A GRAPHICAL RULE-BASED SYSTEM FOR RECOGNIZING ON-LINE HANDWRITTEN DIGITS

STEPHEN E. MALOWANY

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

MAY 1993

ISBN 0-315-90927-7

Abstract

A Graphical Rule-Based System for Recognizing On-line Handwritten Digits

Stephen E. Malowany

A graphical rule-based system for recognizing isolated samples of handwritten digits is presented. On-line pen stroke position data such as that acquired from a digitizer tablet or computer mouse is used as input. The system consists of four major components: a structural feature extractor, a rule-based classifier, a graphical user-interface (GUI), and a control module. The structural feature extractor segments the stroke data into sequentially connected portions from which geometric feature vectors are computed. These features are modeled after high-level structural primitives by which humans could describe the composition of characters. The feature vectors are then fed into a rule-based classifier where a heuristic rule base is applied by a conventional forward chaining inference engine. The classification proceeds by abstracting the quantitative feature vectors into *top level* qualitative elemental primitives. These primitives considered together are then matched against a knowledge base of digit templates. A list of possible digit identities with corresponding certainty factors is then concluded. The graphical user interface running under the X Window System provides "point and click" access to the system's underlying control module, data visualization of extracted features, animated display of digit samples, and the ability to draw new samples with the mouse. The control module simply interprets events generated by the user via the GUI, and dispatches commands to the various component processing modules. Some experimental results are given.

Acknowledgements

I would like to thank first and foremost, my supervisor Dr. Ching Y. Suen. He "booted" me up as a grad student, and offered me a wonderful environment in which to learn at many levels. I am grateful for his support and patience during this thesis work. I also thank Dr. Alun D. Preece, Christine Nadal, and Patrice Scattolin for helpful discussions and reviews on the topic at hand, Dr. T. Radhakrishnan for some much valued encouragement and suggestions, Paul Gill for many fun-filled Unix discussions, and last but not least, Laurie, my parents, and Morie, for that persistent barrage of "Hey Steve, how's the thesis coming along?".

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Researchers have long attempted to apply computer technology to many of our daily mundane tasks. Typical application areas include scientific calculations, business applications such as inventory, billing, payroll, and banking [17], entertainment such as movie animation, electronic music [21], and games. These applications exploit the various advantages of computer technology, namely computational speed, accuracy, and the ability to process vast amounts of data.

Computers have benefited from the recent large scale communication available through the growth of networks. Advances in telecommunications allow world wide access via telephone networks, satellite systems, and fibre optic technologies. These offer continually increasing bandwidth to share information and knowledge [53]. These advances are followed by corresponding evolution of database, on-line, and real time systems [38]. With falling costs and performance improvements of hardware in general, and microprocessors in particular, computers are finding their way into consumer products such as cameras, washing machines, automobiles, and home electronics. Thus computer researchers are striving to develop suitable hardware and software algorithms for an ever expanding range of uses including video images, sound, medical diagnostics, teaching, and aerospace.

1.1 Expert Systems

One area of computer applications which has made significant progress in the last two decades is the domain called artificial intelligence(AI), including expert systems and neural networks. In the broadest sense, artificial intelligence has been defined as "...the study of ideas that enable computers to be intelligent"[1]. An expert system has been defined as "a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice"[2]. MYCIN is a renowned early expert system which assists medical professionals in the treatment of blood infections [46]. Other successful expert systems include Prospector for the analysis of geological samples [13], and DENDRAL for the determination of the molecular structure of organic compounds [32]. Probably the most successful commercial application of an expert system is XCON developed by Digital Equipment Corporation [5]. This system is used to configure VAX computer systems according to customer orders. Besides providing expert calibre solutions, current expert systems are now expected to provide meaningful explanations for the purposes of analysis and training of non-expert users [50]. A wide spectrum of successful expert system applications can be found in the literature [24].

Many current expert systems are being implemented in the rule-based paradigm for which a number of development shells are available, such as Nexpert Object, CLIPS, ART-IM, Level5, EXSYS, and Kappa. Some of these environments offer objects as an additional scheme for knowledge representation. Early expert systems were mainly implemented in the LISP language [60], and as such were large and slow programs which required expensive special purpose hardware to run. This was an obstacle to the commercial proliferation of expert systems. However, with the development of fast and efficient pattern matching algorithms, such as the Rete [14] and Treat [37] algorithms, today's expert system shells written in the C language offer significant performance increases on general purpose computers compared to LISP-based systems [35].

[1][59], p. 1.
[2][27], p. 3.

1.2 Man-Machine Communication

In the early days of computing, punched cards and line printer listings were the standard way of programming and interfacing with computers. This was the era of batch processing. Interactive on-line computing followed with the development of teletypes and alphanumeric cathode ray tube (CRT) terminals attached to a central computer system. Ivan Sutherland then pioneered the age of interactive computer graphics with the invention of his Sketchpad system in the early 1960's [49]. Following this a rapid evolution of both hardware and software for computer generated graphics occurred. These advances included video frame buffers, light pens, digitizer tablets, and algorithms for performing rendering, clipping, shading, anti-aliasing, and hidden line and surface removal for two and three dimensional displays [18].

The advantages of utilizing graphical user interfaces (GUI) to increase the efficiency of human-computer interaction are now becoming widely known and accepted. Window-based user interfaces are now a common feature of most computer systems, and as a result users have come to expect all applications to have polished user-friendly interfaces. However, a user interface that is easy to use is seldom easy to build. In a recent survey of user interface developers, approximately half of the code and development time for their applications is dedicated to the user interface portion [39]. Hence the emergence in recent years of user interface management systems (UIMS) and GUI building tools [1]. The X Window System, a de-facto standard in the workstation environment, provides a hardware-independent window platform that allows application programmers to spend more time improving their programs and less time porting to new systems. X permits these GUI environments to be exploited in a distributed computing environment based on the client-server model [61].

Windows, icons, menus, and pointing devices constitute today's computer graphical user interface environment which has evolved from the pioneering work done at XEROX in the mid-seventies [28]. Current personal computer systems and workstations offer high resolution color bit-mapped displays to support such environments. Interactive multimedia adds compact disk(CD) quality stereo audio, in addition to full motion video [9, 22]. The modern computing trends focus on visual programming [54] and data visualization [20, 33].

The leading research in man-machine interaction is evolving in the domain known as *virtual reality*, or simply VR [41]. Multiple new input/output devices are involved in these man-machine interfaces. They include: voice recognition for speech input, 3D spatial audio outputs, head helmet with built-in position and orientation sensors, stereoscopic eye glasses display and eye position sensor for determining the user's point of focus, body suit for measuring user limb positions, gloves capable of measuring hand gestures and generating force reflection. The glove can optionally heat or cool the user's fingers! "Virtual reality presents a synthetically generated environment to the user through visual, auditory, and other stimuli. ... Users perceive themselves as being inside the scene, and they have a 3D spatial understanding of objects' locations with respect to their own body"[3].

Leading researchers in this area include Jaron Lanier of VPL Research Inc., creator of the "eyephones", the "data glove", and the "data suit", and Tom Furness of the University of Washington's Human Interface Technology Laboratory [8]. Virtual Reality promises major breakthroughs in many fields such as sports, medicine, architecture, design, as well as entertainment [10].

1.3 Character Recognition

Another interesting area of research is in the domain of handwriting and character recognition. This field is enjoying a resurgence of interest due to recent advances in recognition algorithms and I/O hardware, as exemplified in the pen-based notebook computer [43].

Character recognition is a specialization within the broader field of pattern recognition which concentrates on the interpretation and understanding by computers of machine and handwritten information. On-line character recognition involves the user inputting data directly into the computer from a writing device. Typically, the data consists of a stream of x and y position coordinates, and sometimes pressure and velocity information are also captured [40]. On the other hand, optical character

[3][41], p. 79

recognition (OCR) addresses the processing of handwritten or machine printed documents which are subsequently scanned and inputted for analysis. Reliable computer recognition of machine printed characters has been achieved and commercial systems exist for this purpose. However, computer recognition of handwritten characters remains difficult and requires continued research efforts. Comprehensive surveys on the state of the art in both on-line and optical character recognition are presented in [52] and [11].

The goal of character recognition is to identify symbols from a given alphabet or language. For example, the 26 letters of the English alphabet, Arabic characters [44], or Chinese characters [45], which may represent entire words by themselves. Classification methods can usually be categorized as either statistical or syntactical. Many different approaches to the problem of on-line character recognition have been taken utilizing a wide variety of feature types. Some of the more common methods include recognition algorithms based on Fourier coefficients [23], curve matching and other signal processing techniques [25, 26], elastic matching with local affine transformation [55], and stroke codes [45].

At Concordia University's Centre for Pattern Recognition and Machine Intelligence (CENPARMI), researchers have been studying and developing algorithms for pattern recognition of handwritten information, including document analysis, segmentation of connected handwriting, application of multi-expert combinations [12], human performance analysis [31], as well as recognition of isolated characters and numerals [29]. This thesis work combines current expert systems and man-machine communication technologies for research into character recognition at CENPARMI. A prototype system named OLDRES, an on-line digit recognition expert system, was developed and is presented here. Chapter 1 provided an introduction and overview of some relevant literature in the areas of expert systems, man-machine interface, and character recognition. Chapter 2 explains the OLDRES system architecture and the graphical user interface. Chapter 3 describes the structural feature extraction methods used, while Chapter 4 develops the rule-based classification system. Chapter 5 presents sample outputs, test results, and an analysis of the system performance before concluding in Chapter 6.

Chapter 2

On-Line Digit Recognition Expert System

This chapter describes the functionality and features of the On-Line Digit Recognition Expert System (OLDRES) and its graphical user interface (GUI). Its purpose is to facilitate the development of the recognition sub-systems by providing the researcher with an integrated environment offering efficient access to data and presenting it in an intuitive, meaningful fashion. The data visualization capabilities were especially important in refining the algorithms for the feature extraction sub-system, as well as the classification knowledge base. The functionality of the system as a demonstrational tool was also an important feature.

The first section presents an overview of the OLDRES system including the design objectives and system architecture. The services provided by OLDRES are then described from the user's perspective followed by some implementation highlights of the system, ending with a discussion of the control module.

2.1 OLDRES System Overview

2.1.1 Objectives

In the design stage of development, the basic system requirements were identified as the following:

Data Management

- Loading of selected digit samples from the training and testing databases into the system.
- Inputing of samples to the system by drawing with the mouse.

Data Processing

- Interface to and coordinate the execution of the feature extraction and rule-based classification sub-systems.

Data Visualization

- Animating the graphical display of the component strokes of digit samples.
- Graphical display of extracted structural features.

The OLDRES system implementation has realized these objectives.

2.1.2 System Architecture

OLDRES is an interactive window-based application. In contrast to the conventional command line style of programming, window applications are *event-driven*. Conventional text-based interactive programs have an internal command loop which prompts the user for input, reads in some characters, and performs some actions. This sequence repeats until the user keys-in a request to exit. Window-based applications on the other hand, surrender execution control to the event dispatcher of the window system. The application simply registers what events it is interested in, and which functions it

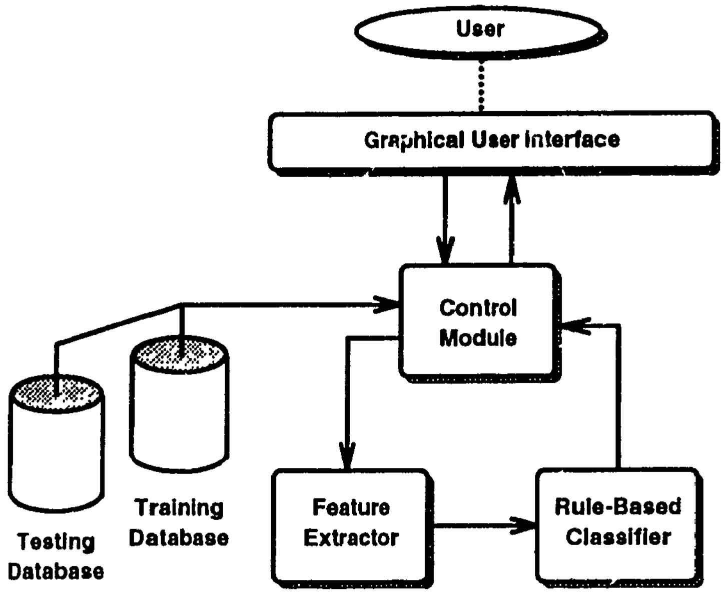
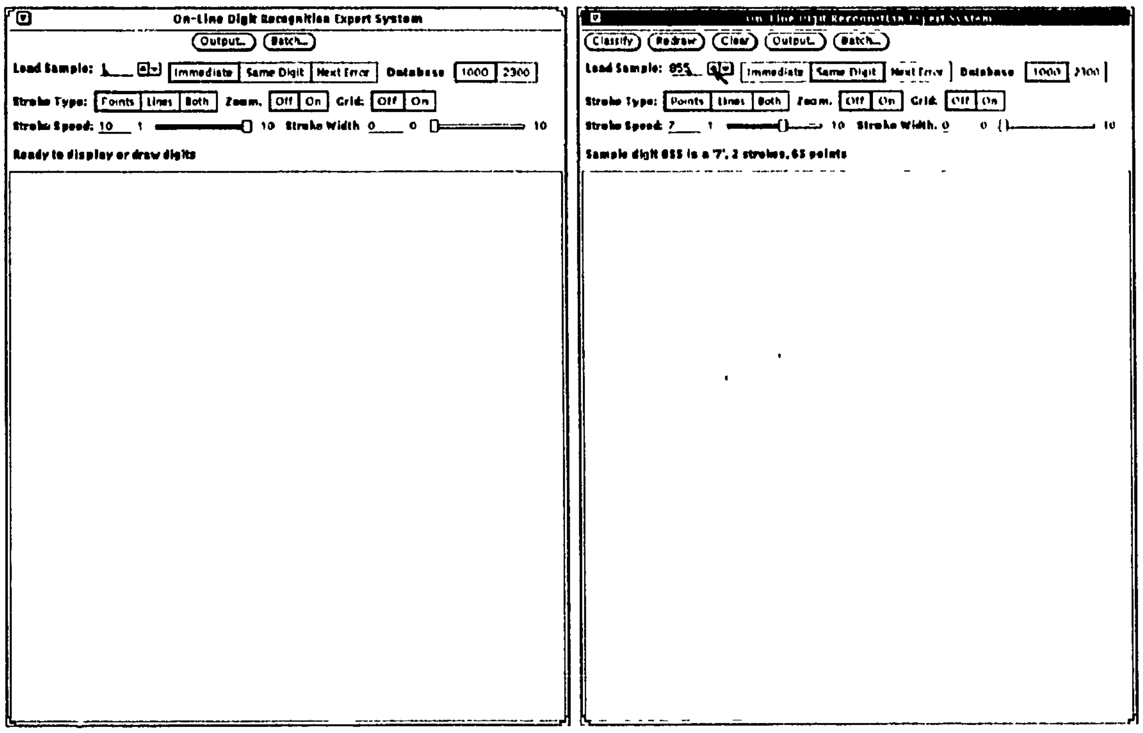


Figure 1: Block diagram of the OLDRES system architecture.

wishes to have called as a result of those events. Such functions are known as *callback functions*.

OLDRES is comprised of four principal software modules, namely the graphical user interface, the feature extractor, the rule-based classifier, and the control module. Figure 1 shows the interrelationships among these modules. Using the mouse and keyboard, the user interacts with the system by manipulating GUI objects displayed on the screen. These input events are then passed on to the control module for interpretation and processing. The control module is a layer of code consisting of callback functions which have been registered with the window system during GUI creation. These callback functions have been "attached" to interface objects, and are executed by the window system in response to prespecified event(s) (e.g. mouse clicks, key presses). The control module also handles the reading in of digit samples from database files on disk, and those that are drawn by the user with the mouse. The feature extraction module processes the raw digit sample data into geometric feature vectors, which are then passed into the rule-based classifier. Here, the digit classification knowledge base written in the CLIPS expert system language[4], is applied by the forward chaining CLIPS inference engine in an attempt to classify the



Load Sample: 1 Database 1000 2300
Stroke Type: Points Zoom: Off Grid: Off
Stroke Speed: 10 Stroke Width 0
Ready to display or draw digits

Load Sample: 855 Database 1000 2300
Stroke Type: Points Zoom: Off Grid: Off
Stroke Speed: 7 Stroke Width: 0
Sample digit 855 is a '7', 2 strokes, 65 points

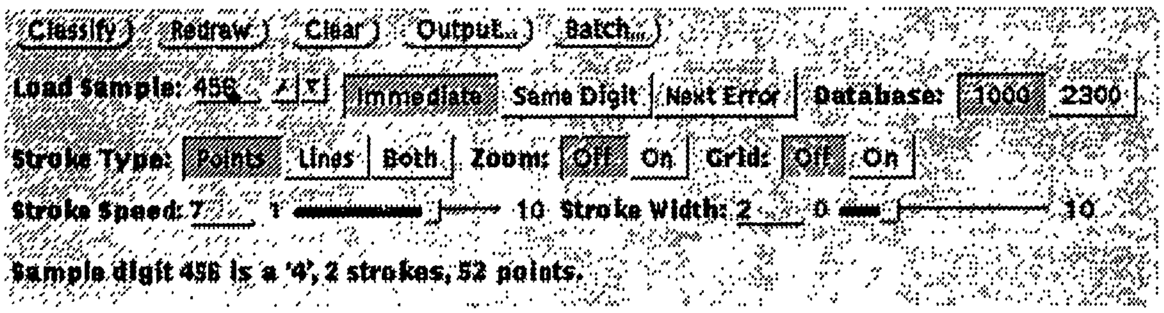
(a) (b)

Figure 2: The main OLDRES window (a) without a sample currently loaded, (b) with a sample currently loaded from a database.

patterns. The resultant features and classifications are then displayed back to the user. Training the system consists of developing the feature extraction capabilities and classification rules to process the features. Separate databases were used during the training and testing phases of development. The testing phase evaluates the combined performance of the tuned system on an independent data set.

2.2 Using OLDRES

Figure 2(a) shows the main OLDRES window as it appears at start-up. It contains two sub-windows, a control panel at the top, and a large drawing canvas underneath. The control panel is where the various parameters and options are set, and processing actions are triggered. There are five rows of panel items here. The top row contains push buttons which are used to initiate processing or cause other windows to appear.



Load Sample: 456 Database: 1000 2300
Stroke Type: Points Zoom: Off Grid: Off
Stroke Speed: 7 1 10 Stroke Width: 2 0 10
Sample digit 456 is a '4', 2 strokes, 52 points.

Figure 3: The OLDRES control panel items.

Notice that in Figure 2(b) there are three extra buttons which do not appear in Figure 2(a). These three items are context sensitive and only appear when a sample is currently loaded.

The second row of items is used to access digit samples from the databases. It consists of a numeric field and two exclusive choice settings. Exclusive settings are those where only one of the choices can be selected, or *active* at any one time. A numeric field is a text item where only numeric ASCII characters can be entered or displayed. Items from the third and fourth rows, exclusive settings and sliders, control various options for the graphic display of digit samples and extracted features. The bottom row of the control panel sub-window contains a single textual message item. The message string shown here changes dynamically. It is used to display system state information, data sample statistics, as well as the results of classification.

