications on Pentop Computers

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

WeeSan Lee

March 2009

The Dissertation of WeeSan Lee is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

First of all, I would like to thank the members of my committee for their time and valuable suggestions. Especially, I am deeply indebted to my advisor, Professor Thomas F. Stahovich, for his constant advice and support, as well as the endless hours he has spent revising my papers and presentations. Without that, this dissertation literally stops right here.

I also would like to thank my lab mates for their help on my research and the fun time we spent together. I especially thank Rumi de Silva for her help on statics problems and user studies, Tyler Bischel for sharing my view on software design and for easing my mood with funny jokes, Eric Peterson for proofreading my papers and providing car fixing tips, and Ryan Rusich for his encouragement and countless rides to my car.

A special thanks to Terri Phonharath for lending a hand when I needed it the most, Dr. Tom Payne and Victor Hill for providing their support on the rainy days.

Thanks to my parents, sisters and brother, for simply believing in me and being supportive from the other side of the globe.

Thanks to my sons, Frank and Jeffrey, for their purest and endless hugs, which magically turned my bad days into bright and shiny ones. Last but not least, to my wife, Fui Lin, for her patience and support throughout the course of my Ph.D. studies.

To my parents, Seng Kiang and Siew Hong,

and my sisters and brother, Siow Hoon, Siow Yen and Wee Chuan,

and my sons, Frank and Jeffrey.

Finally and most importantly, my wife, Fui Lin.

ABSTRACT OF THE DISSERTATION

Enabling Instructional Applications on Pentop Computers

by

WeeSan Lee

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, March 2009
Dr. Thomas F. Stahovich, Chairperson

While traditional computer interfaces based on the mouse and keyboard are ubiquitous, they are ill suited to many common application domains. This is particularly true in education, where recent research suggests that students perform better when instructional interfaces are more similar to work practice. Thus, the goal of our work is to create computational techniques and user interface design principles to enable natural, pen-based tutoring systems that scaffold students in solving problems in the same way they would ordinarily solve them with paper and pencil. In our work, we have focused on interfaces suitable for a "pentop computer," a writing instrument that is used with special dot-patterned paper, and that has an integrated digitizer and embedded processor. A pentop is capable of producing dynamic output in the form of synthesized speech and recorded sound clips.

Accurate shape recognition is an essential foundation for developing pen-based interfaces. We created a trainable, multi-stroke recognizer that is insensitive to orientation, non-uniform scaling, and drawing order. Symbols are represented internally as attributed relational graphs describing both the geometry and topology of the symbols. Symbol recognition is accomplished by finding the definition symbol whose attributed relational graph best matches that of the unknown symbol. We developed five efficient approximate matching techniques to perform the graph matching.

To explore instructional and interface design issues, we created Newton's Pen, a pentop-based statics tutor. This system, which is intended for undergraduate education, scaffolds students in the construction of free body diagrams and equilibrium equations. Newton's Pen employs a finite state machine architecture that effectively models the student's problem-solving progress, thus serving as a convenient means for providing context-sensitive tutorial feedback. User studies suggest that Newton's Pen is an effective teaching tool, and that students are satisfied with the interface.

A key issue in the design of pentop interfaces is how to provide effective feedback to the user. To explore this issue, we developed PaperCAD, a system that enables users to query geometric information from printed CAD drawings. PaperCAD employs two methods of feedback: audio feedback with an adjustable level of conciseness, and a PDA that provides a video display of the portion of the drawing near the pen tip. This system also employs a novel technique that uses a hidden Markov model to correct interpretation errors in handwritten equations. Results of a user study suggest that users are highly satisfied with the interface and prefer it to a traditional WIMP interface.

Contents

List of Figures **xi**

List of Tables **xiv**

1 Introduction **1**
- 1.1 Shape Recognizer . . . 3
- 1.2 Newton's Pen . . . 4
- 1.3 PaperCAD . . . 7
- 1.4 Organization of the Dissertation . . . 9

2 Literature Review **10**
- 2.1 Shape and Symbol Recognition . . . 10
  - 2.1.1 Graph-based Recognizers . . . 10
  - 2.1.2 Feature-based Recognizers . . . 12
  - 2.1.3 Recognizers Using Other Techniques . . . 13
- 2.2 Intelligent Tutoring Systems . . . 14
- 2.3 Pen-based User Interfaces . . . 16
- 2.4 Paper-based User Interfaces . . . 17
- 2.5 Equation Recognition, Interpretation and Correction . . . 18

3 An Efficient Graph-Based Recognizer for Hand-Drawn Symbols **20**
- 3.1 Introduction . . . 20
- 3.2 Representation . . . 22
- 3.3 Measuring Similarity . . . 25
  - 3.3.1 Primitive Count Error . . . 27
  - 3.3.2 Primitive Type Error . . . 28
  - 3.3.3 Relative Length Error . . . 28
  - 3.3.4 Number of Intersections Error . . . 29
  - 3.3.5 Intersection Angle Error . . . 30
  - 3.3.6 Intersection Location Error . . . 31
- 3.4 Graph Matching . . . 32

3.4.1 Stochastic Matching . . . 33
3.4.2 Error-Driven Matching . . . 35
3.4.3 Greedy Matching . . . 36
3.4.4 Hybrid Matching . . . 36
3.4.5 Sort Matching . . . 37
3.5 Training . . . 39
3.6 Results . . . 43
3.6.1 Experiment One: Learning Rate . . . 44
3.6.2 Experiment Two: Drawing Order . . . 47
3.7 Discussion . . . 50
3.8 Summary . . . 54

4 Newton's Pen: A Pen-based Tutoring System for Statics 57
4.1 Introduction . . . 57
4.2 System Overview . . . 59
4.3 System Design . . . 64
4.3.1 System Architecture . . . 64
4.3.2 Recognition . . . 69
4.3.3 Free Body Diagram Interpreter . . . 75
4.3.4 Equation Interpreter . . . 76
4.3.5 Tutoring . . . 79
4.4 User Studies . . . 80
4.4.1 Study 1: Pilot Study . . . 81
4.4.2 Study 2: User study in an individual setting . . . 83
4.4.3 Study 3: User study in a group setting . . . 97
4.5 Discussion and Future Work . . . 102
4.6 Summary . . . 105

5 PaperCAD - A Paper-based CAD System 107
5.1 Introduction . . . 107
5.2 System Overview . . . 111
5.3 Prototyping Environment: ProtoPaper . . . 114
5.4 System Details . . . 116
5.4.1 Extension Line Recognizer . . . 117
5.4.2 Arrow Recognizer . . . 118
5.4.3 Interpreting Dimensions . . . 122
5.4.4 Tracing Gesture Recognizer . . . 123
5.4.5 Text Label and Edit Gesture Recognizer . . . 124
5.5 Equation Interpretation . . . 124
5.6 User Study . . . 128
5.7 Discussion and Future Work . . . 133
5.8 Summary . . . 134

6 Conclusion 136

Bibliography **141**

List of Figures

1.1 The FLY pentop computer. 2

1.2 (a) Worksheet for drawing free body diagrams. (b) Worksheet for writing equilibrium equations. 5

1.3 A drawing formatted for PaperCAD. 7

3.1 Top: An ideal square drawn with a single, counter-clockwise pen stroke. Arrows show the direction of drawing. Bottom: The corresponding ARG. I = number of intersections, A = intersection angle, L = intersection location, R = relative length. 23

3.2 Gaussian probability density function and modified probability density function for $\mu = 0$ and $\sigma = 1$. 27

3.3 The squash function from Equation 3.7. 30

3.4 Graph matching: assignments **b-2** and **c-3** are correct, while **a-1** and **d-4** are not. 34

3.5 Pseudo code for stochastic matcher. 35

3.6 Pseudo code for error-driven matcher. 37

3.7 Pseudo code for greedy matcher. 38

3.8 Pseudo code for hybrid matcher. 38

3.9 Pseudo code for sort matcher. 40

3.10 Proximity matching of two pivot symbols. The reference symbol is shown with bold primitives. The arrows indicate the assignments of primitives. . . 41

3.11 Computing the directed distance from primitive A to primitive B. Each point on A is mapped to the closest point on B. 42

3.12 Symbols from the user study drawn by one of the participants. 44

3.13 (Left) Top-one and (Right) top-three accuracy vs. the number of training examples. Stochastic: max-iterations = 300. Greedy1 = one round of greedy search. Greedy2 = two rounds. Error-driven: no-improvement-limit = 150 iterations, max-iterations = 300. 46

3.14 (Left) Top-one and (Right) top-three accuracy vs. the number of random changes to the drawing order. One random change consists of swapping the drawing order of a randomly selected pair of primitives. Sxxx = stochastic search with max-iterations = xxx. Greedy1 = one round of greedy search. Greedy2 = two rounds. Exxx = error-driven search with no-improvement-limit = xxx iterations and max-iterations = 300. 48

4.1 A typical problem from Newton's Pen. This problem was used for the pretest in the user studies. 60
4.2 (a) Worksheet for drawing free body diagrams. (b) Worksheet for writing equilibrium equations. 61
4.3 Symbol sheet used for correcting recognition errors when writing equations. 62
4.4 A typical force. 67
4.5 A portion of the finite state machine corresponding to the free body diagram in Figure 4.2a. The system is currently in state *Forces*. 68
4.6 Resampled arrow. Inverse curvature at point A is $cos(\theta)$. 73
4.7 Inverse curvature of the arrow from Figure 4.6. 73
4.8 Problem concerning a mass on an inclined plane. This problem was used as both a model and transfer problem in the user studies. 81
4.9 Problem concerning a mass subject to an applied force and friction. This problem was used as both a model and transfer problem in the user studies. 82
4.10 Problem-solving performance on free body diagrams for Study 2: the average number of problem-solving events per user. See Table 4.4 for a description of the events. 91
4.11 Usability issues encountered while drawing free body diagrams for Study 2: the average number of usability events per user. See Table 4.4 for a description of the events. 91
4.12 Problem-solving performance on equations for Study 2. "Physics Help: Hint" = average number of taps on "HELP" button to obtain problem-solving hints. "Physics Help: Answer" = average number of taps on "HELP" button to obtain the answer (i.e., the correct equation). "Physics Errors" = average number of errors in the equations. "Attempts" = average number of attempts required to derive the two equilibrium equations. "Equations Copied" = average number of equations directly copied from the "HELP" button. . . . 92
4.13 Usability issues encountered while deriving equilibrium equations for Study 2: the average number of usability events per user. See Table 4.4 for a description of the events. 92

5.1 A drawing formatted for PaperCAD. 109
5.2 Typical dimensions. 112
5.3 ProtoPaper prototyping environment used to implement PaperCAD. System users do not see the PC. 115
5.4 Ambiguous extension line. 118

5.5 The inverse curvature of (a) a single-headed arrow and (b) a double-headed arrow. 120
5.6 (a) Simple bounding box approach to clustering. (b) Rotating the equation 11° counter-clockwise before constructing bounding boxes results in more accurate clustering. 125
5.7 Drawings from user study. 130

List of Tables

3.1 Error metrics and corresponding weights. 25
3.2 Average time to classify a symbol in ms. Stochastic: max-iterations = 300. Greedy1 = one round of greedy search. Greedy2 = two rounds. Error-driven: no-improvement-limit = 150 iterations, max-iterations = 300. 45
3.3 Average time to classify a symbol in ms. Sxxx = stochastic search with max-iterations = xxx. Greedy1 = one round of greedy search. Greedy2 = two rounds. Exxx = error-driven search with no-improvement-limit = xxx iterations and max-iterations = 300. 49

4.1 The matrix representation of Equation 4.1. 78
4.2 Rubric for scoring the free body diagram from the pretest. 85
4.3 Rubric for scoring the equations from the pretest. 85
4.4 Event categories used for analyzing the video from the experimental sessions. 86
4.5 Overall Performance for Study 2. See Table 4.2 and Table 4.3 for the scoring rubrics. Time is in minutes. 87
4.6 Perceived ease of drawing correctly recognized objects and equations for Study 2. 10 = easy, 1 = hard. 95
4.7 Perceived usability for Study 2. 95
4.8 Perceived educational usefulness for Study 2. 96
4.9 Overall reaction to Newton's Pen from Study 2. 96
4.10 Perceived ease of drawing correctly recognized objects and equations for Study 3. 10 = easy, 1 = hard. 100
4.11 Perceived usability for Study 3. 100
4.12 Perceived educational usefulness for Study 3. 101
4.13 Overall reaction to Newton's Pen from Study 3. 101

5.1 Legal state transitions for equation HMM. ***digit*** = digits from '0' to '9'; ***alpha*** = all uppercase letters; ***alpha_cs*** = all uppercase letters except 'C' and 'S'; ***op*** = '+', '-', '*', and '/'. 127
5.2 Perceived ease of drawing correctly recognized objects and equations. 10 = easy, 1 = hard. 131

5.3 Perceived usability. 132
5.4 Overall reaction to PaperCAD. 133

Chapter 1

Introduction

While traditional computer interfaces based on the mouse and keyboard are ubiquitous, they are ill suited to many common application domains. This is particularly true in education where recent research [54, 59] suggests that students perform better when instructional interfaces are more similar to work practice. Thus, the goal of our work is to create computational techniques and user interface design principles to enable natural, pen-based tutoring systems that scaffold students in solving problems in the same way they would ordinarily solve them with paper and pencil.

Our focus is on interfaces suitable for "pentop computers," specifically the LeapFrog FLY [2] shown in Figure 1.1. This device is a writing instrument (pen) with an integrated digitizer and an embedded 96 MHZ ARC processor. It is used in conjunction with paper preprinted that is with a specially designed dot pattern. (The FLY is based on Anoto technology [1].) As one writes on this "digital paper," the digitizer uses the dot pattern to locate the pen tip on the page and digitize the pen stroke. Creating software that runs

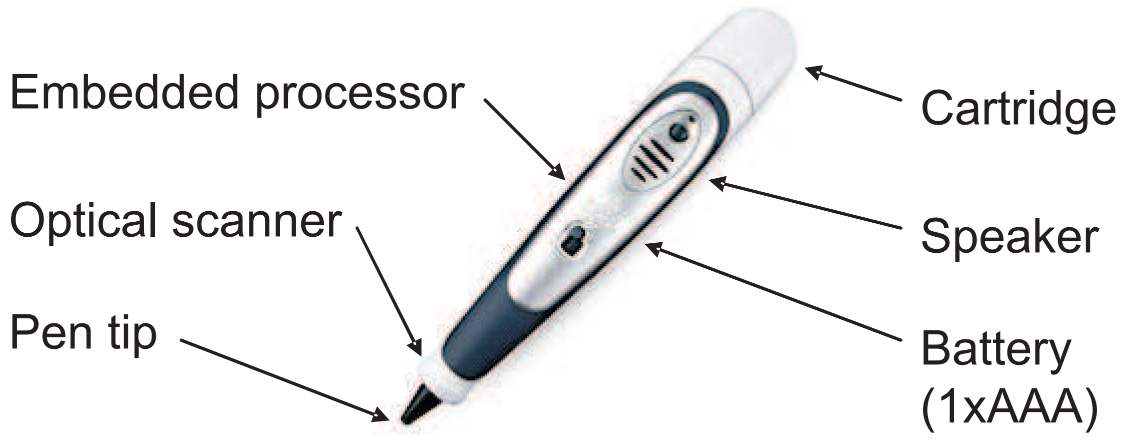


Figure 1.1: The FLY pentop computer.

directly on the FLY's embedded processor presents significant challenges because of the limited memory and computational power available. While there is ample read-only memory for code and static data, there is only about 4 kilobytes of RAM available for applications. The platform also presents substantial user interface design challenges because audio is the only form of dynamic output. The system has a speech synthesizer and can play recorded sound clips.

This dissertation encompasses three major projects: the development of a general purpose shape recognizer; the development of Newton's pen, a pentop-based tutoring system for statics; and the development of PaperCAD, a test bed for exploring user interface design issues for pentop computers.

1.1 Shape Recognizer

Accurate shape recognition is an essential foundation for pen-based interfaces. Researchers have developed a variety of approaches for recognizing hand-drawn shapes and symbols, but many of the current approaches have important limitations. For example, some are limited to single-stroke shapes drawn in preferred orientations [68]. Others consider only aggregate properties of a shape and can confuse dissimilar shapes that have similar aggregate properties [68, 18]. Other approaches require shapes to be drawn with a consistent pen stroke order [69].

Our work is aimed at overcoming some of these limitations. In particular, we developed a trainable, multi-stroke symbol recognizer that is insensitive to orientation, non-uniform scaling, and drawing order. Symbols are represented internally as attributed relational graphs describing both the geometry and topology of the symbols. Symbol definitions are statistical models, which makes the approach robust to variations common in hand-drawn shapes. Symbol recognition requires finding the definition symbol whose attributed relational graph best matches that of the unknown symbol. Much of the power of our approach derives from the particular set of attributes used, and our metrics for measuring similarity between graphs. One challenge addressed in the current work is how to perform the graph matching efficiently. We developed five approximate matching techniques: stochastic matching, which is based on stochastic search; error-driven matching, which uses local matching errors to drive the solution to an optimal match; greedy matching, which uses greedy search; hybrid matching, which uses exhaustive search for small problems and stochastic matching for larger ones; and sort matching, which relies on geometric information to accelerate the matching.

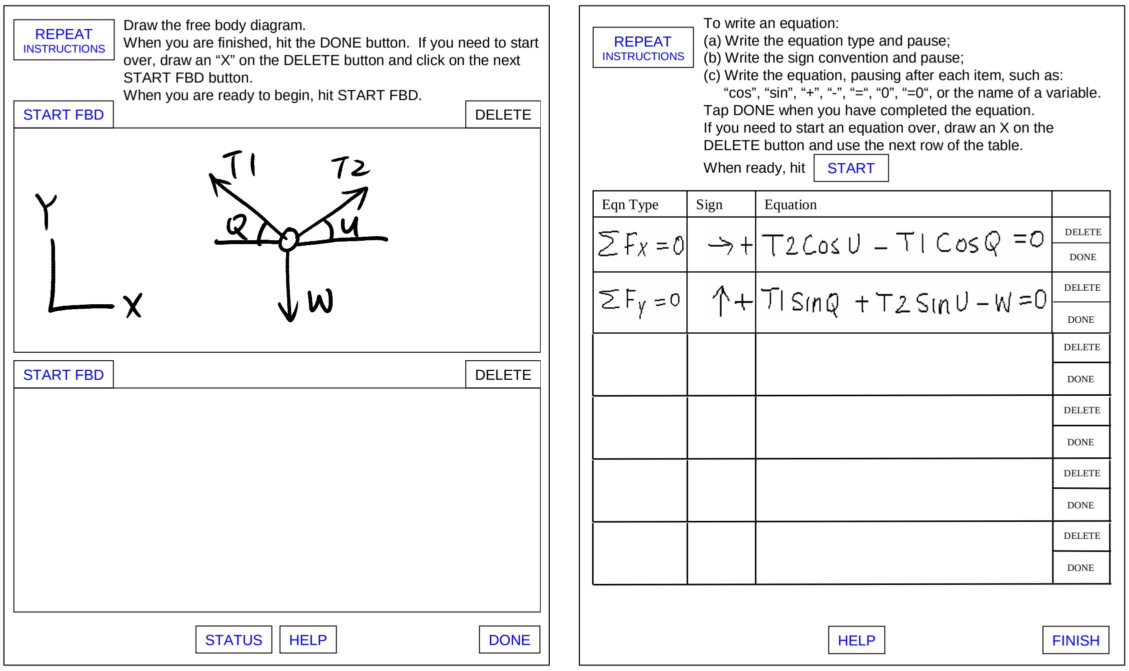
We conducted a user study to evaluate the performance of our graph-based recognizer. The results suggest that it is both fast and accurate enough for interactive applications. The study also revealed tradeoffs between our matching techniques. For example, for most applications, the greedy search technique has the best combination of speed and accuracy. However, when symbols have uniform orientation, sort matching may be preferable.

We developed our general-purpose, graph-based shape recognizer prior to selecting the pentop as the platform for our research. (We did not have access to software development tools for the FLY until after we completed the recognizer.) Although our recognizer is efficient and achieves interactive performance on tablet computers, it is not well suited to pentops with their limited computational power and memory. Instead we use light-weight, special-purpose recognizers for our pentop applications.

1.2 Newton's Pen

To explore instructional and interface design issues, we created Newton's Pen, a statics tutoring system designed for the FLY pentop. Statics is the sub-discipline of engineering mechanics concerned with the equilibrium of objects subjected to forces. Newton's Pen scaffolds students in the construction of free body diagrams and equilibrium equations. The system interprets the student's hand-written solutions to equilibrium problems and provides context sensitive tutorial feedback.

Problems are solved on digital paper preprinted with user interface objects, such


REPEAT INSTRUCTIONS

Draw the free body diagram.
When you are finished, hit the DONE button. If you need to start over, draw an "X" on the DELETE button and click on the next START FBD button.
When you are ready to begin, hit START FBD.

REPEAT INSTRUCTIONS

To write an equation:
(a) Write the equation type and pause;
(b) Write the sign convention and pause;
(c) Write the equation, pausing after each item, such as:
"cos", "sin", "+", "-", "=", "0", "=0", or the name of a variable.
Tap DONE when you have completed the equation.
If you need to start an equation over, draw an X on the DELETE button and use the next row of the table.
When ready, hit START

Eqn Type	Sign	Equation	
$\Sigma F_x = 0$	$\rightarrow +$	$T2 \cos U - T1 \cos Q = 0$	DELETE DONE
$\Sigma F_y = 0$	$\uparrow +$	$T1 \sin Q + T2 \sin U - W = 0$	DELETE DONE
			DELETE DONE
			DELETE DONE
			DELETE DONE
			DELETE DONE

(a) (b)

Figure 1.2: (a) Worksheet for drawing free body diagrams. (b) Worksheet for writing equilibrium equations.

as "HELP" and "DONE" buttons. Figure 1.2(a) shows a worksheet for drawing a free body diagram. To begin, the student taps the pen on the "START FBD" button, and draws the free body diagram in the space provided. After each graphical element is drawn, the system provides interpretive audio feedback. If the student makes a problem-solving error, or the input is not recognized, the system informs the student via synthesized speech. The student can tap "HELP" at any time for audio hints about what to do next or for guidance after an error. Once the student has correctly completed the free body diagram, he or she hits the "DONE" button, and then writes the equilibrium equations on the worksheet in Figure 1.2(b). The system interprets the equations and provides tutorial feedback, guiding the student to the correct solution.

Newton's Pen employs an instructional model based on a structured solution process with feedback provided at each problem-solving step. For example, when drawing a free body diagram, the student must first draw a coordinate system, then the body, and finally the forces. This has proven to be an effective instructional design. This model, which is designed to match the pentop's unusual user interface capabilities, is implemented using a finite state machine architecture. Each state contains a list of legal inputs (i.e., things the user can write or draw), a list of next states, and help messages.

This architecture has a number of important benefits. First, it provides a convenient method for providing context sensitive help. Because the program's state mirrors the student's problem-solving progress, the help messages associated with a state are always relevant. Second, the architecture allows new problems to be easily coded. A new problem is created by instantiating the appropriate set of states. Third, the architecture allows the system to be run in a tutorial mode that guides the student in the use of the system. Each state contains an explanation of what should be drawn in that state. In tutorial mode, when a state is entered, the explanation is announced to the student.

We conducted three user studies to evaluate the usability and educational value of Newton's Pen. The studies consistently suggested that Newton's Pen was an effective tutoring system. In all three studies, students were initially unable to complete a pretest, but after about an hour with the system, were able to successfully complete a new transfer problem as well as the original pretest. Survey results suggested that students perceive both the instructional and user interface designs to be effective.

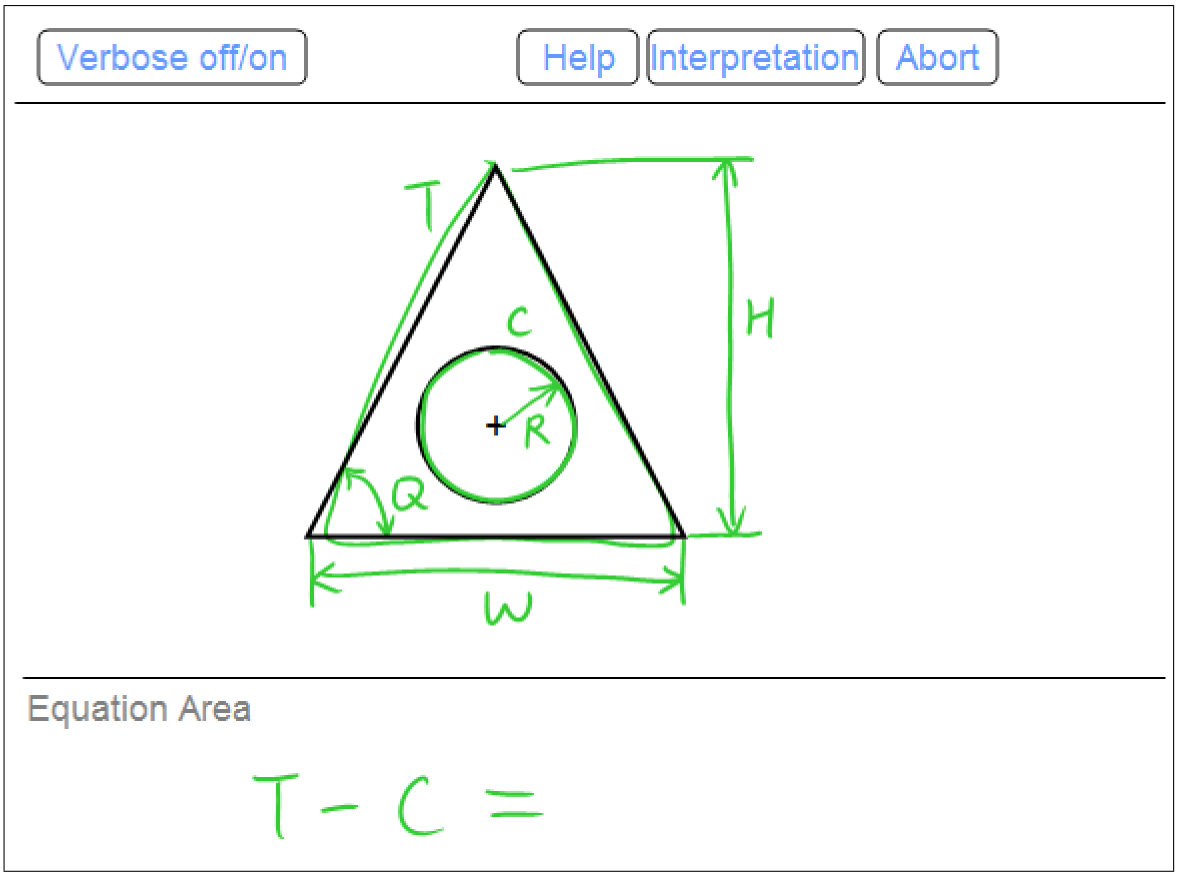


Figure 1.3: A drawing formatted for PaperCAD.

1.3 PaperCAD

On pen-based hardware with a dynamic display, such as a tablet computer, the results of shape recognition and diagram interpretation can be displayed with color coding, text labels, and the like. On a pentop platform, feedback must be provided via audio. Consequently, Newton's pen employs a "draw-pause-interpret" model in which the user must pause after drawing a graphical object to wait for the system to provide interpretive feedback. While users of the system were generally pleased with the interface, many expressed the desire for faster interaction. Our PaperCAD project explores improved interaction methods in the context of a pentop interface to computer-aided design (CAD) models.

PaperCAD enables users to query geometric information from CAD drawings printed on digital paper. Figure 1.3 shows an example of a simple drawing formatted for use with the system. The user measures dimensions by drawing conventional dimension lines (arrows), such as those used to dimension the triangle in Figure 1.3. As each dimension is drawn, the system announces its value with synthesized speech. The values are obtained by querying a digital model of the drawing, and are not scaled from the paper drawing. A symbolic label is associated with each dimension. These can be used to construct equations to compute various geometric quantities.

PaperCAD employs both verbose and concise audio modes. In the verbose mode, the system provides all feedback via synthesized speech. In concise mode, the system uses short tones to indicate whether or not objects have been successfully recognized, and uses synthesized speech only to report the values of dimensions. By using tones rather than words, the concise mode results in faster feedback. We also augmented the FLY's output capabilities by using a PDA as a dynamic video display. After the user draws on the paper drawing, the beautified ink and CAD drawing in the neighborhood of the pen stroke are displayed on the PDA. To view a different part of the drawing, the user taps the FLY at the desired location on the paper.

In creating PaperCAD, we also addressed several important problems related to interpreting hand-drawn input. First, we developed a recognizer for dimension arrows capable of recognizing arrows with a wide range of aspect ratios. Second, we developed a method that uses contextual information to interpret ambiguous dimensions. Third, we developed a technique that uses a hidden Markov model to automatically correct interpretation errors in hand-written equations. Fourth, we created a prototyping environment that accelerates the development process for the FLY by enabling code to be compiled, executed, and debugged on a PC.

We conducted a user study to evaluate PaperCAD's usability. The results suggest that users are highly satisfied with the interface and prefer it to a traditional WIMP (windows, icons, menus, and pointers) interface.

1.4 Organization of the Dissertation

The rest of the dissertation is organized as follows. The next chapter reviews related work, providing background for this dissertation. This is followed by the details of the three systems we built: our graph-based recognizer, Newton's Pen, and PaperCAD. Finally, conclusions are presented.

Chapter 2

Literature Review

This thesis touches upon several research areas including shape and symbol recognition, intelligent tutoring systems, pen-based and paper-based interface design, and equation interpretation. This chapter reviews related work in these areas.

2.1 Shape and Symbol Recognition

Shape and symbol recognition is an active area of research. An extensive overview of the literature can be found in [48]. Here, we present a representative sample of the literature.