The canvas sub-window is where digit samples are displayed and drawn, and extracted features are graphically illustrated in colour. The main OLDRES window can be arbitrarily resized using facilities provided by the window manager.

2.2.1 Loading Samples from Disk

Before doing anything useful, OLDRES needs to have a currently loaded digit sample to work with. Digit samples can be loaded in one at a time from either of two database files (the testing database and the training database). When a sample is loaded, it is automatically displayed in the drawing canvas according to the current control panel display settings (discussed in Section 2.2.3). Access to digit samples from the two databases is controlled by the second row of panel items (see Fig. 3). They are shown here in a state where a digit sample has been loaded in, but recognition has not been attempted yet.

There are three possible modes in which samples can be loaded from disk, namely *immediate mode*, *same digit mode*, and *next error mode*. The load mode setting is the middle item of the row. Immediate mode simply loads and displays the sample number specified in the "Load Sample" numeric field. Same digit mode scans sequentially through the database for the next sample with the same identity as that of the current sample. In other words, if the currently loaded sample is a '4', we scan through the database for the next sample of a '4', then load and display it. Next error mode scans sequentially through the database looking for the next sample which is not correctly classified by the system. In this mode, each sample is successively loaded but not displayed, and processed by the recognition system. When the first non-correct classification is encountered, the search halts and that sample is displayed. The loading action is triggered by typing enter into the "Load Sample" numeric field, or by clicking on either of the increment/decrement buttons (small squares with triangles pointing upward and downward) associated with the numeric field. In immediate mode, the increment/decrement buttons step the "Load Sample" value by one. In the other two modes, they select the direction of the search. The panel item labeled "Database" specifies which database to load samples from, the 1000 member training set or the 2300 member testing set.

2.2.2 Drawing Samples with the Mouse

Besides loading in samples from disk, the user can also enter in new patterns by drawing them onto the canvas sub-window with the mouse. Whenever the mouse pointer enters this window, the cursor glyph changes to that of a pencil (see Fig. 6(a)). Upon leaving the drawing area, the cursor glyph reverts back to whatever is appropriate. This is to indicate to the user the intended input modality, i.e. that handwriting can be effected here. To draw a new sample, the user simply presses down and holds any of the three mouse buttons, moves the mouse to form the pattern, and releases the mouse button at the end of the stroke. While the mouse button is held down, its x, y location within the window is measured at the maximum possible rate, typically around 20-25 points per second. Stroke points are continually drawn as they are sampled, giving real-time feedback to the user. An arbitra y number of strokes may be entered up to a maximum of 500 individual points.

Even though the computer mouse is generally considered a very unnatural writing instrument, and the locator sampling rate is somewhat slow, it is adequate for an experimental system and is universally available on systems supporting the X Window System.

2.2.3 Digit Display Options

The appearance of displayed digit samples in the drawing canvas can be controlled via the third and fourth rows of control panel items. The "Stroke Type" exclusive setting specifies whether the pattern is to be drawn using just points, just line segments, or with both points and line segments. Points are rendered in white as single screen pixels. Line segments are drawn in blue at the width specified by the "Stroke Width" slider item. Digit samples can be animated at varying speeds by adjusting the value of the "Stroke Speed" slider. A stroke speed value of 10 results in instantaneous display of the entire pattern (no animation), whereas a speed of 0 yields very slow animation of strokes at the approximate rate of ten points per second.

The "Zoom" and "Grid" items are boolean settings. "Zoom" specifies whether or not to magnify the sample image on the canvas. This is useful for viewing small patterns with increased detail. The "Grid" item specifies whether or not to draw a dashed bounding box around the pattern. Also, when viewing features (discussed in section 2.2.5), feature end point coordinates are illustrated by projecting normals to the bounding box. Both "Zoom" and "Grid" are set to "On" in the sample screen outputs of Figure 4.

The "Redraw" push button causes the loaded sample to be redrawn according to the current display options mentioned above. The "Clear" button purges the loaded sample from the system and clears the drawing canvas.

2.2.4 Classifying Samples

Once a digit sample has been loaded into OLDRES (e.g. Fig. 2(b)), recognition of the pattern is attempted by clicking on the "Classify" push button. The pattern is then processed by the feature extraction sub-system, followed by the rule-based expert system module. Results of classification are displayed textually in the control panel message item (e.g. Fig. 4(a)). Results of feature extraction may be observed in color graphical format as described in the next sub-section.

2.2.5 Viewing Structural Features

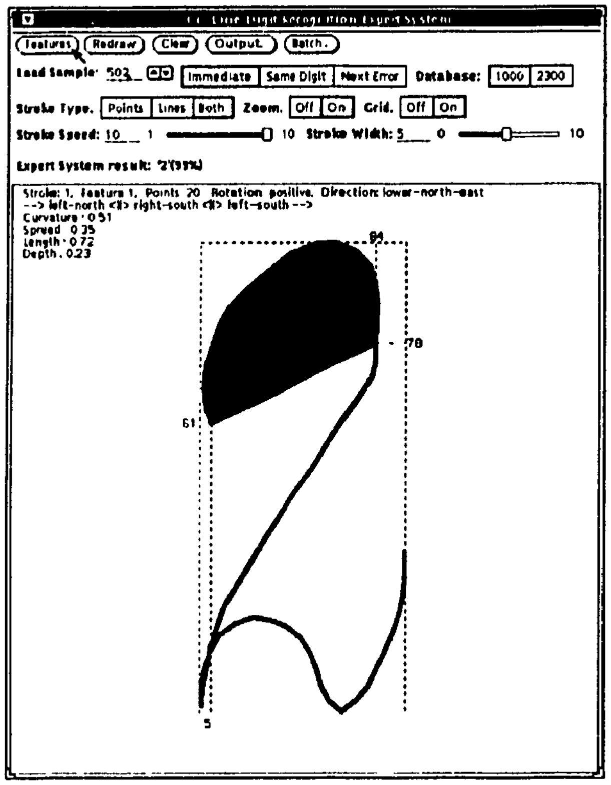
Once classification of the sample has been performed, the label of the "Classify" button changes to "Features", and the extracted structural feature segments are indicated graphically by alternating the colour used to draw the points and/or line segments (Fig. 4(b)). Individual features may then be sequentially selected and examined by repeatedly clicking on the "Features" button. One such sequence of three features is illustrated in Figure 4(c) through 4(e). Textual and numeric feature vector values for the currently selected feature are displayed in the upper-left corner of the canvas. The feature vector elements are described in section 3.3.4. Features are graphically displayed in the following manner:

1. A yellow line segment joining the feature start and end points is drawn, illustrating the *trajectory* and *end point spread* feature vector components.

2. A red line segment is drawn joining the feature *apex point* to the mid-point of the line segment described in (1), illustrating the *direction of concavity* and *apex point* of the feature.

3. The area enclosed by the stroke segment defined by the feature and the line segment in (1) is filled in cyan, thus highlighting the *degree of curvature* of the feature.

4. If the "Grid" control panel setting is "On", normals in the form of dashed line segments are drawn from the feature's start and end points to the four sides



Load Sample: 503

Database: 1000 2300

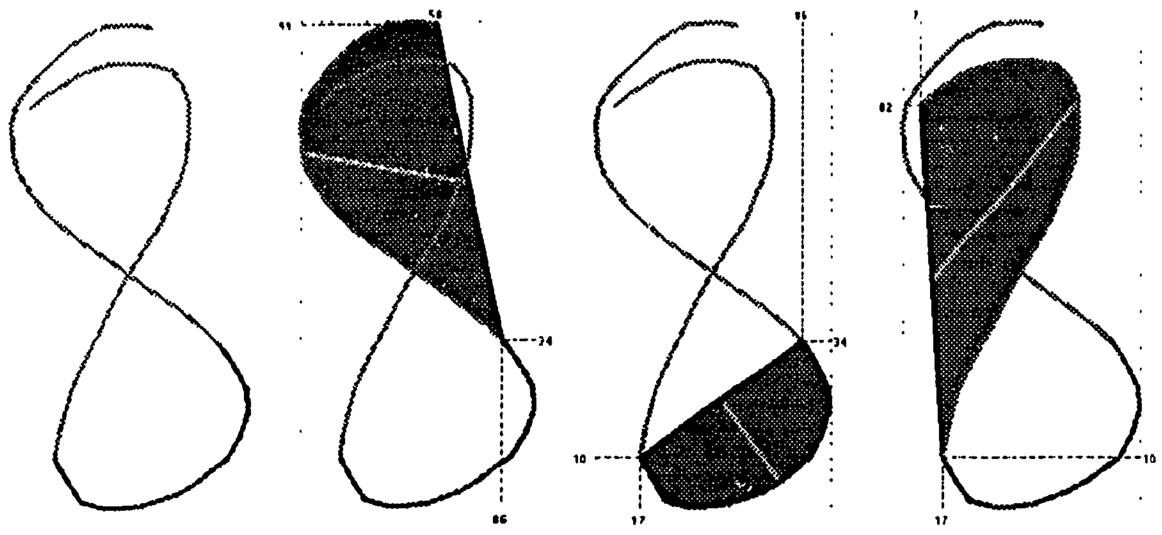
Stroke Type. Points Lines Both Zoom. Off On Grid. Off On

Stroke Speed: 10 1 10 Stroke Width: 5 0 10

Expert System result: '2'(93%)

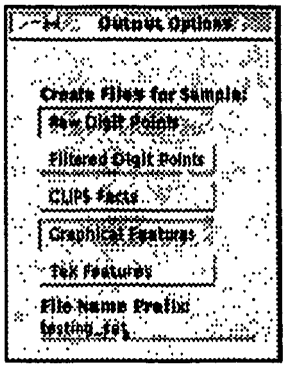
```
Stroke: 1, Feature 1, Points 20  Rotation positive, Direction: lower-north-east
--> left-north <||> right-south <||> left-south -->
Curvature : 0.51
Spread  0.35
Length : 0.72
Depth . 0.23
```

(a)



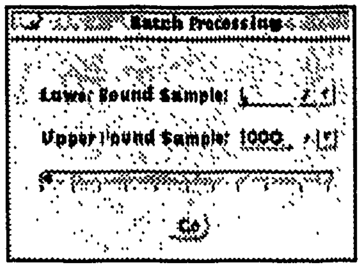
(b) (c) (d) (e)

Figure 4: Color graphical display of extracted features. (a) A single feature at a time is displayed on the pattern. (b) - (e) A composite sequence of displayed features.



File Name Prefix: testing_fet

(a)



Lower Bound Sample: 1

Upper Bound Sample: 1000

(b)

Figure 5: OLDRES pop-up frames. (a) The output options pop-up frame. (b) The batch classification pop-up frame.

of the displayed bounding box. The x, y coordinate values normalized over the range [0..100] are drawn where the normals meet the bounding box.

2.2.6 Creating Output Files

OLDRES has the ability to generate useful output files in various formats from the digit sample data. Five files containing the following types of information may be created:

1. **Raw Digit Points** A graphics file in the PostScript language[2, 3] which depicts digit sample data in the form of dots before filtering.

2. **Filtered Digit Points** A PostScript graphics file illustrating post-filtered digit sample data in the form of dots.

3. **CLIPS Facts** A file containing extracted structural feature vectors in the form of CLIPS facts.

4. **Graphical Features** A PostScript file which graphically illustrates the stroke segments used to calculate feature vectors.

5. **TEX Features** A file containing the textual and numeric feature vector elements formatted as a LATEX table[30].

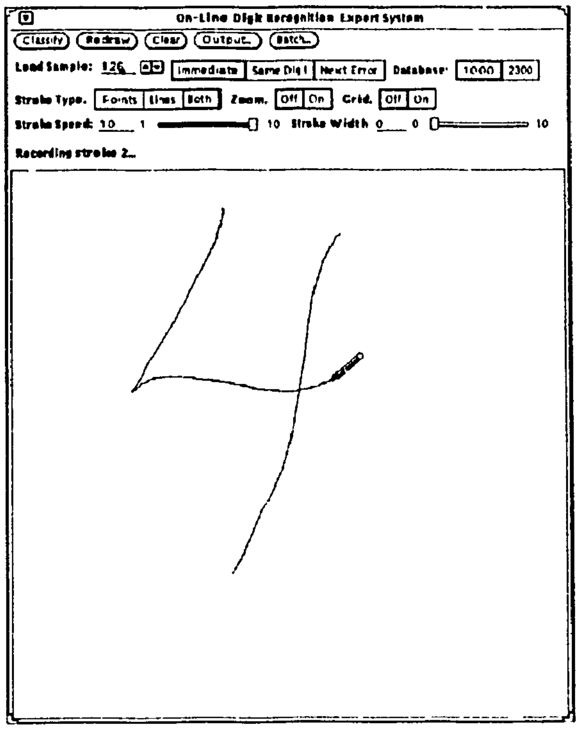
The output options pop-up frame, shown in Figure 5(a), is used to select which of the five types files to create for digit samples. It is brought up by clicking on the control panel push button labeled "Output". Unlike all the other selection items in OLDRES, the "Create Files for Sample" item on the panel of this pop-up frame is an *inclusive* setting. Hence, any combination of the five options may be chosen. The "File Name Prefix" text field allows the user to specify the base name of files. A separate disk file is created for each of the selected file types, each with a unique file name suffix.

2.2.7 Batch Classification

Clicking on the control panel button labeled "Batch" brings up the batch processing pop-up frame (Fig. 5(b)). Here, a desired range of samples from the currently active database can be selected for classification in batch mode. Two numeric fields are provided to specify the lower and upper bound sample foı the run. The "Go" push button is pressed to initiate processing. A thermometer-like gauge tracks the progress of the batch run as it progresses. During batch classification the drawing canvas is used to display two tables, ⁻ confusion table and a performance table (see Fig. 6(b)). Their format is explained in section 5.2. The tables are continually updated after each sample is processed during the batch run. Various colours are used in order to facilitate their interpretation. For example, numbers in the diagonal of the confusion table correct classifications - are green. Rejections are indicated in yellow, and other non-zero entries represe' ting misclassifications are in red.

2.3 System Implementation

The OLDRES system implementation is written entirely in the C programming language, except for the digit classification knowledge base which is written in the CLIPS version 4.3 expert system language. It was developed on the Sun SPARCstation platform under the SunOS 4.1.1 (UNIX) operating system running the X Window System, version 11, release 4. The system involves approximately 5000 lines of C code and



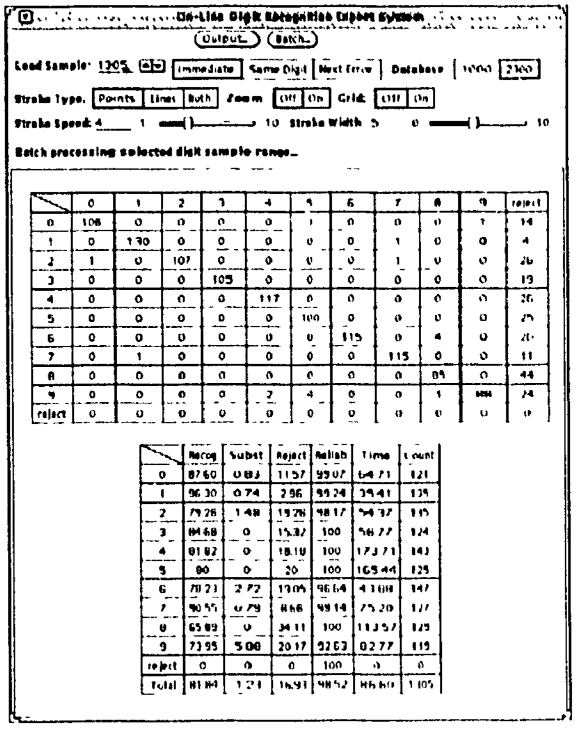
Load Sample: 126

Database: 1000 2300

Stroke Type. Zoom. Grid.

Stroke Speed: 10 1 10 Stroke Width 0 0 10

Recording stroke 2...



Load Sample: 1305

Database: 1000 2300

Stroke Type. Zoom. Grid.

Stroke Speed: 4 1 10 Stroke Width 5 0 10

Batch processing selected digit sample range...

	0	1	2	3	4	5	6	7	8	9	reject
0	106	0	0	0	0	1	0	0	0	1	14
1	0	130	0	0	0	0	0	1	0	0	4
2	1	0	107	0	0	0	0	1	0	0	26
3	0	0	0	105	0	0	0	0	0	0	19
4	0	0	0	0	117	0	0	0	0	0	26
5	0	0	0	0	0	100	0	0	0	0	25
6	0	0	0	0	0	0	115	0	4	0	28
7	0	1	0	0	0	0	0	115	0	0	11
8	0	0	0	0	0	0	0	0	89	0	44
9	0	0	0	0	2	4	0	0	1	88	24
reject	0	0	0	0	0	0	0	0	0	0	0

	Recog	Subst	Reject	Reliab	Time	Count
0	87.60	0.83	11.57	99.07	64.71	121
1	96.30	0.74	2.96	99.24	39.41	135
2	79.28	1.48	19.24	98.17	54.37	135
3	84.68	0	15.32	100	56.77	124
4	81.82	0	18.18	100	173.71	143
5	80	0	20	100	165.44	125
6	78.23	2.72	19.05	96.64	43.08	147
7	90.55	0.79	8.66	99.14	75.20	127
8	65.89	0	34.11	100	113.57	135
9	73.95	5.08	20.17	92.63	82.77	119
reject	0	0	0	100	0	0
Total	81.89	1.21	16.91	98.52	86.60	1305

(a) (b)

Figure 6: (a) Drawing a digit sample with the mouse. (b) OLDRES during batch classification.

3000 lines of CLIPS code. Using SunOS shared libraries, the OLDRES executable image is 393K bytes in size, and uses about 1.7 Mbytes of virtual memory space at run-time.[1] Test results were obtained on a Sun SPARCstation 2 with 32 Mbytes of RAM.

2.3.1 The CLIPS Expert System Shell

CLIPS (C Language Integrated Production System) is a forward-chaining rule-based expert system shell developed by NASA at the Lyndon B. Johnson Space Center. Based on the *Rete* pattern matching algorithm[14] and written entirely in C, it is one of the fastest and most portable expert system shells available[34]. The CLIPS language is syntactically similar to LISP and provides rich pattern matching capabilities.

[1]as reported by the UNIX "ps" (process status) command.

The rule-based classification module is implemented as an embedded CLIPS run-time module and linked into the OLDRES executable image. This is achieved by first loading the knowledge base into CLIPS, and then issuing the CLIPS command `(rules-to-c)`. This results in the translation of the KB into seven C source files, which when compiled and linked with a CLIPS run-time function library, produces the run-time module.

2.3.2 The XView Toolkit

The OLDRES graphical user interface was created using XView version 2.0 toolkit. XView (X window system-based Visual/Interactive Environment for Workstations) is a user interface toolkit to support interactive, graphics-based applications running under the X Window System[2]. It provides a collection of pre-built user interface objects (e.g. push buttons, pop-up menus, scrollbars, etc.), the appearance and functionality of which comply with the OPEN LOOK Graphical User Interface (GUI) specification [47, 48]. This is the GUI standard for UNIX System V Release 4 supported by Sun Microsystems Inc. and AT&T. XView was developed by Sun as the X-based successor to its SunView toolkit. SunView is Sun's dated proprietary window system which ran only on Sun hardware. In contrast to the network based X Window System, SunView is kernel based, thereby restricting graphical display of applications to the local display of the machine which they are running on. XView is being freely distributed in source form, hence it has been ported to run on many UNIX workstation platforms, including Digital Equipment Corporation's DECstation line running the Ultrix operating system, IBM's RS/6000 running AIX, and IBM-compatible PC's under LINUX, among others.