2.1.1 Graph-based Recognizers

Lee [43] presents a graph-based recognizer in which the graph represents the precise geometry of the object. The approach is suitable for precisely drawn symbols with uniform scaling. For example, the approach has been used to recognize machine drawn symbols,

symbols drawn using templates, and precise hand-drawn symbols. Lee's approach requires manual selection of key vertices during training.

Mahoney and Fromherz [51] present a technique for matching models of curvilinear configurations to hand-drawn sketches. The technique is intended for matching articulated figures rather than sketches with a more fixed shape. The approach has been implemented only for human stick figures and requires hand-coded shape definitions.

Rocha [66] uses graph matching for optical character recognition (OCR). The matching technique assumes that the characters have a fixed orientation and no disconnected parts. The approach relies on hand-coded definitions and has not been applied to hand-drawn shapes.

Lladós *et al.* [47] present an error-tolerant subgraph matching technique for matching region adjacency graphs. The technique is intended for recognizing symbols composed of adjacent polygonal regions, rather than general arrangements of low-level primitives.

Calhoun *et al.* [11] developed a graph-based approach that considered both topology and geometry, but our approach differs in several important respects. With their approach, a definition is an average graph, with each attribute represented by a single value. We describe attributes statistically, making our approach significantly more robust to pen stroke segmentation errors and drawing variations. Their approach is incapable of representing symbols with disconnected parts, while ours can. Furthermore, to achieve interactive performance, their approach requires the user to maintain a consistent drawing order. Their system does have a mode that allows variable drawing order, but it is considerably less efficient than our approach. Similarly, their approach requires the training examples to have a consistent drawing order and pen-stroke direction, and is intolerant of segmentation errors. Our approach overcomes these limitations.

In addition to symbol recognition, graph-based techniques have been used for a variety of other pattern recognition problems. Conte *et al.* [13] provides an extensive overview of graph matching techniques and their applications. According to the taxonomy presented there, our stochastic, error-driven, and greedy matching techniques can be considered approximate matching techniques based on continuous optimization.

2.1.2 Feature-based Recognizers

Many existing approaches to symbol recognition rely on feature-based representations. Fonseca *et al.* [18] use features such as the smallest convex hull that can be circumscribed around the shape, the largest triangle that can be inscribed in the hull, and the largest quadrilateral that can be inscribed. Because their classification relies on aggregate features of the pen strokes, it might be difficult to differentiate between similar shapes.

Rubine [68] describes a trainable gesture recognizer designed for gesture-based interfaces. The recognizer is applicable only to single-stroke symbols, and is sensitive to the drawing direction and orientation. Pereira *et al.* [60] have extended Rubine's method to multi-stroke symbols. However, each symbol must be drawn with a consistent set of strokes. Additionally, they have developed a graph-based symbol recognizer, but it is not trainable. Matsakis [53] describes a system for converting handwritten mathematical expressions into a machine-interpretable typesetting command language. Each symbol requires a multitude of training examples, where each example must be preprocessed to eliminate variations in drawing directions and stroke orderings. However, the preprocessing makes their approach sensitive to rotations. Gennari *et al.* [21] describe a trainable recognizer that uses nine geometric features to construct concise probabilistic models of input symbols. The approach is suitable for multi-stroke symbols with arbitrary drawing orders and orientations. However, the features are an abstraction of the topology, and it is possible for shapes with different topologies to have the same features.

Hse and Newton [28] developed a particularly accurate recognizer based on Zernike moments, which provide a rotation invariant representation. However, a preprocessing step in which the image size is normalized makes the approach sensitive to orientation. The preprocessing results in insensitivity to some forms of non-uniform scaling, but not cases in which different parts of a symbol are scaled differently (e.g., the circle in a pivot is drawn large, while the triangle is drawn small).

2.1.3 Recognizers Using Other Techniques

In addition to graph-based and feature-based methods, researchers have also explored a variety of other representations and approaches. For example, Sezgin and Davis [69] present a technique based on hidden Markov models. The approach requires shapes to be drawn with a consistent pen stroke ordering. Hammond and Davis [26] developed a recognizer that relies on hand-coded shape descriptions. Their representation is similar to ours in that both contain topological information. However, their descriptions are hand-coded while ours are learned from training examples. In later work [27], they extended their approach to use hand-drawn and machine-generated examples to assist the developer in interactively creating shape descriptions. Shilman *et al.* [70] present a sketch recognition approach that requires a manually encoded visual grammar. A large corpus of training examples is used to learn the statistical distributions of the geometric parameters used in the grammar, resulting in a statistical model. Composite objects are defined hierarchically in terms of lower-level, single-stroke symbols, which are recognized using Rubine's method [68].

Gross' [23] approach relies on a 3x3 grid inscribed in the symbol's bounding box. The sequence of grid cells visited by the pen distinguishes each symbol. Because of the coarse resolution of a 3x3 grid, this approach may not be able to handle symbols with small features. Kara and Stahovich [38] developed a recognizer based on a bitmap representation. One advantage of the approach is that it is tolerant of over-stroking and variations in line styles. However, the approach is sensitive to non-uniform scaling.

Parametric methods such as polygon, B-spline, and Bezier curve fitting techniques have also been considered in shape representation and classification [30, 65]. A benefit of these approaches is that there is no need to segment the pen stroke into geometric primitives such as lines and arcs. Additionally, since only a few parameters are needed for shape description, these methods are computationally efficient. Similar to Rubine's method, however, these methods are primarily applicable to single-stroke symbols or gestural commands.

2.2 Intelligent Tutoring Systems

Intelligent tutoring systems have been developed for a wide variety of domains, such as: medicine [75], law [73], IT security [29], SQL [57], computer programming [17], algebra [76], electric circuits [8, 10, 14], free body diagrams [67], and physics [78]. Nearly all of these systems are based on WIMP (windows, icons, menus, and pointer) or keyboard interfaces. Our work, by contrast, is focused on building pedagogically-sound, pen-based interfaces for tutoring systems. While conventional tutoring systems work from unambiguous input provided with a keyboard and mouse, we focus on the challenges of working from ambiguous, hand-drawn input.

While sketch understanding techniques have been used for a wide range of applications, the impact on tutoring systems has been limited. [6] describes a classroom interaction system that allows students and instructors to communicate wirelessly in lecture environments using tablet computers. However, this system does not interpret what is written, nor is it intended to provide tutoring capabilities.

Researchers have recently begun to explore issues related to the development of pen-based instructional tools. For example, Oviatt [59] compared student performance using paper and pencil, an Anoto [1] digital pen (a pen that digitizes ink but does not run embedded applications), and a Tablet PC. Students used these platforms solely as a recording medium for problem solving; no tutoring capabilities were provided. The work demonstrated that student performance is better when the computer interface is more similar to familiar work practice. This speaks to the importance of good user interface design in creating effective educational systems. Anthony *et al.* [7] describes a related study aimed at comparing various methods for entering mathematical equations into a computer. The goal was to explore interface issues for tutoring systems. Although the user input was not interpreted, the study suggests that pen-input of equations is substantially more efficient than keyboard entry, and is greatly preferred by users.

2.3 Pen-based User Interfaces

Research on pen-based interfaces is quite active at present. Examples of existing experimental applications include: tools for simulating hand-drawn mechanical devices [4, 35], a tool for sketching user interfaces [42], a UML diagram tool [25], tools for interpreting hand-drawn equations [53, 71], a tool for understanding military tactics [19], circuit design and analysis tools [58, 22, 21], a control system analysis tool [36], and tools for sketching 3D shapes [80, 52]. For the most part, this work is focused on algorithms for interpreting sketches and diagrams. Our work differs in that we address the challenges of user interface design for a visually static medium, namely ink on paper.

In addition to recognition-based techniques, researchers have also developed a variety of text input and menu selection techniques for pen-based interfaces. For example, Quikwriting [61] enables the user to enter text by continuous movements of the stylus within a 3x3 grid centered on the pen-down location. Marking Menus [41, 40] are on-the-spot radial pop-up menus that combine object and verb selection in a single pen stroke. For example, a group of files can be selected and then deleted without lifting the stylus. Marking Menus support both novice and expert modes. In the novice mode, a radial menu is displayed when the pen tip is held down for a brief period of time, and a menu item is selected by drawing a “mark” (a line) through it. In the expert mode, a mark can be drawn immediately, before the menu pops up. Control Menus [63] extend Marking Menus to support operations with parameters. For example, immediately after a zoom menu item is selected, moving the pen to the right or left changes the zoom level. FlowMenu [24] extends Marking Menus and combines them with Quikwriting. Not only does FlowMenu combine object and verb selection into a single stroke gesture, but also allows a sequence of verb selections, followed by text entry using Quikwriting.

2.4 Paper-based User Interfaces

XAX [31] is an example of an early paper-based interface. It uses optical character recognition to interpret commands written on the coversheet of a document with an ordinary pen. For example, instructions for faxing a document can be placed on a cover sheet, and the system can fax and scan the document without additional input from the user.

Ariel [49] was an early attempt to introduce computational capabilities into paper engineer drawings. It employs a WIMP (Window, Icon, Menu, and Pointer) interface projected onto the drawing with a video projector. A light pen is used only as a pointer.

Several recent research efforts have aimed at using pentop technology to build paper-based applications. The PapierCraft [45] system enables users to edit documents by writing on paper printouts with an Anoto [1] digital pen. Annotations and command gestures are captured by the digital pen, and are then uploaded to a PC to be interpreted and executed on a digital version of the document. ModelCraft [72] is used to apply annotations and edits to a 3D geometric model constructed from dot-patterned paper. Again, ink is captured with Anoto pens, and is processed offline when the pen is docked. PaperCAD, by contrast, is an interactive paper-based interface.

The NISMap system [12] is a military planning tool, also based on the Anoto digital pen. As the user annotates a paper map, digitized pen strokes are wirelessly transmitted to a PC for processing. The pen input is processed in real time, but is done so on a traditional computer. Liao *et al.* [46] have begun exploring methods of providing real-time user feedback directly from a pentop using LEDs, voice coils, and audio speakers. These techniques were evaluated in laboratory experiments, rather than software applications.

Recent work has explored methods of integrating paper-based interfaces with digital devices. For example, ButterflyNet [79] enables field biologists to link digital photos to paper notebooks by drawing a gesture. The system relies on an Anoto Pen wirelessly connected to a digital camera. Similarly, PocketPad [3] enables users to annotate digital documents displayed on a PDA by writing with an Anoto pen and paper. These systems do little recognition of the pen input, and the final document is produced only when the digital device and pen are docked with a computer.

The A-book system [50] introduced the concept of an "interaction lens," a PDA placed over a paper document to provide a digital image of the contents. A graphical interface on the PDA is used to manipulate the digital model of the document. While A-book relies on a dynamic video display for interaction, our PaperCAD system works directly from the paper, and requires only audio output.

2.5 Equation Recognition, Interpretation and Correction

Previous approaches to correcting interpretation errors in equations have relied on heuristics. For example, heuristics in $MathPad^2$ [32] can replace $5in$ with sin. Kirchhoff's Pen [14] employs a similar approach based on domain knowledge for electrical circuit equations. Our HMM technique in PaperCAD provides a more principled approach to error correction in that it is based on a grammar and statistics about recognition accuracy.

HMMs have been used for other sketch interpretation tasks. For example, Sezgin and Davis [69] use an HMM approach to recognize objects in hand-drawn sketches. A separate HMM is trained to recognize each kind of shape in the domain. A global optimization of the probability is used to determine the overall interpretation of a sketch. Kosmala *et al.* [39] describe a similar approach for recognizing mathematical expressions. Their approach, however, does not make use of a grammar to improve recognition accuracy.

Chapter 3

An Efficient Graph-Based Recognizer for Hand-Drawn Symbols

3.1 Introduction

Researchers have developed a variety of approaches for recognizing hand-drawn shapes and symbols. However, many of the current approaches have important limitations. For example, some methods are limited to single-stroke shapes drawn in preferred orientations [68]. Others consider only aggregate properties of a shape and can confuse dissimilar shapes that have similar aggregate properties [68, 18]. Other approaches require shapes to be drawn with a consistent pen stroke order [69].

Our work is aimed at overcoming some of these limitations. Our goal is to create an efficient, trainable, multi-stroke symbol recognizer that is insensitive to orientation, scaling, and drawing order. This is achieved via a graphical representation. Specifically, a symbol is represented with an attributed relational graph (ARG) describing its geometry and topology. The nodes in the graph represent the geometric primitives, and the edges represent the geometric relationships between them. Representing a symbol in terms of its topology allows us to achieve invariance to rotation, and uniform and non-uniform scaling, including cases in which different parts of the same symbol are scaled differently. Because of the later capability, our approach is particularly tolerant of large variations in the shape of a hand-drawn symbol. This is an important advantage over approaches such as those presented in [28, 68, 38, 37].

Symbol definitions are statistical models learned from training examples of a symbol. Using statistical descriptions makes the approach robust to variations common in hand-drawn shapes. Symbol recognition involves finding the definition symbol whose ARG best matches that of the unknown symbol. Much of the power of our approach derives from the particular set of attributes encoded in the ARGs, and our metrics for measuring similarity between two ARGs.

With our approach, symbol recognition is a graph matching or "graph isomorphism detection" [55] problem. The general problem of subgraph isomorphism detection [77, 15] is known to be NP-complete [20]. Here, the problem is made more difficult because of noise that comes from variations in how the symbols are drawn as well as from processing errors. For example, it is not uncommon for a symbol to have extra or missing geometric primitives, and thus extra or missing nodes in its ARG.

There has been considerable research in developing efficient graph matching techniques for a variety of applications [13]. We have created and evaluated five techniques specifically designed for recognizing hand-drawn shapes. These techniques are designed to be efficient enough for interactive performance, and to be tolerant of the noise inherent in hand-drawn symbols. Furthermore, some of these techniques are able to take advantage of consistent drawing order to achieve higher efficiency.

Our recognizer assumes that the individual symbols in a sketch have been located prior to recognition. Techniques such as those in [36, 21] can be used for this purpose.

The remainder of the chapter contains the details of our approach, results of a user study, and a discussion of the user study results, including a discussion of the tradeoffs between our five graph-matching techniques.

3.2 Representation

We represent a symbol with an attributed relational graph (ARG) describing its geometry and topology. The nodes in the graph represent the geometric primitives, and the edges represent the geometric relationships between them.

Each node is characterized by the type of the primitive – line or arc – and its relative length. The primitives are obtained from the raw pen strokes via a speed-based pen stroke segmenter [74]. The relative length of a primitive is defined as the ratio of its length (in pixels) to the total length of the primitives comprising the symbol. For example, each of the four line segments in a perfect square would have a relative length of 0.25. Defining length on a relative basis results in a scale-independent recognizer.

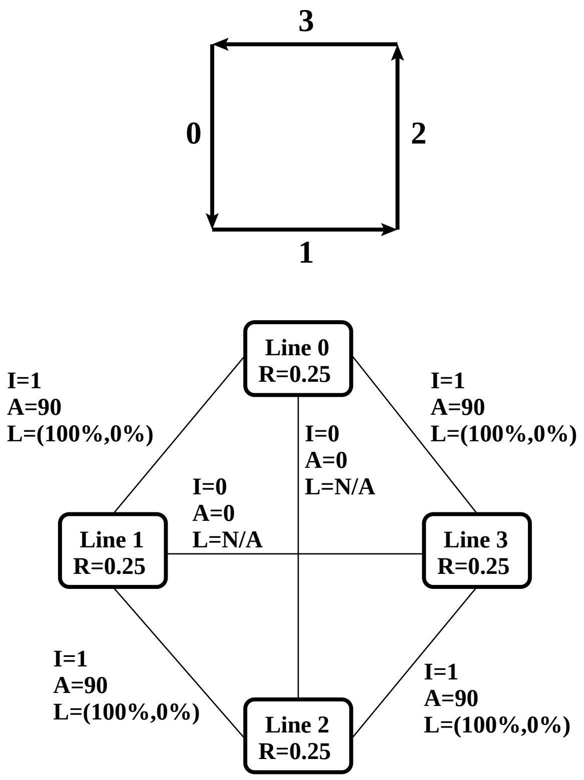


Figure 3.1: Top: An ideal square drawn with a single, counter-clockwise pen stroke. Arrows show the direction of drawing. Bottom: The corresponding ARG. I = number of intersections, A = intersection angle, L = intersection location, R = relative length.

The edges in a graph represent the geometric relationships between the primitives. Each pair of primitives is characterized by the number of intersections between them, the relative locations of the intersections, and for lines, the angle of intersection. When extracting intersections from a sketch, a tolerance of 10% of the length of the primitives is used to allow for cases in which an intersection was intended but one of the primitives was a little too short. Intersection locations are measured relative to the lengths of the two primitives. For example, if the beginning of one line segment intersects the middle of another, the location is described by the coordinates (0%, 50%). The intersection angle is defined as the acute angle between two line segments. It is defined for both intersecting and non-intersecting line segments. Defining an intersection angle for non-intersecting segments allows the program to represent the topology of disconnected symbols, such as the dashpot in Figure 3.12. Intersection angle is undefined for an intersection between an arc and another primitive.

Figure 3.1 shows an example of an ARG for an ideal square. Each side of the square has a relative length of 0.25 and intersects two other sides with an intersection angle of 90°. Because of the drawing directions used in this example, all intersections are located at the end of one line segment and the beginning of another.

A definition for a symbol is created by constructing an "average" ARG from a set of training examples. (Additional details of the training process are described in Section 3.5.) The number of nodes in a definition is taken to be the most frequently occurring number of nodes in the training examples. Each node in the definition is assigned the primitive type that occurred most frequently for that node in the training data. The number of intersections assigned to a pair of primitives is determined in an analogous fashion. A pair of primitives is assigned two intersections if at least 70% of the examples had two. If less than 70% had two intersections, but there was at least one intersection 70% of the time, the pair is assigned one. Otherwise, the pair is assigned zero intersections. The remaining properties of the ARG – relative length, intersection angle, and intersection location – are continuous valued properties. These are characterized by the means and standard deviations of the values from the training examples.

3.3 Measuring Similarity

During recognition, it is necessary to compare the ARG of the unknown symbol to that of each definition symbol to find the best match, and hence the classification of the unknown. The match between an unknown and a definition is quantified in terms of a dissimilarity score, which is computed using an ensemble of error metrics that consider both the intrinsic properties of the geometric primitives and the relationships between them. The former are encoded in the nodes of the ARG, the latter in the edges.

Table 3.1 lists our six error metrics and the weights applied to them when computing the dissimilarity score. The weights, which are based on empirical studies, reflect the relative importance of the various error metrics for discriminating between symbols. For the purposes of recognition, the dissimilarity score is converted to a *Similarity Score* in the obvious way:

$$\textit{Similarity Score} = 1 - \sum_{i=1}^{6} w_i E_i \tag{3.1}$$

where the E_i are the error metrics and the w_i are the weights listed in Table 3.1.

Error Metrics (E_i)	**Weight** (w_i)
E_1: Primitive count error	20%
E_2: Primitive type error	20%
E_3: Relative length error	20%
E_4: Number of intersections error	15%
E_5: Intersection angle error	15%
E_6: Intersection location error	10%

Table 3.1: Error metrics and corresponding weights.

The error metrics for relative length, intersection angle, and intersection location involve comparing properties of the unknown symbol to distributions of those properties encoded in a definition. For example, it is necessary to compare the relative length of each primitive in the unknown to the mean and standard deviation of the relative length of the corresponding primitive in the definition. Ordinarily, this is done with a Gaussian probability density function. As an alternative, we have developed a modified probability density function (MPDF) that is better suited to our recognition task:

$$P(x) = exp[-\frac{1}{50.0} \cdot \frac{(x-\mu)^4}{\sigma^4}] \tag{3.2}$$

Here, μ and σ are the mean and standard deviation of the features from the training examples. This function was designed empirically such that its top is flatter than the Gaussian probability density function for the same μ and σ. This makes it easier to detect matches that are in the "vicinity" of the definition. Additionally, we have found that the Gaussian distribution dies off too quickly towards its tails, which decreases its usefulness for recognition. For comparison, Figure 3.2 shows both the Gaussian probability density function and our modified probability density function for $\mu = 0$ and $\sigma = 1$.

The six error metrics used for computing the similarity score are described in the following sections. Here we use the term "unknown" to refer to the symbol to be recognized, or equivalently, the ARG of that symbol. Likewise, the term "definition" refers to the ARG of a definition symbol. Note also that each metric is normalized to the range [0, 1] so that the weights in Table 3.1 have predictable influences.

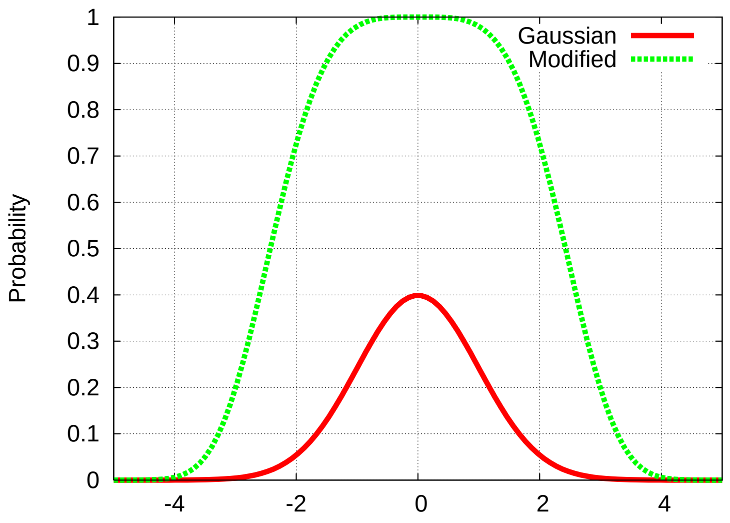


Figure 3.2: Gaussian probability density function and modified probability density function for $\mu = 0$ and $\sigma = 1$.

3.3.1 Primitive Count Error

The primitive count error is the difference between the number of nodes in the unknown and definition ARGs, normalized by the minimum number of nodes:

$$E_1 = min(1.0, \frac{|N_U - N_D|}{N_{min}}) \tag{3.3}$$

Here, N_U and N_D are the numbers of nodes in the unknown and definition ARGs, respectively, and $N_{min} = min(N_U, N_D)$. We normalize by N_{min} to quantify the significance of the mismatch in the primitive count. The fewer primitives there are, the more significant a given mismatch is. The error saturates at one so that all errors have the same range of [0, 1].

3.3.2 Primitive Type Error

The primitive type error is the number of node pairs with mismatched types, normalized by the minimum number of nodes:

$$E_2 = \frac{\sum_{i=1}^{N_{min}} [1 - \delta(Type(U_i), Type(D_i))]}{N_{min}} \tag{3.4}$$

Here, U_i is a node from the unknown, D_i is the corresponding node from the definition, $Type(X)$ is a function that returns the primitive type (arc or line) of node X, and $\delta(p, q)$ is one when $p = q$, and zero otherwise.

3.3.3 Relative Length Error

Each primitive from the unknown should have a relative length similar to that of the corresponding primitive from the definition. If not, an error is assigned. Here, similarity is measured using the MPDF defined in Equation 3.2. The error is computed as:

$$E_3 = \frac{\sum_{i=1}^{N_{min}} [1 - P(R(U_i))]}{N_{min}} \tag{3.5}$$

where $R(U_i)$ is the relative length encoded in node U_i of the unknown ARG, and $P(x)$ is evaluated using the mean and standard deviation from the corresponding node in the definition. Note that whereas $P(x)$ is the probability of match, $1 - P(x)$ is the probability of mismatch.

3.3.4 Number of Intersections Error

A pair of primitives in the unknown should have the same number of intersections as the corresponding pair in the definition. If not, an error is assigned. The total error is computed as:

$$E'_4 = \frac{\sum_{i=1}^{N_{min}-1} \sum_{j=i+1}^{N_{min}} |I(U_i, U_j) - I(D_i, D_j)|}{min(M_U, M_D)} \tag{3.6}$$

where $I(X, Y)$ is the number of intersections between the primitives in nodes X and Y, and M_U and M_D are the numbers of edges in the unknown and definition ARGs, respectively. This error is normalized by the number of potentially intersecting pairs of primitives. However, because a pair of primitives can intersect as many as two times, E'_4 has a range of [0, 2]. So that all error metrics have the same range of [0, 1], the value of E'_4 is "squashed" with:

$$S(x) = \frac{1}{1 + exp[6(1 - x)]} \tag{3.7}$$

This squash function, which is shown in Figure 3.3, was chosen such that small differences are attenuated while larger ones are preserved. During our experiments, we found that this choice provides better performance compared to a linear squashing function. As a result, the "Number of Intersections Error" is defined as:

$$E_4 = S(E'_4) \tag{3.8}$$

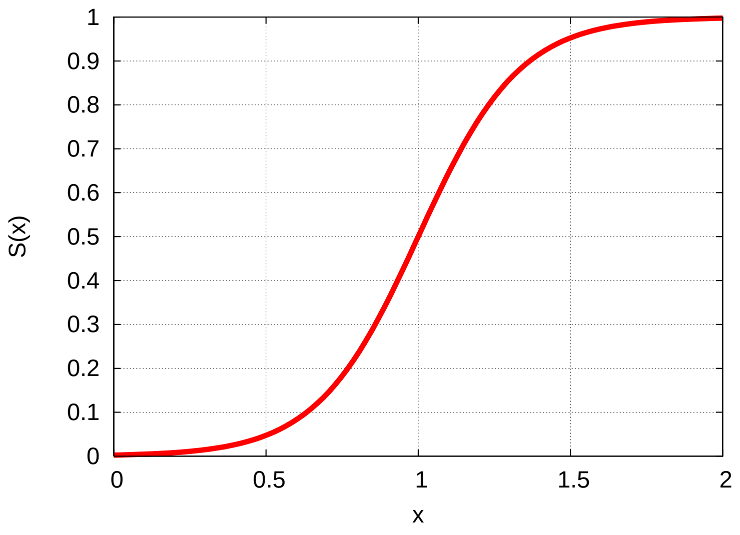


Figure 3.3: The squash function from Equation 3.7.

3.3.5 Intersection Angle Error

The intersection angle of a pair of lines in the unknown should be similar to that of the corresponding pair of lines in the definition. (Intersection angle is defined only for pairs of lines.) If not, an error is assigned. Here, similarity is again measured using the MPDF defined in Equation 3.2. The error is computed as the sum of the intersection angle errors normalized by the number of line pairs the unknown and definition have in common:

$$E_5 = \frac{\sum_{i=1}^{N_{min}-1} \sum_{j=i+1}^{N_{min}} [1 - P(A_{ij})]}{\sum_{i=1}^{N_{min}-1} \sum_{j=i+1}^{N_{min}} Lines(U_i, U_j, D_i, D_j)} \tag{3.9}$$

Here, A_{ij} is the angle at which the primitive from node i of the unknown intersects the primitive from node j of the unknown. $P(A_{ij})$ is evaluated using the mean and standard deviation from the corresponding pair of primitives from the definition. Note that if the two primitives are not lines, A_{ij} is undefined and $P(A_{ij})$ is taken to be one. $Lines(U_i, U_j, D_i, D_j)$ is one when all of the arguments are nodes representing lines, and zero otherwise.

3.3.6 Intersection Location Error

The locations of the intersections between a pair of primitives from the unknown should be similar to those of the corresponding pair of primitives from the definition. If not, an error is assigned. Here, similarity is again measured using the MPDF defined in Equation 3.2. Because intersection location is defined by two coordinates, the MPDF is applied twice for each intersection. The total error is computed as:

$$E_6 = \frac{\sum_{i=1}^{N_{min}-1} \sum_{j=i+1}^{N_{min}} \sum_{k=1}^{I(D_i,D_j)} \left([1 - P(L_i^k)] + [1 - P(L_j^k)]\right)}{\sum_{i=1}^{N_{min}-1} \sum_{j=i+1}^{N_{min}} 2 \cdot I(D_i, D_j)} \tag{3.10}$$

where (L_i^k, L_j^k) is the coordinates of the k^{th} intersection between the primitives from nodes i and j of the unknown. $I(D_i, D_j)$ is the number of intersections between the primitives from nodes i and j of the definition. In cases where a pair of primitives intersect in the unknown but not in the definition, or vice versa, both $P(L_i^k)$ and $P(L_j^k)$ are set to zero. This error is normalized by twice the number of intersections, as two coordinates can contribute error to each intersection.

3.4 Graph Matching

The previous section described how to compute the similarity between two graphs. This assumed that each node in the unknown ARG was assigned to a specific node in the definition ARG. This section describes how these assignments are obtained. This is a graph matching, or graph isomorphism problem. If the user always draws each symbol with a consistent number of primitives and a consistent drawing order, the graph matching problem is trivial. In this case, drawing order would directly provide the correct node-pair assignments. In practice, however, users do not always maintain a consistent drawing order. Furthermore, the problem is made more difficult because of noise. Noise comes from variations in how the symbols are drawn as well as from processing errors. For example, it is not uncommon for there to be extra or missing nodes in the unknown (*i.e.*, extra or missing geometric primitives). Likewise, a primitive that was intended to be a line can be misinterpreted, either through ambiguity or processing errors, as an arc, or vice versa.

We have developed five efficient, approximate matching techniques to find the best match between two ARGs. These are: stochastic matching, error-driven matching, greedy matching, hybrid matching, and sort matching. The first four methods are based on search. The fifth method avoids search by assuming symbols are drawn with a consistent orientation.

The search-based methods make initial node-pair assignments based on drawing order. Assignments are then swapped until the best match is obtained. The quality of the match at each iteration is determined using the similarity score defined in the previous section (Equation 3.1). Our four search-based approaches differ in the way they select the assignments to swap at each iteration.

If the two graphs being matched do not have the same number of nodes, the smaller one is “padded” with empty nodes. This ensures that every node in one graph has a match with a unique node in the other, and hence that every node is considered by the swapping process. When evaluating the error metrics, a pairing with an empty node produces the maximum possible local error. For example, the addition of empty nodes does not reduce the primitive count error, E_1.