2.3.3 User Interface Objects

XView is an object oriented toolkit based on the following fundamental principles of object-oriented programming:

- Objects are represented in a class hierarchy

[2][19], p. xxiii

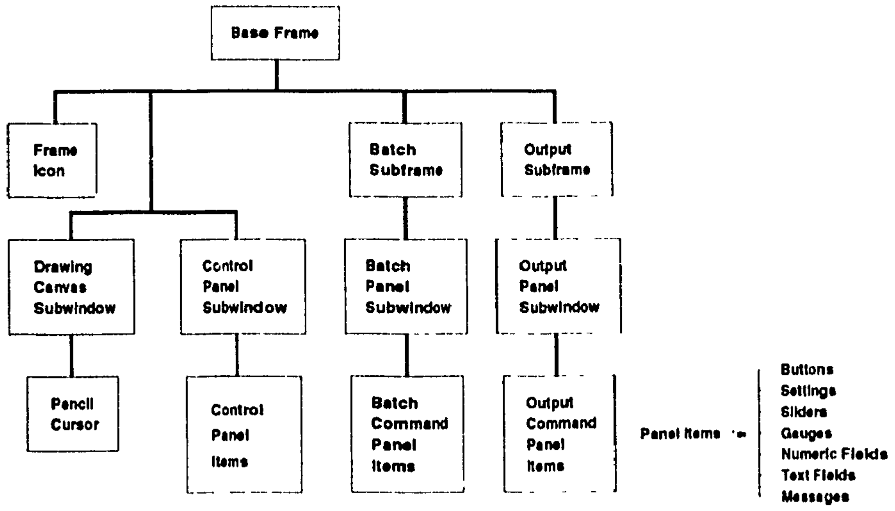


Figure 7: Hierarchy of OLDRES GUI objects.

- Objects are opaque data types.
- Objects have attributes which can be set by message passing functions.
- Objects may have callback procedures that are triggered by events.

XView objects include both visual and non-visual objects. Some examples of visual objects are frames, panels, buttons, and scrollbars. Non-visual objects maintain state information which affect the display of visual objects. For example, screen, server, and font objects. Static sub-classing with chained inheritance is used such that all objects of a particular class inherit properties from its parent class, or *superclass*.

Figure 7 shows the hierarchy of user interface objects in the OLDRES system. The base frame is the parent, or owner object of the panel and canvas sub-windows which it contains. It is also the parent of the application's two pop-up sub-frames, and the frame icon. Both pop-up sub-frames contain a child panel and each of the three panels is in turn the parent of numerous panel item objects.

2.3.4 XView Application Programmer's Interface

All user interface objects visible on the OLDRES base frame are created immediately at program start-up. Creation of the output options and batch processing pop-up frames and their descendant objects are delayed until the first attempt to access them is made. This contributes to faster system startup, and conserves memory since one might not always use the pop-up frame services during every session with OLDRES. User interface objects are created and managed using the XView application programmers interface (API) functions. The XView toolkit API consists of six primary generic C functions.

1. **xv_init()** Establishes the connection to the server, initializes the event dispatcher and the Resource Manager database, processes any passed attributes, and installs a default error handler. The return value is a handle to the X server object being used. `attrs` is a variable length sequence of attribute-value pairs, terminated by a NULL.

```
Xv_server
xv_init(attrs)
    <attribute-value list> attrs;
```

2. **xv_create()** Creates an object. `owner` is the handle of the parent object. **package** specifies the type or class of object to be created. Any attributes of the object not specified in the `attrs` list will assume default values. The return value of the function is a handle to the newly created object. This handle can then be used to reference the object in subsequent calls to `xv_destroy()`, `xv_get()`, and `xv_set()`, or as the `owner` in calls to `xv_create()`.

```
Xv_object
xv_create(owner, package, attrs)
    Xv_object      owner;
    Xv_pkg         package;
    <attribute-value list> attrs;
```

3. **xv_destroy()** Destroys an object.

```
int
xv_destroy(object)
    Xv_opaque object;
```

4. **xv_find()** Finds an object that meets certain criteria; or if the object doesn't exist, creates it. This is the default behavior which can be defeated by specifying the attribute-value `XV_AUTO_CREATE, FALSE` in `attrs`.

```
Xv_opaque
xv_find(owner, package, attrs)
    Xv_object      owner;
    Xv_pkg         package;
    <attribute-value list> attrs;
```

5. **xv_get()** Gets the value of a single attribute.

```
Xv_opaque
xv_get(object, attr)
    Xv_object     object;
    Attr_attribute attr;
```

6. **xv_set()** Sets the value of one or more attributes.

```
Xv_opaque
xv_set(object, attrs)
    Xv_object     object;
    <attribute-value list> attrs;
```

In general, an XView application must first call `xv_init()` to initialize the system, and then `xv_create()` to create the application's top level window, or *base frame*. This is the application's root object. From here, one or more sub-windows are created within the base frame as child objects of the base frame. Optionally, pop-up sub-frames containing sub-windows may also be created. Frames are free floating windows which may overlap each other. Frames do not exist by themselves; they contain sub-windows which are bound by the frame and *tiled* within it, i.e. they do not overlap one another. XView provides the following types of sub-windows:

Canvas Sub-windows A basic planar drawing surface onto which graphics can be rendered. A single canvas window is used in OLDRES.

Text Sub-windows An area where a sequence of ASCII characters may be displayed and edited using the built in editing facilities. OLDRES does not use any text windows.

Panel Sub-windows A control area where buttons, controls and settings are displayed. Three such windows are used in OLDRES.

TTY Sub-windows A terminal emulator in which arbitrary command line or screen based applications can be run. No TTY windows are necessary in OLDRES.

Objects specific to each sub-window class are then created. Notification callback procedures and select input events are registered, after which the application calls the `xv_main_loop()` function to initiate the dispatching of events. The user then generates keyboard and mouse events asynchronously, which are then delivered to the application's registered callback functions for processing. Event processing continuous until the quit condition is signalled,[3] at which point `xv_main_loop()` returns and the program exits gracefully by using `xv_destroy_safe()` to shut down the frame object and all its descendants.

2.3.5 Colour Graphics

For efficiency reasons, the recommended method for rendering graphics under X is Xlib[15]. The Xlib function library is the lowest level API to X, on top of which all higher level toolkits are built. However, toolkits are the preferred choice for managing higher level user interface objects such as windows, menus, scrollbars, etc. OLDRES makes use of the following Xlib calls for management and display of color graphics in the drawing canvas sub-window: XClearArea. XClearWindow, XDefaultGC, XDrawLine, XDrawLines, XDrawPoints, XDrawRectangle, XDrawSegments, XDrawString, XFillPolygon, XQueryFont, XSetBackground, XSetFont, XSetForeground, XSetLineAttributes, XSetWindowBackground, XTextWidth.

[3]OPEN LOOK specifies this to be the responsibility of the window manager program.

2.4 The Control Module

The control module consists of a number of user interface callback functions, various disk input/output (I/O) routines, interfacing code for the feature extraction and classification sub-systems, as well as miscellaneous utility functions for maintaining system state information.

A single callback function is assigned to handle all control panel buttons, while another function handles inputting of user drawn patterns in the canvas window. All other interactive panel items (i.e. everything except the control panel message item which is output only) have their own private callbacks attached. Collectively, the callback functions maintain internal system state buffers, manage user interface objects, and execute processing actions requested by the user.

Disk I/O duties performed by the control module include the reading in of selected digit samples from the databases, and the formatting and writing of output files selected via the output options pop-up frame. Colour graphics are rendered onto the drawing canvas using the Xlib functions mentioned in the previous section. Coordinate system transformations are also required here to map the digitizer tablet data from the databases onto physical screen pixels in the dynamically resizable canvas window.

This completes the presentation of the OLDRES system architecture and graphical user interface. The OLDRES feature extraction and rule-based classification modules will be explained in the next two chapters.

Chapter 3

Extraction of Structural Features

The structural feature extraction method presented in this chapter is extremely simple and efficient, both in theory and in its implementation. The basis for definition of its feature space is that of *functional attributes* which humans use to differentiate difficult cases [57]. Thus the goal is to identify and describe characters using high-level primitives in much the same fashion as a human might do when presented with the image of a character and asked to elaborate on its structural composition. The feature extraction process consists of segmenting strokes into one or more descriptive subcomponents, which at the highest level, compose the character. For example, the character '5' may be described as having three major structural components; a horizontal bar, a vertical bar, and a curve with its concavity facing leftward. Relative positioning, size, connectivity, and directional information are also required to more precisely specify the inter-relationships among the major components.

3.1 Input Data

Data for a single character is input to the feature extraction system as a sequence of one or more strokes. Each stroke is in turn composed of a sequence of (x, y) coordinate value pairs. Coordinate pairs, or points, are obtained from an input device, such as a digitizer tablet [56], by sampling the (x, y) position of a locator with respect to the origin of a reference plane at constant, discrete time intervals. The origin of the reference plane is assumed to be the lower left-hand corner, with values of x increasing to the right, and values of y increasing upwards. Hence, x and y are non-negative integers. Sampling of points for each stroke is initiated asynchronously by an input device *pen–down* event, and terminated by a *pen–up* event.

If we use the coordinate pair (E, E) to indicate the end of each stroke[1], we may represent an entire input character as a linear array of coordinate pairs $C(i) = (x_i, y_i)$, where $i = 1, 2, \ldots, N$, and N is the number of points, including end of stroke indicators, which compose the character. This is the representation of input characters used by the system.

Based on the stroke data for each character, four values are maintained for size normalization purposes (discussed in the next section). These values are x_{min}, y_{min}, x_{max}, y_{max}. They are the minimum x and y, and maximum x and y coordinate values, respectively.

3.2 Definitions

3.2.1 Size Normalization

We wish to extract features from *completely unconstrained* input characters. By this we mean that there are no assumptions made as to where the character is positioned, what the size of the character is, how skewed it is, etc. Obviously we must impose some basic constraints, such as that the character must be drawn somewhere within the boundaries of input device, and it must be drawn large enough that pen movement and direction changes can be reasonably detected by the resolution of the input device. Therefore, in order to allow for input characters to be completely unconstrained, we must normalize our measurements with respect to the size of the character currently under consideration [7].

We define the **bounding box** of a character as being the smallest rectangle that fully encloses all points which compose the character. Hence, it is the rectangle defined by the diagonally opposite vertices (x_{min}, y_{min}) and (x_{max}, y_{max}) (see Figure 8).

[1]Making sure that (E, E) is unique with respect to all valid (x, y) coordinate pairs

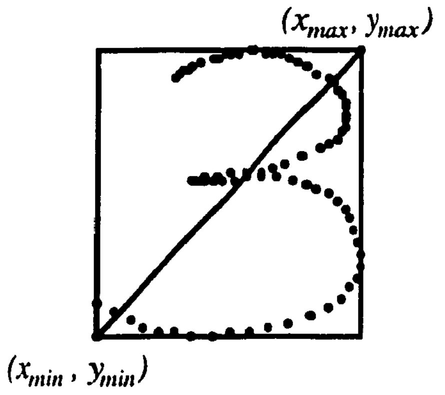


Figure 8: The bounding box of a character.

We define the **size** of a character as being the length of the diagonal of its bounding box, or more formally as

$$\textbf{size} = \sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}.$$

Henceforth, when we speak of lengths of line segments, and distances between points, we mean Euclidean distances of the type $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, where the line segment end points, or the two points, are given by (x_1, y_1) and (x_2, y_2).

3.2.2 Discrete Directions

For the purposes of the extraction process, as well as for the description of the resultant features, the range of possible directions over two dimensions is divided into sixteen equal sub-ranges. The names and arrangement of these discrete direction classifications are given in Figure 9. Therefore, given any two ordered pairs of points, the slope, $\Delta y/\Delta x$, so defined dictates the classification of the direction defined from the start point to the end point. Henceforth, when we speak of direction, we are referring to the sixteen possible discrete directions given in Figure 9.

Also, when we refer to the direction at point $C(j)$, we mean the discrete direction defined from the point $C(j-1)$ to the point $C(j)$, for some index j into the character data array C. Positive rotations are defined to be clockwise.

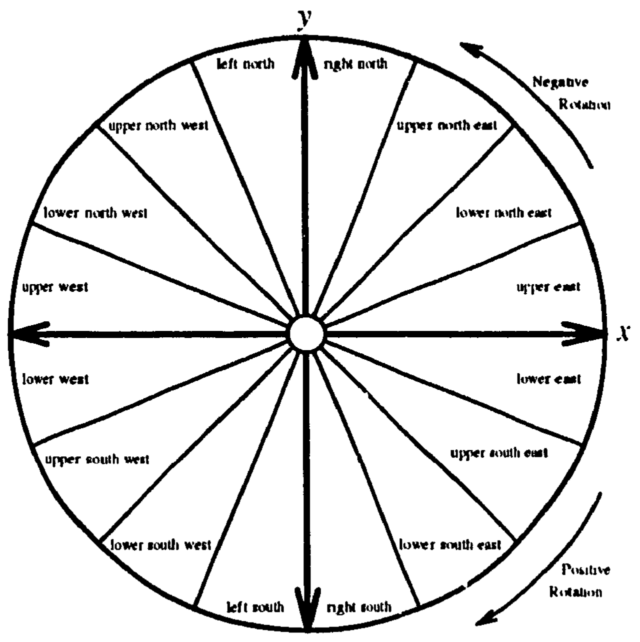


Figure 9: The sixteen discrete directions.

3.3 The Feature Extraction Process

3.3.1 Filtering the Input Data

It is noted that due to the constant sampling rate used in the data input process, it is possible to infer certain temporal properties of stroke segments from inter-point distances, e.g. pen speed and acceleration. However, since our objective is to extract structural features, we do not consider such temporal characteristics, and in fact filter out the basis for their determination. The only temporal aspect exploited is the data point sequence. Input characters tend to be composed of many more point samples than are actually required to structurally represent the character. Therefore, in order to increase the efficiency (speed) of the algorithm, extraneous points are filtered out such that the distance between neighboring points is greater than or equal to some threshold value [51]. The two opposing factors in determining this threshold level are (1) to minimize the number of points needed to represent the character while (2) maintaining sufficient stroke smoothness for the algorithm to perform well. This minimum inter-point threshold distance was experimentally determined, and set to

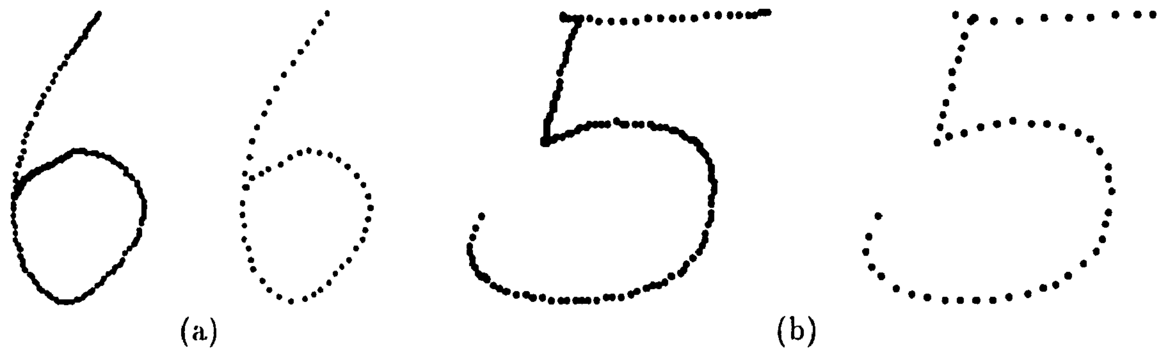


Figure 10: Examples of input data filtering. (a) The number of points is reduced from 195 to 54. (b) The number of points is reduced from 171 to 58.

be 3% of the size of the character. Values higher than this did not yield a sufficient degree of stroke smoothness. Examples of pre- and post-filtered data are given in Figure 10.

3.3.2 Identifying Raw Features

Given our initial representation of the character as a linear array of $C(i) = (x_i, y_i)$, where $i = 1, 2, \ldots, N$ for N composite coordinate pairs, the process of identifying features proceeds iteratively, and is summarized by the following four steps:

1. Given the starting point of the current feature F_c as the index i_s^c into C, find the end point of F_c as the index i_e^c into C such that the structural segment between the points $C(i_s^c)$ and $C(i_e^c)$ does not violate the constraints imposed on features, but the structural segment between the points $C(i_s^c)$ and $C(i_e^c + 1)$ does violate these constraints.

2. If $(i_e^c + 1) = N$, the current feature F_c is the last feature of the character.

3. If the point $C(i_e^c + 1)$ is *not* an end of stroke indicator, set the start point index i_s^{c+1} of the next feature F_{c+1} to i_e^c.

4. If the point $C(i_e^c + 1)$ *is* an end of stroke indicator, set the start point index i_s^{c+1} of the next feature F_{c+1} to $i_e^c + 2$.

The constraints which stipulate whether a structural segment is a valid feature or not are discussed in the following section.

Each feature is thus uniquely identified by its start and end points, $C(i_s)$ and $C(i_e)$, with connected features of the same stroke sharing a common point (the previous feature's end point is the next feature's start point). The result of this process is to segment each stroke into one or more raw features. These features are placed in an ordered list, the elements of which are the feature vectors that describe the input character.

Connected raw features may subsequently be combined according to a set of preset rules in order to minimize the number of features required to describe the character. The need to aggregate features and the rules which permit connected features to be combined are discussed in Section 3.4.

3.3.3 Constraints Which Define Features

The determination of a feature end point $C(i_e)$ given a start point $C(i_s)$ is performed in a sequential, point by point fashion. We define the *first direction* of a feature as the direction at point $C(i_s + 1)$ (i.e. the discrete direction defined from the point $C(i_s)$ to the point $C(i_s + 1)$). Initially, the *rotation* of the feature is undefined. Then, starting with the index $i = i_s + 2$ into the character data array C, we proceed with $i = i + 1$ until any one of the following four constraints is violated:

1. **end of stroke constraint:** The point $C(i)$ cannot be (E, E) (i.e. the point representing end of stroke). This is obvious, a feature cannot span multiple strokes.

2. **smoothness constraint:** The directions at the points $C(i-1)$ and $C(i)$ must either be the same or adjacent directions (adjacent directions share a common boundary). For example, the Upper North East is adjacent to the direction Right North, whereas Lower North East is not, see Fig. 9.

3. **rotation constraint:** The rotation of a feature becomes defined on the first occurrence where the direction at point $C(i)$ is not the same as the direction at

point $C(i-1)$. If the direction at $C(i)$ is the next clockwise direction from the direction at $C(i-1)$, the rotation of the feature is said to be ***positive***, otherwise the rotation is *negative*. Subsequent changes of direction from $C(i-1)$ to $C(i)$ must maintain this same rotation.

4. **direction constraint**: The total number of cumulative contiguous discrete directions traversed by the feature must be no more than eight. Recall that we have sixteen circularly contiguous directions, hence this constraint ensures that a curved feature will have an arc of less than 180°.

Once we encounter a constraint violation at the point $C(i)$, we define the feature's end point by setting $i_e = i - 1$, since the point $C(i-1)$ was the last point which did not violate any constraints. Finally, we define the *last direction* of the feature as being the direction at point $C(i_e)$.