Figure 3.4 illustrates the typical search process. For ease of explanation, the figure shows hypothetical symbols rather than ARGs. Finding the correct node-pair assignments is equivalent to finding the correct assignment of the primitives of the unknown to the primitives of the definition. Here, the primitives of the definition symbol are numbered according to a typical drawing order. Likewise, the primitives of the unknown are labeled with letters indicating the order in which they were actually drawn. Based on drawing order, primitive **a** of the unknown is initially assigned to primitive **1** of the definition, **b** is assigned to **2**, and so on. It is clear that assignments **b-2** and **c-3** are correct, while **a-1** and **d-4** are not. Swapping the latter to produce the assignments **d-1** and **a-4** is what is needed. The success of this swap can be measured by the resulting increase in the similarity score.

The following sections describe our five matching techniques in detail.

3.4.1 Stochastic Matching

This approach is based on stochastic search. To begin, the initial node-pair assignments are saved as the current best. Then, three node-pair assignments, which we will

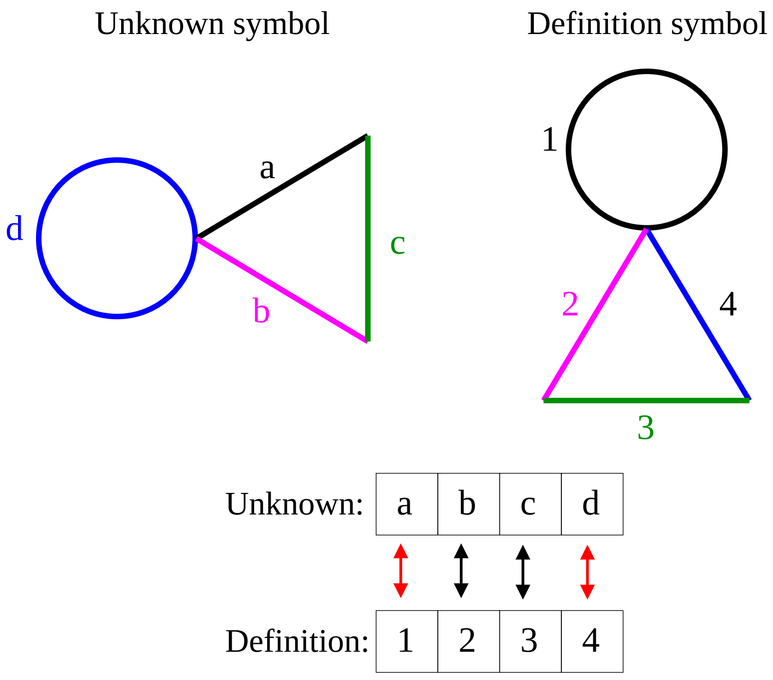


Figure 3.4: Graph matching: assignments **b-2** and **c-3** are correct, while **a-1** and **d-4** are not.

call A, B, and C, are randomly selected. A and B are swapped producing assignments A' and B'. B' is then swapped with C. If the new similarity score is better than the current best score, the new assignments are saved as the new current best. This process is repeated a fixed number of times, and the current best node-pair assignments are returned as the best match. In practice, we use a limit of between 100 and 300 iterations. As the number of iterations is fixed (but adjustable), the only cost that varies with problem size is the cost of evaluating the similarity score. This cost is $O(n^2)$, where n is the number of nodes. Pseudo code for this matcher is shown in Figure 3.5.

```
Stochastic_Matcher(unknown, def)
  current_assignment = assign_using_drawing_order(unknown, def)
  best_assignment = current_assignment
  best_score = similarity_score(best_assignment)
  for i = 1 to max_iterations
    (A, B, C) = randomly_pick_node_pairs(current_assignment)
    swap(A, B)
    swap(B, C)
    new_score = similarity_score(current_assignment)
    if (new_score > best_score)
      best_score = new_score
      best_assignment = current_assignment
    endif
  endfor
  return (best_assignment, best_score)
```

Figure 3.5: Pseudo code for stochastic matcher.

3.4.2 Error-Driven Matching

With this approach, a local matching error determines the probability that a node-pair assignment will be selected to be swapped. For example, if a node from the unknown was a line, and the corresponding node from the definition was an arc, there would be a relatively high local matching error, and correspondingly high probability that the node-pair would be selected for swapping. The local matching error of a node-pair is defined as the portion of the dissimilarity score related to that node-pair. This includes all intersection angle, intersection number, and intersection location errors involving the primitives from that node-pair. Likewise, the local matching error also includes primitive type and relative length errors.

At each iteration, the local matching error of each node-pair is computed and selection probabilities are assigned. Based on these probabilities, two node-pairs are selected and swapped. If the similarity score improves, the new assignments are kept. Otherwise, the swap is rejected. This continues until there is a certain number (I_0) of consecutive iterations with no improvement, or until the total number of iterations reaches a limit (I_{MAX}). In practice, we use a value of 300 for I_{MAX}, and a value of between 50 and 200 for I_0. The computational complexity of this approach is similar to that of the stochastic matching approach. Pseudo code for this matcher is shown in Figure 3.6.

3.4.3 Greedy Matching

This approach uses greedy search to find good node-pair assignments. The program first considers the best assignment for the first node of the unknown. If there are n nodes, the program considers all $n - 1$ cases in which the first node-pair is swapped with another. Whichever assignment produces the best similarity score is selected for the first node, and this node-pair is removed from further consideration. This is repeated for the second node-pair and so on. In all, $O(n^2)$ sets of node-pair assignments are considered. The entire search process can be repeated for increased accuracy. We have found that one repetition produces a significant improvement in accuracy, but additional repetitions produce minimal improvement. Pseudo code for this matcher is shown in Figure 3.7.

3.4.4 Hybrid Matching

For symbols with a small number of primitives, it is practical to use exhaustive search to find the optimal node-pair assignments. Our hybrid approach uses exhaustive search when there are six node-pairs or less. Otherwise it uses stochastic matching with

```
Error_Driven_Matcher(unknown, def)
  current_assignment = assign_using_drawing_order(unknown, def)
  best_assignment = current_assignment
  best_score = similarity_score(current_assignment)
  iterations = 0
  no_improvement = 0
  while (true)
    iterations = iterations + 1
    compute_local_matching_error(current_assignment)
    (A, B) = pick_node_pairs_using_local_error(current_assignment)
    swap(A, B)
    new_score = similarity_score(current_assignment)
    if (new_score > best_score)
      best_score = new_score
      best_assignment = current_assignment
      no_improvement = 0
    else
      swap(A, B) // undo swap
      no_improvement = no_improvement + 1
    endif
    if iterations == iteration_limit
       or no_improvement == no_improvement_limit
      return (best_assignment, best_score)
    endif
  endwhile
```

Figure 3.6: Pseudo code for error-driven matcher.

a limit of 720 iterations. Thus, regardless of the size of the problem, a maximum of 720 search states are explored. Pseudo code for this matcher is shown in Figure 3.8.

3.4.5 Sort Matching

This approach does not rely on search. Instead, the nodes are sorted based on the locations of their primitives. Each line segment is characterized by its minimum x and y-coordinates. Each arc is characterized by the coordinates of its center. The primitives are then sorted in ascending order of their x-values. Ties are broken using the y-values sorted in

```
Greedy_Matcher(unknown, def, number_of_rounds)
  current_assignment = assign_using_drawing_order(unknown, def)
  best_score = similarity_score(current_assignment)
  for repeat = 1 to number_of_rounds
     for i = 0 to number_of_node_pairs - 2
        best_swap = i
        for j = i+1 to number_of_node_pairs - 1
           swap(node_pair[i], node_pair[j])
           new_score = similarity_score(current_assignment)
           if (new_score > best_score)
              best_score = new_score
              best_swap = j
           endif
           swap(node_pair[i], node_pair[j]) // undo swap
        endfor
        swap(node_pair[i], node_pair[best_swap])
     endfor
  endfor
  return (current_assignment, best_score)
```

Figure 3.7: Pseudo code for greedy matcher.

```
Hybrid_Matcher(unknown, def)
  if(number_of_node_pairs <= 6)
    return(Exhaustive_Matcher(unknown, def))
  else
    return(Stochastic_Matcher(unknown, def))
  endif
```

Figure 3.8: Pseudo code for hybrid matcher.

ascending order. The sorted order of the nodes determines the node-pair assignments. The definitions for the sort matcher are learned with a special training procedure (see Section 3.5) and have pre-sorted nodes. Pseudo code for this matcher is shown in Figure 3.9.

This approach is useful only when the drawing orientation is fixed. However, even with a fixed orientation, variations in drawing can result in different sorted orders. For example, if the top edge of a horizontal square were drawn too long, it could be the leftmost primitive rather than the left edge of the square. Nevertheless, as Section 3.6 describes, the approach often works reasonably well in practice. Additionally, because this approach is particularly efficient, it is suitable for devices with little computational power, such as PDAs.

3.5 Training

The recognizer is trained by providing a set of training examples for each symbol class. As described in Section 3.2, the program constructs an "average" ARG for each class. This entails another graph matching problem. To learn a definition, the program must match the ARGs of the various training examples to one another. This task is different from the previous matching problem because a similarity score cannot be computed until after a definition has been learned. For example, during training, the primitive type error cannot yet be determined because the expected primitive type of each node is yet to be determined.

We have explored two solutions to this problem. The first is to require the training examples to be drawn with a consistent drawing order. In this case, the matching problem is avoided as the drawing order uniquely identifies the nodes in the ARG. The second approach, which we call "proximity matching," requires the user to draw symbols with a consistent orientation. In this case, geometric information is used for the matching.

With the proximity matching approach, the training examples are first scaled to have unit bounding boxes and are then translated to the origin. One of the symbols with the most frequently occurring number of primitives is selected as a reference symbol. For

```
Sort_Matcher(unknown, def)
  sort_primitives_in_x_and_y(unknown)
  current_assignment = assign_using_sorted_order(unknown, def)
  best_score = similarity_score(current_assignment)
  return (current_assignment, best_score)
```

Figure 3.9: Pseudo code for sort matcher.

example, if five of the examples have six primitives, and one example has seven, an example with six primitives will be selected as the reference. Each of the remaining symbols is then matched to the reference symbol.

To match a symbol, U, to the reference symbol, R, the scaled symbols are first overlayed on top of each other as shown in Figure 3.10. Then the directed distance from each primitive in U to each primitive in R is computed. To facilitate this, the primitives are resampled to have a uniform point spacing of 50 pixels. (We have found that resampling at 50 pixel intervals produces good accuracy with reasonable cost.) The distance from primitive A in symbol U to primitive B in symbol R is computed by finding, for each point a in A, the closest point b in B:

$$d(A, B) = \frac{1}{N_a} \sum_{a \in A} \underset{b \in B}{min} \|a - b\| \tag{3.11}$$

where, N_a is the number of points in A. Multiple points in A may be closest to the same point in B, and there may be some points in B that are not the closest point of any point in A. Figure 3.11 show an example of the directed distance from a line to an arc.

The best match between U and R is defined as an assignment of each primitive in U to a unique primitive in R such that the sum of the directed distances is minimized.

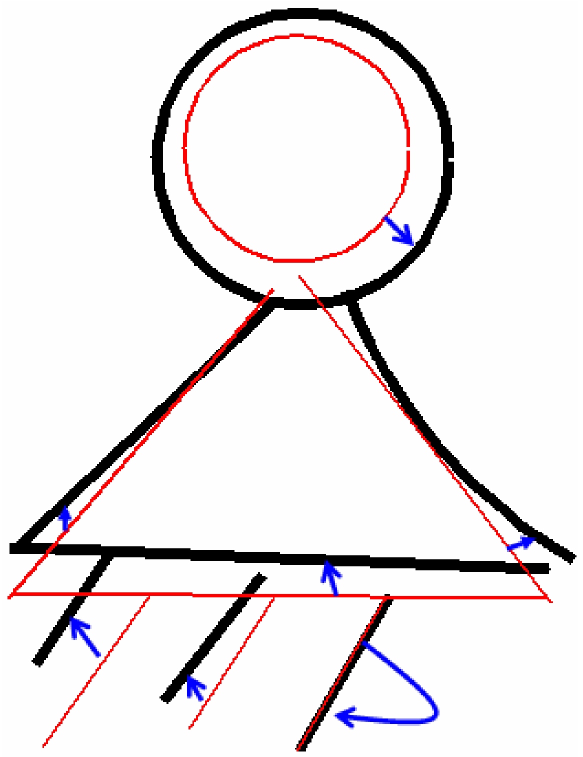


Figure 3.10: Proximity matching of two pivot symbols. The reference symbol is shown with bold primitives. The arrows indicate the assignments of primitives.

This can be expressed mathematically as:

$$BestMatch = \underset{m \in M}{argmin} \sum_{A \in Segs(U)} d(A, m(A)) \tag{3.12}$$

where M is the set of all one-to-one mappings of the primitives of U to the primitives of R. If R has fewer primitives than U, dummy primitives are added to R so that the number of primitives is the same for both.

To find the best match defined by Equation 3.12, we use depth first search with branch and bound. During the search, a partial match is pruned if the sum of the directed distances thus far exceeds that of the current best solution. Efficiency can be further

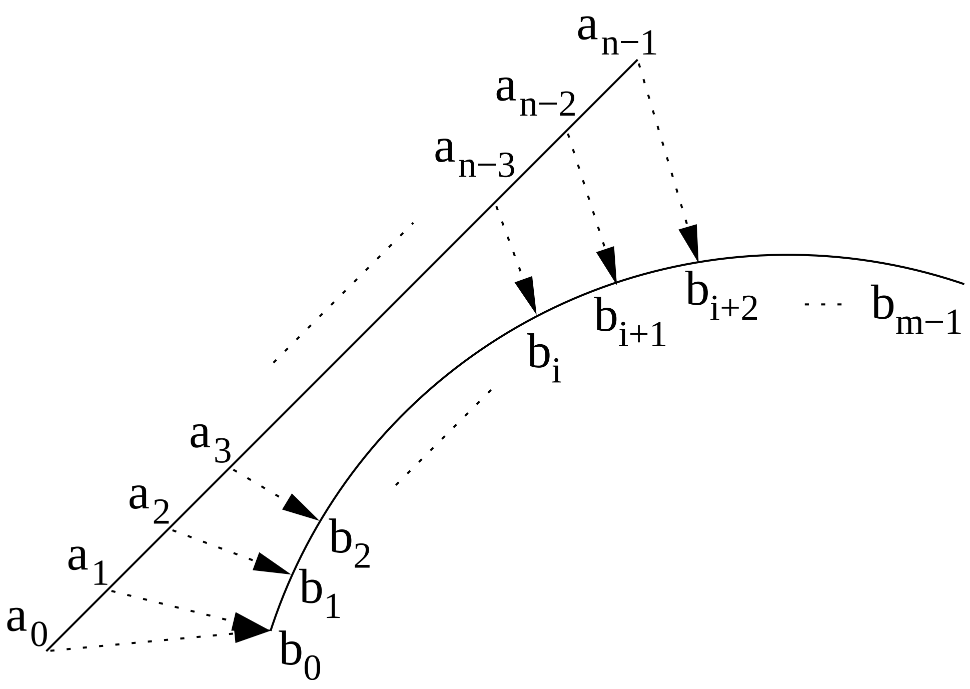


Figure 3.11: Computing the directed distance from primitive A to primitive B. Each point on A is mapped to the closest point on B.

improved through the use of a heuristic under-estimate. The heuristic distance for a given unmatched primitive from U is the minimum directed distance from that primitive to a primitive in R. If the sum of the directed distances for the matched primitives, plus the sum of the heuristic distances for the unmatched primitives exceeds the current best match, the partial match can be pruned. Because training is done off-line, efficiency has not been an issue and we have not implemented this heuristic approach.

Once all of the training symbols have been matched to the reference training symbol, the average ARG is constructed, thus forming the definition of the symbol.

To be consistent with the assumptions underlying the sort matcher, definitions for it are learned using a special procedure. Rather than using geometric proximity to match the training examples to one another, they are matched based on the sorted locations of their primitives. As before, lines are characterized by their minimum x and y-coordinates, and arcs are characterized by their centers. The primitives are sorted in ascending order of their x-values, and ties are broken using the y-values sorted in ascending order.

3.6 Results

We conducted a user study to evaluate the performance of our five matching techniques. The study involved nine participants, consisting primarily of engineering and computer science graduate students. Two had minimal prior experience with pen-based systems and the rest had essentially none. Because the participants were novices, this is a worst-case evaluation of our system. We expect that even better results would have been obtained if the participants had prior experience using our system.

Each participant was asked to provide fifteen sets of the 22 symbols[1] shown in Figure 3.12. Participants were instructed to draw naturally but reasonably carefully, and to not intentionally try to trick or break the system. They were also instructed to avoid over-stroking, and to instead redraw a symbol if necessary. (This was rarely done.) Data was collected using a tablet computer, which displayed only the raw ink rather than the processed (segmented) ink. Recognition accuracy was computed after the data was collected so that the participants would receive no feedback that could bias their performance.

[1]Data was collected for another symbol class, but it was not used. An anomaly with some examples of this symbol from one particular participant caused slow training. Ordinarily, this would not be a problem, but the experiments reported below required repeating the training process 1800 times.

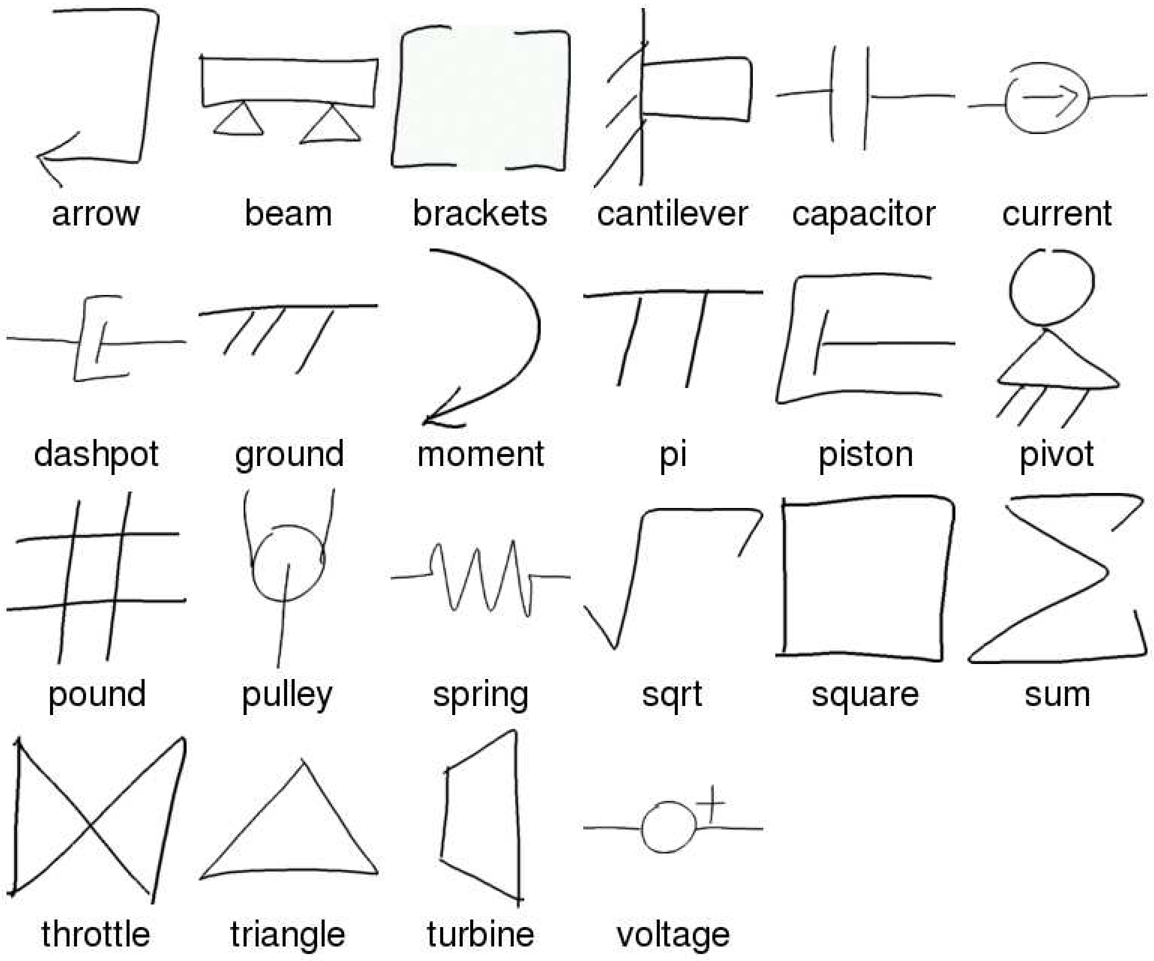


Figure 3.12: Symbols from the user study drawn by one of the participants.

3.6.1 Experiment One: Learning Rate

As one measure of performance, we evaluated recognition accuracy as a function of the number of training examples, n_t. We computed both the "top-one" accuracy, the rate at which the class ranked highest by the recognizer is indeed the correct class, and the "top-three" accuracy, the rate at which the correct class is one of the three highest ranked classes. The results are shown in Figure 3.13. The average recognition times for this experiment are listed in Table 3.2. All tests were conducted on a Pentium 4 machine with a 3.2GHz processor and 1GB of memory. Note that recognition time is independent of the number of training examples.

For this experiment, the maximum number of iterations for the stochastic matcher was set to 300. Likewise, the error-driven matcher was limited to 150 consecutive iterations with no improvement ($I_0 = 150$) or a total of 300 iterations ($I_{MAX} = 300$).

The recognizer was evaluated separately for each user using a cross validation approach. Each test consisted of randomly selecting n_t of the user's fifteen symbol sets for training, and using the remaining $15 - n_t$ sets for testing. The test was then repeated nine times, and the results averaged. For each value of n_t, the results were then averaged across all nine participants. Thus, each data point in Figure 3.13 represents an average of ninety iterations: ten cross-validation iterations for each of nine participants.

For this experiment, the hybrid matcher, stochastic matcher, and greedy matcher with two rounds of greedy search all achieved nearly the same performance. With only five training examples, these methods achieved top-one and top-three accuracies of better than 93.3% and 98.4%, respectively. With ten training examples, they achieved top-one and top-three accuracies of better than 96.0% and 99.0%, respectively. The hybrid approach achieved the highest top-one accuracy of 96.7% with ten training examples. The hybrid

Hybrid	**Stochastic**	**Greedy2**	**Greedy1**	**Error-driven**	**Sort**
41.8	35.9	4.0	2.1	34.9	0.24

Table 3.2: Average time to classify a symbol in ms. Stochastic: max-iterations = 300. Greedy1 = one round of greedy search. Greedy2 = two rounds. Error-driven: no-improvement-limit = 150 iterations, max-iterations = 300.

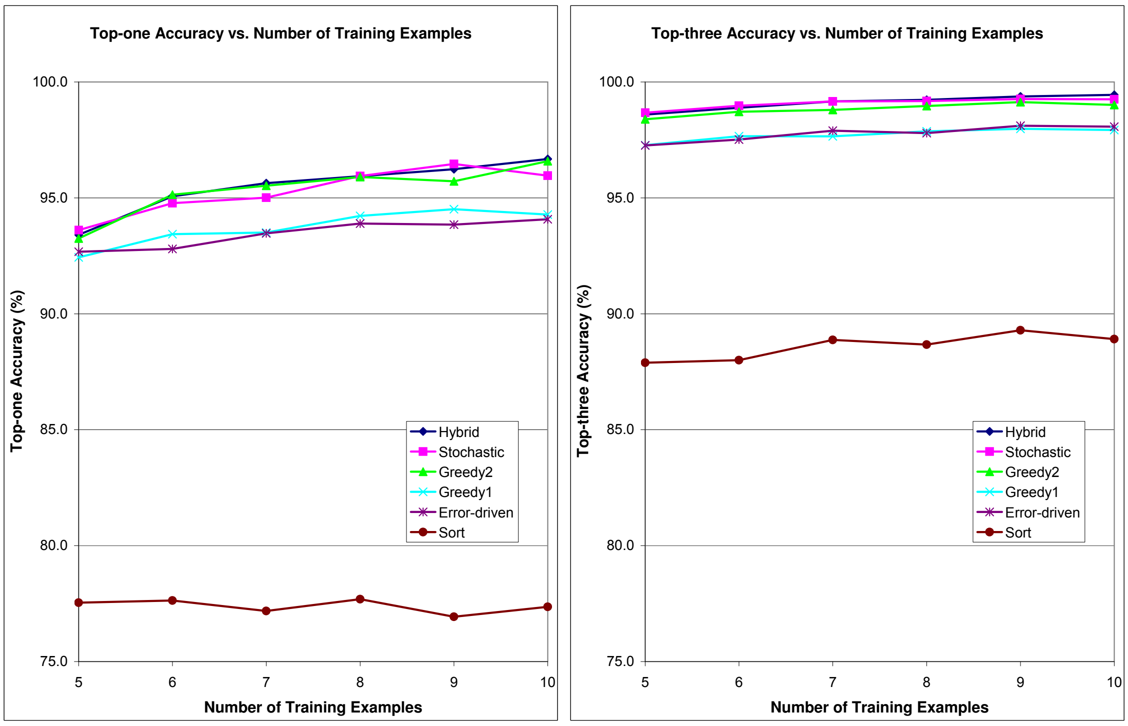


Figure 3.13: (Left) Top-one and (Right) top-three accuracy vs. the number of training examples. Stochastic: max-iterations = 300. Greedy1 = one round of greedy search. Greedy2 = two rounds. Error-driven: no-improvement-limit = 150 iterations, max-iterations = 300.

matcher took on average 41.8ms to classify a symbol, while the stochastic matcher took on average 35.9ms. The greedy matcher with two rounds of greedy search was much faster, requiring on average only 4.0ms to classify a symbol.

The error-driven matcher and greedy matcher with one round of greedy search also achieved nearly identical performance. With only five training examples, they achieved top-one and top-three accuracies of about 92.4% and 97.3%, respectively. With ten training examples, they achieved top-one and top-three accuracies of about 94.1% and 97.9%, respectively. The error-driven matcher took on average 34.9ms to classify a symbol, while the greedy matcher took on average only 2.1ms.

The sort matcher was the least accurate method, but it is exceptionally fast, requiring on average only 0.24ms to classify a symbol. With only five training examples, the sort matcher achieved top-one and top-three accuracies of 77.5% and 87.9%, respectively. With ten training examples, it achieved top-one and top-three accuracies of 77.4% and 88.9%, respectively. The sort matcher requires a consistent drawing orientation. The participants in our study tended to draw this way. If they had varied the orientation, the performance would have been lower.

3.6.2 Experiment Two: Drawing Order

As a second test of performance, we measured accuracy as a function of the randomness of the drawing order of the symbols. If the user maintains a consistent drawing order, matching is easier because all of our matching techniques, except sort matching, use the drawing order to construct the initial search state. Thus, randomizing the drawing order provides a good means of evaluating the robustness of our techniques.

In this experiment, the drawing order of the symbols was randomized by selecting pairs of primitives and swapping their drawing orders. The experiment was conducted with one, two, and five random swaps per symbol. Five swaps is a severe test, as it results in as many as ten primitives having random positions in the drawing order. To provide a baseline for comparison, accuracy was also measured for the original, "un-randomized" drawing order.

Figure 3.14 shows the top-one and top-three recognition accuracy as a function of the number of random swaps applied to the drawing order of each symbol. For the stochastic

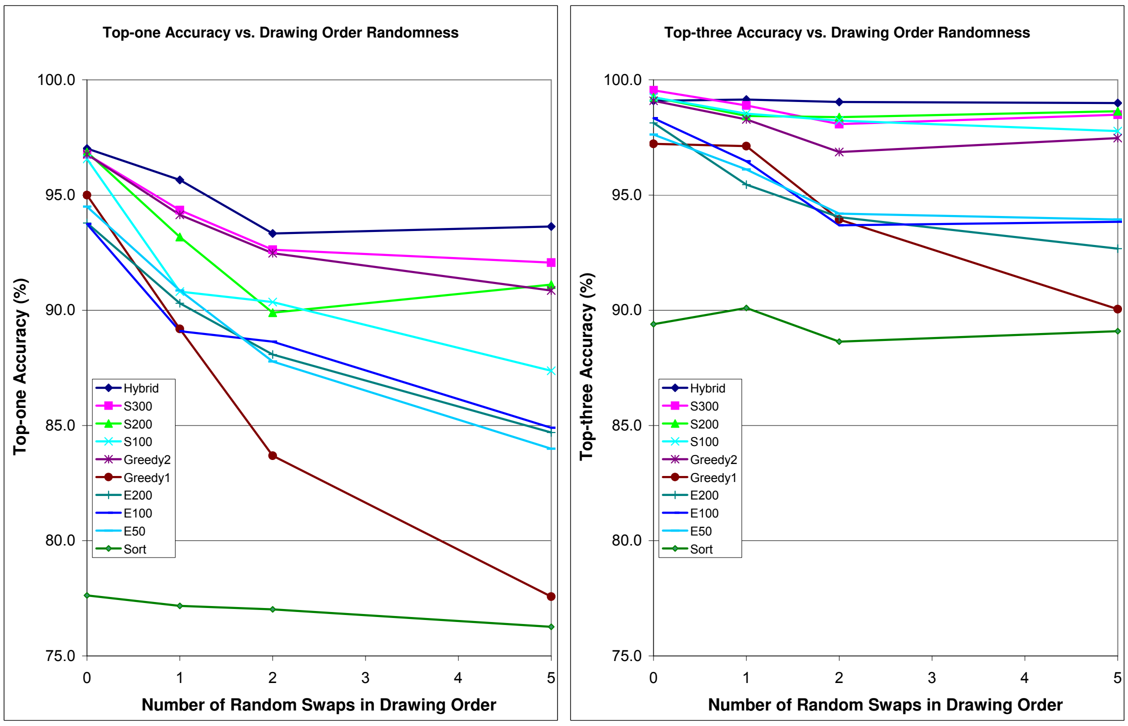


Figure 3.14: (Left) Top-one and (Right) top-three accuracy vs. the number of random changes to the drawing order. One random change consists of swapping the drawing order of a randomly selected pair of primitives. Sxxx = stochastic search with max-iterations = xxx. Greedy1 = one round of greedy search. Greedy2 = two rounds. Exxx = error-driven search with no-improvement-limit = xxx iterations and max-iterations = 300.

matcher, accuracy is reported for cases in which the number of iterations is limited to 100, 200, and 300. Likewise, for the error-driven matcher, accuracy is reported for cases in which the maximum number of iterations with no improvement (I_0) is limited to 50, 100, and 200. In all cases, the maximum total number of iterations for the error-driven approach is limited to 300 ($I_{MAX} = 300$). The average recognition times for this experiment are listed in Table 3.3.