The procedure described in this section merely identifies the set of points which constitute a feature. The descriptive attributes which we use to characterize features are defined in the next section.

3.3.4 Calculation of Feature Vectors

Given the start point $C(i_s)$, and end point $C(i_e)$ of a feature, the following eight structural and geometric properties may then be calculated to describe the feature (refer to Fig. 11).

1. **end point spread**: The length of the line segment $S = \overline{C(i_s)C(i_e)}$, defined by the feature's start and end points.

2. **apex point**: The point $C(i_a)$ on the feature such that

$$\forall i\,[(i_s < i < i_a) \wedge (i_a < i < i_e)] : L_n(C(i_a), S) \geq L_n(C(i), S),$$

where $L_n(p, l)$ is the length of the normal from point p to the line l. In other words, the point along the feature which is farthest away from line segment S.

3. **apex depth**: The length of the normal from the apex point $C(i_a)$ to the line segment S

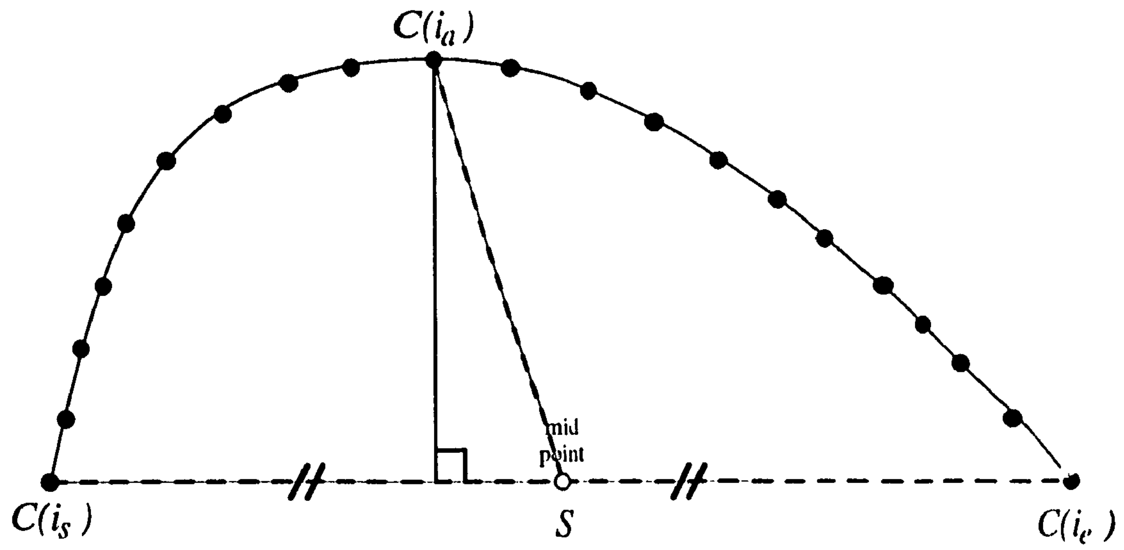


Figure 11: A sample feature.

4. **length**: The sum of the distances between each successive pair of points along the entire feature.

$$\sum_{i=i_s}^{i_e-1} D\left(C(i), C(i+1)\right),$$

where $D(p_1, p_2)$ denotes the distance from the point p_1 to the point p_2.

5. **degree of curvature**: Defined as 1 minus the *end point spread* divided by the *length* of the feature. Indicates the relative amount of curve closure, 0.0 being a perfectly straight line, 1.0 being a completely closed loop.

6. **direction of concavity**: The discrete direction defined from the *apex point* $C(i_a)$ to the mid-point of the line segment S.

7. **trajectory**: The discrete direction defined from the feature start point $C(i_s)$ to the feature end point $C(i_e)$.

8. **rotation**: The rotation indicated by rotating from the feature's *first direction* through its intermediate directions to its *last direction*. Can be either positive, negative, or undefined (i.e no rotation, the first and last directions are the same).

3.4 Rule-Based Aggregation of Features

3.4.1 Justification

The feature extraction algorithms implied in the previous sections perform best when curved segments have smooth, gradual, and definitive direction changes, and connected straight segments form crisp vertices with minimal rounding. In reality however, these characteristics are often unattainable due to the imprecision of, and noise introduced by the input device, the digitization process itself, as well as unsteadiness of hand of the writer [52].

Spurious points which single-handedly interrupt an otherwise consistent rotation over a larger segment cause premature termination of a feature during the sequential, point by point process of feature identification. The application of various pixel averaging smoothing techniques [51] to the input data was attempted in order to circumvent the problem of premature feature termination. However, in order to approach the degree of stroke smoothness required over widely varying sizes of input characters, these methods tended to obscure otherwise desirable and detectable feature boundaries.

In the presence of moderately noisy input data, such as that obtained from a digitizer tablet or from a mouse, the problem of premature feature termination can be solved by aggregating selected pairs of features into one. Using a set of heuristic rules to compare two connected features, we may select appropriate candidates which when combined would form a larger more representative segment. In the presence of high quality input data, the feature aggregation process is unnecessary due the fact that the raw extracted features are already optimal. An optimal set of raw features which requires no aggregation can occur in practice, albeit rarely.

Occasionally, seemingly straight and relatively smooth segments yield multiple raw features (Fig. 12(a)(b)(c)). This may happen when the trajectory of a straight segment is very close to a boundary between two discrete directions. Discrete direction classification of successive point pairs along such a segment may oscillate back and forth between the adjacent discrete directions, even though the difference between their actual slopes is minimal. This in turn violates the constraint on features that

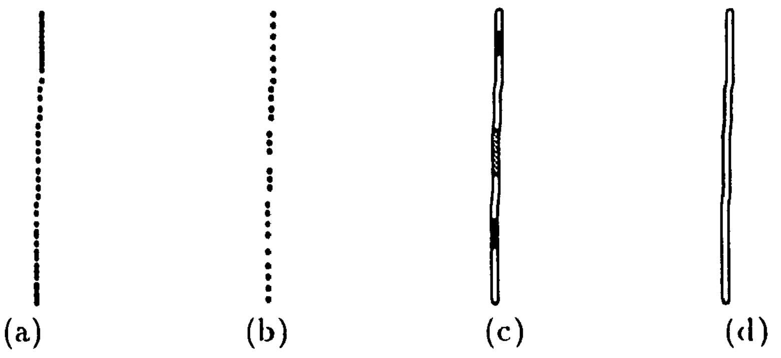


Figure 12: An example of how a seemingly straight segment can result in multiple raw features being extracted. (a) 62 original data points. (b) 26 data points after filtering. (c) 7 raw features extracted. (d) 1 remaining feature after aggregation.

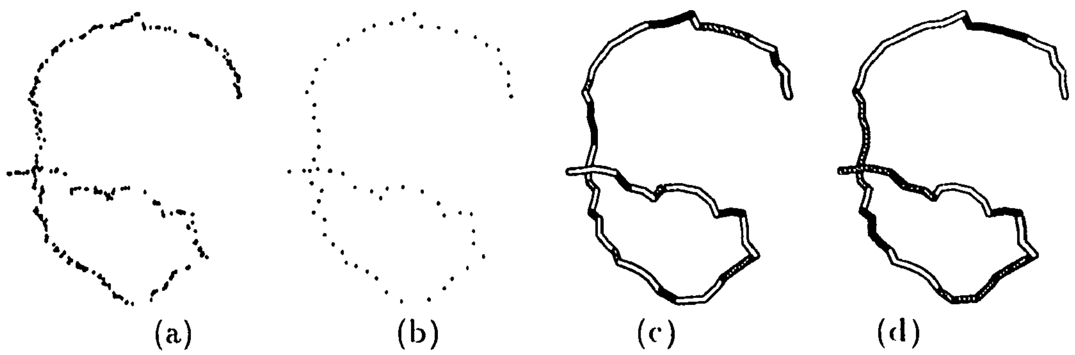


Figure 13: An example of an exceedingly jittery input character. (a) 316 original data points. (b) 64 data points after filtering. (c) 27 raw features extracted. (d) 12 features after aggregation.

they must have a single rotation, thus preventing the entire segment from being identified as a single raw feature. The feature aggregation stage rectifies this problem however, and the entire segment is reconciled into one feature (Fig. 12(d)). If input characters are exceedingly jittery however, the sensitivity of the feature identification process and the limits of the feature aggregation rules cannot yield a set of features to adequately represent the character (Figure 13).

3.4.2 Method of Aggregation

Heuristics

Two features are said to be connected when they share a common point. Specifically, the ending point of one feature is the starting point of another feature, $C(i_e^1) = C(i_s^2)$. Feature aggregation takes two such connected features, and merges them by setting the start point of the new merged feature to be the start point of the first feature, and the end point of the new merged feature to be the end point of the second feature,

$$C(i_s^N) = C(i_s^1) \qquad C(i_e^N) = C(i_e^2).$$

The creation of a new aggregated feature is not subject to the constraints imposed on raw features defined in Section 3.3.3, but rather their creation is governed by a set of feature aggregation rules. These are four prioritized heuristics which compare two connected features and decide either for or against their aggregation.

Each of the four rules has a unique level of priority. All rules at higher priorities are attempted before considering the aggregation of features by rules at a lower priority. This scheme implements a conflict resolution strategy for cases in which a feature is doubly connected, and could be justifiably aggregated with either of its connected peers. But, the decision should be made one way or the other based on the level of confidence we have that the resulting aggregated feature will in the long run, minimize the number of features, while maximizing the *appropriateness of representation* of the structural features for the actual character. Higher priority rules take precedence over lower priority rules because we have more confidence in their aggregated results. The requirements for aggregation of features are thus relaxed as rules descend in priority.

The next section lists the four heuristics for feature aggregation in a pseudo-rule format. We note here that the symbolic expressions used in the pseudo-rules are actually implemented as numeric comparisons or other procedures on the descriptive properties calculated for each feature as described in Section 3.3.4. For example, the expression (feature-1 is straight) really tests if the **degree of curvature** of feature-1 is less than or equal to a threshold value, namely 0.05. The threshold levels for numeric tests have been experimentally determined and contribute to the heuristic nature of the aggregation process. The numeric thresholds and procedures for determining truth values are listed following the four pseudo-rules.

Rules for Feature Aggregation

```
  Rule 1:  Combine-Same-Rotations
 Purpose:  Aggregates features which have the same rotation.
           The not-expression prevents straight lines from
           being aggregated, unless one or both of them is small.
Priority:  1, highest
     If:   (feature-1 and feature-2 are connected)
     and   (feature-1 and feature-2 have the same rotation, not undefined)
     and   not (     (feature-1 and feature-2 are both straight)
                 and (feature-1 and feature-2 are both not very small))
     and   (feature-1 and feature-2 are smoothly connected)
     and   (feature-1 and feature-2 do not over-rotate)
     and   (feature-1 and feature-2 do not potentially intersect)
    Then:  (aggregate feature-1 and feature-2)
```

```
     Rule 2:  Combine-Curve-and-Straight
    Purpose:  Aggregate a curved feature with a straight feature.
              The block of or-expressions prevents a curve and
              straight line of opposing rotations from being aggregated,
              unless the straight line is small or very straight.
   Priority:  2
         If:  (feature-1 and feature-2 are connected)
        and   (feature-1 is curved)
        and   (feature-2 is straight)
        and   (   (feature-1 and feature-2 do not have opposing rotations)
              or  (feature-2 is small)
              or  (feature-2 is very straight))
        and   (feature-1 and feature-2 are smoothly connected)
        and   (feature-1 and feature-2 do not over-rotate)
        and   (feature-1 and feature-2 do not potentially intersect)
       Then:  (aggregate feature-1 and feature-2)

     Rule 3:  Combine-Straight-Lines
    Purpose:  Aggregates straight segments with the same trajectory.
              The second block of or-expressions prevents straight
              lines of opposing rotations from being aggregated,
              unless one of them is small or extremely straight.
   Priority:  3
         If:  (feature-1 and feature-2 are connected)
        and   (feature-1 and feature-2 are both straight)
        and   (   (feature-1 and feature-2 have the same trajectory)
              or  (feature-1 and feature-2 have adjacent trajectories)
              or  (feature-1 and feature-2 have second adjacent trajectories))
        and   (   (feature-1 and feature-2 do not have opposing rotations)
              or  (either feature-1 or feature-2 is small)
              or  (either feature-1 or feature-2 is extremely straight))
       Then:  (aggregate feature-1 and feature-2)
```

Rule 4:	$\boxed{\text{Combine-Tiny-Feature}}$
Purpose:	Aggregate tiny features with their connected neighbors
Priority:	4, lowest
If:	(feature-1 and feature-2 are connected)
and	(feature-1 is tiny)
Then:	(aggregate feature-1 and feature-2)

Symbolic Rule Expressions

- *feature-1 and feature-2 are connected*: i_e of feature-1 = i_s of feature-2.
- *feature is curved*: **degree of curvature** > 0.05.
- *feature is straight*: **degree of curvature** ≤ 0.05.
- *feature is very straight*: **degree of curvature** ≤ 0.02.
- *feature is extremely straight*: **degree of curvature** ≤ 0.01.
- *feature is small*: $i_e - i_s < 5$, i.e. feature has 5 or fewer points.
- *feature is very small*: $i_e - i_s < 3$, i.e. feature has 3 or fewer points.
- *feature is tiny*: $i_e - i_s = 1$, i.e. feature has 2 points.
- *opposing rotations*: positive and negative.
- *adjacent trajectories*: Any 2 of the 16 discrete directions which are connected, i.e. share a common boundary.
- *second adjacent trajectories*: Any 2 of the 16 discrete directions which are separated by one discrete direction.
- *feature-1 and feature-2 are smoothly connected*: The first direction of feature-2 is either equal to, or an adjacent direction to the last direction of feature-1. Additionally, if one of the features is a curve with a defined rotation, the first

direction of feature-2 may also be the second adjacent direction on the rotation side of feature-1's last direction.

- *feature-1 and feature-2 do not over-rotate*: the trajectory of feature-2 is less than eight direction away from the trajectory of feature-1 when rotating in the direction indicated by feature-1. i.e. the trajectory of feature-2 is less than the opposite direction of the trajectory of feature-1.
- *feature-1 and feature-2 do not potentially intersect*: The line defined by the last direction of feature-2 does not intersect the line segment defined by the start and end points of feature-1. Also the line defined by the first direction of feature-1 does not intersect the line segment defined by the start and end points of feature-2.

Method of Inference

The inference process for selection of features to aggregate is an exhaustive linear search. Successive passes over the feature list are performed at each priority level, starting with the highest level. During each pass, the rule at the current priority level is attempted against each pair of connected features. If at the end of a pass over the feature list, there has been at least one feature aggregation, we reset to the highest priority level for the next pass. If there has been no aggregations at the end of a pass, we lower the priority level by one for the next pass. Once we complete a pass over the feature list at the lowest priority level with no resultant aggregations, the inference process is halted.

This is a simple, albeit inelegant method of inference, but due to the small number of data items (raw features) which we are dealing with, usually no more than ten, it is adequate. The extra overhead needed to implement a more elaborate strategy is not necessary. We can however realize an increase in speed by taking advantage of *short-circuited logic* and arranging rule expressions so that the simplest, most discriminating expressions come before more detailed less discriminating ones.

The next chapter will discuss how the features are used to classify digits.

Chapter 4

Rule-Based Classification of Digits

In this chapter, we present the OLDRES digit classification knowledge base (KB), the function of which is to determine a list of possible identities for the input sample based on the extracted structural features. This list is then passed back to the control module which will decide the final outcome reported by the system. The conclusion will either be that the input sample represents one of the digits 0..9, or that the sample has been rejected as un-classifiable.

In the approach taken here, we attempt to recognize individual characters purely on the basis of their shape. Hence, the system must have knowledge as to what shapes constitute valid digits, and which do not. Knowledge about which structural patterns do not represent valid digits is not directly built in to the system. Rather, a *rejection by failure* scheme is used whereby after considering the input features, if a classification cannot be made with a high enough confidence level, we conclude that the sample is not one of the digits 0..9 (i.e. we conclude "reject").

The knowledge which the system possesses to classify digits is represented by *top-level* structural components. The KB contains a set of *structural prototypes* for each digit expressed in terms of these top-level components. Each prototype describes a particular variation of a digit using from one to three top-level components. For instance, a '1' can be described as being composed of one component (a simple vertical bar), two components (a vertical bar with a hat on top, like an upright harpoon), or three components (a vertical bar, a hat, and a horizontal bar as a base). Thus,

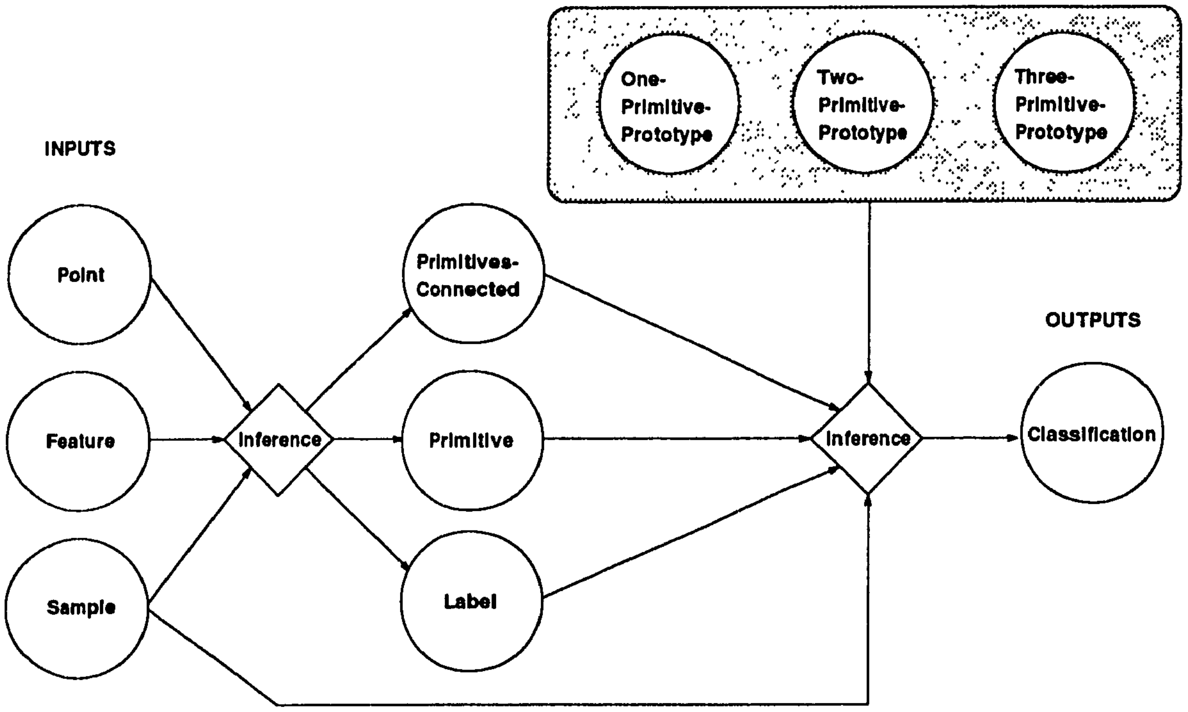


Figure 14: Facts and inferences for asserting classifications.

multiple prototypes for each digit are incorporated into the KB to account for the diversity of writing styles which occurs in the general population.

Given this internal representation of valid digit shapes, the recognition process becomes one of *labeling* the input features, either individually or in groups, as being suitably representative of one or more top-level digit components. Once all appropriate labels have been inferred, they are collectively matched against prototypes in the KB, the outcome of which is a list of possible identities for the input sample. The weighted proportion of a sample's features attributable towards a specific classification is used to calculate a *degree of belief*, or certainty factor, associated with that classification. Figure 14 shows the facts and inferences involved in asserting digit classifications. All of these facts will be described in detail in the remainder of the chapter.

The next section presents a general overview of the digit classification KB, after which CLIPS integration procedures are discussed, and the rule base control strategy is explained. Section 4.4 then proceeds with a detailed discussion of the KB rule

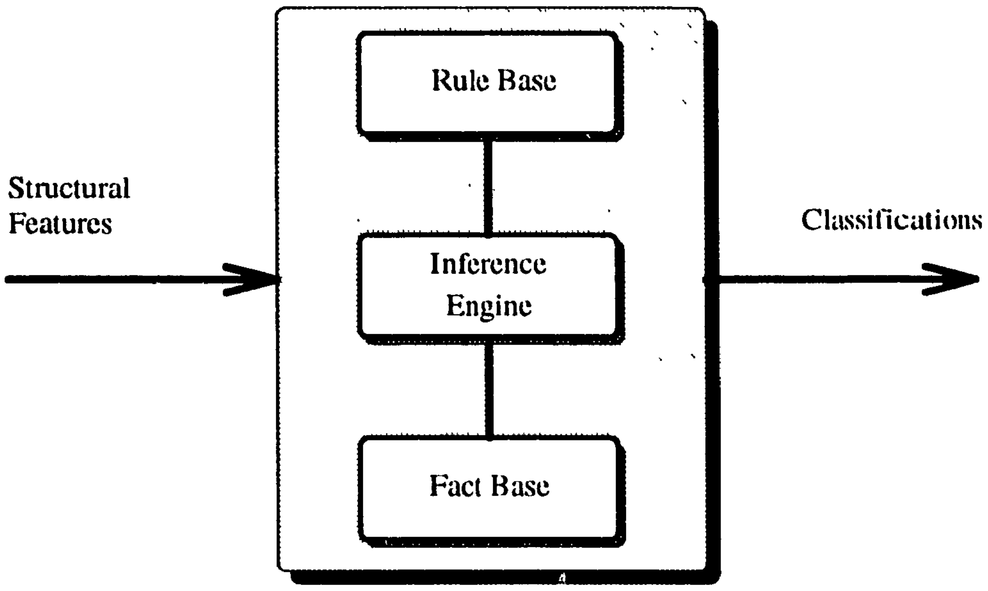


Figure 15: Components of the rule-based classification system.

groups. In section 4.5, the chapter concludes by elaborating on the knowledge acquisition and refinement procedures employed in the development of the knowledge base.

4.1 Overview

The OLDRES digit classification knowledge base is implemented as an embedded, run-time CLIPS expert system module. The CLIPS environment supports the classic forward-chaining, rule-based system paradigm, where rules are of the form **if** ***conditions*** **then** ***actions***. Figure 15 illustrates the three main sub-systems of this environment, namely the rule base, the fact base, and the inference engine. Based on the existence or non-existence of facts in the fact base, the inference engine selects a rule whose left-hand-side (LHS) conditions are satisfied, and *fires* it. When a rule fires, the actions specified on its right-hand-side (RHS) are performed. This process, called the *recognize-act cycle*[16], continues until some desired conclusion is reached, or there are no more fireable rules (i.e. no more rules whose LHS conditions are satisfiable). In our case, the inputs to the system are the structural features representing the input sample, and the outputs are possible identity classifications for the sample.

The KB consists of 93 rules, 14 fact templates, and 4 fact blocks, represented in the form of the three primary CLIPS knowledge constructs, *defrule*, *deftemplate*, and *deffacts* [4]. The fact templates serve as an aid in the KB verification and validation (V&V) process. By defining the structure and allowed slot values for all valid facts used throughout the knowledge base, it is possible to use automated tools to assist in the detection of some KB anomalies. Three of the four fact blocks contain digit prototype information defining known digit patterns in terms of top-level components. There is one fact block each for prototype groups consisting of one, two, and three top-level components. The fourth fact block contains sixteen facts describing the inter-relationships of the sixteen discrete directions used by the system (i.e. which directions are opposite to one another, which directions are adjacent to one another).

4.2 Interfacing with the CLIPS Environment

4.2.1 Initialization

During OLDRES start-up and initialization, the following three CLIPS C functions are called:

```
init_clips();
init_c_rules_1();
reset_clips();
```

`init_clips()` initializes the embedded CLIPS run-time system. `init_c_rules_1()` loads the rule module '1'. This rule module ID corresponds to the ID supplied to the `(rules-to-c)` function when it created the C source files for the knowledge base. `reset_clips()` performs an analogous function to the `(reset)` CLIPS environment command; it clears the agenda and fact list, then asserts all facts listed in the *deffacts* statements into the fact list.

4.2.2 Input

Once the feature vectors representing the input sample have been determined, this information is then passed into the CLIPS fact base by the OLDRES control module via multiple calls to the CLIPS supplied C function `assert()`. This function accepts as an argument, a single formatted text string representing the desired fact to assert. There are three fact structures used to assert facts representing the input sample in the KB. They are the *sample*, *point*, and *feature* facts and are described now.

```
(sample (number               <integer>)
        (database             <symbol>)
        (number-of-features   <integer>)
        (number-of-strokes    <integer>)
        (total-length         <real>)
)
```

The *sample* fact contains information about the input sample as a whole. Only one *sample* fact is asserted per digit sample. The *number* and *database* slots serve to uniquely identify the input sample, if possible. If the sample did not come from a database (i.e. a user drawn sample), an appropriate marker is placed to indicate this. The *number-of-features* slot indicates the number of structural features which have been extracted from the input data. A corresponding number of *feature* facts are asserted into fact base. The *number-of-strokes* slot indicates the number of distinct strokes which compose the input sample. The *total-length* slot is the sum total of all the *length* slot values of all the features of the current sample.

```
(point (ID      <symbol>)
       (type    <symbol>)
       (stroke  <integer>)
       (X-coor  <real>)
       (Y-coor  <real>)
)
```

In the *point* fact pattern, the *ID* slot is an identifier which serves to distinguish it from other *point* facts. The *type* slot is a categorical attribute which can take on one of two values, *terminator* or *hinge*. Each stroke has exactly two terminator points, one at the beginning of the stroke, and one at the end (i.e. the pen-down and pen-up points). A hinge point is a common point shared between two successive features of the same stroke (i.e. the end-point of one feature ≡ start-point of the next feature). For an input sample with N_f features and N_s strokes, the number of *point* facts asserted is $(N_f + N_s)$.

```
(feature (ID           <symbol>)
         (start-pt-ID  <symbol>)
         (end-pt-ID    <symbol>)
         (trajectory   <symbol>)
         (concavity    <symbol>)
         (rotation     <symbol>)
         (length       <real>)
         (spread       <real>)
         (depth        <real>)
)
```

In the *feature* fact pattern, the *ID* slot is an identifier which distinguishes it from other *feature* facts. The *start-pt-ID* and *end-pt-ID* slots indicate the *point* fact *ID* of the start-point and end-point of the feature. The remaining seven slots, ***trajectory, concavity, rotation, length, curvature, spread,*** and ***depth***, are the computed values for the feature vector components, as described in the previous chapter.

Once all the *sample*, *point* and *feature* facts have been asserted into the fact base, execution of the knowledge base is triggered by calling the CLIPS C function **run()**. This transfers control to the CLIPS inference engine. An integer value specifying how many rules to fire is the only argument to this function. A value of -1 is used for our purposes which causes the inference engine to continue firing rules until the agenda is exhausted (i.e. there are no more fireable rules).

4.2.3 Output

Output is passed from the knowledge base via a user defined function. CLIPS provides for the use of user-defined functions on both the LHS and RHS of rules. Hence, we define a function (report-result) to CLIPS, and use it on the RHS of a rule to pass data back into the OLDRES control module. The function takes two arguments:

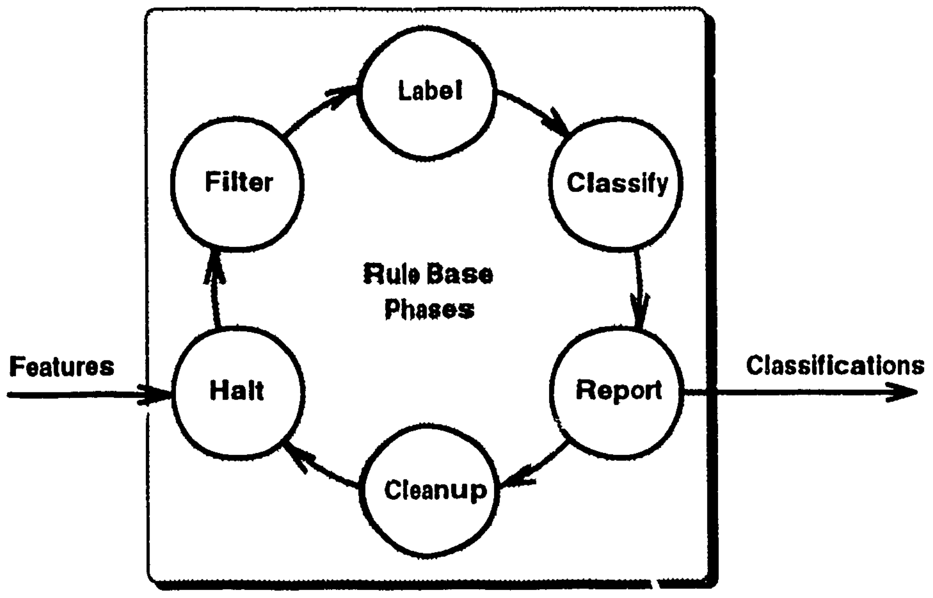


Figure 16: The six phases of execution for rule-based digit recognition.

(report-result <identity> <confidence-level>)

The <identity> parameter specifies one of the digits 0..9, while <confidence-level> is a measure of how certain we are that this classification is correct. This certainty level is expressed in the form of a real number over the range 0 to 1 with 1 representing absolute certainty.

4.3 Controlling the Execution Phase

For each input sample, execution of the rule base proceeds sequentially through six phases, namely, *filter, label, classify, report, cleanup*, and *halt*. These phases are illustrated in a cyclic fashion in Figure 16 because the KB maintains its state from one input sample to the next. The state is maintained by *not* performing a reset before processing each sample. Hence exactly one cycle is performed in order to classify each digit sample. If the system was organized as a linear processing sequence, a reset would be necessary before each new sample is processed, thus re-asserting each of the four *deffacts* fact blocks - 68 facts in total. The cyclic phase structure allows us to avoid this overhead altogether.

Rule-Group	Number of Rules	Function
Filter	2	Purge features determined to be *noise*
Label	82	Label features as digit component primitives
Classify	4	Match primitives to top-level digit prototypes
Report	2	Report inferred classifications
Cleanup	2	Purge inferred facts from the fact-base
Control	1	Select currently active rule-group

Table 1: Breakdown of rule-group size and function

The rule base is partitioned into discrete, non-overlapping sub-sets, or *rule groups*, corresponding to each of these phases, except for the halt phase which has no rules associated with it. To supervise the execution of these rule-groups, there is a single control rule, the sole purpose of which is to affect phase changes. Rules such as this which specify how other rules are to be used are called *meta-rules* [58]. Table 1 gives size and function information for each rule group. This sequentially phased, rule group control strategy is realized by the use of a *phase fact*:

```
(phase (current <symbol>))
```

Since the inference engine matches LHS rule patterns from first to last, we can achieve the desired effect by making the first pattern on the LHS of every rule a literal phase fact. For example:

```
(defrule FILTER-rule
        (phase (current filter))
        (<pattern-2>)
              ⋮
        (<pattern-N>)
=>
        (<action-1>)
              ⋮
        (<action-M>)
)
```

In this way, pattern matching attempts will only proceed on those rules whose phase fact corresponds to the current phase. During each phase, satisfiable rules in the current rule group continue to fire at the discretion of the inference engine. When there are no more applicable rules in the current rule group, the control rule will fire and change the execution context to the next phase. This is guaranteed to happen because the LHS of the phase control rule is *always* satisfied. It is guaranteed to happen at the right time because the rule has a lower salience level (-100) than all other rules in the rule-base, which have the default salience level of 0. Therefore, even though the control rule is always on the agenda, always in a fireable state, the inference engine will only execute it when there are no other rules of a higher salience left on the agenda.

The fact block declaration for phase sequencing, and the phase control rule are shown in Figure 17. Note the use of the procedural (if then) construct on the RHS of the phase control rule, the consequent of which is to execute the CLIPS function **`(halt)`**. Recalling that the expert system was originally invoked by a call to the C function **`run()`**, this will cause that function to return, hence returning control to the OLDRES control module. This active detection and stoppage technique is required to break an otherwise cyclic inference chain. In order to implement phase control in a single rule, we must circularly link the assertion of phase facts, i.e. (phase-after (current halt) (next filter)). That is to say, at the end of processing for a given sample, we must reinitialize the phase fact in preparation for the next sample to be processed. But, if this was performed unchecked, the continuously fireable nature of the control rule would result in an endless loop.

The phase control rule and phase sequence facts also serve as a simple example of how dynamically bound variables are used as field constraints during pattern matching (refer again to Figure 17). The first LHS pattern of the control rule simply declares its salience level in relation to other rules. No pattern matching takes place on this pattern. In pattern two of the rule, the *phase* pattern, the unbound single field variable "?p" occurs for the first time in the slot named *current*. Hence it is treated as a wild card and will match any single value in that slot. Given the fact block in Figure 17(a), the fact (phase (current filter)) matches this pattern, so the value *filter* is bound to variable "?p". The "<-" operator binds the *fact address* of the matched *phase* fact to the variable "?Fph" for subsequent use on the RHS of the rule. In the

```
(deffacts  phase-sequence-information
    (phase-after  (current  filter)   (next  label))
    (phase-after  (current  label)    (next  classify))
    (phase-after  (current  classify) (next  report))
    (phase-after  (current  report)   (next  cleanup))
    (phase-after  (current  cleanup)  (next  halt))
    (phase-after  (current  halt)     (next  filter))
    (phase        (current  filter))
)
```

(a)

```
(defrule  CONTROL-change-phase
    (declare  (salience        -100))
    ?Fph<-    (phase (current  ?p))
    (phase-after     (current  ?p)
                     (next     ?np))
=>
    (modify  ?Fph    (current  ?np))
    (if  (eq ?p halt)  then
        (halt))
#if DEBUG
    (printout t "CONTROL-change-phase" crlf)
#endif
)
```

(b)

Figure 17: (a) The phase sequence fact block. (b) The phase control meta-rule.

third pattern, *phase-after*, "?p" is also used in the slot named *current*. Since "?p" is already bound at this point, it acts as a field constraint, thereby restricting the *current* slot of the fact matching this pattern to have the value *filter* as well. The *next* slot in this pattern introduces a new unbound variable "?np". Since this is the first occurrence of "?np", it is treated as a wild card and will match any single value. The fact (phase-after (current filter) (next label)) matches this pattern (i.e. its field values satisfy the field constraints of the pattern), hence binding the value *label* to the variable "?np". Therefore, the facts

(phase (current filter))
(phase-after (current filter) (next label))

satisfy all the patterns on the LHS of the rule, and it becomes instantiated based on these two facts (i.e. placed on the agenda)

On the RHS of the rule, we change the value of the slot named *current* of the fact specified by address "?Fph", to the value bound to the variable "?np". (i.e. (phase (current filter)) becomes (phase (current label)). The value bound to "?p" is then checked, if it is *halt*, the inference process is stopped.

This ability to bind variables dynamically, pattern-match against them, and use them on the RHS allows for us to write more general, and therefore fewer rules than would otherwise be possible. Without this capability, for example, one would need to write N rules for N possible phase transitions.

4.4 Knowledge Base Rule Groups

4.4.1 Filtering Rules

There are two filtering rules, the purpose of which is to purge features considered to be noise. One rule deletes tiny features which occur at the beginning or end of a stroke. Such features are often the result of *hooks* at the end of a stroke or hesitance on the part of the writer at the beginning of a stroke. The other filtering rule deletes tiny isolated features which constitute a stroke unto themselves. Such noise results from a rapid pen-down-pen-up succession with minimal intermediate lateral pen movement. These also are usually caused by writer hesitation, or when the pen-tip just grazes the writing surface during relocation of the pen in preparation for the next intended stroke.

4.4.2 Labeling Rules

By far, the majority of rules in the digit classification knowledge base are labeling rules. There are eighty-two of them. These rules embody the specific detailed criteria for determining which features, either individually or in combinations, represent the appropriate top-level components of specific digits. Table 2 lists the forty-seven labels used in the knowledge base. These labels were determined heuristically by trial and error, and were influenced primarily by the types of features resulting from the feature extraction system. Some digits have more possible labels than others. This is mostly a result of certain digits having more structural variability than others, but also reflects the need for increased labeling precision with some forms in order to differentiate

Digit	Labels
1	base-of-1, hat-of-1, stem-of-1
2	left-curve-of-2, base-of-2
3	top-bar-of-3, angle-bar-of-3, top-curve-of-3, bottom-curve-of-3, frill-of-3
4	stem-of-4, angle-bar-of-4, middle-bar-of-4, top-curve-of-4, up-stem-of-4
5	vertical-bar-of-5, top-bar-of-5-rightward, S-curve-of-5, top-bar-of-5-leftward, bottom-curve-of-5, top-right-curve-of-5
6	bottom-curve-of-6, right-curve-of-6, top-right-curve-of-6, bottom-loop-of-6
7	stem-of-7, middle-bar-of-7, top-bar-of-7, top-hook-of-7
8	top-loop-of-8, bottom-loop-of-8, S-curve-of-8, Z-curve-of-8, top-hook-of-8, top-right-curve-of-8, top-left-curve-of-8, bottom-right-curve-of-8, stem-of-8, bottom-left-curve-of-8
9	stem-of-9, top-loop-of-9, closed-hook-of-9
0	loop-of-0, left-curve-of-0, right-curve-of-0, over-rotation-of-0
1 or 7	hat-of-1-or-7

Table 2: Top-level structural component labels.

structurally similar digits.