In this experiment, the recognizer was again evaluated separately for each user, using a cross validation approach. Each test consisted of randomly selecting fourteen of the user's fifteen symbol sets for training, and using the remaining set for testing. The test was then repeated nine times, and the results averaged. The results were then averaged across all nine participants. Thus, each data point in Figure 3.14 represents an average of ninety iterations: ten cross-validation iterations for each of nine participants.

Examination of Figure 3.14 reveals that as the drawing order is increasingly randomized, more search is needed to achieve a given level of accuracy. For example, with five random swaps and an iteration limit of 100, stochastic search achieved a top-one accuracy of 87.4% and a top-three accuracy of 97.8%. When the iteration limit was increased to 300, the top-one and top-three accuracies increased to 92.1% and 98.5%, respectively.

The hybrid approach is the best performing method in cases where the drawing order varies. This is because it explores more of the search space than the other methods: It uses exhaustive search for small problems, and 720 iterations of stochastic search for larger ones. With five random swaps, the hybrid approach still achieved a top-one accuracy of 93.6% and a top-three accuracy of 99.0%. Despite exploring more of the search space, the hybrid approach is still fast, requiring on average only about 41.6ms to classify a symbol.

As expected, the performance of the error-driven approach also increased with increased iteration limits. However, for a given amount of processing time, this approach was not as accurate as the stochastic approach. This is due, in part, to the fact that the

Hybrid	**S300**	**S200**	**S100**	**Greedy2**	**Greedy1**	**E200**	**E100**	**E50**	**Sort**
41.6	35.6	23.9	12.2	4.0	2.1	44.9	24.3	13.5	0.29

Table 3.3: Average time to classify a symbol in ms. Sxxx = stochastic search with max-iterations = xxx. Greedy1 = one round of greedy search. Greedy2 = two rounds. Exxx = error-driven search with no-improvement-limit = xxx iterations and max-iterations = 300.

error-driven approach must compute both a local error and the dissimilarity score, while the stochastic approach computes only the latter.

The performance of the greedy matcher with one round of greedy search did degrade with increasing randomness in the drawing order. This is to be expected as greedy search methods tend to find only local maxima. However, applying a second round of greedy search substantially improved the performance. For example even with five random swaps in the drawing order, the greedy matcher with two rounds of search achieved top-one and top-three accuracies of 90.9% and 97.5%, respectively. Furthermore, it achieved this high level of performance while requiring on average only 4.0ms to classify a symbol.

The sort matcher is insensitive to drawing order and thus its performance did not vary significantly in this experiment. The small variations that did occur are likely a result of variations due to the random selection of training data in the cross-validation process.

3.7 Discussion

We believe that the results of our user study are quite promising when compared to results reported in the literature. For example, Landay and Myers [42] report an accuracy of 89% on a set of five single-stroke editing gestures. In our case, however, there are 22 symbol definitions which can be drawn with multiple strokes. In a study involving seven multi-stroke and five single-stroke shapes, Fonseca and Jorge [18] report an accuracy of 92%. Hse and Newton [28] report an accuracy of 97.3% using 30 training examples on a database of 13 symbols. Our hybrid matcher achieves an accuracy ranging from 93.6% to 97.0%, depending on the amount of randomness in the drawing order. Furthermore, our approach is insensitive to rotation and non-uniform scaling, where their approach may not be (see Chapter 2). On a database of 20 symbols, Kara and Stahovich [38] report an accuracy of 97.7%, where each symbol was trained with 14 examples. That method is based on image matching techniques, and thus is sensitive to non-uniform scaling. Also, our methods, particularly our greedy matcher, are faster than those in [38].

The experiments described in the previous section reveal various tradeoffs between our five matching techniques. The hybrid matcher is the most accurate, but the most expensive matcher. It achieves the best accuracy because it explores more of the search space than the other methods.

The stochastic matcher is the second most accurate method. One benefit of this approach is that it can take advantage of consistency in the drawing order. If the user maintains a consistent drawing order, relatively little computation is needed. If the drawing order varies greatly, the amount of computation can be directly adjusted to maintain high accuracy.

Like the stochastic-matcher, the error-driven matcher can take advantage of consistency in the drawing order. However, for a given processing time, this approach is not as accurate as the stochastic approach. This is due, in part, to the fact that each iteration of the error-driven approach requires the error metrics to be evaluated two times rather than one. They must be evaluated once to determine the local matching error and once to determine the dissimilarity score. The code could be optimized to eliminate the redundant computation. Also, the error-driven approach uses a form of hill-climbing: node-pair swaps are rejected if the similarity score decreases. It is possible that this hill-climbing strategy causes the match to become stuck in local maxima. It may be desirable to use a simulated annealing approach in which there is some probability that a decrease in the similarity score will be allowed on any given iteration.

The greedy matcher has the best tradeoff between accuracy and cost. Even with the drawing order randomized five times (*i.e.*, five random swaps in the drawing order), it achieved top-one and top-three accuracies of 90.9% and 97.5%, respectively. Furthermore, it took on average only 4.0ms to classify a symbol.

The sort matcher works by sorting the coordinates of the primitives. Thus, it is useful only when the drawing orientation is fixed. However, even for a fixed orientation, variations in the shape can result in different sorted orders, and consequently recognition errors. One advantage of this method is that it is very fast, requiring on average only 0.24ms to classify a symbol with a library of 22 definitions. Because the approach is so inexpensive, it would be suitable for devices with little computational power, such as PDAs. Furthermore, the top-three accuracy of 89% is still relatively high. This suggests that it might be possible to use the sort matcher as an inexpensive pre-recognizer to eliminate low ranked definitions. This would then reduce the amount of computation needed for a more accurate but more expensive matcher, such as the hybrid matcher.

The results in Figure 3.14 suggests that the participants in our user study tended to draw with a relatively consistent drawing order. This allowed our program to achieve high accuracy with little computation. Artificially randomized the drawing order necessitated more computation to achieve high accuracy. The consistency of the drawing order in our experiment was likely due to the nature of the data collection task. We expect that when pen-based applications are used for real-world tasks, there will indeed be variation in the drawing order. However, we do not expect it to be entirely random. Rather, we expect that users will likely have a few preferred ways of drawing each symbol. For example, when drawing a pivot, one is likely to draw the triangle and then the circle, or vice versa. Furthermore, it is unlikely that one would draw part of the triangle, then the circle, then the rest of the triangle. As a consequence, we plan to explore the possibility of learning the most common drawing orders for each symbol. Combining these with a small amount of search using one of our matching methods may prove to be an effective approach.

Regardless of the amount of randomness in the drawing order, our experiments indicate that the hybrid, stochastic, and greedy matching techniques provide suitable accuracy. There is little variation in their performance when applied to data with two random drawing order swaps vs. five. In the later case, the drawing order is essentially random because the typical symbol in our study had about five primitives. Note that each random swap actually results in two primitives having random positions in the drawing order.

Our experiments did not explicitly evaluate our recognizer's insensitivity to orientation and non-uniform scaling. As our representation is entirely insensitive to orientation, we expect that our overall approach is insensitive to orientation. Likewise, because our representation describes the topology of a shape, our approach is tolerant of non-uniform scaling. However, it would be useful to conduct further studies to quantify this.

We do not yet have an automated means of evaluating the performance of the proximity matcher used for training. We do, however, have a tool that allows us to visually inspect the matches it produces. We have informally examined some of the data and have found that the approach is reliable. Although we have not separately evaluated the performance of the proximity matcher in a systematic way, its performance can be inferred from the overall performance of our system. Because the system as a whole performed well, the proximity matcher must have performed well.

One important area of future work is the use of feedback from our recognizer to improve segmentation. The pen stroke segmenter performed accurately, but examination of the data revealed that some of the symbols did contain missing or extra primitives. Likewise, some primitives that were intended to be lines were classified as arcs, and vice versa. Some of these errors were due to sloppy drawing and ambiguity, while others were a result of processing errors. Once a symbol has been classified, the segmenter could be informed of mismatches between the segmentation of the unknown symbol and that of the matching definition so that the segmentation could be improved.

3.8 Summary

In this chapter, we have presented a trainable symbol recognizer for pen-based user interfaces. The approach is suitable for multi-stroke symbols, and is insensitive to rotation, and tolerant of uniform and non-uniform scaling (*i.e.*, different parts of a shape being scaled differently). Furthermore, our user studies have demonstrated that our approach allows a symbol to be drawn with any drawing order. If, however, the user maintains a relatively consistent drawing order, our techniques can take advantage of this and operate more efficiently.

A symbol is represented internally as an attributed relational graph that describes both its geometry and topology. A symbol definition is also represented as an attributed relational graph, but the attributes are learned from training examples and are described statistically. Using a statistical representation makes our approach robust to the types of variations common in hand-drawn shapes.

Symbol recognition involves finding the symbol definition whose attributed relational graph best matches that of the unknown symbol. We have developed a novel set of metrics for comparing graphs in this domain. Much of the power of our approach derives from these metrics, and the particular set of attributes encoded in the graphs.

One challenge addressed in the current work is how to perform graph matching in an efficient fashion so as to achieve both tolerance for drawing variation and interactive performance. We presented five approximate graph matching techniques: stochastic matching, which is based on stochastic search; error-driven matching, which uses local matching errors to drive the solution to an optimal match; greedy matching, which uses greedy search; hybrid matching, which uses exhaustive search for small problems and stochastic matching for larger ones; and sort matching, which relies on geometric information to accelerate the matching. We have also developed a "proximity" matcher which is used for training purposes.

Our user studies have revealed a number of tradeoffs between these techniques. The hybrid matcher is the most accurate but most expensive approach. Nevertheless, it is still sufficiently fast for interactive use. The stochastic matcher achieves high accuracy and can take advantage of consistency in the drawing order. If the user maintains a consistent drawing order, relatively little computation is needed. If the drawing order varies greatly, the amount of computation can be directly adjusted to maintain high accuracy. The greedy matcher provides the best tradeoff between accuracy and cost: It achieves relatively high accuracy and is fast. The error-driven matcher is accurate, but for a given amount of computation, is not as accurate as the stochastic matcher. The sort matcher is the least accurate approach and requires consistent drawing orientation. However, because this method is particularly efficient, it may be a good solution when computational resources are constrained, such as with a PDA.

Chapter 4

Newton's Pen: A Pen-based Tutoring System for Statics

4.1 Introduction

Intelligent tutoring systems have been widely studied and have been applied to a variety of domains [75, 73, 29, 57, 17, 76, 8, 10, 67, 78]. Most current systems are based on WIMP (windows, icons, menus, and pointers) interfaces. However, research suggests that transfer of skills from training to testing is higher when the testing and training environments are similar [56]. Thus, there is a clear benefit to creating tutoring systems with interfaces that match real-world problem-solving environments.

The goal of our work is to create computational techniques and user interface design principles to enable natural, pen-based tutoring systems that scaffold students in solving problems in the same way they would ordinarily solve them with paper and pencil.

This goal is consistent with recent research comparing student performance across different user interfaces showing that "as the interfaces departed more from familiar work practice..., students would experience greater cognitive load such that performance would deteriorate in speed, attentional focus, meta-cognitive control, correctness of problem solutions, and memory" [59]. While that work used systems that provided no problem-solving assistance (i.e., they were not tutoring systems), the findings provide compelling evidence of the potential benefits of well-designed, pen-based instructional tools.

As a step toward our goal, we have built Newton's Pen, a pen-based statics tutor designed for the LeapFrog FLY pentop computer, shown in Figure 1.1. Statics is the subdiscipline of engineering mechanics concerned with the equilibrium of objects subjected to forces. Newton's Pen scaffolds students in the construction of free body diagrams and equilibrium equations, two essential components of equilibrium analysis.

Newton's Pen runs entirely on the FLY's embedded processor, which created significant challenges because of the limited memory and computational power available. The platform also presented substantial user interface design challenges because audio is the only form of dynamic output on the FLY. The system has a speech synthesizer and can play recorded sound clips. Our current implementation relies exclusively on synthesized speech for audio output.

The next section presents an overview of the Newton's Pen system. This is followed by the details of the system design. Finally, results of three user studies are presented and discussed.

4.2 System Overview

Newton's Pen is deployed on the LeapFrog FLY pentop computer (Figure 1.1), which is based on Anoto [1] digital pen and paper technology. "Digital paper" is ordinary paper printed with a special dot pattern. The pentop contains a digitizer (camera) that uses the dot pattern to locate the pen tip coordinates. The digitizer, which is near the tip of the pen, is activated when the pen is pressed against the paper. The FLY produces two kinds of output: it leaves ink on the paper, and can produce synthesized speech and sound clips through a speaker or headphones.

The Newton's pen system consists of software and specially designed worksheets. The software is contained on a flash memory cartridge and runs entirely on the FLY's embedded processor. The worksheets, which are printed on digital paper, contain instructions for using the system, user interface objects (i.e., buttons), and spaces for solving problems. Each problem is laid out in three pages: one page (on ordinary paper) for the problem description (Figure 4.1), one for the free body diagram (Figure 4.2a), and one for the equilibrium equations (Figure 4.2b).

Figure 4.1 shows a typical problem in which the student is asked to draw a free body diagram for a ring of weight W supported by two cords. Figure 4.2a shows a worksheet for drawing free body diagrams. The top of the page has instructions for using the system. Depending on the operating mode, the system can provide additional instructions via synthesized speech. The "REPEAT INSTRUCTIONS" button at the top of the page is used to repeat the last audio instruction. Below the printed instructions are three blocks for drawing free body diagrams (only two are shown in the figure), which allow the student

Problem:
A ring *C* of weight *W* hangs from a ceiling by means of two cords as shown below. The angles between the horizontal and cords *AC* and *BC* are *Q* and *U,* respectively. Draw the free body diagram for the ring, and write the equilibrium equations.

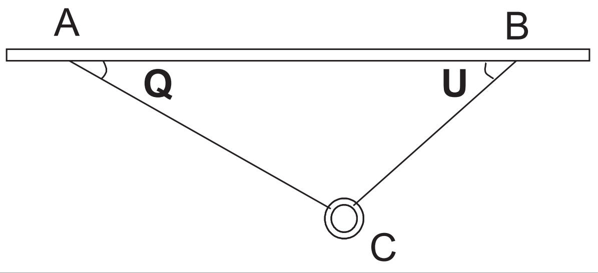
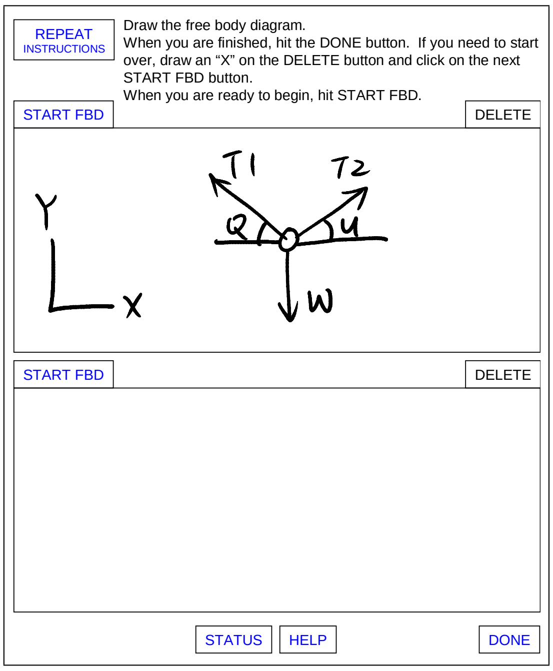


Figure 4.1: A typical problem from Newton's Pen. This problem was used for the pretest in the user studies.

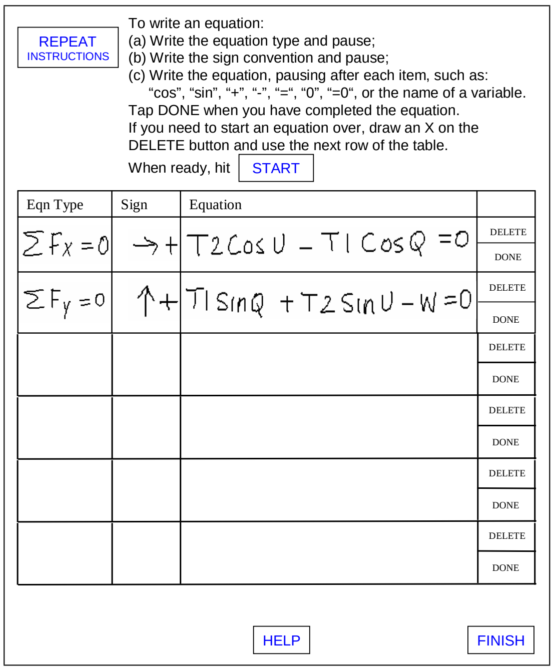
to make three attempts at solving the problem. There are two buttons at the top of each block: "START FBD" and "DELETE". To begin, the student first taps the pen on "START FBD". The student then draws the coordinate system, the body, and the forces in the space provided.

After drawing each graphical element such as a force arrow, coordinate frame, body, label, etc., the student must pause until the system provides audio feedback indicating what was recognized. If an element cannot be interpreted, the system announces that it was "not recognized." If an element is interpreted but is incorrect (i.e., there is a problem-solving error), the system announces the interpretation and a suitable error message. For example, if the student draws a rectangle for the body, but a circle was expected, the system announces "rectangle is incorrect." In an earlier implementation, our system simply



Draw the free body diagram.
When you are finished, hit the DONE button. If you need to start over, draw an "X" on the DELETE button and click on the next START FBD button.
When you are ready to begin, hit START FBD.

(a)



To write an equation:
(a) Write the equation type and pause;
(b) Write the sign convention and pause;
(c) Write the equation, pausing after each item, such as:
"cos", "sin", "+", "-", "=", "0", "=0", or the name of a variable.
Tap DONE when you have completed the equation.
If you need to start an equation over, draw an X on the DELETE button and use the next row of the table.
When ready, hit START

Eqn Type	Sign	Equation	
$\Sigma Fx = 0$	$\rightarrow +$	T2 Cos U − T1 Cos Q = 0	DELETE DONE
$\Sigma F_y = 0$	$\uparrow +$	T1 Sin Q + T2 Sin U − W = 0	DELETE DONE
			DELETE DONE
			DELETE DONE
			DELETE DONE
			DELETE DONE

(b)

Figure 4.2: (a) Worksheet for drawing free body diagrams. (b) Worksheet for writing equilibrium equations.

announced "try again" for both uninterpreted and incorrect input. This caused many users to repeatedly draw the same incorrect input. For example, a student might repeatedly draw a rectangle when a circle was expected. The current approach of announcing the identity of all recognized objects, even those that are incorrect, has proven to be a more effective strategy.

If the student makes several mistakes while drawing a free body diagram, he or she can start over in the next block. Before doing so, the student marks an "X" on the "DELETE" button to indicate that the current work should be discarded. The student then taps the "START FBD" button of the next block, and begins as before.

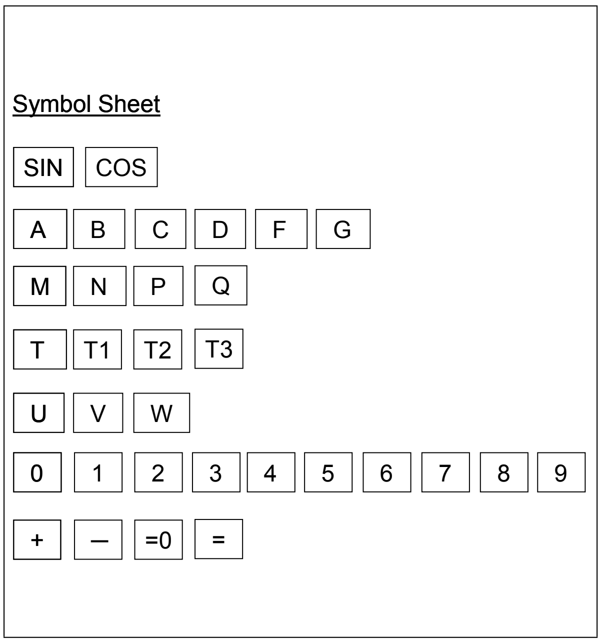


Figure 4.3: Symbol sheet used for correcting recognition errors when writing equations.

There are three additional buttons at the bottom of the page: "HELP", "STATUS", and "DONE". The "HELP" button provides help both about using the program and about solving statics problems. The help is context sensitive. For example, if the student is drawing the coordinate system, the "HELP" button will provide assistance with the proper way to draw a coordinate frame. If the student is drawing forces, the "HELP" button provides assistance with the proper way to label force arrows, and it also provides assistance with identifying the forces needed to complete the diagram. Students can use the "HELP" button for hints about what to do next or for guidance following uninterpreted or incorrect input.

The "STATUS" button informs the student of which forces have been correctly drawn and how many forces remain. The "DONE" button initiates analysis of the free body diagram. If the diagram is complete, the student is congratulated. If forces are missing, the system notifies the student of the number of missing forces. The “STATUS” and “DONE” buttons perform similar functions. However, the student uses the former to query for help, and the latter to indicate that the problem has been solved. This distinction will eventually be used in assigning credit.

Figure 4.2b shows the worksheet for writing equilibrium equations. It provides a table for the student to write equations, where each row contains a space for the equation type, the sign convention, and the equation. The equation type denotes the generic equilibrium equation, and the sign convention indicates the positive coordinate direction. For example, “$\Sigma Fx = 0$” and “$\rightarrow +$” indicate an equilibrium equation for the x-direction, with positive defined to the right. Requiring the student to provide this information has several benefits: It supports meta-cognition, is consistent with the way statics is commonly taught, and allows Newton’s Pen to provide focused feedback.

To facilitate correction of interpretation errors, equations are written token-by-token. “Tokens” consist of force and angle labels (e.g., “T1” and “Q”), “sin”, “cos”, “+”, “-”, “=”, and numerical digits. As each token is written, the system announces the interpretation. If the token is not recognized or is misinterpreted, the student can simply choose to rewrite it. However, such errors are more easily corrected by tapping on the intended interpretation in the “symbol sheet” shown in Figure 4.3. For example, if the student writes “T2”, but it is misinterpreted as “T3”, the student can fix this by tapping on “T3” on the symbol sheet. The symbol sheet is not problem-specific, but rather contains the complete set of symbols that could appear in equilibrium equations.

In addition to the normal problem-solving mode, Newton's Pen has a demonstration mode that familiarizes students with the system. The demonstration begins by providing step-by-step instruction for drawing and labeling forces. The student is then shown an example of a force and is asked to copy it. Next, the student is shown a sample problem, and the complete free body diagram. The system asks the student to copy the diagram, and provides step-by-step audio instruction for doing so. Likewise, the student is shown the equilibrium equations and is guided through the process of copying them. Before starting the demonstration, we typically ask students to practice using commercial FLY applications such as the calculator or music keyboard ("FLY Tones") applications. This helps students become familiar with basic interaction concepts, such as pausing after drawing and waiting for audio feedback.

4.3 System Design

The following sections describe how Newton's Pen performs its tasks. In particular the system architecture, recognition techniques, interpreters for free body diagrams and equations, and tutoring techniques are presented.

4.3.1 System Architecture

Newton's Pen is designed around a systematic problem-solving approach. For example, when drawing the free body diagram, the student must first draw a coordinate system, then the body, and finally the forces. Similarly, when writing an equilibrium equation, the student must first specify the equation type and sign convention. We initially selected a systematic approach for both performance and pedagogical reasons. Constraining the problem-solving order limits the number of graphical objects that are expected at any given time. This reduces recognition cost and increases recognition accuracy by reducing opportunities for confusion. Additionally, the systematic problem-solving approach is consistent with the way many instructors teach. With recent improvements in our recognizers, the performance advantages of enforcing a systematic problem-solving approach are less important then they were initially. However, the pedagogical advantages remain. In fact, many of the participants in our user studies explicitly indicated that they liked the system's step-by-step approach.

To accommodate a systematic problem-solving approach, our system architecture is essentially a non-deterministic finite state machine. Here, a "state" is a C++ object that contains a list of legal inputs, a list of next states, and context sensitive help. Each problem actually has two finite state machines: one for the free body diagram, and one for the equilibrium equations. Inputs to states consist of hand-drawn graphical objects, such as arrows, text, leader lines, etc. The expected properties of each graphical object are encoded in the state. For example, a force arrow is associated with an expected orientation and location on the body. Each legal input for a state is associated with a specific recognizer that can classify that kind of input (see Section 4.3.2). The finite state machines for a given problem are automatically constructed from a description of the problem which includes the shape of the body (e.g., a rectangle) and the list of forces. This is done using a set of macros that generate "const" data structures so as to conserve available RAM.

After the student draws an object and pauses (or double-taps the pen), the system calls each of the recognizers associated with the current state to determine the object's classification. If the object is recognized, and it is both a legal input for the state and has the expected properties, the system announces the classification of the object and advances to the appropriate next state. Otherwise, the system reports an error and remains in the same state. In particular, if the object is not recognized, the system announces "not recognized." Conversely, if the object is recognized, but it is either an illegal input to the state, or its properties are incorrect, the system announces the identity of the object and provides a suitable error message. Typically, the error message is that the input "is incorrect" (i.e., not expected), or that some property of the input, such as its location or orientation, is incorrect. For forces, however, a second error message is possible. Force symbols are comprised of an arrow and a force label such as "F". Forces not aligned with a coordinate axis also include a leader line, an arc, and an angle label such as "Q". Figure 4.4 shows a typical example. The student is required to draw all of the parts of one force before beginning the next. If the student draws some other recognized object before completing a force, the system announces the identity of the object and instructs the student to "go back and complete the previous force."

There are a few special issues with the interpretation of equations. After each token of an equation is written, the system either announces the interpretation of the token, or announces "not recognized" if it cannot be interpreted. In the latter case, the student can use the symbol sheet in Figure 4.3 to manually specify the intended meaning. Similarly, if the system misclassifies the token, the student can correct this with the symbol sheet.

Our approach of providing instantaneous interpretive feedback is necessary because

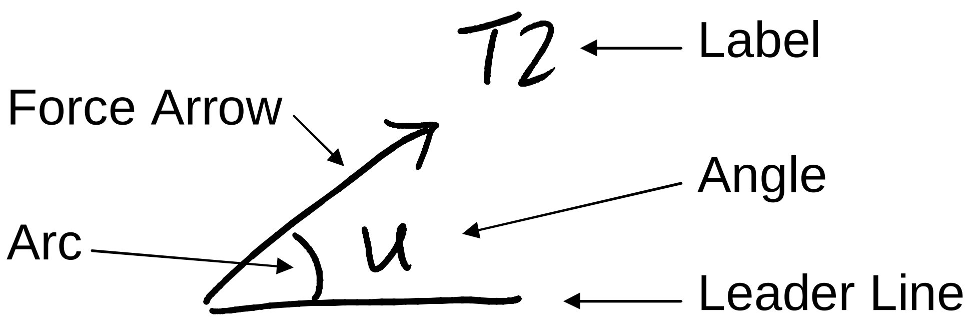


Figure 4.4: A typical force.

of the unusual user interface capabilities of a digital pen – the only means of providing feedback is via audio. The most direct way to relate the feedback to a specific object is to provide it immediately after the object is drawn. If, instead, several objects were drawn at once, it would be difficult for the student to know which feedback went with which object. This would be particularly troublesome in cases in which objects were not recognized, because then there would be no convenient method of referring to the unrecognized objects. In addition to the usability advantages, providing instantaneous feedback also has pedagogical advantages. It is beneficial to provide instructional feedback to the student at the instant that a problem-solving error occurs.

To illustrate our finite state machine architecture, consider the free body diagram on the worksheet in Figure 4.2a, describing a particle subject to three forces $T1$, $T2$, and W. A portion of the corresponding finite state machine is shown in Figure 4.5. At the instant shown, the student has drawn the coordinate system and body, and the system is in state *Forces*. There are three legal inputs to this state: Arrow1, the arrow corresponding to force $T1$; Arrow2, the arrow corresponding to force $T2$; and Arrow 3, the arrow corresponding

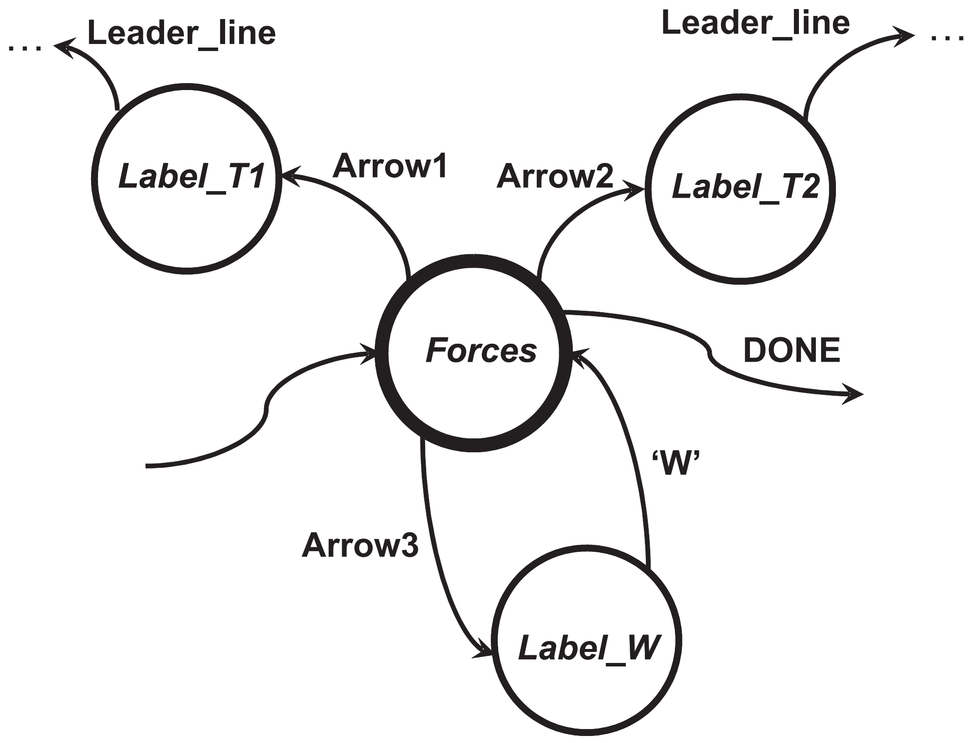


Figure 4.5: A portion of the finite state machine corresponding to the free body diagram in Figure 4.2a. The system is currently in state *Forces*.

to force W. If the student draws an arrow matching Arrow3, for example, the system will transition to state *Label_W* in which the only legal input is the text label "W". If the student writes a "W" while in this new state, the system will transition back to state *Forces* and wait for the other forces to be drawn. Thus, our C++ "state" object actually represents multiple logical states.