Inferring Primitives

We wish to attach these labels to features individually, in connected pairs, and in connected triples, thereby abstracting the structural features into higher-level descriptive components, or primitives. To represent these primitives, facts of the following structure are used:

```
(primitive (ID          <symbol>)
           (start-pt-ID <symbol>)
           (end-pt-ID   <symbol>)
           (weight      <number>)
)
```

The *ID* slot is, as usual, an identifier to distinguish it from other *primitive* facts. The *start-pt-ID* and *end-pt-ID* slots specify the ID of the start and end point facts for the primitive. The *weight* slot is the sum total of the lengths of all the features from which the primitive has been inferred. Figure 18 illustrates this concept for a primitive derived from three connected features. Each primitive is unique in its elemental

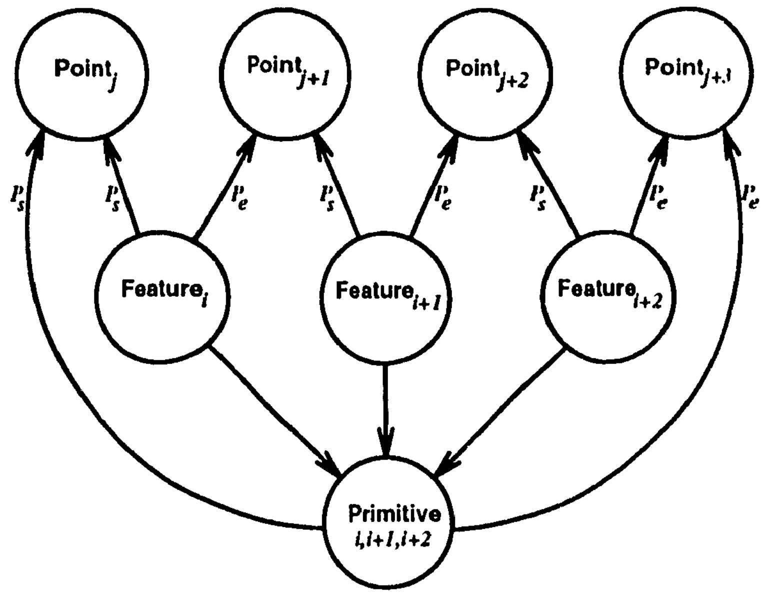


Figure 18: A primitive derived from three connected features. P_s links indicate starting points, P_e links indicate ending points

composition. That is to say, for any given set of features, only one *primitive* fact is asserted. Individual features however, can be an element of more than one primitive. For example, given the three features in Figure 18, we may assert primitives based on the following sets of features (**Feature** has been abbreviated to F):

$$\{F_i\}, \{F_{i+1}\}, \{F_{i+2}\}, \{F_i, F_{i+1}\}, \{F_{i+1}, F_{i+2}\}, \{F_i, F_{i+1}, F_{i+2}\}$$

Primitives are not necessarily constructed for every possible set however. Some combinations may not be useful or representative of a desired shape. But, for each *individual* feature, we automatically assert a corresponding primitive (i.e. for each F_i). There is a rule which does just this. Composite primitives on the other hand are only asserted simultaneously and in conjunction with the inference of an appropriate label. This is explained further in the next subsection.

Inferring Primitive Labels

The process of determining labels for primitives involves detailed pattern matching against feature vector components and end-point positions. Therefore, primitives are

```
(defrule  LABEL-bar-diagonal-top-right-downward
    (phase  (current  label))
    (point  (ID       ?pt)
            (Y-coor   ?yt &:(> ?yt 0.80))
    )
    (feature  (start-pt-ID  ?pt)
              (end-pt-ID    ?pb)
              (curvature    ?c &:(<= ?c 0.10))
              (direction    upper-south-west | lower-south-west)
              (length       ?l &:(> ?l 0.20))
              (ID           ?id)
    )
    (point  (ID       ?pb)
            (Y-coor  ?yb &:(> ?yb 0.20))
    )
=>
    (assert (label (name  hat-of-1)           (primitive-ID  ?id)))
    (assert (label (name  angle-bar-of-3)     (primitive-ID  ?id)))
    (assert (label (name  angle-bar-of-4)     (primitive-ID  ?id)))
    (assert (label (name  vertical-bar-of-5)  (primitive-ID  ?id)))
    #if DEBUG
    (printout t "LABEL-bar-diagonal-top-right-downward: " ?id crlf)
    #endif
)
```

Figure 19: Sample rule to assert single feature primitive labels.

labeled as representing top-level structural components according to their position, size, shape, and orientation. Labeling information is asserted using the following fact structure:

```
(label (name          <symbol>)
       (primitive-ID  <symbol>)
)
```

The *name* slot contains one of the top-level labels from Table 2. The *primitive-ID* slot contains the ID of the primitive to which the label is attached. Figure 19 shows a sample rule for labeling a single-feature primitive. In this rule, four labels are asserted for the primitive in question. This is because when considered in isolation, the primitive could reasonably be any one of the labeled top-level digit components. Hence, the labels-to-primitives relation is many-to-one. We do not need to assert a primitive fact here as all single-feature primitive facts get asserted by a separate, single rule. Figure 20 shows an example of a labeling rule for a compound-feature primitive. Note that in this rule, as in all compound-feature primitive labeling rules,

```
(defrule LABEL-two-part-wavy-base
    (phase  (current  label))
    (point  (ID       ?pbl)
            (X-coor  ?xbl &:(<= ?xbl 0.25))
            (Y-coor  ?ybl &:(<= ?ybl 0.31))
    )
    (feature  (start-pt-ID  ?pbl)
              (end-pt-ID    ?pbm)
              (rotation     positive | undefined)
              (direction    lower-north-east | upper-east        |
                            lower-east       | upper-south-east )
              (length       ?l1)
              (ID           ?id1)
    )
    (feature  (start-pt-ID  ?pbm)
              (end-pt-ID    ?pe)
              (rotation     negative | undefined)
              (direction    right-north      | upper-north-east |
                            lower-north-east | upper-east       |
                            lower-east       | upper-south-east )
              (length       ?l2)
              (ID           ?id2)
    )
=>
    (bind  ?cid  (format nil "%s-%s" ?id1 ?id2))
    (assert  (primitive  (ID  ?cid)  (weight       =(+ ?l1 ?l2))
                                     (start-pt-ID  ?pbl)
                                     (end-pt-ID    ?pe)))
    (assert  (label  (name  base-of-2)  (primitive-ID  ?cid)))
    #if DEBUG
    (printout t "LABEL-two-part-wavy-base: " ?cid crlf)
    #endif
)
```

Figure 20: Labeling rule for a compound-feature primitive.

we assert both a *label* fact, and a *primitive* fact.

Inferring Primitive Connectivity

The final function performed during the labeling phase is to determine connectivity information of primitives. In order to consider the input sample as a whole, we need to know which primitives are connected (sequentially) to each other and which are not. This information is represented by facts of the following structure:

```
(primitives-connected  (status  <boolean>)
                       (ID1     <symbol>)
                       (ID2     <symbol>)
)
```

These facts are concluded on the RHS of two rules. One which asserts for sequentially connected primitives, and one which asserts for non-connected ones. In order to infer connectedness, the former rule simply matches the *end-pt-ID* of one primitive to the *start-pt-ID* of another. Connected primitives are used to infer the overall shape of an entire stroke. The latter rule which asserts facts for non-connected primitives, only considers features belonging to different strokes. We are not interested in finding non-connected primitives belonging to the same stroke. This is because we use non-connectedness specifically as an indication that primitives do not belong to the same stroke.

4.4.3 Classification Rules

Once all the appropriate *primitive*, *label*, and *primitives-connected* facts have been asserted, the classification phase commences. During this stage of processing, subsets of the primitives are matched against the fact base of known top-level digit prototypes to infer digit identity classifications for the input sample. As mentioned in section 4.1, a collection of single, double, and triple component prototypes are used for this purpose. The fact structures for these top-level prototypes are as follows:

```
(one-primitive-prototype  (digit    <symbol>)
                          (label-1  <symbol>)
)


(two-primitive-prototype  (digit               <symbol>)
                          (label-1             <symbol>)
                          (connected-1-to-2    <boolean>)
                          (label-2             <symbol>)
)
```

```
(defrule CLASSIFY-two-primitives
    (phase (current      classify))
    (sample (total-length ?wt))
    (label      (name           ?label1)
                (primitive-ID  ?prim1))
    (primitive (ID             ?prim1)
                (weight        ?w1))
    (primitives-connected  (status  ?cnct1-2)  (ID1  ?prim1)
                                               (ID2  ?prim2))
    (label      (name           ?label2)
                (primitive-ID  ?prim2))
    (primitive (ID             ?prim2)
                (weight        ?w2))
    (two-primitive-prototype  (digit             ?digit)
                              (label-1           ?label1)
                              (connected-1-to-2  ?cnct1-2)
                              (label-2           ?label2))
=>
    (assert  (classification  (character-ID  ?digit)
                              (certainty     =(/ (+ ?w1 ?w2) ?wt))))
#if DEBUG
    (printout t "CLASSIFY-two-primitives " ?digit
                " : " ?label1 "(" ?prim1 ") " ?label2 "(" ?prim2 ")" crlf)
#endif
)
```

Figure 21: Classification rule for two primitive prototypes.

```
(three-primitive-prototype  (digit              <symbol>)
                            (label-1            <symbol>)
                            (connected-1-to-2   <boolean>)
                            (label-2            <symbol>)
                            (connected-2-to-3   <boolean>)
                            (label-3            <symbol>)
)
```

For each of these three prototype classes, the *digit* slot contains the identity of the numeral for which the prototype describes (i.e. '0'..'9'). The *label-N* slots indicate the top-level component labels for the prototype, while the *connected-M-to-N* slots specify whether or not the *label-M* primitive is connected to the *label-N* primitive.

Given a set of valid digit descriptors represented in this fashion, we use a total of three rules to match inferred primitive labels to these prototypes. One rule concludes identity classifications for each of the one, two, and three component prototypes. The classification rule for two component prototypes is shown in Figure 21. These rules also compute an associated certainty fa^tor (CF) for each classification. The CF is simply a percentage measure of the total length of the sample represented by the primitives under consideration. This is calculated by summing the *weight* slot values of the one, two, or three primitives, and dividing by the total weight of the sample – the *weight* slot value of the *sample* fact. In other words, it is the total length of all the features contributing to the classification, divided by the total length of all features of the sample. Hence, if the primitives supporting the instantiation of one of these three classification rules are collectively derived from all the features of the sample, a CF of 100% will be concluded. A fourth classification rule is used to purge redundant conclusions since it is possible to infer a given digit identity more than once. In this case, the conclusion with the highest CF is retained while the others are discarded.

4.4.4 Reporting Rules

The report rule group consists of only two rules. The first is used to report inferred classifications back to the OLDRES control module, the second is used for debugging purposes. Classifications and their associated certainty factors are passed back into the control module by means of a CLIPS user-defined function on the RHS of the first rule. Only high confidence classifications are passed back however, those with certainty factors of 80% or higher. This threshold value was experimentally determined. The control module then sorts the returned values and concludes the identity with the highest CF. If two or more identities are tied for the top spot, we conclude rejection. The debugging rule is used to skip over samples which have been marked as *bad samples* by the developer during the knowledge acquisition phase (discussed in section 4.5).

4.4.5 Cleanup Rules

Once the reporting phase has returned all classifications to the OLDRES control module, the cleanup phase begins. Here, we retract all facts which were asserted over the course of processing the current input sample. These include the externally asserted *feature*, *sample* and *point* facts, for which there is one rule, and the internally deduced *label, primitive, primitives-connected*, and *classification* facts, for which there is another rule. Recalling that the KB maintains its state from processing one sample to the next, retraction of these facts is necessary in order to prepare the fact base for processing of the next digit sample.

4.5 Knowledge Acquisition

This section describes the steps followed during the iterative process of acquiring, modeling, implementing, and testing the knowledge which went into the digit classification knowledge base. The facilities provided by the OLDRES graphical user interface (GUI) were an integral and vital part of the knowledge acquisition and refinement process. Without the availability of these services built-in to the system, development of the rule base would have been much more time consuming, more tedious, and less effective overall. A training database composed of one-thousand digit samples was used for system development. This set of cases is the basis from which the knowledge base was created. More details about the training set are given in Chapter 5.

Once the general concept was decided upon for attempting to recognize digits based on their structural composition, a rule-based expert systems approach was selected for the implementation because it allows for the formalizing of qualitative heuristics in a modular and easily extensible fashion. First, preliminary *conceptual models* [6] were drawn up in a mostly graphical fashion. These informal designs were basically bubble graphs with links showing simulated inference nets and external interfaces. The inputs for these designs were already decided upon as being the outputs from the structural feature extractor, which was developed beforehand. This laid the groundwork for the basic KB structure. A few approaches were considered and some quick prototypes were created to test out the feasibility of the designs. A crude form of rule-induction was attempted, along with a design that used short, single-rule inference chains of the form *features* $\rightarrow$ *classifications*. These designs met with limited success in the prototyping stage, and were eventually abandoned in favor of the present architecture which proved to be more flexible.

Having determined a working model for the rule base which abstracts feature vectors up to descriptive top-level components, the process of incrementally expanding the capabilities of the KB was done in a seven stage spiral-like progression. At the outset, the system had no primitive labeling rules, and no digit prototypes with which to match the labels against. Therefore, no classifications could be made and all samples were rejected. So, starting with first sample of the training set (i.e. sample 1 of 1000), we iterate through the following seven steps:

1. Load the current sample into OLDRES and extract the structural features. View the results and decide if the sample is *good.* If so, proceed to the next step. If not, mark it as a *bad sample* and repeat this step for the the next test case. Marking a sample as bad is a subjective judgement where it is felt that either the sample is not generally representative of a valid formation for a particular character class, or its identity is ambiguous.

2. Make necessary additions and/or modifications to the KB for it to correctly recognize the current sample.

3. Rebuild the entire system, thereby incorporating the new knowledge gained from the current test case. Syntactic errors in the KB are found at this stage. If any occur, return to step 2.

4. Run the KB through the automated checking tool CRSV[1] in an attempt to detect possible semantic errors. If any are detected, return to step 2.

5. Load the current sample once again and attempt to recognize it. If it is not classified correctly, examine the rule-trace debugging output and return to step 2.

6. Perform a complete regression test of all the previous test cases to see if the modifications made to the KB for the current sample have adversely affected other test cases. If one of the previous test cases now results in a misclassification, either revoke the KB modification just made and return to step 2, or accept the new modifications as correct, set the misclassified previous test case to be the current sample and return to step 2.

[1]The CLIPS Cross-Reference, Style, and Verification utility

7. Seek forward for the next misclassified test case, set this sample to be the current sample and return to step 1.

This procedure is referred to as a spiral in the sense that the cycle proceeds to train the KB on all ten digits at the same time, while ensuring that previous test cases (those from inner cycles of the spiral) remain unaffected. In effect, we are building the KB outwards in all directions. Since the occurrence of particular digits in the training set is random, a relatively even mix of sample identities will be encountered by training the system on the samples in the order that they occur in the database. This is in contrast with the alternative approach of training the system on only one class of digit at a time before moving on to the next. Moreover, if we were to proceed with a testing window consisting of only the current sample, new modifications may result in undesirable rule interactions which cause previously successful test cases to fail. Thus, we could wind up thrashing, and locating the source of the problem could be difficult.

In the next chapter, we shall present and review some experimental results from the system.

Chapter 5

Results and Analysis

This chapter presents some sample results obtained from two data sets. The first data set was used to train the system. This involved the development and refining of the feature extraction algorithms, as well as the digit classification knowledge base. The second data set was used as independent test cases for the purpose of assessing system performance.

The system was trained on a first set of 1000 samples of the ten numerals 0 through 9. This balanced training set was obtained from ten individuals, all members of the Department of Computer Science at Concordia University. A balanced set is one which contains an equal representation of samples from each character class. A mix of right and left-handed individuals were involved. Ten instances each of the ten numerals were requested in a random order from each of the ten subjects, for a total of one hundred samples per subject. To generate this training set, a Summagraphics SummaSketch MM1201 digitizer tablet and electronic pen, interfaced to a personal computer was used. Since the subjects did not have any prior experience with using the digitizer tablet, they were given the opportunity to familiarize themselves with the equipment by inputting a few practice characters. No special instructions or constraints were imposed regarding the size, placement, orientation, complexity, etc. of the characters. The data acquisition program running on the personal computer instructed the subjects on which character to draw. In order to avoid biasing the shape of the patterns, subjects were prompted using words rather than numerals.

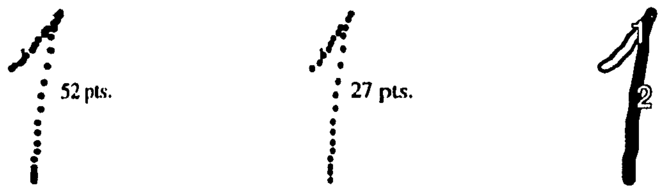
For example, "Please draw the digit 'six' ", instead of "Please draw the digit 6".

The system was evaluated on a balanced test set of 2300 samples obtained from 23 other members of our department. These test cases were generated under similar conditions as the training set, except that a few of the writers were character recognition experts from our research team. These individuals were asked to draw difficult and confusing samples in a variety styles. The same data acquisition hardware and software was used to generate both data sets.

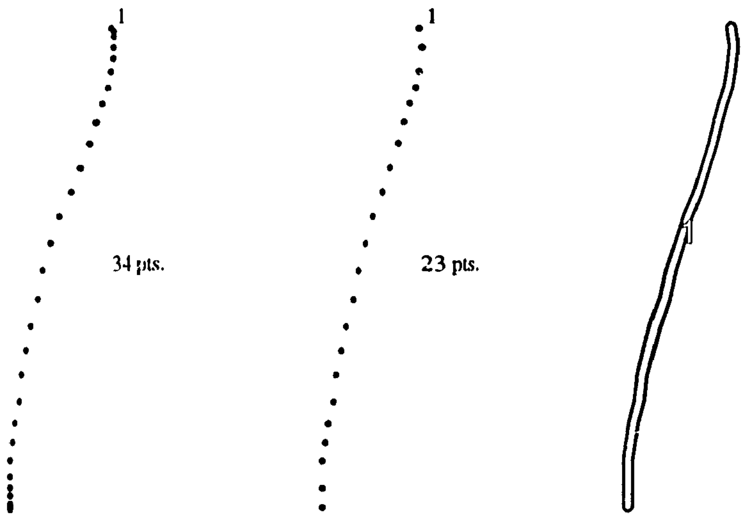
5.1 Structural Feature Extraction

This section gives some sample results of structural feature extraction by presenting four samples of each of the digits 0 through 9. They were randomly selected from the training set with the intention of demonstrating the diversity of writing styles present in the database. For each sample, three instances of the pattern are displayed across a single row, accompanied underneath by a tabular summary of the feature vectors. For each row, the image on the left shows the original raw data points, while the center image shows the filtered sample data points. The number of points is indicated just to the right of each of these patterns. The number, order, and direction of strokes are indicated by marking the first point of each stroke with a '1', '2', etc. for the first, second, etc. strokes. The right image of each row displays the stroke segments used in calculating the feature vectors. These segments are numerically marked for cross-referencing into the included feature vector table just below each row of images. This information was automatically generated using the facilities provided by the OLDRES output options pop-up frame (Sec. 2.2.6). All 40 of these samples were correctly classified by the system.

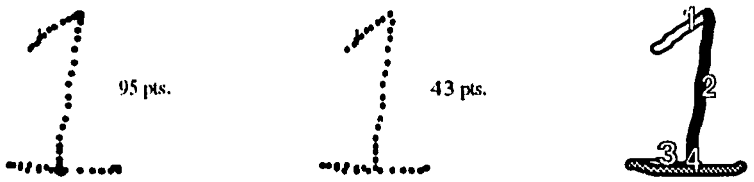
5.1.1 Samples of the digit '1'



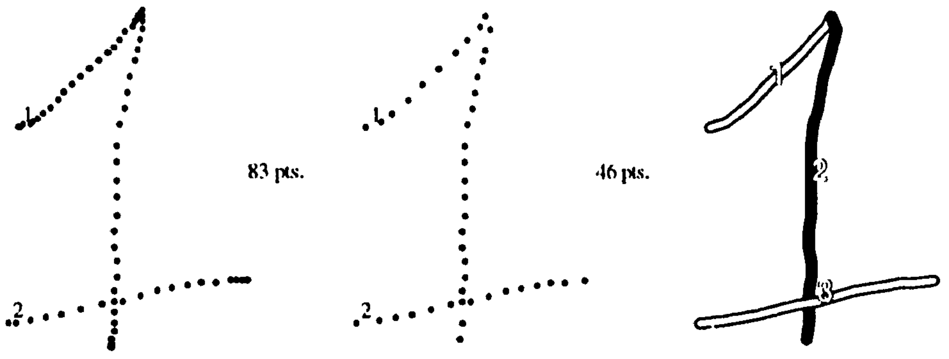
Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	upper-north-east	right-north	negative	0.40	0.01	0.40	0.02	(0.00,0.67),(0.97,1.00)
2	left-south	lower-east	negative	0.98	0.00	0.97	0.03	(0.97,1.00),(0.50,0.00)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left-south	right-north	undefined	1.00	0.01	0.99	0.03	(0.98,1.00),(0.00,0.01)

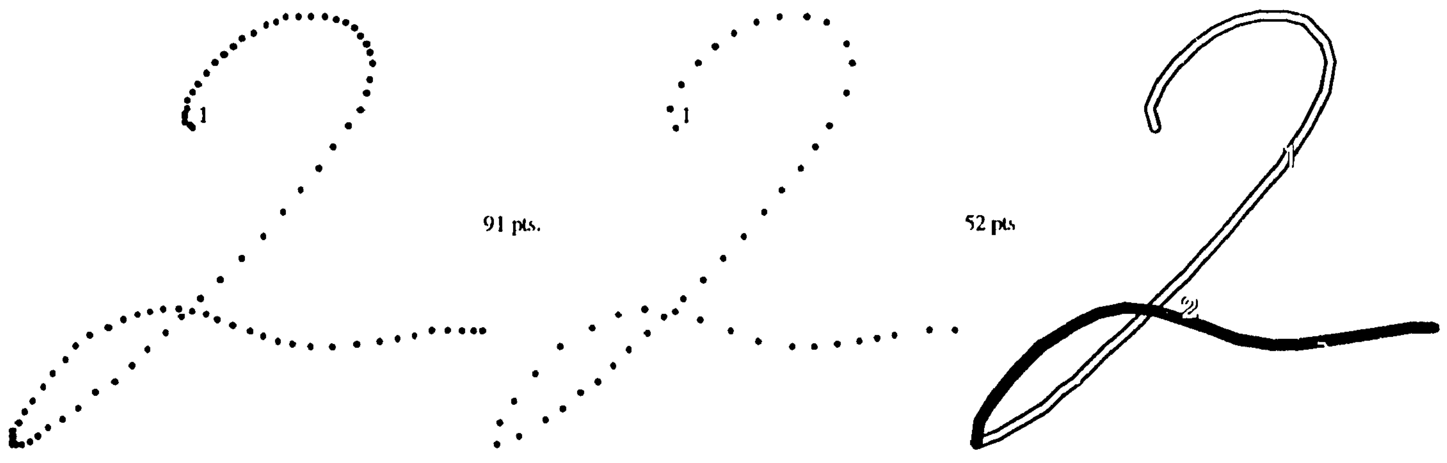


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-north-east	upper east	positive	0.31	0.01	0.31	0.02	(0.23,0.77),(0.65,0.99)
2	left-south	left-south	undefined	0.80	0.01	0.79	0.02	(0.65,0.99),(0.51,0.03)
3	upper west	upper-west	undefined	0.29	0.00	0.29	0.01	(0.51,0.03),(0.00,0.05)
4	lower-east	upper east	negative	0.57	0.00	0.57	0.02	(0.00,0.05),(0.98,0.02)

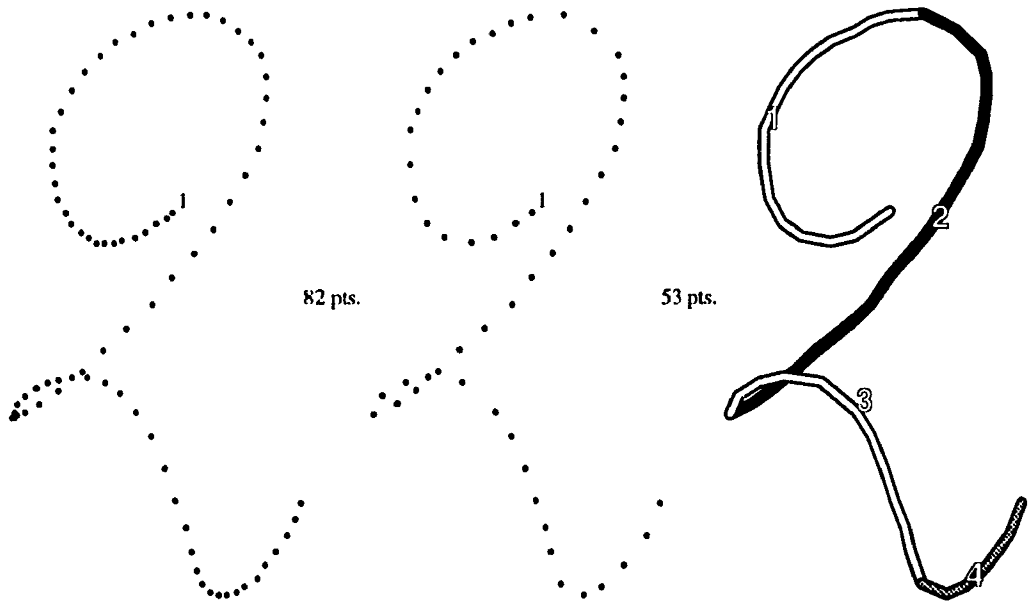


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-north-east	upper-north-east	negative	0.40	0.02	0.39	0.02	(0.05,0.65),(0.55,0.99)
2	left-south	right-south	negative	0.79	0.01	0.79	0.02	(0.55,0.99),(0.43,0.02)
3	upper-east	lower-west	undefined	0.56	0.00	0.56	0.01	(0.00,0.07),(0.96,0.20)

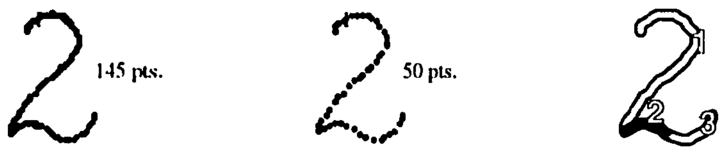
5.1.2 Samples of the digit '2'



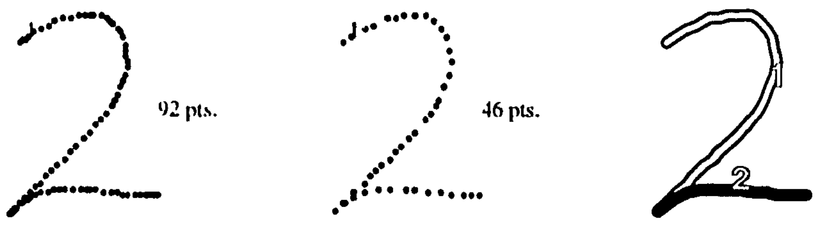
Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower south-west	upper-south west	positive	1.25	0.55	0.57	0.21	(0.38,0.74),(0.01,0.00)
2	upper-east	upper-south east	positive	0.84	0.11	0.74	0.15	(0.01,0.00),(0.98,0.27)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	right north	upper-east	positive	0 65	0 52	0 31	0 21	(0 55,0 66),(0 65,1 00)
2	lower south west	lower-west	positive	0 82	0 17	0 68	0 16	(0 65,1.00),(0 01,0.31)
3	upper south east	right-south	positive	0 52	0 25	0 39	0 13	(0 01,0 31),(0 66,0 02)
4	lower-north-east	right-north	negative	0 23	0 15	0 20	0 05	(0 66,0 02),(1.00,0.16)

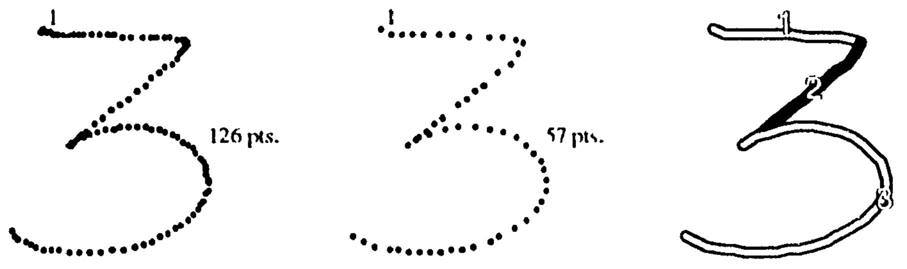


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left south	upper-south-west	positive	1 26	0 47	0 66	0 36	(0 19,0.85),(0 01,0.06)
2	lower-east	right-south	positive	0 35	0 11	0 31	0 07	(0 01,0 06),(0 56,0 01)
3	lower-north-east	upper-north west	negative	0 34	0 14	0 29	0 07	(0 56,0 01),(1 00,0 20)

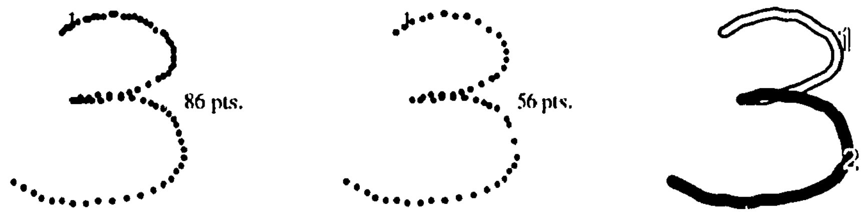


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left south	upper-south-west	positive	1 34	0 50	0 67	0 43	(0.06,0 86),(0 02,0 02)
2	upper east	upper south-east	positive	0 60	0 04	0 58	0 06	(0 02,0 02),(0 98,0.09)

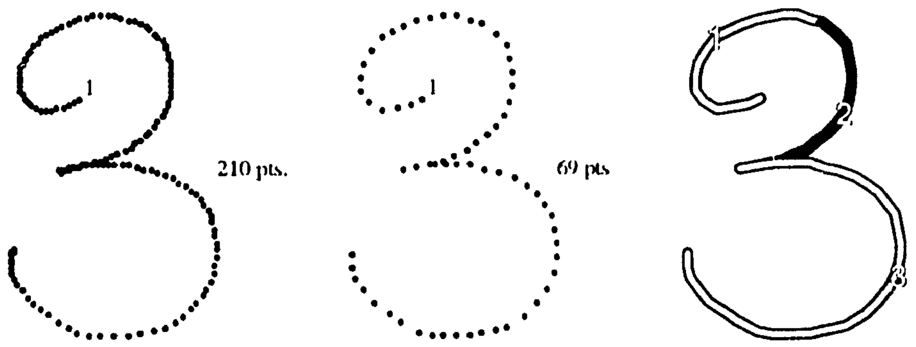
5.1.3 Samples of the digit '3'



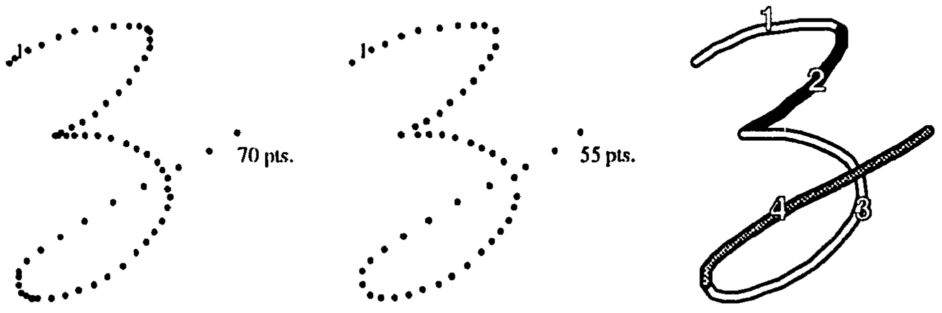
Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-east	upper-west	undefined	0.48	0.01	0.48	0.01	(0.16,1.00),(0.88,0.94)
2	upper-south-west	upper south-west	positive	0.52	0.01	0.51	0.02	(0.88,0.94),(0.30,0.48)
3	lower-south west	upper-west	positive	1.34	0.73	0.36	0.48	(0.30,0.48),(0.01,0.07)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	right-south	lower-west	positive	0.98	0.71	0.28	0.41	(0.29,0.91),(0.39,0.54)
2	lower south-west	upper-west	positive	1.32	0.69	0.41	0.47	(0.39,0.54),(0.00,0.11)

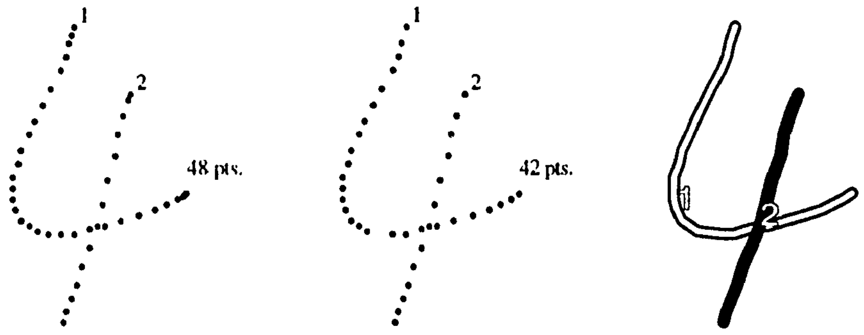


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	upper-north-east	lower east	positive	0.63	0.60	0.25	0.19	(0.35,0.74),(0.63,0.98)
2	lower-south-west	upper-west	positive	0.58	0.25	0.43	0.16	(0.63,0.98),(0.26,0.52)
3	lower-south-west	lower-north west	positive	1.31	0.80	0.26	0.49	(0.26,0.52),(0.00,0.26)

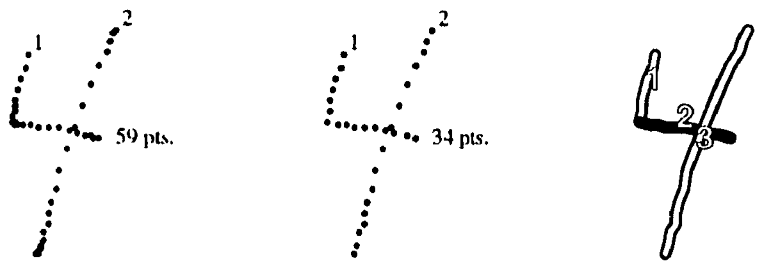


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	upper east	right south	positive	0 43	0 04	0 41	0 05	(0 00,0 86),(0 62,0 98)
2	lower-south west	lower-west	positive	0 41	0 05	0 39	0 05	(0 62,0 98),(0 21,0 60)
3	left south	lower west	positive	0 98	0 55	0 44	0 35	(0 21,0 60),(0 05,0 05)
4	lower-north east	upper-east	positive	0 76	0 03	0 74	0 05	(0 05,0.05),(1 00,0.60)

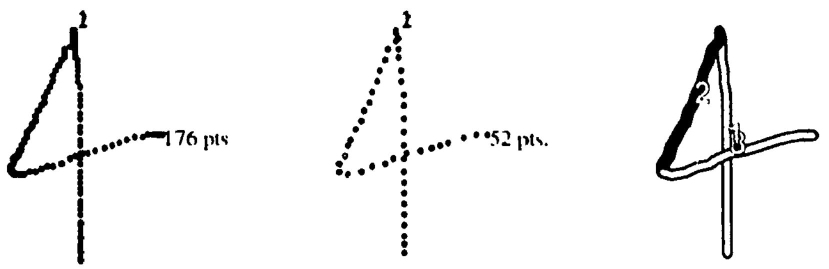
5.1.4 Samples of the digit '4'



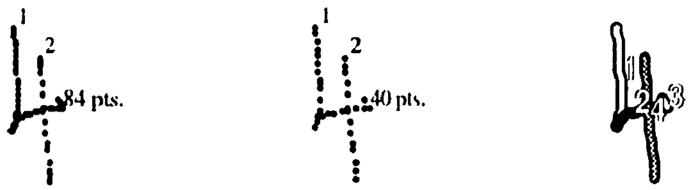
Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower south-east	lower north-east	negative	1 12	0 47	0 59	0 45	(0 35,1.00),(1.00,0 43)
2	left-south	upper-south west	positive	0 69	0 00	0 69	0 01	(0 69,0 77),(0 30,0 00)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left south	upper north-east	negative	0 28	0 00	0 28	0 01	(0.15,0 89),(0 00,0 59)
2	lower east	lower-west	positive	0 36	0 01	0 36	0 01	(0 00,0 59),(0 84,0 52)
3	left south	left-south	undefined	0 96	0 00	0 96	0 02	(1 00,1 00),(0 25,0 00)

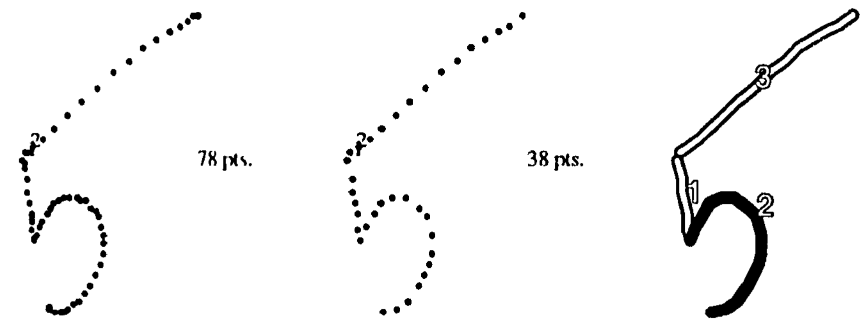


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	right-south	left-south	positive	0.81	0.00	0.81	0.02	(0.39,1.00),(0.43,0.03)
2	lower-south-west	upper-north east	undefined	0.55	0.01	0.54	0.01	(0.39,0.98),(0.01,0.38)
3	upper-east	upper-east	negative	0.56	0.01	0.55	0.02	(0.01,0.38),(0.99,0.55)

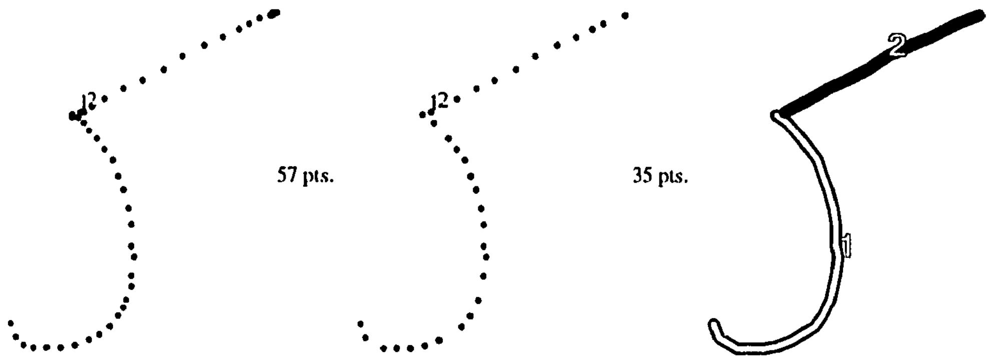


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left-south	left-north	positive	0.61	0.01	0.60	0.03	(0.02,1.00),(0.01,0.36)
2	upper-east	lower-east	positive	0.37	0.05	0.35	0.05	(0.01,0.36),(0.97,0.49)
3	upper-north-west	upper-north-west	undefined	0.05	0.00	0.05	0.00	(0.97,0.49),(0.90,0.54)
4	right-south	left-north	undefined	0.73	0.00	0.73	0.01	(0.54,0.80),(0.76,0.03)