This architecture has a number of important benefits. First, it provides a convenient method for providing context sensitive help. Each state is associated with help messages that are announced to the student when the help button is pressed. Because the program's state mirrors the student's problem-solving progress, the help messages are always relevant. Second, the architecture allows new problems to be easily coded. A new problem is created by instantiating the appropriate set of state objects. Third, the architecture allows the system to be run in a demonstration mode that guides the student in the use of the system (see Section 4.2). Each state contains an explanation of what should be drawn in that state. In demonstration mode, when a state is entered, the explanation is announced to the student.

4.3.2 Recognition

Newton's Pen must recognize a variety of hand-drawn objects including: text, force arrows, leader lines and arcs used to indicate angles, and bodies. In most recognition-based systems, the task is to determine which of many possible classifications should be assigned to an arbitrary symbol. By contrast, our task is to determine whether or not a given object matches the expected classification. For example, our system must determine if a symbol is or is not an arrow. Because of the nature of our task, we can use a set of efficient, special-purpose recognizers, each suited for specific types of symbols. The current state of the finite state machine determines which recognizer is used at any given time. This is less expensive than using a single, general-purpose recognizer. Also, because our task is typically to verify the class of a symbol, rather than to select the classification from multiple possibilities, our recognizers need not be as robust as those required for typical recognition-based interfaces. The following sections describe the various recognizers we use.

Character Recognizer

The FLY has an integrated character recognizer which we use to recognize force and angle labels, and the characters in equations. To improve accuracy, we bias the character recognizer to the text expected to appear in the solution of the problem. Biasing the recognizer helps to prevent situations when “T1” is misclassified as “TI”, for instance. To prevent false positives, we rely on the confidence value of the recognition result. If this value is below a threshold, Newton’s Pen considers the ink to be unrecognizable.

Body Recognizer

For the problems we have considered, bodies are either circles or horizontal or inclined rectangles. (The latter are typically inclined between about 30^o and 45^o from the horizontal.) We use a neural network to recognize these shapes. The inputs to the neural network are the pixel values of a low-resolution binary bitmap image of the shape. To construct the bitmap, it is first necessary to frame the image. A bounding box aligned with the edges of the paper is constructed. The shortest dimension of the bounding box is then expanded, without changing the location of the box’s center, to produce a square. The result is that the shape is centered in a square frame, without necessarily filling it. This preserves the original aspect ratio. The frame is then sampled to produce a 17x17 binary bitmap. The digitized pen strokes are typically sparse, thus when sampling the framed image, it is necessary to interpolate between consecutive points in the pen strokes.

Our neural network has 289 inputs and one hidden layer with six nodes. There are four outputs corresponding to a circle, a horizontal rectangle, an inclined rectangle, and other shapes (i.e., negative examples). The network was trained with circles and rectangles provided by a number of individuals. The training data included both single-stroke and multi-stroke rectangles. The negative training examples included arrows and triangles.

Once a body has been recognized, it is necessary to determine its geometric parameters. For a circle, the center is approximated as the center of the bounding box. The radius is approximated as the average distance from the points in the pen strokes to the center. The corners of a horizontal rectangle are taken to be the corners of a coordinate-aligned bounding box. The corners of an inclined rectangle are obtained in a similar fashion, except that the ink is first rotated before the bounding box is constructed. The program considers two different rotation angles: 35^o and 45^o clockwise. The rotation that results in the bounding box with the smallest area is taken to be the best fit. The smallest bounding box is then rotated back to the original orientation to determine the corners of the rectangle.

Arrow Recognizer

To recognize arrows, we use a technique described in [34]. Our implementation is suitable for both single-stroke and two-stroke arrows drawn from tail to tip. Two-stroke arrows are handled by joining the last point of the first stroke to the first point of the second stroke so that both kinds of arrows become single pen strokes. The (equivalent) single pen stroke is resampled to produce 20 evenly spaced points. The cosine of the angle between adjacent segments is then computed, as shown in Figure 4.6. The cosine is inversely related to the curvature. For example, if two consecutive segments are nearly colinear, the cosine is close to 1.0. If there is a large discontinuity, such as a 90^o bend, the cosine is close to

0.0. For this reason, the cosine of the angle between adjacent segments is called “inverse curvature.” The inverse curvature is undefined for the first and last points on a pen stroke, thus the value at these points is assumed to be 1.0. However, to eliminate hooking problems, the first two sample points are actually discarded by the recognizer.

Figure 4.7 shows the inverse curvature representation of the arrow from Figure 4.6. Notice that the inverse curvature is approximately 1.0 for most points on the arrow, but is much smaller (in this case, less than 0.0) for the three discontinuities at the head of the arrow. It is these discontinuities that enable the technique to identify arrows.

The arrow recognizer is a neural network comprised of an eighteen-node input layer, one hidden layer with nine nodes, and a single-node output layer. The inputs to the network are the 18 inverse curvature values for the arrow. The output is the classification: arrow or non-arrow. We trained the network using 966 arrows and 1143 non-arrows from four different individuals. The non-arrows consisted of alpha-numeric characters.

Once an arrow has been recognized, it is necessary to compute the two endpoints, which are later used to determine if the arrow (i.e., the force) has the correct location and orientation. To facilitate this, the arrow is resampled to produce 20 evenly-spaced points. A line is constructed connecting the first and sixth (20/3) samples. Finally, the sample points of the arrow are projected onto the line to determine the two endpoints. The head and tail are differentiated by the fact that arrows are drawn tail to head.

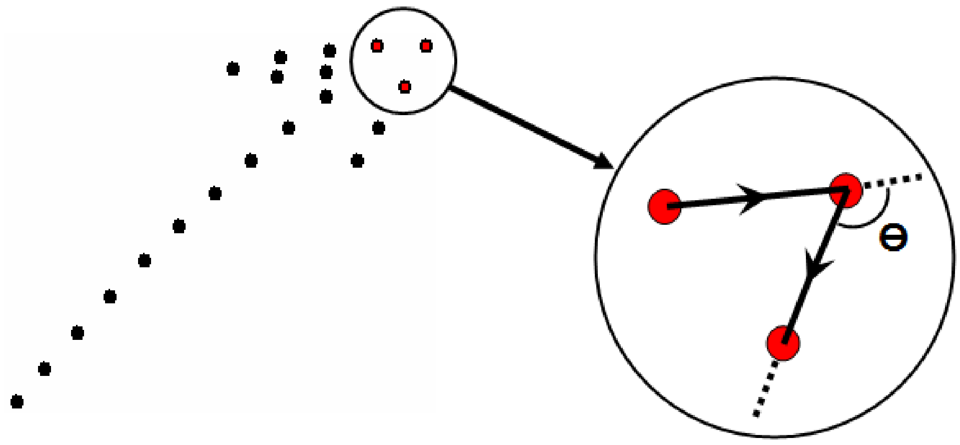


Figure 4.6: Resampled arrow. Inverse curvature at point A is $cos(\theta)$.

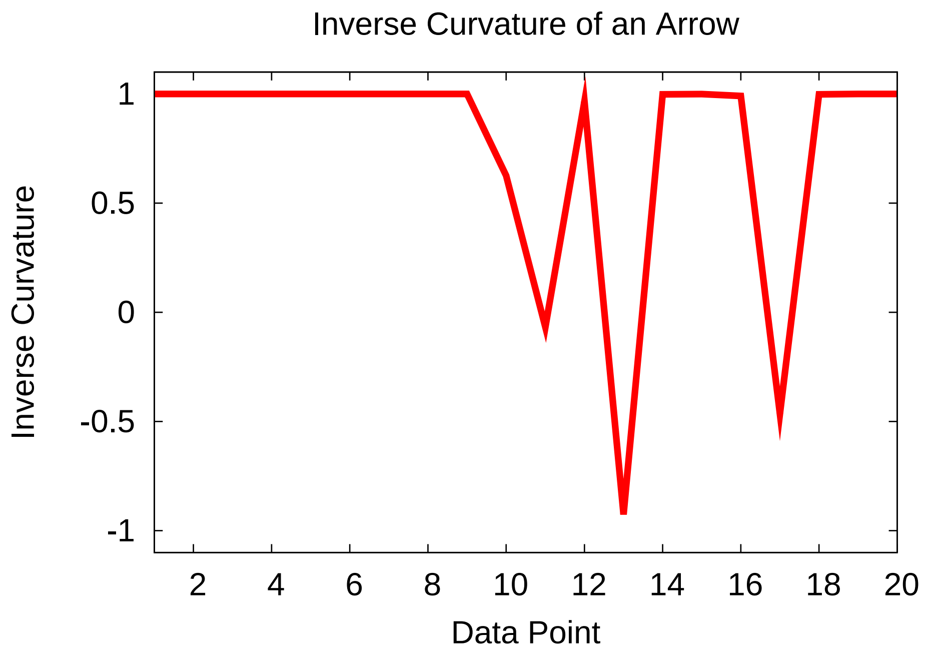


Figure 4.7: Inverse curvature of the arrow from Figure 4.6.

Image-Based Recognizer

Our system includes a computationally-inexpensive, image-based recognizer. It is based on one previously developed by our group [38], but is modified for the limited computational resources of the FLY. We currently use it to recognize the sign convention portion of equations, such as "$\rightarrow +$". In our first implementation of Newton's Pen, we also used it to recognize arrows. However, we now use our inverse-curvature approach (see above) for arrows, as it has proven to be more accurate.

With the image-based recognizer, symbols are represented as binary bitmap images called "templates." As with the body recognizer, the bitmaps are 17x17. However, whereas the body recognizer uses uniform scaling when framing the image, the image-based recognizer applies nonuniform scaling to the ink to produce a square bounding box. We have found nonuniform scaling to increase accuracy for our application.

We compare templates using the Modified Hausdorff Distance (MHD) [16]. An unknown symbol is classified by the definition that matches with the minimum MHD. However, if the minimum value is greater than a threshold, the symbol is considered to be unrecognizable. The threshold is approximately 10% of the length of the diagonal of the 17x17 grid.

Angle-Indicator Recognizer

The angle of a force is indicated with a leader line and arc as shown in Figure 4.4. A leader line is a pen stroke that is roughly straight, and approximately horizontal or vertical. Here we make use of the "straightness ratio" defined as the ratio of the distance between the first and last points and the total stroke length. If the straightness ratio is greater than or equal to 0.8, the stroke is considered to be a line, otherwise it is considered to be an arc. A line is assumed to be horizontal if its slope is less than 0.3. Vertical lines are defined analogously.

4.3.3 Free Body Diagram Interpreter

As described in Section 4.3.1, after each element of a free body diagram is drawn, the system announces the interpretation. Arrows are described by one of eight qualitative directions which are spaced at 45^o intervals. For example, an arrow that points up and to the right is announced as "arrow forty five degrees."

After announcing the identity of an element, Newton's Pen verifies that the element is correct. Force arrows must be in the correct location on the body and have the correct orientation. Each force label must be near the correct force. The leader line for a force must intersect it near its tail and have the correct orientation (either horizontal or vertical). The corresponding arc must connect the arrow and leader line. Finally, angle labels must be near their arcs. If these conditions are not satisfied, the system provides a suitable error message. For example, if the student draws an upward-pointing force but none exists in the problem, the system will announce: "arrow ninety degrees is incorrect." Similarly, if the student draws a leader line that is not near an arrow, the system will announce: "leader line is too far away."

When the student has completed the free body diagram, he or she taps the "DONE" button, and the program checks if the diagram is complete. If it is, the student is congratulated. Otherwise, the system notifies the student that forces are missing.

4.3.4 Equation Interpreter

Our current system can interpret only a limited class of equations appropriate for the set of problems we have considered thus far. Currently, the system can interpret force equilibrium equations, but not moment equilibrium equations. The latter are unnecessary for particle equilibrium problems. Also, equations cannot have parentheses.

In our first implementation of Newton's Pen, the student wrote the entire equation before receiving interpretive feedback. If there was a recognition error, the student was forced to rewrite the entire equation. As a remedy, we modified the system so that equations are written token-by-token. As described above, "tokens" consist of force and angle labels, "sin", "cos", "+", "-", "=", and numerical digits. As each token is written, the system announces the interpretation. If a token is either unrecognizable or is incorrectly recognized, the student can immediately correct this by tapping on the intended interpretation in the symbol sheet (Figure 4.3). (If input is unrecognized, the student can also simply rewrite it.) The token-by-token approach and symbol sheet have made it considerably easier for students to write correctly interpreted equations. We have even observed some students using the symbol sheet to correct their own problem solving mistakes when they realize they have written the wrong token.

After the student completes writing an equation, the "DONE" button must be tapped to initiate interpretation. The token-by-token approach facilitates interpretation, because an equation is essentially tokenized as it is written. However, it is possible for a single force label to be written as two tokens. For example, the label "T1" could be written as "T" and "1", as both are legal tokens. We use a simple tokenizer to correct this. (The tokenizer is actually capable of taking an entire equation as a single character string and identifying the tokens it comprises.) After the equation is tokenized, the system announces the interpretation (i.e., it speaks the equation), and the equation is then sent to a simple parser to enable error checking.

The results of parsing are stored in a matrix representation. For the class of problems we consider, a term in an equilibrium equation will consist of a sign, a force label, and a force component. The component is either the sine or cosine of an angle, or 1.0. Thus, all possible terms in an equation can be represented by a matrix in which the rows correspond to the possible force labels, and the columns correspond to the sines and cosines of the possible angles, and 1.0. A "1" in the matrix indicates that the term exists in the equation. A "-1" indicates that the term exists, and has a negative sign. A "0" indicates that the term does not exist in the equation.

Consider the problem in Figure 4.2a which has three forces, $T1$, $T2$, and W, and two angles, U, and Q. Equilibrium equations for this problem can be represented by the 3x5 matrix shown in Table 4.1. Equation 4.1 shows one of the two equilibrium equations for this problem. The first term in the equation, "T1 SIN Q", is represented by the "1" in the upper left corner of the matrix. The second term, "T2 SIN U", is represented by the "1" in row two, column three. The third term, "$-W$", is represented by the "-1" in the lower right corner.

$$T1SINQ + T2SINU - W = 0 \tag{4.1}$$

	$SINQ$	$COSQ$	$SINU$	$COSU$	1
$T1$	1	0	0	0	0
$T2$	0	0	1	0	0
W	0	0	0	0	- 1

Table 4.1: The matrix representation of Equation 4.1.

The program derives the correct equation matrices for each problem directly from the problem description. When the student's equations have been parsed, the resulting matrices are compared to the correct ones to detect errors. By comparing the matrices element by element, the program can identify missing and extra terms, and sign errors. Errors are typically announced with a brief message such as "extra F SIN Q". If the student has written the wrong force component, for example he or she has written "F SIN Q" rather than "F COS Q", there will be two errors: "F SIN Q" will be an extra force and "F COS Q" will be a missing force. In this case, the program will combine the two errors into one and announce: "did you mean F COS Q instead of F SIN Q?"

If there are multiple error messages, the system announces only one. This provides the student with an opportunity to examine the equation and possibly identify the other errors. The one message that is announced may provide the student sufficient clues to identify the other errors.

Once the student has completed all of the equations, he or she taps the "FINISH" button. If the student has completed all of the equations, he or she is congratulated. Otherwise the system informs the student that more equations remain.

4.3.5 Tutoring

Newton's Pen provides instructional guidance through two means: diagnostic error messages provided throughout the solution process, and help provided by the "HELP" button. The diagnostic error messages have been describe above. (See, for example, Section 4.3.3 and Section 4.3.4.)

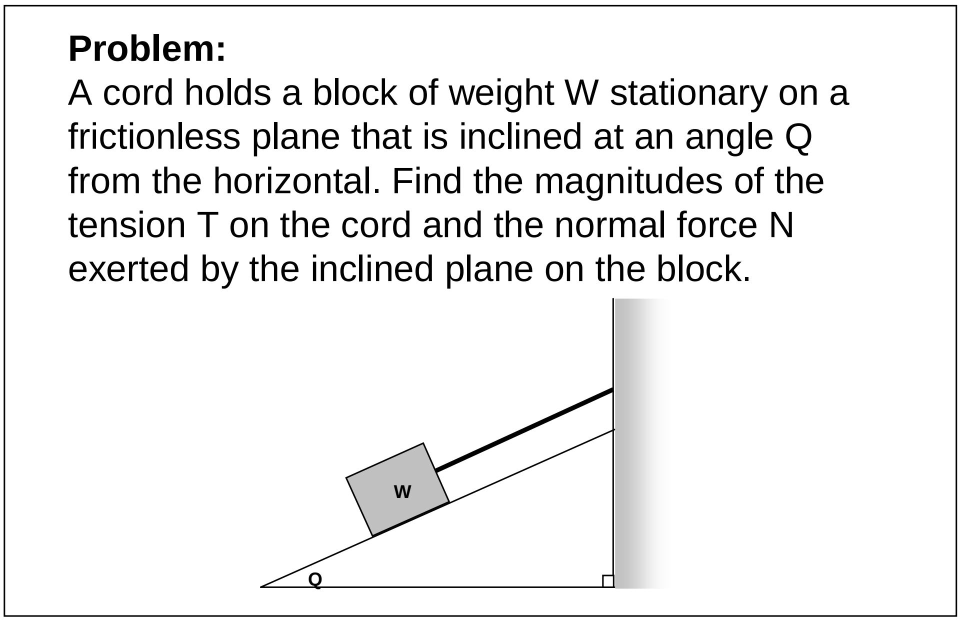
The help provided by the "HELP" button is implemented as part of Newton's Pen's finite state machine architecture. As discussed in Section 4.3.1, each state contains a set of help messages that are announced to the student with a speech synthesizer when the "HELP" button is tapped. Because the program's state mirrors the student's problem-solving progress, the help messages are always relevant. The first time a student taps the "HELP" button in a given state, he or she receives general help. Successive taps of the button result in more specific help. The first level of help is typically "program help," either describing what action the system expects the user to take, or describing how to perform an action. The second level of help typically provides a problem-solving hint, i.e., a "physics hint." The third level of help typically provides the answer to the particular problem-solving step, i.e., a "physics answer." For example, when the current state requires the student to draw the body for the free body diagram in Figure 4.2a, the sequence of help messages is: "draw the body", "the body is a ring", and "draw a circle". Similarly, when the student is writing the equilibrium equation for the x-direction, the sequence of help messages is: "write an equilibrium equation", "sum the forces in the x-direction", and finally the equation itself (e.g., "-T1 COS Q + T2 COS U = 0").

4.4 User Studies

We conducted three studies to evaluate the usability and educational value of Newton's Pen. The first study evaluated an early partial implementation of the system. The other studies evaluated the complete system described above.

All of the studies employed a similar protocol. Sessions lasted 60 to 90 minutes and consisted of five parts. First, the student used FLY's calculator and music keyboard applications to become familiar with basic interaction concepts, such as pausing after drawing and waiting for audio feedback. Second, the student solved a pretest problem, shown in Figure 4.1, using ordinary pen and paper. Third, the student used Newton's Pen in demonstration mode to "solve" a model problem. More precisely, Newton's Pen provided step-by-step instructions guiding the student in copying the provided solution to the model problem. This taught the student how to use the system and provided an example of a solved problem. Students were encouraged to carefully examine the meaning of the solution as they copied it. Fourth, the student solved a transfer problem using Newton's Pen (i.e., with scaffolding). Fifth, the student used Newton's Pen to make a second attempt at solving the pretest problem. We refer to this as the "retest" problem.

The model and transfer problems are shown in Figure 4.8 and Figure 4.9. Students who received the former for the model problem received the latter for the transfer problem, and vice versa. In most studies, we alternated the problem selections such that each problem would be used as the model problem for half of the students and as the transfer problem for the rest. However, in the group setting study (Section 4.4.3) we used the problem in Figure 4.8 as the model problem and the one in Figure 4.9 as the transfer problem for all



Problem:
A cord holds a block of weight W stationary on a frictionless plane that is inclined at an angle Q from the horizontal. Find the magnitudes of the tension T on the cord and the normal force N exerted by the inclined plane on the block.

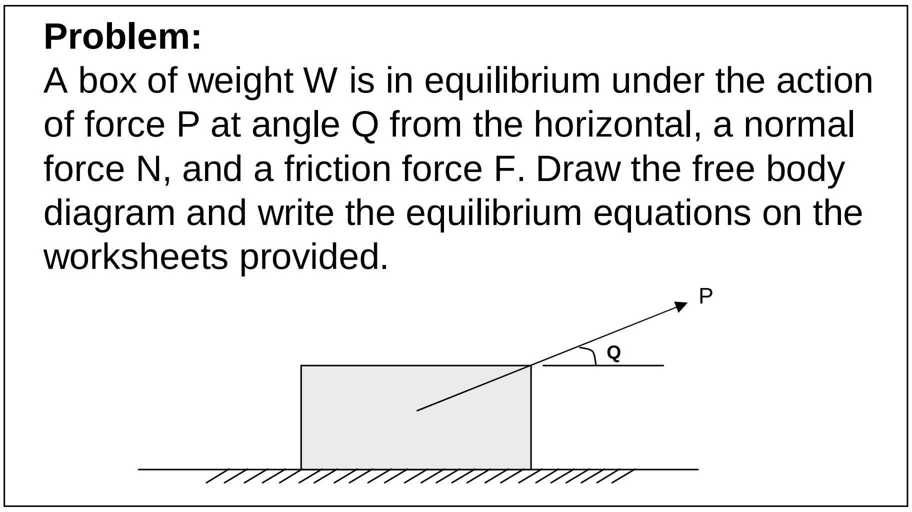
Figure 4.8: Problem concerning a mass on an inclined plane. This problem was used as both a model and transfer problem in the user studies.

students.

The remainder of this section describes the results of our three studies.

4.4.1 Study 1: Pilot Study

Early in the development of Newton's Pen, we conducted a pilot study to assess its educational value and to obtain a preliminary evaluation of the usability of its user interface. The study included nine volunteer test subjects from an introductory physics class (Physics 40A) intended for engineering and physical sciences students at the University of California, Riverside. The subjects had just completed lectures on free body diagrams and had been assigned homework problems, but had not yet begun solving them.



Problem:
A box of weight W is in equilibrium under the action of force P at angle Q from the horizontal, a normal force N, and a friction force F. Draw the free body diagram and write the equilibrium equations on the worksheets provided.

Figure 4.9: Problem concerning a mass subject to an applied force and friction. This problem was used as both a model and transfer problem in the user studies.

At the time of the pilot study, the free body diagram interpreter was implemented. The equation interpreter and the portion of the demonstration mode (see Section 4.2) in which the student practiced drawing an isolated force were simulated with a Wizard-of-Oz approach. A member of the research team read a script to simulate the audio feedback the system would have provided had this functionality been implemented.

From an educational perspective, the results were clear and encouraging: Every student was stymied by the pretest. Every student then worked carefully through the model problem in demonstration mode. Every student successfully solved the first transfer problem, and was then successful when retested on the pretest (one student did not have an opportunity to do the retest). The "diagram time" – the time from presentation of the problem to completion of the free body diagram – decreased steadily from the model to the retest. Thus, learning with evidence of transfer occurred over the course of about an hour.

Most students eventually became proficient with the interface, but initially needed assistance in using the system. The user interface clearly needed refinement, which we addressed in the current implementation. Despite the rough edges, students were extremely enthusiastic about the system. They made comments such as “it answered all of the questions I would have asked the professor” and “it was like having the professor in my hand.”

4.4.2 Study 2: User study in an individual setting

Our second user study involved student volunteers from the same introductory physics course used in the pilot study. This study was conducted one year after the pilot study. The students were given a \$10 gift certificate for their participation. At the time of the study, they had had only 10 to 15 minutes of lecture on equilibrium problems, and had not yet done any homework on the topic. These conditions represent a challenging test case, as our system is intended to by an adjunct to lecture, rather than a replacement.

Each experimental session involved a single student. The sessions were videotaped with an overhead camera that captured the student’s writing and the audio from the pen. At the conclusion of each session, the student was asked to complete a survey evaluating Newton’s Pen.

We evaluated the free body diagrams and equations from the pretests using the scales in Table 4.2 and Table 4.3, respectively. We analyzed the video by constructing event logs with events selected from a list of 133 possible event types in six categories listed in

Table 4.4. The results of the video analysis, pretests, and surveys are presented below.

Note that the software used in this study differs slightly from the description in Section 4.3. In particular, the software employed an earlier body recognizer described in [44], rather than the one described in Section 4.3.2. The former required rectangles to be drawn with four separate pen strokes, whereas the latter allowed any number of strokes, including single strokes. Additionally, the software required arrows to be drawn with a single stroke.

Score	Criteria
0	Nothing drawn.
1	No understanding, but something drawn.
2	Minimal understanding. For example, the diagram included only the body and the weight force.
3	Some understanding. The body and some forces were drawn, but there were significant errors or omissions.
4	Good understanding. The diagram was mostly correct but there were some minor errors.
5	Correct Solution. The diagram was complete and correct.

Table 4.2: Rubric for scoring the free body diagram from the pretest.

Score	Criteria
0	Nothing written.
1	No understanding, but something written.
2	Minimal understanding. The equations demonstrated some notion of equilibrium.
3	Some understanding. The equations demonstrated a clear notion of equilibrium and were partially correct.
4	Good understanding. The equations were mostly correct but there were some minor errors.
5	Correct Solution. The equations were complete and correct.

Table 4.3: Rubric for scoring the equations from the pretest.

Event Category	Description
Program Help	Clicking the “HELP” button to obtain assistance with using the program, such as help with the proper way to draw a coordinate frame.
Physics Help – Hints	Clicking the “HELP” button to obtain hints about solving a problem, such as “sum the forces in the x-direction.” Clicking the “STATUS” button is also considered a physics hint.
Physics Help – Answers	Clicking the “HELP” button to obtain the answer for a particular step in the solution, such as “draw a force up and to the right.”
Physics Errors	Errors in the free body diagram or equations, such as a force drawn in the wrong direction or a missing term in an equation.
Drawing Convention Errors	Situations in which the user does not follow the program’s drawing conventions, such as failing to draw an arrow from tail to head or failing to pause after drawing an object.
Recognition Errors	Situations in which the program incorrectly recognizes or fails to recognize correctly drawn input, such as when the system fails to recognize a correctly drawn arrow.

Table 4.4: Event categories used for analyzing the video from the experimental sessions.

Overall Performance

The average overall problem solving performance achieved on the pretest, transfer problem, and retest is presented in Table 4.5. As would be expected, students with only 10 to 15 minutes of lecture time were not able to complete the pretest accurately. The average score on the free body diagram portion of the pretest was 2.1, representing a minimal understanding. Similarly, the average score on the equation portion of the pretest was 1.2, representing a lack of substantial understanding of equilibrium.

		Pretest, Score	Transfer Problem, Time	Retest, Time
Free body diagram	Mean	2.1	4.6	3.1
	Stdev	2.0	3.7	1.6
Equations	Mean	1.2	4.7	5.2
	Stdev	1.9	2.4	2.9

Table 4.5: Overall Performance for Study 2. See Table 4.2 and Table 4.3 for the scoring rubrics. Time is in minutes.

After completing the model problem using Newton's Pen in demonstration mode, most students were then able to correctly complete the transfer and retest problems with assistance from the system. There were four exceptions. Two students were in the process of writing the correct final equation for the retest problem when the software malfunctioned. (The problem had to do with the built-in character recognizer; we later remedied the problem.) Rather than having the students restart the system, we estimated the time it would have taken them to write the final equation. A third student did not complete the y-equilibrium equation for the transfer problem. The force "N" was expected, but the student variously tried "N COS Q" and "N SIN Q". After several attempts at correcting the mistake, the student moved on to the retest problem. So that this student's data could be included in our statistics, we extrapolated from the time already spent on the equation to obtain an estimate for how long it would have taken the student to complete the problem. A fourth student arrived late to the session and was unable to complete the equations for the retest. The student nearly completed the x-equilibrium equation, but did not begin the y-equation. We again extrapolated from the time already spent on the x-equation to obtain an estimate for how long it would have taken the student to complete the problem.

On average it took students 4.6 minutes to draw the free body diagram for the transfer problem, and 4.7 minutes to write the equilibrium equations. Similarly, on average it took students 3.1 minutes to draw the free body diagram for the retest problem, and 5.2 minutes to write the equilibrium equations. Thus, the time to draw the free body diagram decreased 33% from the transfer problem to the retest, which is a substantial improvement in performance. There was actually an 11% increase in the time required to write the equations. It should again be emphasized, however, that there was a substantial improvement compared to performance on the pretest. On average, students demonstrated little understanding of equilibrium in the pretest, but they were typically able to derive the equations for the transfer and retest problems with assistance from the system.

Performance on Free Body Diagrams

Figure 4.10 presents a more detailed analysis of the students' performance in drawing free body diagrams. On average students required about two physics hints per problem on the transfer problem, but only about one on the retest. Similarly, for both problems, students tapped the "HELP" button to obtain the answer for a particular step in the solution an average of about two times per problem. Answers include the direction for a force, the shape of the body, and so on. For both problems, students averaged four problem-solving errors per problem. The errors include drawing a force arrow in the wrong direction, labeling a force or angle incorrectly, and so. While there was significant improvement compared to the pretest, students were still making use of the help system to solve problems.

Figure 4.11 presents usability issues encountered while drawing free body diagrams. This data suggests that with only a small amount of practice, students quickly become adept at using the system. Students accessed program help an average of 48% fewer times on the retest than on the transfer problem. Similarly, the number of times in which the student did not follow the program's drawing conventions decreased, on average, by 36%. The drawing conventions include pausing after drawing an object, drawing arrows tail to head, and so on. Finally, the number of recognition errors decreased, on average, by 53%. The overall number of usability problems encountered suggests that the program is reasonably easy to use. For example, students averaged only about two drawing convention errors and two recognition errors on the retest.

Performance on Equilibrium Equations

Figure 4.12 presents an analysis of the students' performance in deriving equilibrium equations. Each problem had two equations, thus the values shown in the figure must be normalized by two to obtain the statistics on a per-equation basis. For both the transfer and retest problems, students accessed the help system to obtain hints and answers an average of roughly one to 1.5 times each. For the transfer problem, students made an average of 2.1 problem-solving errors; for the retest, they made an average of 3.8. Errors include missing and extra force terms, incorrect force components (e.g., sine vs. cosine), and sign errors. One equation can contain multiple errors. For both problems, students required an average of roughly 3.8 equation writing attempts to obtain the two correct equations. This corresponds to an average of slightly less than two attempts for each of the two equations for a problem. If the "HELP" is tapped a sufficient number of times, it will provide the final answer, which the student can then copy as the solution. Students copied an average of 0.9 equations on the transfer problem, and 0.7 equations on the retest. Thus, on average, students were able to derive only a little more than half of the two equations for each problem. This is still a considerable improvement over the pretest, in which most students were unable to write any form of equation.

Figure 4.13 presents usability issues encountered while writing equilibrium equations. For the transfer problem, students accessed help on program usage an average of 3.1 times; for the retest, they accessed it an average of only 2 times. The program help includes help with the proper way to write the equation type and sign convention, and includes a prompt instructing the student to write the actual equation. Students had little trouble with the drawing conventions for writing equations. For example, on average there were only 0.2 such problems on the retest. Recognition accuracy was more of an issue. For both problems, there was an average of about 4 recognition errors per problem. "SIN" was frequently misrecognized because the program expected capital letters, but students frequently wrote this in lowercase. Additionally, FLY's character recognizer would sometimes malfunction after repeated use. (We were later able to fix this problem.)

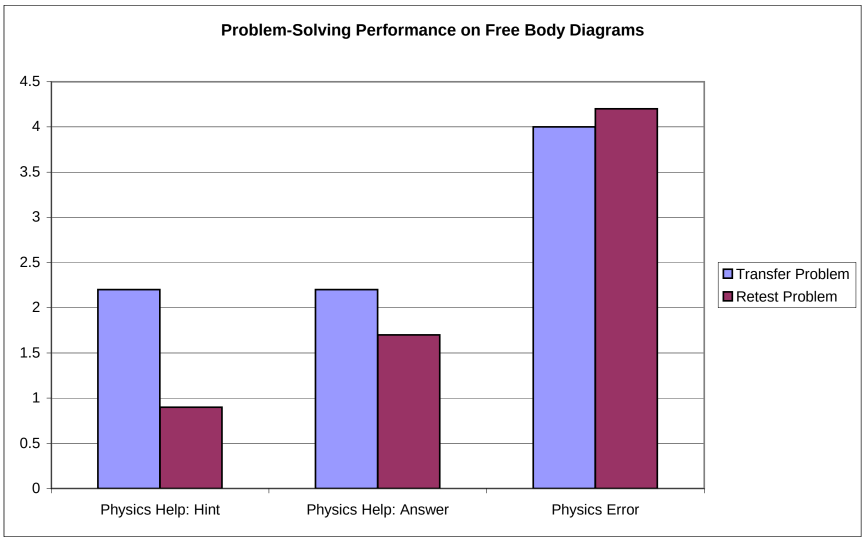


Figure 4.10: Problem-solving performance on free body diagrams for Study 2: the average number of problem-solving events per user. See Table 4.4 for a description of the events.

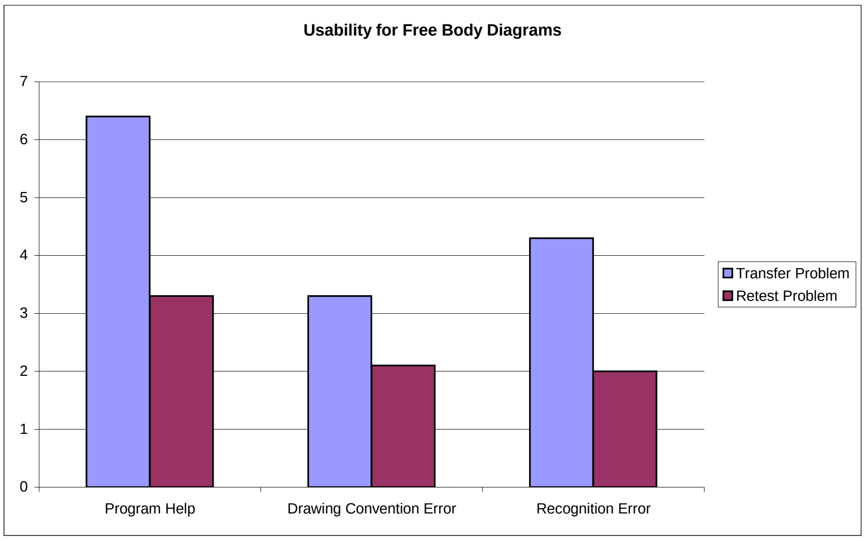


Figure 4.11: Usability issues encountered while drawing free body diagrams for Study 2: the average number of usability events per user. See Table 4.4 for a description of the events.

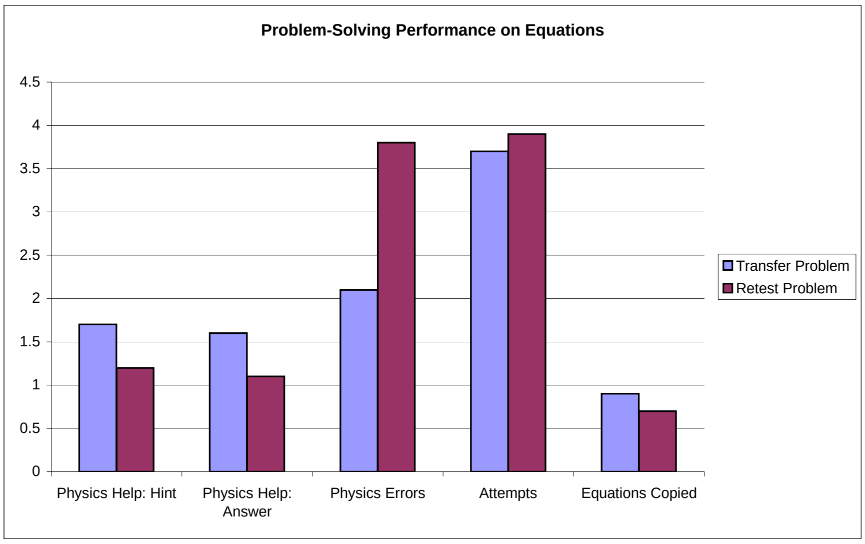


Figure 4.12: Problem-solving performance on equations for Study 2. “Physics Help: Hint” = average number of taps on “HELP” button to obtain problem-solving hints. “Physics Help: Answer” = average number of taps on “HELP” button to obtain the answer (i.e., the correct equation). “Physics Errors” = average number of errors in the equations. “Attempts” = average number of attempts required to derive the two equilibrium equations. “Equations Copied” = average number of equations directly copied from the “HELP” button.

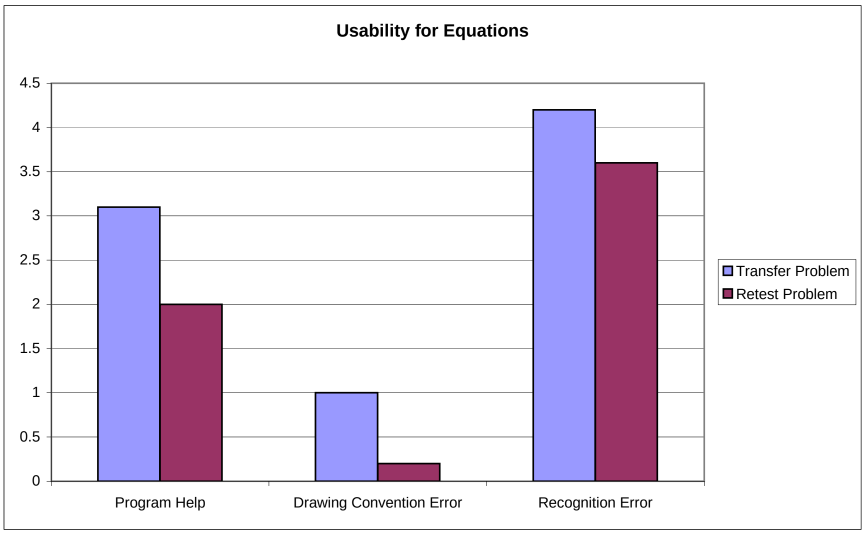


Figure 4.13: Usability issues encountered while deriving equilibrium equations for Study 2: the average number of usability events per user. See Table 4.4 for a description of the events.

Survey Results

The survey included 21 questions in which the students rated various aspects of the system including ease of drawing, usability, educational usefulness, and overall reaction. The questions for the latter topic were taken from the user study reported in [33].

Table 4.6 summarizes survey results about the ease of drawing various objects, such as coordinate systems and bodies. The data suggests that students perceived little difficulty in drawing with Newton's Pen. Arrows and bodies were rated slightly more difficult to draw than the other objects. This is likely due to the fact that arrow recognizer required single-stroke arrows, and the body recognizer required rectangles to be drawn with four separate strokes. (This study used earlier versions of our arrow and body recognizers.)

Table 4.7 summarizes survey results about usability. Students found the program to be easy to learn, and as a whole very usable. Similarly, they found recognition accuracy to be sufficient. Students uniformly liked the symbol sheet for correcting recognition errors for equations. Students perceived the system to be minimally similar to paper and pencil: They rated the system a six on scale in which 10 represents great similarity to paper and pencil, and one represents great dissimilarity. We suspect that this is a result of the basic interaction method on a pentop: after drawing an object, it is necessary to wait for interpretive feedback before proceedings. This can interrupt the flow of drawing. (See Section 4.5.) Students rated pen-based interfaces to be moderately preferable to traditional WIMP interfaces.

As shown in Table 4.8, students found that Newton's Pen presented useful tutorial feedback, and that it was useful for learning to solve equilibrium problems. Students also indicated a high likelihood of using Newton's Pen in their physics course if the system were available. Similarly, they indicated that they would likely use similar software if it were available for their other courses.

As described in Table 4.9, the students' overall reaction to the Newton's Pen was quite positive. Students considered the system to be near "wonderful" on a scale of "wonderful" vs. "terrible," near "easy" on a scale of "easy" vs. "hard," near "satisfying" on a scale of "satisfying" vs. "frustrating," and near "stimulating" on a scale of "stimulating" vs. "dull."

	Ease of Drawing	
	Mean	StdDev
Coordinate systems	9.0	1.4
Bodies	7.8	2.1
Arrows	7.5	2.5
Labels	8.8	1.2
Leader lines / arcs	8.9	1.0
Eqn types & signs	9.0	0.9
Equations	8.2	1.0

Table 4.6: Perceived ease of drawing correctly recognized objects and equations for Study 2. 10 = easy, 1 = hard.

	Usability	
	Mean	StdDev
Ease of learning program 10 = easy, 1 = hard	9.0	0.7
Recognition accuracy 10 = sufficient, 1 = insufficient	8.6	0.8
Ease of using symbol sheet 10 = easy, 1 = hard	10.0	0.0
Overall usability 10 = very usable, 1 = not usable	9.2	1.0
Similarity to paper & pencil 10 = very similar, 1 = very dissimilar	6.0	2.4
Prefer pen-based to WIMP 10 = prefer pen-based, 1 = prefer WIMP	7.2	3.2

Table 4.7: Perceived usability for Study 2.

	Educational Usefulness	
	Mean	StdDev
Usable tutorial feedback 10 = very usable, 1 = not usable	9.4	0.8
Useful for equilibrium problems 10 = very useful, 1 = not useful	9.4	0.8
Likely to use for PHY40A 10 = likely, 1 = unlikely	7.9	2.6
Likely to use for other courses 10 = likely, 1 = unlikely	7.8	2.3

Table 4.8: Perceived educational usefulness for Study 2.

	Overall Reaction	
	Mean	StdDev
Wonderful = 10, Terrible = 1	9.3	1.1
Easy = 10, Difficult = 1	8.6	1.0
Satisfying = 10, Frustrating = 1	8.0	1.4
Stimulating = 10, Dull = 1	8.1	2.1

Table 4.9: Overall reaction to Newton's Pen from Study 2.

4.4.3 Study 3: User study in a group setting

Our third user study included 19 volunteer test subjects from an introductory physics class (Physics 2A) intended for biological sciences students at the University of California, Riverside. The study was conducted during two discussion sections, but participation was optional. Students were provided with snacks and soft drinks for participating. There were 12 volunteers for the first section, and seven for the second. Each student was provided with a set of worksheets and a FLY pentop containing the Newton's Pen software. Students were allowed to work together, and some did.

This study employed the same protocol as the others. To accommodate the group setting, however, we used a document camera and projector to guide students both through the calculator and music keyboard warm up exercises and through the model problem with Newton's Pen in demonstration mode. Five members of the research team were available to assist students. Students used headphones with the system. The need for this was immediately apparent when 12 pentops were turned on simultaneously.

There are small differences between the software used in this study and that used in Study 2. In particular, Study 2 employed an earlier body recognizer described in [44], while this study used the recognizer described in Section 4.3.2. Additionally, the version used in Study 2 required arrows to be drawn with a single stroke, whereas the version used here allowed two-stroke arrows. Finally, the software used for this study included remedies for various bugs identified during Study 2. One noteworthy bug fix concerned FLY's character recognizer which sometimes stopped functioning after repeated use.

All students used the problem in Figure 4.8 for the model problem, and the one in Figure 4.9 for the transfer problem. We did not videotape the sessions, but did collect the completed worksheets, including pretests. At the end of the session, students completed the same survey used in Study 2.

At the time of the user study, the students had already completed two 50-minute lectures and had solved some homework problems on free body diagrams and equilibrium equations. The mean score on the free body diagram portion of the pretest was 3.4 (stdev = 1.6), indicating moderate understanding of the principles. The mean score on the equilibrium portion was 1.3 (stdev = 1.8), indicating a lack of substantial understanding of equilibrium. After completing the model problem using Newton's Pen in demonstration mode, all students were able to correctly solve the transfer and retest problems.

Survey Results

Table 4.10 summarizes survey results about the ease of drawing the various parts of both free body diagrams and equations. The students perceived little difficulty in drawing the parts of free body diagrams. In fact, they generally rated drawing the parts of a free body diagram easier than did the students in Study 2. This could be a result of the improved arrow and body recognizer used in this study. The students did perceive more difficulty in drawing the parts of equations, including the equation types and sign conventions, than did the students in Study 2. This is surprising given that we had fixed the bug in the character recognizer used by the equation interpreter.

Table 4.11 summarizes survey results about usability. Similar to study 2, the students found the program to be easy to learn, and as a whole very usable. Likewise, they found recognition accuracy to be sufficient, and they uniformly liked the symbol sheet for correcting recognition errors in equations. The students perceived the system to be more similar to paper and pencil than did the students in Study 2. This may be a result of increased fluidity of interaction resulting from the improved recognizers and bug fixes. The students rated pen-based interfaces to be moderately preferable to traditional WIMP interfaces, just as the students in Study 2 did.

The students rated the educational usefulness much the same way the students in Study 2 did. As shown in Table 4.12, students found that Newton's Pen presented useful tutorial feedback, and that the system was useful for learning to solve equilibrium problems. Students also indicated a high likelihood of using Newton's Pen in their physics course if the system were available. Similarly, they indicated that they would likely use similar software if it were available for their other courses.

Just as in Study 2, the students overall reaction to the system was quite positive. As described in Table 4.13, the students considered the system to be near "wonderful" on a scale of "wonderful" vs. "terrible," near "easy" on a scale of "easy" vs. "hard," near "satisfying" on a scale of "satisfying" vs. "frustrating," and near "stimulating" on a scale of "stimulating" vs. "dull."

	Ease of Drawing	
	Mean	StdDev
Coordinate systems	9.0	1.3
Bodies	8.6	1.5
Arrows	8.6	1.3
Labels	8.8	1.2
Leader lines / arcs	8.8	1.1
Eqn types & signs	8.2	1.5
Equations	7.4	2.0

Table 4.10: Perceived ease of drawing correctly recognized objects and equations for Study 3. 10 = easy, 1 = hard.

	Usability	
	Mean	StdDev
Ease of learning program 10 = easy, 1 = hard	8.6	1.3
Recognition accuracy 10 = sufficient, 1 = insufficient	8.8	0.9
Ease of using symbol sheet 10 = easy, 1 = hard	9.5	1.3
Overall usability 10 = very usable, 1 = not usable	8.9	0.9
Similarity to paper & pencil 10 = very similar, 1 = very dissimilar	7.5	1.9
Prefer pen-based to WIMP 10 = prefer pen-based, 1 = prefer WIMP	7.6	2.2

Table 4.11: Perceived usability for Study 3.

	Educational Usefulness	
	Mean	StdDev
Usable tutorial feedback 10 = very usable, 1 = not usable	9.1	1.1
Useful for equilibrium problems 10 = very useful, 1 = not useful	8.9	1.3
Likely to use for PHY2A 10 = likely, 1 = unlikely	8.2	2.2
Likely to use for other courses 10 = likely, 1 = unlikely	7.9	2.2

Table 4.12: Perceived educational usefulness for Study 3.

	Overall Reaction	
	Mean	StdDev
Wonderful = 10, Terrible = 1	8.6	1.8
Easy = 10, Difficult = 1	8.6	1.5
Satisfying = 10, Frustrating = 1	7.8	1.7
Stimulating = 10, Dull = 1	8.5	1.3

Table 4.13: Overall reaction to Newton's Pen from Study 3.

4.5 Discussion and Future Work

Our user studies suggest that Newton's Pen is an effective teaching tool. In all three user studies, students were initially unable to complete the pretest, but after about an hour with the system, were able to do so during the retest of the pretest. Students did still rely, to some degree, on help from Newton's Pen to complete the retest, but this is to be expected. For example, the students in Study 2 had only 10 to 15 minutes of classroom instruction on free body diagrams and equilibrium prior to using our system. It is entirely reasonable, under these circumstances, that the students would not achieve complete mastery of such a complex subject in only one hour with the system.

The survey results were consistently positive. Students found the system to be easy to learn and operate. They found the instructional feedback to be useful, and the system to be effective for learning statics. Furthermore, students indicated that they would be likely to use this system in class if it were available to them. Taken as whole, the survey results suggest that both the instructional and user interface designs are effective.

Despite the positive results with the current prototype, there is still much work to be done. For example, Newton's Pen can handle only a select class of problems having to do with the equilibrium of particles. For the system to be used in an undergraduate statics course, it is necessary to extend the system to a much broader class of problems including: finite bodies requiring moment equilibrium equations, friction, springs, multi-body devices, etc. This extension will require expanding the system's knowledge of free body diagrams and the range of equations it can interpret.

Many students commented that the system was slow. We interpret this to mean that the pace of interaction was slow. The pace is controlled by the interpretive audio feedback provided after each object is drawn. Interaction would clearly be more fluid without the frequent need to pause and listen. One possible remedy would be to develop techniques that enable the user to draw a few objects at once and then receive feedback in a batch. Our PaperCAD system, described in Chapter 5), explores another approach; it employs both verbose and concise audio modes. In the verbose mode, the system uses synthesized speech for feedback, just as Newton's Pen does. In concise mode, it uses short tones to indicate whether or not objects have been recognized, reducing the time necessary for the system to provide feedback. Additionally PaperCAD augments the FLY's output capabilities by using a PDA as a dynamic video display. After the user draws on a paper worksheet, the interpreted ink is displayed on the PDA. To view a different part of the page, the user taps the FLY at the desired location on the paper.

About half of the students in our user studies specifically commented that they like the step-by-step problem solving method that Newton's Pen uses. Conversely, some users have expressed a desire for more flexibility in problem solving. For example, some users like to first draw all of the force arrows, and then go back and label them. Our system requires one force to be completed before the next is drawn. More experimentation will be needed to find the proper balance between structured guidance and flexible problem-solving. We expect that the correct balance will depend on the student's level of expertise: a novice will require a more structured approach, while a student with more experience will prefer greater flexibility.

Newton's pen currently has a simple tutorial help system, which was adequate for our initial prototype. Our technique of providing successively more concrete help with each tap of the "HELP" button has proven to be an effective instructional design. However, we need to improve the content of the tutorial help. For example, if a student draws a force in the wrong direction, the system currently reports that the force is incorrect. A better approach would be to explain what the likely misconception is. For example, if the student draws the weight force inclined from vertical, and perpendicular to a surface on the body, the system could report that the student has confused weight and normal forces: weight forces are vertical while normal forces are determined by the surface normal. In this sense, the improved tutorial help system would be a form of the "buggy rules" approach to intelligent tutoring [9, 17, 5].

From our user studies, it is clear that recognition accuracy for equations needs improvement. Although equation interpretation is similar in some regards to handwriting recognition, the approaches used for improving accuracy with handwriting are not useful here. For example, some handwriting recognizers use a dictionary to improve accuracy [62]. This type of approach is ill-suited to tutoring applications, because biasing interpretation to the known correct answer of a problem could cause the system to overlook actual student problem-solving errors. Our PaperCAD system (Chapter 5) explores a more suitable approach to improving accuracy. PaperCAD corrects interpretation errors by using a hidden Markov model based on a grammar of legal equations and data about how the underlying character recognizer misclassifies.

The interface for writing equations can also use improvement. First, if a student makes a mistake in an equation, Newton's Pen requires the student to rewrite the entire equation, which can be tedious. PaperCAD explores a new interface design that enables a user to modify an equation by crossing out terms and inserting new ones. Second, if an equation has multiple errors, Newton's Pen announces only one. Some students are then able to generalize from the one error to identify other errors. It may be better to provide feedback, perhaps in the form of hints, about all of the errors at once.

Finally, our current system relies on synthesized speech for announcements. This enabled efficient development, but synthesized speech can become irritating. The user experience can be greatly improved by using recorded audio clips rather than synthesized speech.

4.6 Summary

In this chapter, we have presented Newton's Pen, a statics tutor implemented on the LeapFrog FLY pentop computer. A pentop is a writing instrument with an integrated digitizer and embedded processor. Our tutor, intended for undergraduate education, scaffolds students in the construction of free body diagrams and equilibrium equations for a selected class of problems. This project has entailed the development of sketch-understanding techniques and user-interface principles for creating pedagogically-sound instructional tools for pentop computers.

Development on the pentop platform presented novel challenges because of limited computational resources and a visually static, ink-on-paper display (the only dynamic output device is an audio speaker). We have demonstrated that a system architecture based on a finite state machine serves as a convenient means for providing context-sensitive tutorial feedback. We have also demonstrated the effectiveness of an instructional model based on a structured solution process with feedback provided at each problem-solving step. This model is designed to match the pentop's unusual user interface capabilities.

Newton's Pen is a prototype system and represents a first step towards an intelligent, interpretation-based tutoring system for pentop computers. Nevertheless, our three user studies have consistently demonstrated that our system is an effective teaching tool and that students are satisfied with the user interface.

Chapter 5

PaperCAD - A Paper-based CAD System

5.1 Introduction

The user studies we conducted to evaluate our Newton's Pen system (Chapter 4) demonstrated that it is an effective teaching tool and that students are satisfied with its user interface. Nevertheless, these studies revealed a number of open research issues. First, many students desired a faster pace of interaction. Second, recognition accuracy for equations needs improvement. Third, there is a need for a better equation writing interface which allows users to edit their equations. Fourth, for the system to be used in an undergraduate statics course, it is necessary to extend the system to a broader class of statics problems. Our PaperCAD project explores these issues in the context of a pentop interface to computer-aided design (CAD) models.

Despite the widespread availability of CAD software, paper drawings (i.e., blue prints) still play an important role. In many industries, designs are created with CAD software, but are disseminated in paper form. For example, paper drawings are widely used for building construction and in machine shops. There are a variety of reasons for this. First, the harsh conditions of construction sites and machine shop floors are not well suited to computer hardware. Second, paper drawings are inexpensive compared to computer hardware and CAD software. Third, construction workers and machinists have traditionally used paper drawings; use of CAD software would require expensive training.

PaperCAD is a prototype pentop system that introduces some of the essential functionality of computer-based CAD tools into paper-based drawings. In essence, PaperCAD uses the FLY pentop (Figure 1.1) as a bridge between the paper and digital worlds. Figure 5.1 shows an example of a simple drawing formatted for use with the system. The drawing is printed on digital paper which also contains several printed buttons, such as a help button and an abort button, which are used to execute various software functions. The user can measure dimensions by drawing conventional dimension lines. In Figure 5.1, for example, the user has dimensioned the base and height of the triangle, the angle of one of the corners, and the radius of the circle. As each dimension is drawn, the system announces its value with synthesized speech. The values are obtained by querying a digital model of the drawing, and are not scaled from the paper drawing. A symbolic label is associated with each dimension so that the value can be used in equations. The user can also measure the area of a bounded region by simply tracing the boundary. In the current example, the triangle has been traced and its area has been associated with the label T and likewise, the

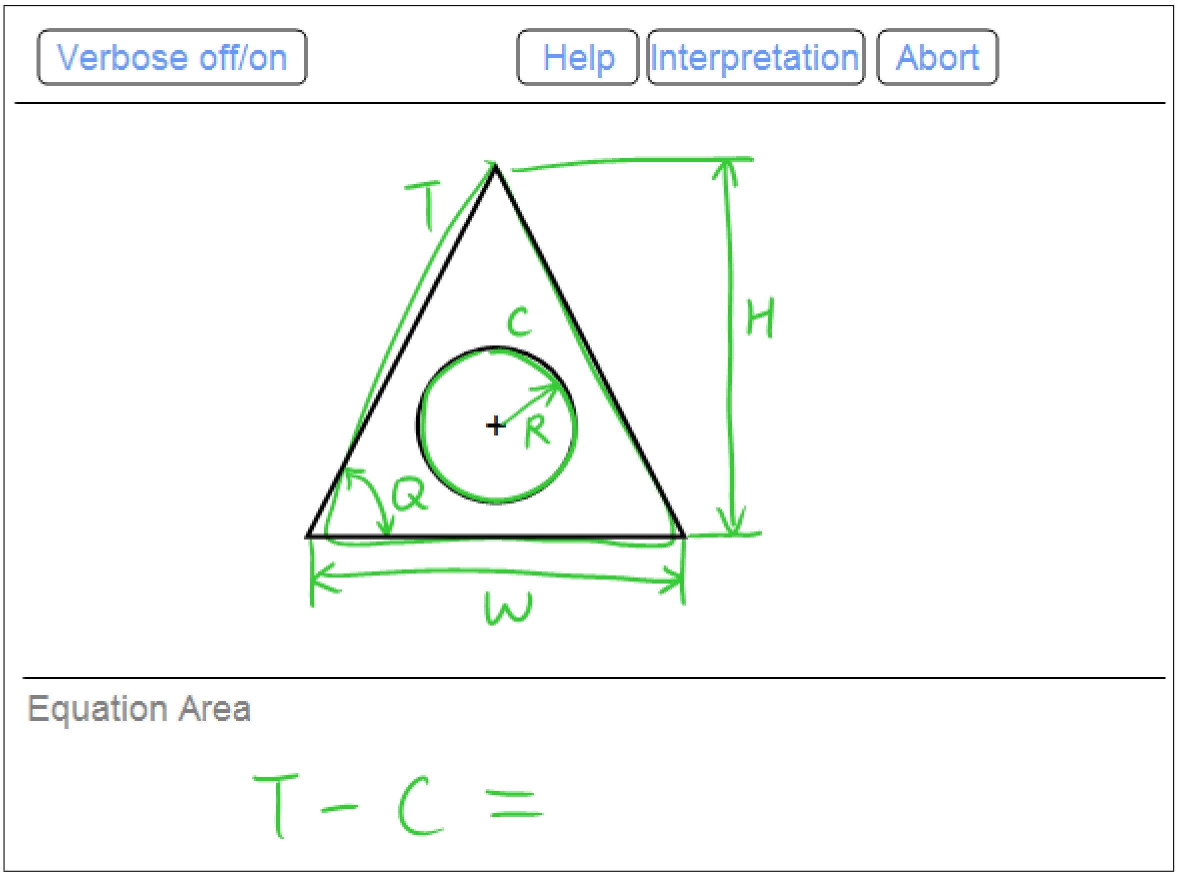


Figure 5.1: A drawing formatted for PaperCAD.

area of the circular hole has been associated with the label C. At the bottom of the page, the user can write algebraic equations. In this example, the user has written the equation "$T - C =$", which computes the area between the triangle and circle. The system interprets such equations and announces the result.

PaperCAD explores new feedback mechanisms aimed at improving the pace of interaction. The FLY is capable of providing only audio feedback. PaperCAD uses this modality and offers both verbose and concise feedback modes. In the verbose mode, the system uses synthesized speech to indicate the identity of recognized objects and to report the values of dimensions. In concise mode, the system uses tones to indicate whether or not objects have been recognized; synthesized speech is still used to report the values of dimensions. Tones allow the system to convey feedback more quickly than speech. We have also augmented the FLY's output capabilities by using a PDA as a dynamic video display. After the user draws on the paper drawing, the beautified ink and CAD drawing in the neighborhood of the pen stroke are displayed on the PDA. To view a different part of the drawing, the user taps the FLY at the desired location on the paper.

In creating PaperCAD, we also addressed several important problems related to interpreting hand-drawn input. First, we developed a recognizer for dimension arrows capable of recognizing arrows with a wide range of aspect ratios. Second, we developed a method that uses contextual information to interpret ambiguous dimensions. Third, we developed a technique that uses a hidden Markov model to automatically correct interpretation errors in hand-written equations. Fourth, developing embedded applications for the FLY is a time-consuming process (e.g., downloading code to the pen is slow). To facilitate efficient development, we created a prototyping environment that enables code to be developed, executed, and debugged on a PC. Because PaperCAD is intended as a test bed for exploring user interface design issues, offloading computation from the FLY to a PC is a reasonable expedient. In principle, however, the code could be ported to run directly on the FLY.

Our prototype provides a small but important set of functions needed by construction workers, machinists, and other users of blue prints. For example, the area measurement function is useful for estimating materials, such as the amount of concrete needed for a slab. The functionality of PaperCAD as a CAD tool is by no means complete. However its functions are important buildings block for extending Newton's Pen to a broader range of statics problems. For example, the area tracing function will be used as a means of specifying a system boundary for a free body diagram in a problem with multiple bodies. Likewise, the functionality for measuring linear and angular dimensions will be used for dimensioning free body diagrams. Similarly, the robust arrow recognizer, capable of interpreting both straight and curved arrows, will be useful for interpreting force and moment arrows in free body diagrams.

The remainder of this chapter includes an overview of the PaperCAD system, a description of our prototyping environment, and the details of PaperCAD's implementation. This is followed by the results of a user study evaluating PaperCAD's usability.

5.2 System Overview

For an engineering drawing to be complete, the size and location of every feature must be specified. However, even when a drawing is fully dimensioned, it is still frequently necessary to compute additional geometric information. For example, the diameter and location of a circular hole may be specified, but a machinist may need to determine the distance between the hole and the edge of the part.

PaperCAD enables a user to measure arbitrary dimensions from paper drawings by simply dimensioning them. Figure 5.2 shows typical examples of dimensions: Dimensions can span directly between edges of the geometric model (A), or can span between extension

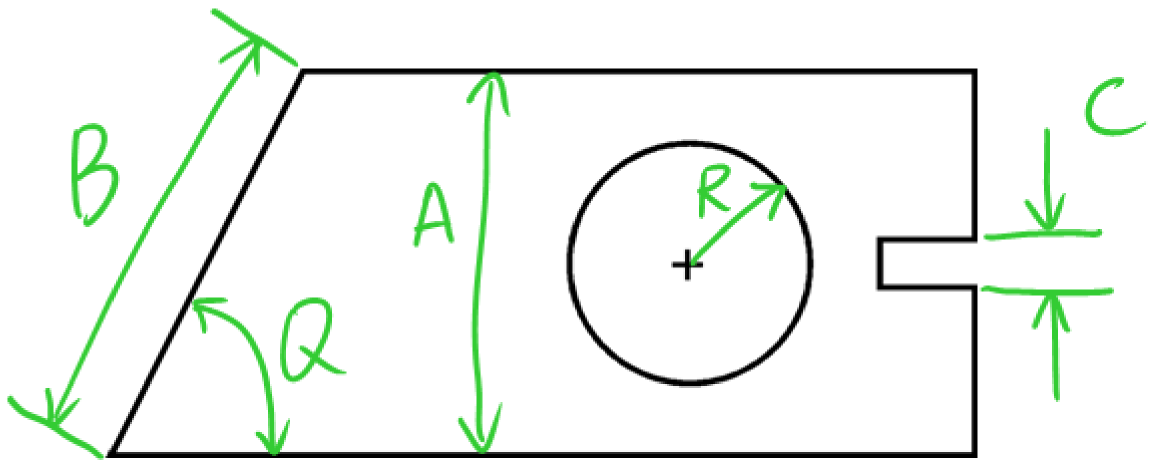


Figure 5.2: Typical dimensions.

lines (B). (An extension line is a short line segment that is parallel or perpendicular to a coordinate axis or model edge, and is drawn near a model vertex.) Linear dimensions can be aligned with a model edge (B) or a coordinate axis (A). A pair of arrows can be used to construct gap dimensions (C). Curved arrows are used to dimension angles (Q). Radii are dimensioned with single-headed arrows (R), and diameters with double-headed arrows. Areas are measured by tracing their boundary.

The user must write a symbolic label, specifically an uppercase letter, for each dimension. Labels serve as a handle for retrieving dimensions; tapping on a label causes the system to announce the value of the dimension. Labels can also be used in equations. PaperCAD has the ability to interpret and evaluate hand-written algebraic equations containing labels, numbers, mathematical operators, and the sine and cosine functions.

Drawings for use with PaperCAD must be printed onto special pages containing Anoto dot pattern. Each page has an equation area at the bottom, and a row of buttons at the top (Figure 5.1). The "Verbose Off/On" button allows the user to toggle between concise and verbose audio feedback. The system defaults to verbose mode, in which synthesized speech is used to provide interpretive feedback for all objects. In concise mode, the system plays "success" and "failure" tones to indicate whether or not objects have been recognized. Even in concise mode, however, synthesized speech is still used to report the values of dimensions. The "Help" button provides guidance for using the system and assistance with errors, such as when the system reports that the input is invalid. The "Abort" button is used to cancel operations.

PaperCAD provides several methods for correcting interpretation errors. The "Interpretation" button allows the user to select alternative interpretation for misrecognized graphical objects. For example, if an extension line is misrecognized as a dimension arrow, the user can tap the "Interpretation" button until the extension line interpretation is selected. Interpretation errors for labels and equations can be corrected by drawing a cross-out gesture ("\") through the symbol or symbols and rewriting them. Additional terms can be added to an equation by drawing a caret ("$\wedge$") at the appropriate location and writing the new terms.

PaperCAD relies on a draw-and-pause paradigm in which the user pauses after drawing each object to receive interpretive audio feedback. For example, once the user has completed drawing a dimension arrow, the system announces the type of dimension, such as model-aligned, and its value. It is important that interpretive feedback be provided after each object is drawn, because otherwise it would be difficult to associate the feedback with individual objects.

In addition to audio feedback, PaperCAD also provides visual feedback via a PDA, although the latter is not required for use of the system. After the user draws on the paper (i.e., the user does not draw on the PDA), the beautified ink and CAD drawing in the neighborhood of the pen stroke are displayed on the PDA. The user can tap the FLY on the paper to view that location on the PDA.

5.3 Prototyping Environment: ProtoPaper

Developing applications for pentop computers is a slow process consisting of repeated cycles of (cross) compiling, downloading, and testing. To facilitate efficient prototyping of pentop applications, we have created a development environment called "ProtoPaper" that combines a LeapFrog FLY pentop computer with a traditional PC, as shown in Figure 5.3. The FLY streams ink data to the PC via a debugging cable. The data is then processed by the prototype application running on the host PC. This development environment enables developers to focus on user interface design issues, rather than issues related to development for embedded processors, such as the lack of a file system and limited memory. In principle, once an application has been developed on the host PC, it can be recompiled to run directly on the FLY. In practice, however, this can require additional debugging to remedy memory issues and other similar problems.

ProtoPaper provides a rich set of feedback mechanisms for the application developer to explore. First, ProtoPaper produces audio output via the PC's speakers. This emulates the functionality of the FLY. ProtoPaper can play audio files and produce speech using a text to speech converter. Second, ProtoPaper has a simulated 32-character display,

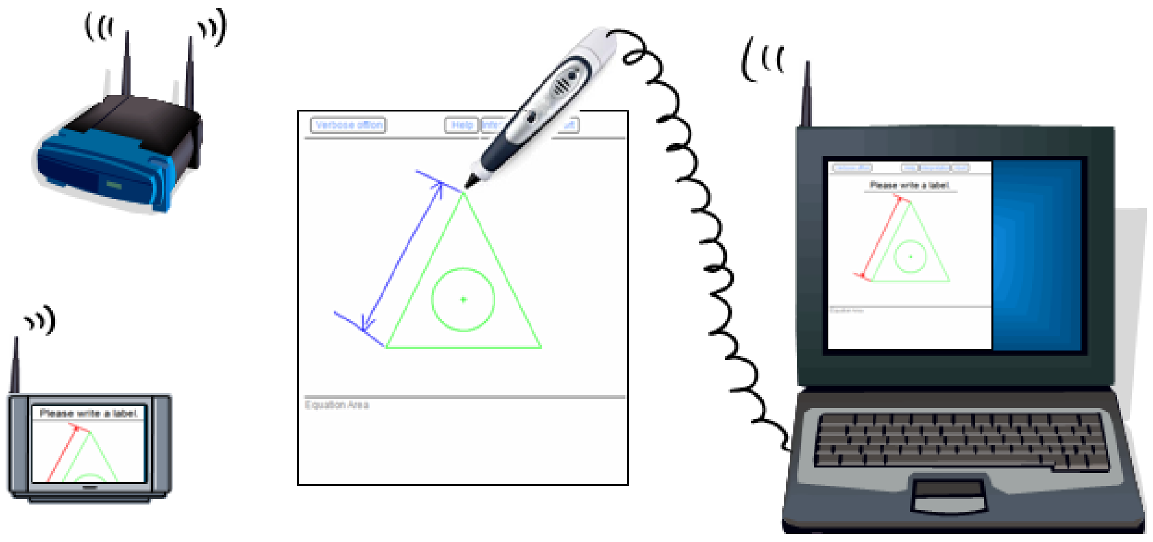


Figure 5.3: ProtoPaper prototyping environment used to implement PaperCAD. System users do not see the PC.

which is rendered as a small window on the host PC's display. Any message played by the text to speech engine is automatically displayed on the simulated character display. Third, ProtoPaper wirelessly transmits text and image data to a Nokia N800, a PDA with an 800x480 color display. Any message played by the text to speech engine is automatically displayed on the N800. Additionally, on each pen-up event, an image of the digital paper near the pen-up location is displayed on the N800, including any graphics preprinted on the page, and beautified versions of the ink. Note that the N800 is used only as an output device.

The N800 communicates with the host PC via a web server integrated into ProtoPaper. ProtoPaper dynamically generates a web page comprised of the text of the most recent audio message and an image of the appropriate region of the digital paper. The web page is regenerated each time a drawing event occurs or an audio message is played. The

N800's web browser refreshes its view of the page at a frequency of one Hz.

To facilitate debugging, ProtoPaper provides a virtual display of the digital paper on the host PC. Ink drawn on the physical paper is automatically rendered on the virtual paper. The developer can also render additional graphical information, such as beautified versions of recognized objects. To facilitate user studies, ProtoPaper generates time-stamped logs containing a variety of data including digital ink, gestures, button presses, audio messages, and internal debugging messages. The system can replay the log files and display the virtual paper on the host PC's display.

5.4 System Details

Each time the user pauses, PaperCAD attempts to recognize the ink. Ink can represent extension lines, dimension arrows, text labels, editing gestures, and equations. If ink drawn in the drawing area of the page (as opposed to the equation area) represents a recognizable object, the system reports its identity. Alternatively, if the ink is unrecognizable, the system announces that the input is invalid. In the equation area, if the ink can be interpreted as an editing gesture, it is processed immediately. Otherwise, the system delays processing until an equal sign is detected, at which time the ink is interpreted as an equation. (Equations must end with an equal sign.) The following sections describe the various interpretation algorithms in detail.

5.4.1 Extension Line Recognizer

An extension line is a straight line segment that is parallel or perpendicular to a coordinate axis or model edge, and that is drawn near a model vertex. Extension lines serve as datums for measuring dimensions. We assume that extension lines are drawn with single pen strokes. A linear least squares line fit is used to determine if a candidate pen stroke is a straight line. Specifically, if the average perpendicular distance from the stroke's data points to the least squares line is less than or equal to 1.5% of the stroke length, the stroke is considered to be a straight line. The linear least squares algorithm fails for nearly vertical lines. To avoid this, we fit x as a function of y, rather than y as a function of x, if the height of the stroke's bounding box is greater than the width.

An extension line is considered to be near a model vertex if one of its endpoints is within a threshold distance to the vertex. We use a tolerance equal to 5% of the average of the height and width of the model's bounding box. This approach is intended to produce a larger tolerance for large drawings, and a smaller tolerance for small drawings.

The intended orientation of an extension line is determined by considering the local context. If an extension line is within 14° (i.e., 4% of 360°) of being parallel or perpendicular to a model edge or coordinate direction, then the program hypothesizes that user may have intended it to be parallel or perpendicular, respectively. If an extension line is nearly parallel or perpendicular to multiple model edges and/or coordinate directions, multiple hypotheses are generated. For example, in Figure 5.4, three hypotheses are generated: the extension line could be perpendicular to edge AB, perpendicular to edge BC, or parallel to the y-axis. The program will maintain all three hypotheses until the extension line is used

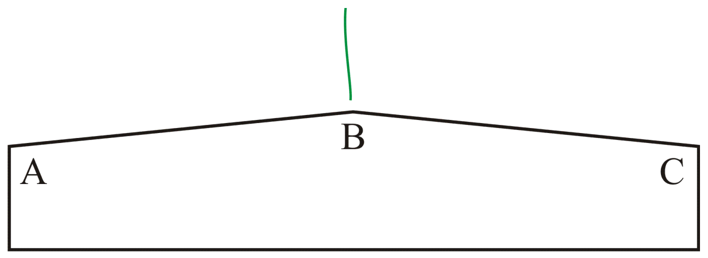


Figure 5.4: Ambiguous extension line.

as datum, at which time additional contextual information will be used to determine the intended orientation. For example, if a dimension line is drawn between this extension line and another that is vertical, this extension line will be interpreted as a vertical one, and the dimension will be reported as a horizontal distance.

5.4.2 Arrow Recognizer

Dimension arrows are challenging to recognize because the aspect ratio varies greatly. For short arrows, the arrowhead width may be comparable to the shaft length, whereas for long arrows the width may be comparatively insignificant. In fact, for long arrows, the arrowhead can be confused with the kind of hooking that often occurs at the ends of digitized pen strokes. To avoid these difficulties, we have developed an arrow recognizer that first decomposes a candidate arrow into a shaft region and head regions. In so doing, the program is able to examine the arrowhead at an appropriate scale regardless of the length of the shaft.

Arrows can be drawn with multiple pen strokes. When this occurs, the strokes are combined into a single equivalent pen stroke prior to recognition. To begin, the stroke that was drawn second is added to the first drawn stroke by joining them at their closest ends with a straight line segment, thus producing a combined stroke. Subsequent strokes are added to the evolving combined stroke until a single combined stroke is produced. We have found that people often draw the shaft of an arrow first. Thus, this procedure effectively attaches the arrowheads to the shaft. Interestingly, however, the approach has proven to work even when the shaft is not drawn first.

Once all of the pen strokes have been combined, the equivalent single pen stroke is resampled to produce 100 evenly spaced points. The inverse curvature, defined in Section 4.3.2, is then computed at each of the sample points. While in Section 4.3.2 the inverse curvature representation was directly used to recognize arrows, here it is used only to distinguish between the shaft and the candidate heads. The shaft of an arrow is identified as the largest set of consecutive data points for which the inverse curvature is greater than zero. For example, Figure 5.5a shows a single-headed arrow and its inverse curvature representation. In this case, the shaft extends from data point 1 to 61. Figure 5.5b shows a double-headed arrow and its inverse curvature representation. In this case, the shaft extends from data point 28 to 81. Data points that precede and follow the shaft are considered to be potential arrowheads. To facilitate recognition of the arrowheads, it is desirable for the candidate arrowheads to contain a small piece of the shaft. Thus, candidate arrowheads are augmented with eight data points from the appropriate end of the shaft. For example, in Figure 5.5b, the two candidate arrowheads consist of data points 1 to 36 and 73 to 100. Augmenting the arrowheads in this fashion also ensures that each candidate arrow has two candidate arrowheads.

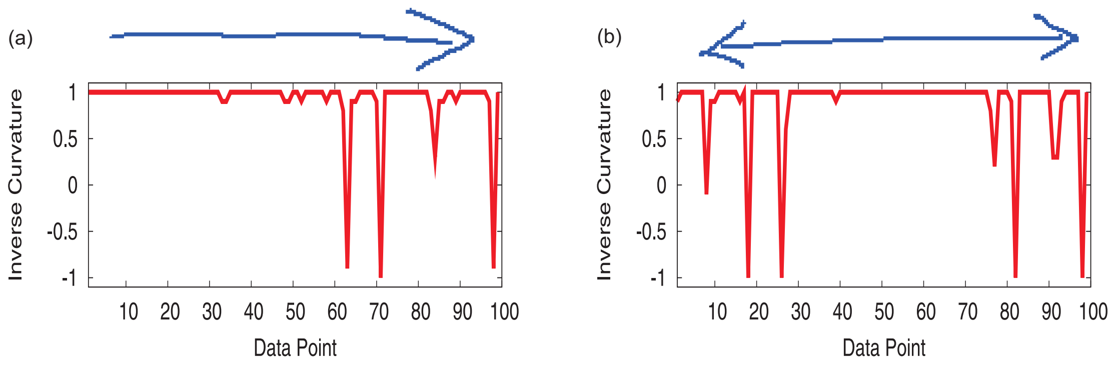


Figure 5.5: The inverse curvature of (a) a single-headed arrow and (b) a double-headed arrow.

Once the two candidate arrowheads for a candidate arrow have been located, we use a bitmap representation and a neural network classifier to determine if the candidate arrowheads actually are arrowheads. To begin, a candidate arrowhead is uniformly scaled and sampled to produce a 17x17 binary bitmap. The bitmap is then transformed into a sort of distance map. Black pixels in the binary bitmap are represented by a +1 in the distance map. Non-black pixels that neighbor black pixels are represented by a 0 in the map. All other pixels are represented by a -1. The map is essentially a gray-scale image in which three shades of gray (black, gray, and white) are used to blur the original image. We have found that using this form of gray-scale image results in higher recognition accuracy than the binary bitmaps.

The gray-scale bitmaps are classified with a neural network consisting of an input layer with 289 units, a hidden layer with 20 units, and an output layer with 12 units. Each input unit corresponds to a pixel in the 17x17 gray-scale image. Each output unit corresponds to a particular arrowhead orientation or to a particular type of non-arrowhead. The network classifies arrowheads as having an orientation of 0^o, 45^o, 90^o, 135^o, 180^o,

225^o, 270^o, or 315^o. The non-arrowhead classifications include horizontal line, vertical line, diagonal line with positive slope, and diagonal line with negative slope.

If both candidate arrowheads from a candidate arrow are classified as arrowheads, the candidate arrow is a double-headed arrow. Likewise, if only one candidate arrowhead is classified as such, the candidate arrow is a single-headed arrow. Otherwise, the candidate arrow is not an arrow. Double-headed arrows are further classified as curved arrows for measuring angles, and straight arrows for measuring linear dimensions. The distinction is based on the straightness of the shaft. If the shaft is a straight line according to the definition of straight used for extension lines (see above), the arrow is assumed to be straight, otherwise it is curved.

To train the neural network, we collected sample arrows from seven engineering students at the University of California, Riverside. Each subject was asked to provide 211 arrows that varied in length and orientation. To ensure systematic data collection, subjects were provided with "targets" for the arrows. For example, subjects were provided with boxes with various lengths and orientations and were asked to draw arrows that spanned them. In this fashion, subjects provided double-headed arrows with lengths of 2, 4, 8, and 16 cm and orientations that varied in 45^o increments. Using a similar approach, subjects provided examples of single-headed arrows, radial dimensions (single-headed arrow), diametral dimensions (double-headed arrows), and angular dimensions (curved double-headed arrows). Finally, to obtain additional variety in the data, each test subject was asked to draw 10 arbitrary linear dimensions, each comprised of a pair of extension lines and a double-headed arrow.

As the neural-network classifies only the arrowhead portion of an arrow, it was necessary to extract the arrowheads from the sample arrows prior to training the network. This was done using the inverse curvature approach described above. As part of the training process, we used a cross validation approach to determine the optimal number of nodes in the hidden layer of the network. The network was trained on data from six subjects and was tested on data from the seventh. (The curved arrows were not used for training, but were used for testing.) We determined that 20 is the optimal number of nodes. For this network topology, the average classification accuracy for the seven iterations of cross validation was 96.8%. Here, accuracy is defined in terms of the number of correctly classified arrows. For an arrow to be correctly classified, both of its candidate arrowheads must be correctly classified.

5.4.3 Interpreting Dimensions

Once a dimension arrow has been recognized, PaperCAD must determine which dimension it represents. The program searches for suitable datums near the ends of the arrow. Datums include extension lines, model edges, and crosses at the centers of arcs. If the arrow spans between compatible datums, the program reports the appropriate dimension. For example, if the tail of a single-headed arrow is at the cross of an arc, and the head is on the arc itself, the program reports the radial dimension. Likewise, if a double-headed arrow connects two parallel extension lines, the program reports the distance between them. Recall that the program may maintain multiple interpretations for the orientation of an extension line. When an ambiguous extension line is related to another datum via an arrow, the program uses the additional context to identify the intended orientation of the extension line. The program selects an orientation that is consistent with the orientation of both the other datum and the arrow.

Context is also used to correct errors from the arrow recognizer. For example, if both ends of a single-headed arrow touch parallel datums, such as two extension lines, the program assumes that the arrow was misrecognized and was intended to be a double-headed arrow. Similarly, if the two ends of a straight double-headed arrow touch datums that intersect one another, the program assumes that the arrow was intended to be an angular dimension (curved arrow). Similar reasoning is used to correct other interpretation errors.

5.4.4 Tracing Gesture Recognizer

A user can query the area of a bounded region by tracing its perimeter with one or more pen strokes. PaperCAD identifies tracing gestures by first identifying the set of model edges that are near the ink. To facilitate this, the model edges are sampled: Each line is sampled with 100 equally spaced points, and each arc is sampled with 180 equally spaced points. If 80% of the sample points of an edge are near ink data points, we assume the edge was traced. The threshold for "nearness" is the same as used with extension lines (i.e., 5% of the average of the height and width of the model's bounding box). The value of 80% was selected as a compromise to allow some amount of sloppiness, without a high risk of false positives. If the set of traced edges forms a closed polygon, the gesture is assumed to be a tracing gesture.

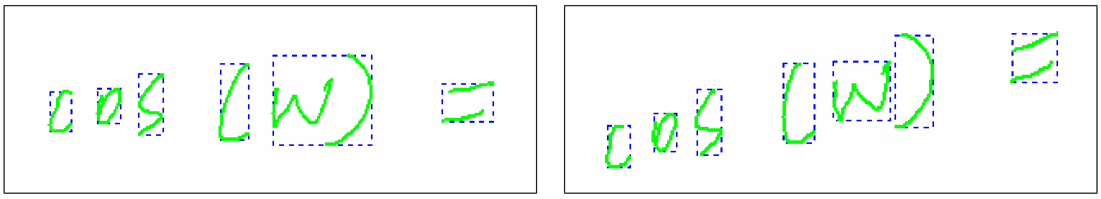
5.4.5 Text Label and Edit Gesture Recognizer

PaperCAD requires the user to provide a symbolic label for each dimension arrow and traced area. To increase recognition accuracy, labels are restricted to single uppercase letters. Labels are recognized using Microsoft's Tablet PC handwriting recognizer [62]. This recognizer is also used to recognized the cross-out gesture ("\") and the caret gesture ("∧") used for inserting new terms into an equation.

5.5 Equation Interpretation

Equations are interpreted using a three step process. First, the ink is segmented into individual characters. Then, the Tablet PC handwriting recognizer is used to classify each of the characters. The resulting interpretation is often unreliable. Thus, the final step is an error correction process in which a hidden Markov model (HMM) is used to correct interpretation errors.

A simple approach to segmenting an equation into characters is to group stokes with overlapping bounding boxes as shown in Figure 5.6a. However, if characters are drawing too close together or are slanted, this simple approach can fail. For example, in Figure 5.6a, "W" and ")" are incorrectly clustered together. We have found that we can obtain more accurate clustering results by rotating the characters a small amount prior to the bounding box calculation. This minimizes the effect of slanting. Empirically, we have found that rotating the entire equation 11° counter-clockwise produces the optimal result. For example, when the equation in Figure 5.6a is rotated by 11°, "W" and ")" are correctly



(a) (b)

Figure 5.6: (a) Simple bounding box approach to clustering. (b) Rotating the equation 11° counter-clockwise before constructing bounding boxes results in more accurate clustering.

clustered as shown Figure 5.6b. After the clustering is computed, the equation is rotated 11° clockwise so that the characters are not distorted when they are sent to the character recognizer. Note that because the two strokes of an equal sign are disconnected, they are treated as a special case during clustering.

Once an equation has been segmented, each cluster is classified with the Tablet PC handwriting recognizer. To improve accuracy, the recognizer is biased to return one of 44 legal symbols which include capital letters, numerical digits, '(', ')', '+', '-', '*', '/', '.', and '='.

The Tablet PC recognizer is intended for cursive rather than printed characters. Consequently the recognition results are somewhat unreliable. Some common errors can be easily identified and repaired. For example, equal signs are often interpreted as a pair of minus signs. Thus, our program reinterprets a pair of consecutive minus signs as an equal sign. However, most errors are more difficult to identify. For example, the term "COS" may be interpreted as "C05", "(OS", etc. We use an HMM to correct these sorts of errors. With this approach, the output of the handwriting recognizer constitutes a sequence of observations (e.g., "C" - "0" - "5"), and the correct interpretations of the clusters constitutes a sequence of hidden states (e.g., “C” - “O” - “S”). We use the Viterbi algorithm [64] to find the most probable sequence of states that could have produced the sequence of observations. This is an instance of the most probable path problem.

An HMM can be represented as a three-tuple: $\lambda(A_{ij}, B_j(k), \pi_i)$. Here, A_{ij} is the state transition probability distribution, describing the probability of transitioning from state i to state j; $B_j(k)$ is the observation probability distribution, describing the probability of observing symbol k in state j; and π_i is the initial state probability distribution, describing the probability of state i being the first state in a sequence. (For a detailed overview of HMMs, please refer to [64].)

Our HMM has 49 states which include the 44 legal characters (see above) as well as 5 special states. The latter include the space character, and four “compound” states that include “SI”, “SIN”, “CO”, and “COS”. The compound states help the system to identify “SIN” and “COS”. For example, if the sequence of observations is “S” - “I” - “N”, the most probable sequence of states is “S” - “SI” - “SIN”. Our HMM has 49 observation symbols corresponding to the 49 states. The most likely observation for most states is the same as the state itself. The compound states are the exception. For example, the most likely observation for state “SIN” is “N”.

We computed the A_{ij} matrix based on a simple grammar for legal equations. Specifically, equations may contain single-letter variables, decimal numbers, “SIN”, and “COS”. Furthermore, multiplication must be explicit (e.g., $X * Y$, rather than XY), the arguments to “SIN” and “COS” must be enclosed in parentheses, and equations must end with an equal sign. From the complete grammar, we generated the set of legal state

S_i	S_{i+1}
‘ ’, ‘=’	‘ ’
op	***digit***, ***alpha***
‘(’	***digit***, ***alpha***, ‘+’, ‘-’, ‘.’
‘)’	***op***, ‘=’
‘.’	***digit***
digit	***digit***, ***op***, ‘=’, ‘)’, ‘.’
alpha_cs	***op***, ‘=’,)
‘C’	***op***, ‘=’, ‘)’, “CO”
‘S’	***op***, ‘=’, ‘)’, “SI”
“CO”	“COS”
“SI”	“SIN”
“SIN”, “COS”	‘(’

Table 5.1: Legal state transitions for equation HMM. ***digit*** = digits from ‘0’ to ‘9’; ***alpha*** = all uppercase letters; ***alpha_cs*** = all uppercase letters except ‘C’ and ‘S’; ***op*** = ‘+’, ‘-’, ‘*’, and ‘/’.

transitions shown in Table 5.1. (The A_{ij} matrix used in our user study was based on a slightly different grammar.) For example, the first row of the table indicates that both a space and an equal sign can be followed by a space. We computed the complete A_{ij} matrix by assuming that all legal transitions from a state were equally likely, with the total probability of such transition summing to 99%. Conversely, we assumed that all illegal transitions from a state were equally likely, with the total probability of such transitions summing to 1%. For example, the probability that any particular digit follows a decimal point is $0.99*(1/10)$, whereas the probability that any particular non-digit follows a decimal point is $0.01*(1/39)$. The π_i matrix was derived from the grammar in a similar fashion.

The $B_j(k)$ matrix was obtained from experimental data. Several subjects (including one of the authors of this paper) provided examples of both individual symbols and complete equations. These were processed with the Tablet PC handwriting recognizer to produce a confusion matrix. This was then used to estimate the $B_j(k)$. Because the confusion matrix was relatively sparse, the $B_j(k)$ matrix contained many zero probability entries. To make the HMM more robust, we added 10% to every value in the $B_j(k)$ matrix and then renormalized each row to 100%. This enabled the system to tolerate recognition errors not observed in the training data.

We have found that our HMM significantly improves recognition accuracy. For example, in one experiment we evaluated the system's performance on the set of equations used to train the $B_j(k)$ matrix. We compared the interpretation accuracy both with and without the HMM. Here, accuracy is defined in terms of the edit distance, which is the minimum number of character changes needed to transform the interpretation result into the correct string. Without the HMM, the average edit distance was 2.81; with the HMM, it was 1.58. The equations contained, on average, 13.17 characters. Thus, without the HMM, on average only 78.6% of each equation was correctly recognized, while with the HMM, 88.0% was correct. While this is a significant increase in accuracy, there is still room for improvement.

5.6 User Study

We conducted a user study to evaluate the performance and usability of PaperCAD. The study included 10 volunteer subjects, none of whom had provided any training data for the system. All of the subjects were Mechanical Engineering students at the University of California, Riverside. Subjects were compensated with a $15 gift certificate. Each session involved a single student and lasted about an hour. Sessions were videotaped with an overhead camera, and all digital data, such as pen strokes and button clicks, was logged to a hard disk.

Each session began with a warm-up exercise in which the subject practiced using the FLY's calculator application. This provided experience with the feel of the device and the draw-and-pause user interaction paradigm. Next, the subject viewed a brief slide presentation illustrating how the system is used to make measurements on a drawing and evaluate equations. Finally, each subject was provided with a one-sheet reference guide for the system. The guide provided examples of typical types of dimensions, a list of editing gestures, and tips for writing equations. With the guide in hand, each subject was asked to perform the following tasks:

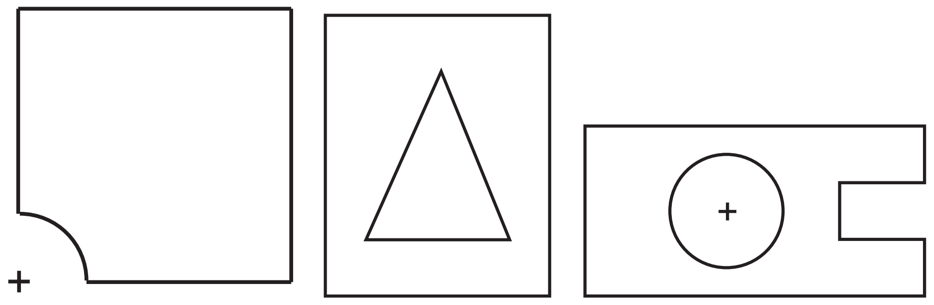
1. Using verbose audio without the LCD: Measure the height and width of a rectangular plate with a quarter circle cutout at one corner (Figure 5.7a). Measure the angle of one corner of the plate. Measure the area of the plate with a tracing gesture. Write an equation to determine the area of the quarter-circle cutout.

2. Using verbose audio without the LCD: Measure the linear and angular dimensions of a triangle, and its area. Construct equations to compute the perimeter and area.

3. Using concise audio without the LCD: Repeat step (2) for another triangle.

4. Using the LCD and audio: Repeat step (2) for a parallelogram.

5. Using desired feedback methods: For a rectangle with a triangular hole (Figure 5.7b), measure the distance from the triangle's vertices to the edges of the rectangle. Compute the area bounded by the rectangle and triangle.



(a) (b) (c)

Figure 5.7: Drawings from user study.

6. Using desired feedback methods: Compute the area and perimeter of a hexagon. Measure the area of the hexagon.

7. Using desired feedback methods: Measure the radius, diameter, and area of two circles. Determine the difference between the diameters of two concentric circles.

8. Using desired feedback methods: Measure the height and width of the notch in the part in Figure 5.7c, and determine the area of the part excluding the hole.

At the conclusion of each session, the subject was asked to complete a survey evaluating the usability of the system. Nine of the users reported similar experiences, while the tenth subject was an outlier. Specifically, on most survey questions, the tenth subject's answers differed from the mean by several standard errors of the mean. For this reason, we excluded this subject from our statistics. Background questions revealed that this was the only subject who had not had a course in engineering drawing.

Table 5.2 summarizes survey results about the ease of drawing various objects, such as arrows and equations. The data suggests that study participants perceived little difficulty in drawing with PaperCAD. Ease of equation writing is rated highly, but slightly lower than the ease of drawing other objects.

Table 5.3 summarizes results related to usability. Participants found equation interpretation accuracy to be good. Likewise, they found it easy to correct equations and labels, and to use the reinterpretation button. Participants perceived that the system provided clear instructions and interpretive feedback. Interestingly, however, there was no strong preference for concise audio feedback versus verbose audio. Similarly, while participants found the LCD to be easy to use, they perceived it to be only mildly useful. Participants found the system to be easy to learn and, overall, very usable. Finally, they found the system to be very similar to paper and pencil, and they strongly preferred this system to systems based on a traditional WIMP interface.

As described in Table 5.4, the study participants' overall reaction to PaperCAD was quite positive. Participants considered the system to be near "wonderful" on a scale of "wonderful" versus "terrible," near "easy" on a scale of "easy" versus "hard," near "satisfying" on a scale of "satisfying" versus "frustrating," and near "stimulating" on a scale of "stimulating" versus "dull."

	Ease of Drawing	
	Mean	StdDev
Extension lines	9.2	0.8
Arrows	9.6	0.7
Area Tracing	9.8	0.4
Labels	9.1	1.7
Equations	8.3	1.7

Table 5.2: Perceived ease of drawing correctly recognized objects and equations. 10 = easy, 1 = hard.

	Usability	
	Mean	StdDev
Equation interpretation accuracy 10 = sufficient, 1 = insufficient	8.4	1.2
Ease of correcting labels/equations 10 = easy, 1 = hard	8.7	1.3
Ease of using reinterpretation 10 = easy, 1 = hard	9.1	1.0
Clear feedback 10 = very clear, 1 = not clear	9.9	0.3
Prefer concise audio mode 10 = concise, 1 = verbose	5.3	3.6
Usefulness of LCD 10 = very useful, 1 = not useful	5.8	3.8
Ease of using LCD 10 = easy, 1 = hard	9.7	0.7
Ease of learning the system 10 = easy, 1 = hard	9.7	0.7
Overall usability 10 = very usable, 1 = not usable	9.6	0.7
Similarity to paper & pencil 10 = very similar, 1 = very dissimilar	8.9	1.1
Prefer pen-based to WIMP 10 = prefer pen-based, 1 = prefer WIMP	8.8	1.3

Table 5.3: Perceived usability.

As a quantitative measure of system performance, we compared the equation interpretation accuracy both with and without the HMM. Here again, accuracy is defined in terms of the edit distance between the actual and correct interpretations. The equations, which were captured in log files, contained on average 6.42 characters. Without the HMM, the average edit distance was 0.79; with the HMM, it was 0.61. Thus, the HMM resulted in a 23% reduction in errors. This is not as significant as the improvement reported in Section 5.5, primarily because the equations in the user study were short and simple.

	Overall Reaction	
	Mean	StdDev
Wonderful = 10, Terrible = 1	9.2	0.7
Easy = 10, Difficult = 1	9.4	0.7
Satisfying = 10, Frustrating = 1	9.2	0.8
Stimulating = 10, Dull = 1	9.7	0.7

Table 5.4: Overall reaction to PaperCAD.

5.7 Discussion and Future Work

The survey results from our user study suggest that users are quite satisfied with PaperCAD's interpretation accuracy and user interface design. However, additional studies are needed to fully evaluate the system. First, while we have analyzed the survey results, we have not yet completely analyzed the quantitative performance of the system. Also, the study involved simple drawings. In future studies, it is necessary to consider more complex drawings typical of those used in industry. Likewise, future studies should involve construction workers, machinists, and members of other populations of blue print users.

Our HMM-based technique for correcting interpretation errors in hand-written equations has proven to be effective. In our experiments, the technique resulted in a 43% reduction in interpretation errors, which is a substantial improvement. However, we believe that additional work is needed to develop a more accurate handwriting recognizer. The 21% error rate of the current handwriting recognizer is difficult to completely overcome, even with an effective error correction technique.

We were surprised that study participants did not find the LCD to be more useful. Further investigation is needed is to understand why. This could be a result of the study design: all participants were asked to use the audio only feedback mode prior to trying the LCD. In retrospect, it would have been useful for half of the participants to use LCD feedback first. It is also possible that the LCD was considered redundant because audio feedback was simultaneously provided. Future studies should consider the use of an LCD only mode and an audio only mode. Finally, it is possible that the slow 1 Hz refresh rate of the display may have been a hindrance. This issue should also be explored in future studies.

We were also surprised that there was no strong preference for concise versus verbose audio. It is possible that the concise audio was insufficiently concise. It is also possible that novice users prefer verbose audio feedback, and more experienced users prefer concise feedback. Again, additional studies will be needed to investigate these issues.

5.8 Summary

We have presented PaperCAD, a prototype software system that bridges between computer-based CAD tools and paper-based drawings. PaperCAD is a pentop computer application that enables users to query geometric information from CAD drawings printed on dot-patterned paper. A key issue in the design of pentop applications is how to provide effective feedback to the user. PaperCAD explores two methods of feedback: audio feedback with an adjustable level of conciseness, and a PDA that provides a video display of the portion of the drawing near the pen tip.

In addition to advances in user interface design, PaperCAD explored improved techniques for interpreting pen input. We developed an arrow recognition technique suitable for recognizing arrows with widely varying aspect ratios. An inverse curvature representation is used to locate candidate arrowheads, and a gray-scale bitmap representation and neural network are used to classify them. We also developed techniques that use context to disambiguate sketched dimensions. Finally, we developed a technique, based on a hidden Markov model, to correct interpretation errors in hand-written equations. The hidden Markov model is based on a grammar for legal equations, and the confusion matrix for the handwriting recognizer.

Experiments have shown our recognition techniques to be accurate. For example, our arrow recognizer achieved 96.8% accuracy, and our technique for correcting interpretation errors in equations achieved a 43% reduction in errors. Additionally, a user study evaluating PaperCAD's performance and usability suggests that users are quite satisfied with the interface and prefer it to traditional WIMP interfaces. While PaperCAD is a limited prototype, and additional studies are needed to further evaluate its design, this work is an important step toward the ultimate goal of bridging between the paper and digital worlds.

Chapter 6

Conclusion

This dissertation has explored user interface design principles and interpretation algorithms to enable natural user interfaces for pentop computers. A "pentop computer" is a writing instrument that is used with special dot-patterned paper, and that has an integrated digitizer and embedded processor. A pentop computer is capable of producing dynamic output in the form of synthesized speech and recorded sound clips. Creating applications for pentops is challenging because of the limited memory and computational power available. The platform also presents substantial user interface design challenges because audio is the only form of dynamic output.

The first major portion of the project was to create a trainable, multi-stroke recognizer for recognizing hand-drawn shapes. Symbols are represented internally as attributed relational graphs describing both the geometry and topology of the symbols. Symbol definitions are statistical models constructed by "averaging" the graphs of training examples. Symbol recognition is accomplished by finding the definition symbol whose attributed relational graph best matches that of the unknown symbol. This work has demonstrated that representing both the geometry and topology of a shape results in insensitivity to orientation, non-uniform scaling, and drawing order. Furthermore, constructing definitions as statistical models results in robustness to variations common in hand-drawn shapes.

We developed five efficient graph matching techniques for use with our graph based recognizer. These include: stochastic matching, which is based on stochastic search; error-driven matching, which uses local matching errors to drive the solution to an optimal match; greedy matching, which uses greedy search; hybrid matching, which uses exhaustive search for small problems and stochastic matching for larger ones; and sort matching, which relies on geometric information to accelerate the matching.

Our user studies have revealed that our recognizer is accurate and fast enough for interactive pen-based applications. The studies also revealed a number of tradeoffs between our graph matching techniques. The hybrid matcher is the most accurate but most expensive approach. Nevertheless, it is still sufficiently fast for interactive use. The stochastic matcher achieves high accuracy and can take advantage of consistency in the drawing order. If the user maintains a consistent drawing order, relatively little computation is needed. If the drawing order varies greatly, the amount of computation can be directly adjusted to maintain high accuracy. The greedy matcher provides the best tradeoff between accuracy and cost: It achieves relatively high accuracy and is fast. The error-driven matcher is accurate, but for a given amount of computation, is not as accurate as the stochastic matcher. The sort matcher is the least accurate approach and requires consistent drawing orientation. However, because this method is particularly efficient, it may be a good solution when computational resources are constrained, such as with a PDA.

The second major portion of the project explored the creation of computational techniques and user interface design principles to enable natural, pen-based tutoring systems that scaffold students in solving problems in the same way they would ordinarily solve them with paper and pencil. Toward this goal, we built Newton's Pen, a pen-based statics tutor that scaffolds students in the construction of free body diagrams and equilibrium equations, two essential components of equilibrium analysis.

Newton's Pen employs an instructional model based on a structured solution process with feedback provided at each problem-solving step. This model is implemented using a finite state machine architecture which naturally encodes the solution structure. Each state contains a list of legal inputs (i.e., things the user can write or draw), a list of next states, and help messages. This architecture provides a convenient method for providing context sensitive help: because the program's state mirrors the student's problem-solving progress, the help messages associated with a state are always relevant.

We conducted three user studies to evaluate the usability and educational value of Newton's Pen. The studies consistently suggested that Newton's Pen is an effective tutoring system. In all three studies, students were initially unable to complete a pretest, but after about an hour with the system, were able to successfully complete a new transfer problem as well as the original pretest. Surveys conducted as part of the user studies suggest that students are highly satisfied with the system's performance and user interface design.

More generally, the user study results speak to the effectiveness of using a structured problem-solving approach for pentop-based tutoring systems. Providing feedback after each problem-solving step is a good match for the pentop's unusual audio-only output capabilities. Also, this approach enables the system to provide relevant tutorial help in response to student problem-solving errors and help queries.

The third and final major portion of the project explored improved methods of providing feedback on pentop computers. As a test bed for this work, we developed PaperCAD, a system that enables users to query geometric information from printed CAD drawings. PaperCAD employs two methods of feedback: audio feedback with an adjustable level of conciseness, and a PDA that provides a video display of the portion of the drawing near the pen tip.

Results of a user study suggest that users are highly satisfied with the interface and prefer it to a traditional WIMP interface. However, we were surprised that users had no strong preference for concise versus verbose audio. It is possible that the concise audio was insufficiently concise. It is also possible that novice users prefer verbose audio feedback, and more experienced users will prefer concise feedback. We were also surprised that, although the study participants found the LCD to be easy to use, they perceived it to be only mildly useful. This could be a result of the study design: all participants were asked to use the audio only feedback mode prior to trying the LCD. In retrospect, it would have been useful for half of the participants to use LCD feedback first. It is also possible that the LCD was considered redundant because audio feedback was simultaneously provided. Additional studies will be needed to investigate these issues.

PaperCAD encompassed several advances in interpreting hand-drawn input. We developed an arrow recognition technique in which an inverse curvature representation is used to locate candidate arrowheads, and a gray-scale bitmap representation and neural network are used to classify them. Explicitly locating candidate arrowheads in this fashion enables the system to robustly recognize arrows with widely varying aspect ratios and shaft shapes. We also developed techniques that use context to improve recognition accuracy for arrows and other parts of dimension lines. Finally, PaperCAD employs a novel technique that uses a hidden Markov model to correct interpretation errors in hand-written equations. The model is based on a grammar of legal equations and data about how the underlying character recognizer misclassifies characters. This technique has proven to be effective at correcting interpretation errors.

Newton's Pen and PaperCAD are prototype systems, and there is clearly much more work to be done. Nevertheless, these systems provide a model for the creation of effective recognition-based interfaces for pentop computer systems. We are particularly encouraged by the suitability of such interfaces for education, where the technology can effectively scaffold students in the way they ordinarily work.

Bibliography

[1] Anoto group AB. `http://www.anoto.com/`.

[2] FLY pentop computer. `http://www.flypentop.com/view/page.home/home`.

[3] Emory Al-Imam and Edward Lank. PocketPad: Using handhelds and digital pens to manage data in mobile contexts. *International Conference on the Digital Society*, 0:13, 2007.

[4] Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural sketch based interface. In *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, pages 1365–1371, 2001.

[5] John R. Anderson and Brian J. Reiser. The LISP tutor: it approaches the effectiveness of a human tutor. *BYTE*, 10(4):159–175, 1985.

[6] Richard Anderson, Luke McDowell, and Beth Simon. Use of classroom presenter in engineering courses. In *Proceedings of Frontiers in Education '05*, pages T2G–13–18, 2005.

[7] Lisa Anthony, Jie Yang, and Kenneth R. Koedinger. Evaluation of multimodal input for entering mathematical equations on the computer. In *CHI '05 extended abstracts on Human factors in computing systems*, 2005.

[8] William Billingsley, Peter Robinson, Mark Ashdown, and Chris Hanson. Intelligent tutoring and supervised problem solving in the browser. In *Proceedings of ICWI*, pages 806–810, 2004.

[9] John Seely Brown, Richard R. Burton, and Kathy M. Larkin. Representing and using procedural bugs for educational purposes. In *ACM '77: Proceedings of the 1977 annual conference*, pages 247–255, New York, NY, USA, 1977. ACM Press.

[10] Brian P. Butz, Michael Duarte, and Susan M. Miller. An intelligent tutoring system for circuit analysis. In *IEEE Transactions on Education*, volume 49, pages 216–223, 2006.

[11] Chris Calhoun, Thomas F. Stahovich, Tolga Kurtoglu, and Levent Burak Kara. Recognizing multi-stroke symbols. In *Proceedings of 2002 AAAI Spring Symposium on Sketch Understanding*, pages 15–23, 2002.

[12] Philip R. Cohen and David R. McGee. Tangible multimodal interfaces for safety-critical applications. *Communications of the ACM*, 47(1), 2004.

[13] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, 18(3):265–298, 2004.

[14] Ruwanee de Silva, David Tyler Bischel, WeeSan Lee, Eric J. Peterson, Thomas F. Stahovich, and Robert Calfee. Kirchhoff's pen: A pen-based circuit analysis tutor. In *2007 Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 75–82, August 2007.

[15] Sven Dickinson, Marcello Pelillo, and Rahmin Zabih. Introduction to the special section on graph algorithms in computer science. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1049–1052, 2001.

[16] Marie-Pierre Dubuisson and Anil K. Jain. A modified hausdorff distance for object matching. In *Proceedings of 12th International Conference on Pattern Recognition*, pages 566–568, 1994.

[17] Robert G. Farrell, John R. Anderson, and Brian J. Reiser. An interactive computer-based tutor for LISP. In *Proceedings of the Fourth Annual Conference on Artificial Intelligence (AAAI'84)*, pages 106–109, 1984.

[18] Manueal J. Fonseca, Cesar Pimentel, and Jaoquim A. Jorge. CALI- an online scribble recognizer for calligraphic interfaces. In *AAAI Spring Symposium on Sketch Understanding*, pages 51–58, 2002.

[19] Kenneth D. Forbus, Ronald W. Ferguson, and Jeffery M. Usher. Towards a computational model of sketching. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*, pages 77–83, 2001.

[20] M.R. Garey and D.S. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. Freeman and Company, 1979.

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

[22] Leslie M. Gennari, Levent Burak Kara, and Thomas F. Stahovich. Combining geometry and domain knowledge to interpret hand-drawn diagrams. In *AAAI 2004 Fall Symposium: Making Pen-Based Interaction Intelligent and Natural*, 2004.

[23] Mark D. Gross. Recognizing and interpreting diagrams in design. In *ACM Conference on Advanced Visual Interfaces.*, pages 88–94, 1994.

[24] François Guimbretiére and Terry Winograd. Flowmenu: combining command, text, and data entry. In *UIST '00: Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 213–216, New York, NY, USA, 2000. ACM.

[25] Tracy Hammond and Randall Davis. Tahuti: A geometrical sketch recognition system for UML class diagrams. In *Proceedings of 2002 AAAI Spring Symposium on Sketch Understanding*, pages 59–68, 2002.

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

[27] Tracy Hammond and Randall Davis. Interactive learning of structural shape descriptions from automatically generated near-miss examples. In *IUI '06: Proceedings of the 11th International Conference on Intelligent User Interfaces*, pages 210–217, New York, NY, 2006. ACM Press.

[28] Heloise Hwawen Hse and A. Richard Newton. Recognition and beautification of multistroke symbols in digital ink. *Computers & Graphics*, 29(4):533–546, 2005.

[29] Ji Hu, Michael Schmitt, Christian Willems, and Christoph Meinel. A tutoring system for IT security. In *Proceedings of World Conference on Information Security Education*, pages 51–60, Monterey, CA, USA, 2003.

[30] Z. Huang and F. Cohen. Affine-invariant b-spline moments for curve matching. *IEEE Transactions on Image Processing.*, 5(10):1473–1480, 1996.

[31] Walter Johnson, Herbert Jellinek, Jr. Leigh Klotz, Ramana Rao, and Stuart K. Card. Bridging the paper and electronic worlds: the paper user interface. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 507–512, New York, NY, USA, 1993. ACM.

[32] Jr. Joseph J. LaViola and Robert C. Zeleznik. *MathPad*^{2}: A system for the creation and exploration of mathematical sketches. *ACM Trans. Graph.*, 23(3):432–440, 2004.

[33] Joseph J. LaViola Jr. An initial evaluation of MathPad^2: A tool for creating dynamic mathematical illustrations. *Computers & Graphics*, 31(4):540–553, 2007.

[34] Levent Burak Kara. *Automatic Parsing and Recognition of Hand-drawn Sketches for Pen-based Computer Interfaces.* PhD thesis, Carnegie Mellon University, 2004.

[35] Levent Burak Kara, Leslie M. Gennari, and Thomas F. Stahovich. A sketch-based interface for the design and analysis of simple vibratory mechanical systems. In *Proceedings of 2004 ASME Design Engineering Technical Conferences, DETC'04*, 2004.

[36] Levent Burak Kara and Thomas F. Stahovich. Hierarchical parsing and recognition of hand-sketched diagrams. In *Proceedings of the 17th annual ACM symposium on User Interface Software and Technology (UIST '04)*, pages 13–22. ACM Press, 2004.

[37] Levent Burak Kara and Thomas F. Stahovich. An image-based trainable symbol recognizer for sketch-based interfaces. In *AAAI 2004 Fall Symposium: Making Pen-Based Interaction Intelligent and Natural*, 2004.

[38] Levent Burak Kara and Thomas F. Stahovich. An image-based, trainable symbol recognizer for hand-drawn sketches. *Computers & Graphics*, 29(4):501–517, 2005.

[39] Andreas Kosmala, Gerhard Rigoll, Stéphane Lavirotte, and Loïc Pottier. On-line handwritten formula recognition using statistical methods. In *Proceedings of the Fourteenth International Conference on Pattern Recognition*, pages 1306–1308, 1998.

[40] Gordon Kurtenbach and William Buxton. User learning and performance with marking menus. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 258–264, New York, NY, USA, 1994. ACM.

[41] Gordon Paul Kurtenbach. *The Design and Evaluation of Marking Menus*. PhD thesis, Toronto, Ont., Canada, Canada, 1993.

[42] James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer*, 34(3), 2001.

[43] Seong-Whan Lee. Recognizing hand-drawn electrical circuit symbols with attributed graph matching. In H S Baird, H Bunke, and K Yamamoto, editors, *Structured Document Image Analysis*, pages 340–358. Springer-Verlag, 1992.

[44] WeeSan Lee, Ruwanee de Silva, Eric J. Peterson, Robert C. Calfee, and Thomas F. Stahovich. Newton's pen - a pen-based tutoring system for statics. In *2007 Eurographics Workshop on Sketch-Based Interfaces and Modeling*, August 2007.

[45] Chunyuan Liao, François Guimbretière, and Ken Hinckley. PapierCraft: a command system for interactive paper. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 241–244, New York, NY, USA, 2005. ACM.

[46] Chunyuan Liao, François Guimbretiére, and Corinna E. Loeckenhoff. Pen-top feedback for paper-based interfaces. In *Proceedings of the 19th annual ACM symposium on User interface software and technology (UIST '06)*, pages 201–210, 2006.

[47] Josep Lladós, Enric Martí, and Juan José Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1137–1143, 2001.

[48] Josep Lladós, Ernest Valveny, Gemma Sánchez, and Enric Martí. Symbol recognition: Current advances and perspectives. In *GREC '01: Selected Papers from the Fourth International Workshop on Graphics Recognition Algorithms and Applications*, pages 104–127, London, UK, 2002. Springer-Verlag.

[49] Wendy E. Mackay, D. S. Pagani, L. Faber, B. Inwood, P. Launiainen, L. Brenta, and V. Pouzol. Ariel: augmenting paper engineering drawings. In *CHI 95 Conference Companion*, pages 421–422, 1995.

[50] Wendy E. Mackay, Guillaume Pothier, Catherine Letondal, Kaare Boegh, and Hans Erik Sorensen. The missing link: augmenting biology laboratory notebooks. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 41–50. ACM, 2002.

[51] James V. Mahoney and Markus P. J. Fromherz. Three main concerns in sketch recognition and an approach to addressing them. In *AAAI Spring Symposium on Sketch Understanding*, 2002.

[52] M. Masry, D. J. Kang, I. Susilo, and H. Lipson. A freehand sketching interface for progressive construction and analysis of 3D objects. In *Proceedings of AAAI Fall Symposium - Making Pen-Based Interaction Intelligent and Natural*, pages 98–105, 2004.

[53] Nicholas Matsakis. Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.

[54] Jeroen J. G. Van Merrienboer and John Sweller. Cognitive load theory and complex learning: Recent developments and future directions. *Educational Psychology Review*, 17(2):147–177, 2005.

[55] Bruno T. Messmer and Horst Bunke. Efficient subgraph isomorphism detection: A decomposition approach. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):307–323, 2000.

[56] J. P. Mestre, editor. *Transfer of learning from a modern multidisciplinary perspective.* Information Age Publishing, Greenwich, CT, 2005.

[57] Antonija Mitrovic. Learning SQL with a computerized tutor. In *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education*, pages 307–311, New York, NY, USA, 1998. ACM Press.

[58] Shankar Narayanaswamy. *Pen and Speech Recognition in the User Interface for Mobile Multimedia Terminals.* PhD thesis, EECS Department, University of California, Berkeley, 1996.

[59] Sharon Oviatt, Alex Arthur, and Julia Cohen. Quiet interfaces that help students think. In *Proceedings of the 19th annual ACM symposium on User interface software and technology (UIST '06)*, pages 191–200, 2006.

[60] Joao P. Pereira, Vasco A. Branco, Nelson F. Silva, Tiago D. Cardoso, and F. Nunes Ferreira. Cascading recognizers for ambiguous calligraphic interaction. In *2004 Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 63–72, 2004.

[61] Ken Perlin. Quikwriting: Continuous stylus-based text entry. In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 215–216, New York, NY, USA, 1998. ACM.

[62] James A. Pittman. Handwriting recognition: Tablet PC text input. *Computer*, 40(9):49–54, September 2007.

[63] Stuart Pook, Eric Lecolinet, Guy Vaysseix, and Emmanuel Barillot. Control menus: Excecution and control in a single interactor. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, pages 263–264, New York, NY, USA, 2000. ACM.

[64] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[65] Chris Raymaekers, Gert Vansichem, and Frank Van Reeth. Improving sketching by utilizing haptic feedback. In *AAAI Spring Symposium on Sketch Understanding*, pages 113–117. AAAI Press, 2002.

[66] J. Rocha and T. Pavlidis. A shape analysis model with applications to a character recognition system. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(4):393–404, 1994.

[67] Robert J. Roselli, Larry Howard, Bryan Cinnamon, Sean Brophy, Patrick Norris, Megan Rothney, and Derek Eggers. Integration of an interactive free body diagram assistant with a courseware authoring package and an experimental learning management system. In *Proceedings of American Society for Engineering Education Annual Conference & Exposition*, 2003.

[68] Dean Rubine. Specifying gestures by example. *Computer Graphics*, 25:329–337, 1991.

[69] Tevfik Metin Sezgin and Randall Davis. HMM-based efficient sketch recognition. In *International Conference on Intelligent User Interfaces (IUI'05).*, New York, 2005.

[70] Michael Shilman, Hanna Pasula, Stuart Russell, and Richard Newton. Statistical visual language models for ink parsing. In *2002 AAAI Spring Symposium on Sketch Understanding*, pages 126–132, 2002.

[71] Steve Smithies, Kevin Novins, and James Arvo. A handwriting-based equation editor. In *Proceedings of Graphics Interface*, pages 84–91, San Francisco, CA, 1999. Morgan Kaufmann Publishers Inc.

[72] Hyunyoung Song, François Guimbretière, Chang Hu, and Hod Lipson. ModelCraft: capturing freehand annotations and edits on physical 3d models. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 13–22, New York, NY, USA, 2006. ACM.

[73] Georges Span. LITES, an intelligent tutoring system for legal problem solving in the domain of Dutch civil law. In *Proceedings of the 4th International Conference on AI and Law*, pages 76–81, 1993.

[74] Thomas F. Stahovich. Segmentation of pen strokes using pen speed. *AAAI 2004 Fall Symposium: Making Pen-Based Interaction Intelligent and Natural*, 2004.

[75] Siriwan Suebnukarn and Peter Haddawy. A collaborative intelligent tutoring system for medical problem-based learning. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI '04)*, pages 14–21, 2004.

[76] Victoria Tsiriga and Maria Virvou. Transferring a standalone intelligent algebra tutor over the WWW. In *Proceedings of the 4th International Workshop on Computer Science and Information Technologies*, Patras, Greece, 2002.

[77] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

[78] Kurt VanLehn, Collin Lynch, Kay Schulze, Joel A. Shapiro, Robert Shelby, Linwood Taylor, Don Treacy, Anders Weinstein, and Mary Wintersgill. The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, 15(3), 2005.

[79] Ron B. Yeh, Chunyuan Liao, Scott R. Klemmer, François Guimbretière, Brian Lee, Boyko Kakaradov, Jeannie Stamberger, and Andreas Paepcke. ButterflyNet: a mobile capture and access system for field biology research. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 571–580. ACM, 2006.

[80] Robert Zeleznik, K.P. Herndon, and J.F. Hughes. SKETCH: An interface for sketching 3D scenes. In *Proceedings of the 23rd annual conference on Computer Graphics and interactive techniques (SIGGRAPH '96)*, pages 163–170, 1996.