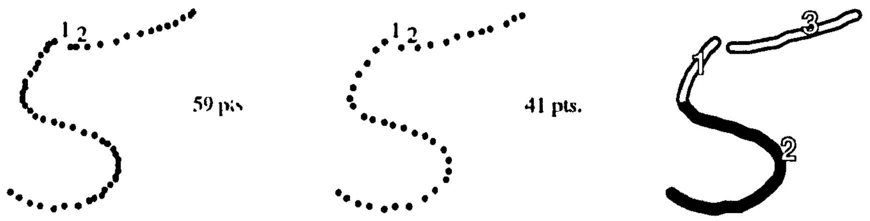
5.1.5 Samples of the digit '5'



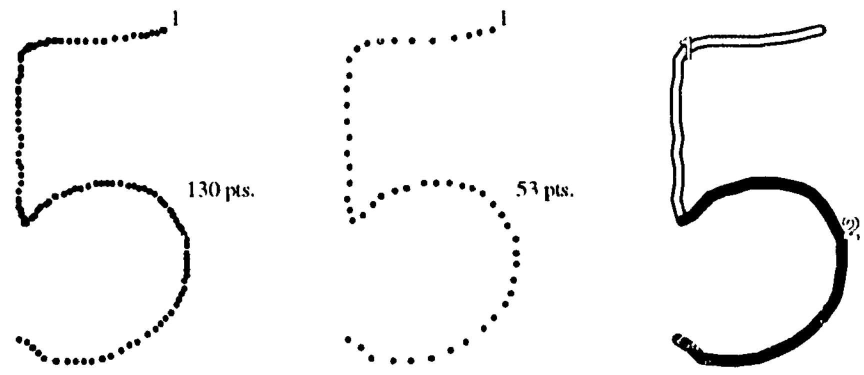
Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	right-south	left-north	undefined	0 24	0 00	0 24	0 00	(0 00,0 51),(0 07,0 24)
2	right-south	lower-south-west	positive	0 62	0 66	0 21	0 20	(0 07,0.24),(0 19,0.00)
3	lower-north east	left south	positive	0 63	0 00	0 63	0.02	(0 01,0 53),(0 97,0 99)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left-south	lower-north-west	positive	0 82	0 39	0 50	0 24	(0 23,0 69),(0 00,0 08)
2	lower-north east	upper-south-west	positive	0 50	0 00	0 49	0 01	(0 26,0 70),(0 97,0 99)

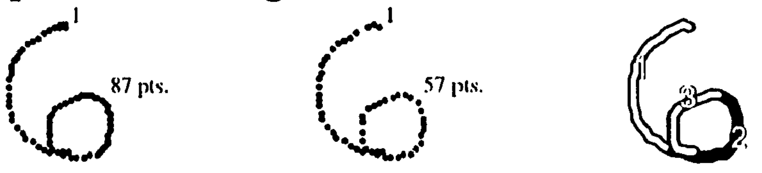


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-south-west	upper-east	negative	0 28	0 04	0 27	0.03	(0 23,0 84),(0 07,0 51)
2	left-south	upper-west	positive	0 93	0 66	0 32	0 38	(0 07,0 51),(0 00,0 09)
3	upper east	lower north east	negative	0 47	0 01	0 47	0.02	(0 32,0 82),(0.98,0 98)

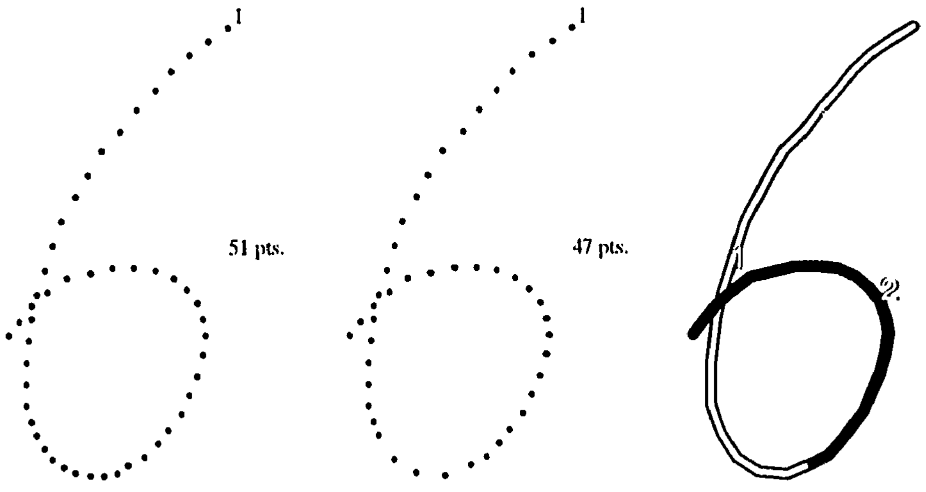


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-south-west	lower-south east	negative	0.85	0.25	0.63	0.27	(0.86,1.00),(0.04,0.42)
2	left-south	lower-west	positive	1.20	0.74	0.31	0.44	(0.04,0.42),(0.00,0.07)

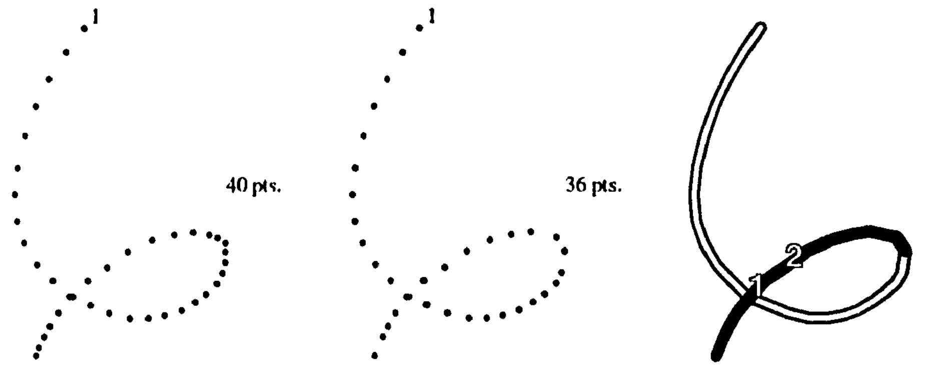
5.1.6 Samples of the digit '6'



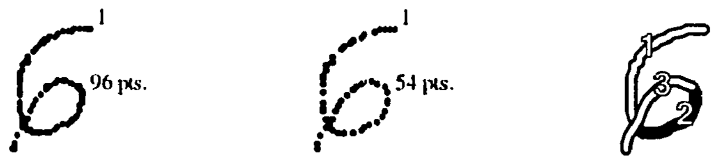
Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left-south	lower-east	negative	1.05	0.28	0.76	0.30	(0.57,0.99),(0.40,0.04)
2	upper-north-east	lower-north-west	negative	0.72	0.42	0.42	0.25	(0.40,0.04),(0.81,0.46)
3	lower-south-west	lower-east	negative	0.59	0.38	0.37	0.18	(0.81,0.46),(0.57,0.04)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left-south	upper-east	negative	1.18	0.23	0.90	0.25	(1.00,1.00),(0.55,0.02)
2	upper-north-west	upper-south-west	negative	0.84	0.57	0.36	0.33	(0.55,0.02),(0.00,0.31)

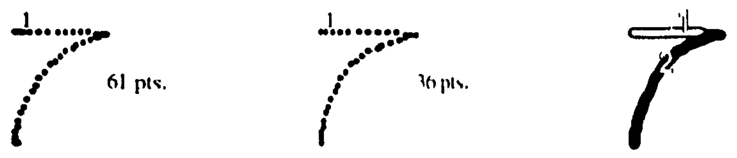


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-south east	upper-north-east	negative	1 28	0 46	0 68	0 42	(0 32,1 00),(1 00,0 31)
2	upper-south-west	right-south	negative	0 64	0 14	0 55	0 12	(1 00,0.31),(0.12,0 00)

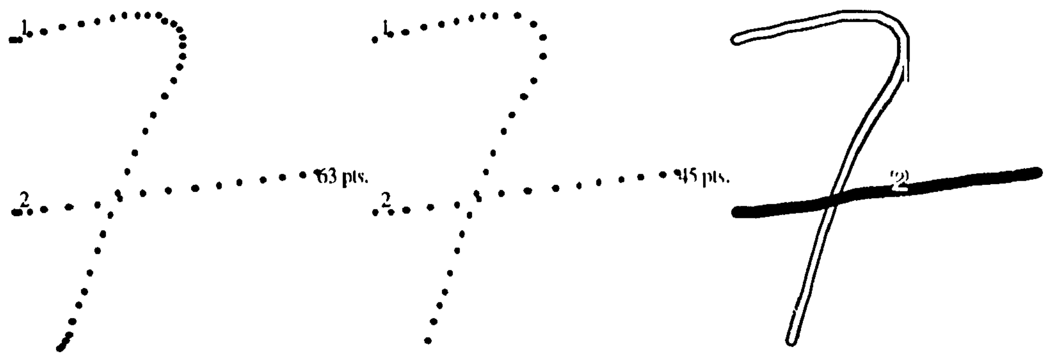


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-south-west	upper south-east	negative	0 99	0.18	0.81	0 25	(1.00,1.00),(0 15,0 20)
2	lower-north-east	upper-north-west	negative	0 62	0 25	0 46	0 16	(0.15,0 20),(0 85,0 52)
3	upper south-west	right-south	negative	0 73	0 14	0.62	0 15	(0.85,0 52),(0 01,0 01)

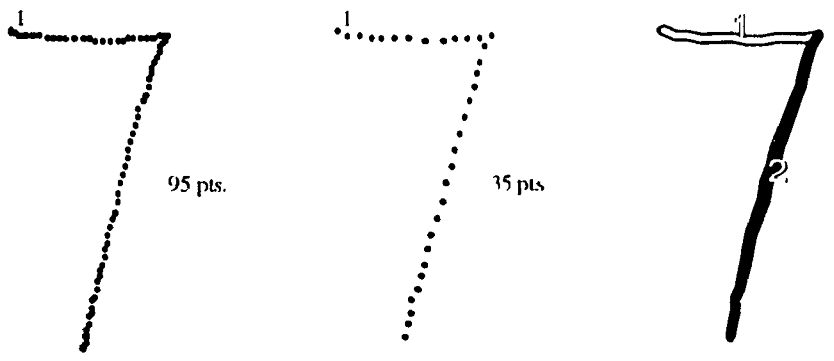
5.1.7 Samples of the digit '7'



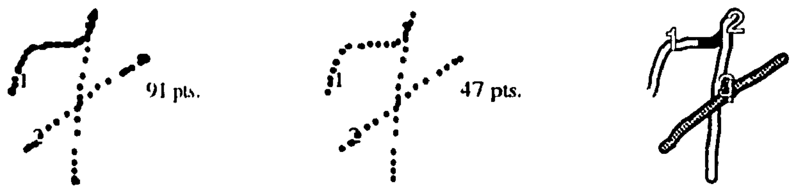
Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-east	lower-west	positive	0.63	0.00	0.64	0.01	(0.00,1.00),(1.00,0.96)
2	lower-south-west	lower-south east	negative	1.05	0.08	0.97	0.18	(1.00,0.96),(0.01,0.01)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	right-south	lower-south-west	positive	1.13	0.40	0.68	0.36	(0.00,0.92),(0.17,0.01)
2	upper-east	upper-east	undefined	0.67	0.00	0.67	0.00	(0.01,0.41),(1.00,0.53)

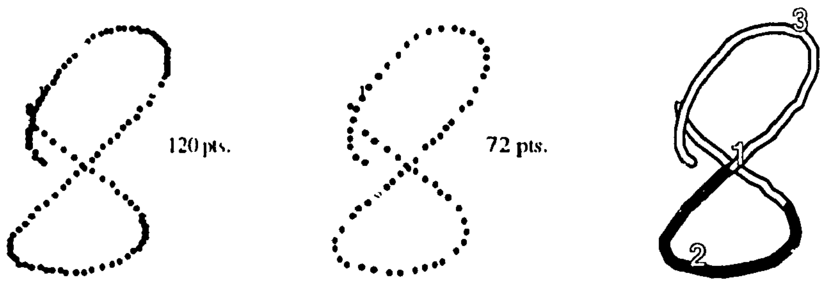


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-east	upper-east	negative	0.45	0.01	0.45	0.02	(0.00,1.00),(1.00,0.99)
2	left-south	left-south	undefined	0.89	0.00	0.89	0.02	(1.00,0.99),(0.17,0.03)

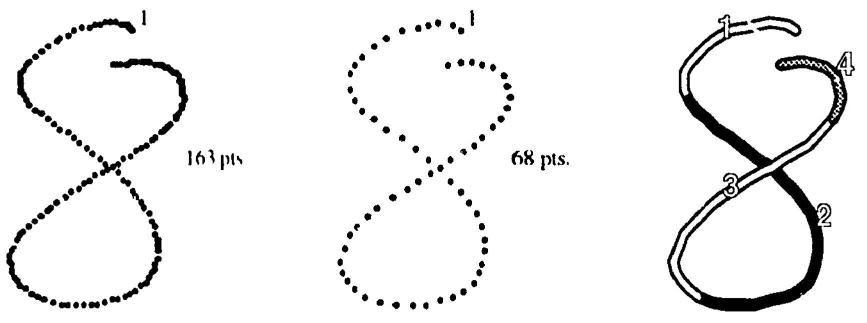


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower north east	lower south east	positive	0.40	0.16	0.33	0.10	(0.00,0.53),(0.41,0.81)
2	upper north east	lower north west	negative	0.21	0.07	0.20	0.03	(0.41,0.81),(0.64,0.98)
3	left south	upper east	negative	0.77	0.01	0.76	0.03	(0.64,0.98),(0.48,0.03)
4	lower north east	lower-south west	positive	0.69	0.00	0.68	0.12	(0.11,0.20),(1.00,0.72)

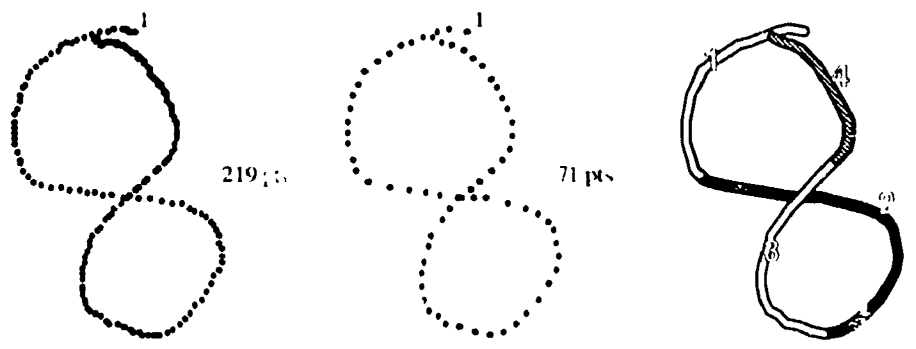
5.1.8 Samples of the digit '8'



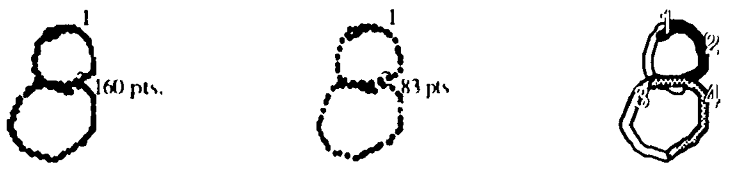
Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	upper south east	upper-south east	negative	0.54	0.03	0.52	0.04	(0.11,0.68),(0.82,0.26)
2	lower north-west	lower-north east	positive	1.02	0.77	0.24	0.40	(0.82,0.26),(0.47,0.43)
3	upper west	lower south west	negative	1.32	0.89	0.15	0.50	(0.47,0.43),(0.20,0.45)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	upper south-west	lower south east	negative	0.48	0.19	0.39	0.10	(0.72,0.98),(0.09,0.72)
2	right south	lower north west	positive	1.08	0.44	0.60	0.37	(0.09,0.72),(0.15,0.03)
3	upper north east	upper east	positive	0.81	0.16	0.68	0.15	(0.15,0.03),(0.95,0.65)
4	upper north-west	upper south west	negative	0.32	0.27	0.23	0.10	(0.95,0.65),(0.64,0.85)

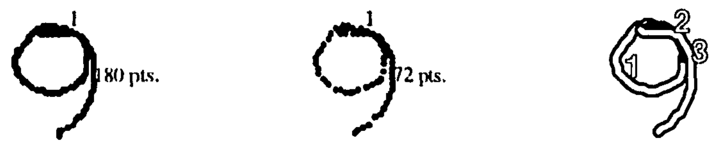


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-south-west	upper-south east	negative	0.63	0.23	0.49	0.17	(0.57,0.99),(0.09,0.50)
2	lower-south-east	lower-west	positive	0.90	0.42	0.52	0.32	(0.09,0.50),(0.65,0.00)
3	right-north	lower-north-east	positive	0.66	0.28	0.47	0.19	(0.65,0.00),(0.72,0.57)
4	upper-north west	upper-west	negative	0.44	0.14	0.38	0.08	(0.72,0.57),(0.40,0.98)

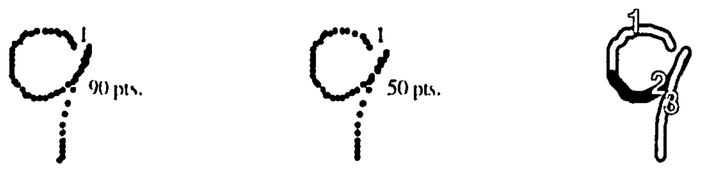


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower-south west	upper south east	negative	0.56	0.29	0.40	0.16	(0.27,0.98),(0.36,0.58)
2	right-north	upper-west	negative	0.86	0.66	0.29	0.32	(0.36,0.58),(0.41,0.92)
3	left-south	lower-east	negative	0.91	0.56	0.42	0.32	(0.69,0.49),(0.55,0.01)
4	left north	upper-south west	negative	[illegible]	0.46	0.48	0.32	(0.55,0.01),(0.31,0.55)

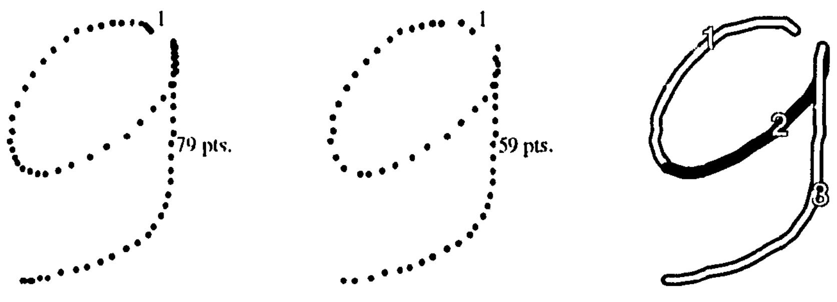
5.1.9 Samples of the digit '9'



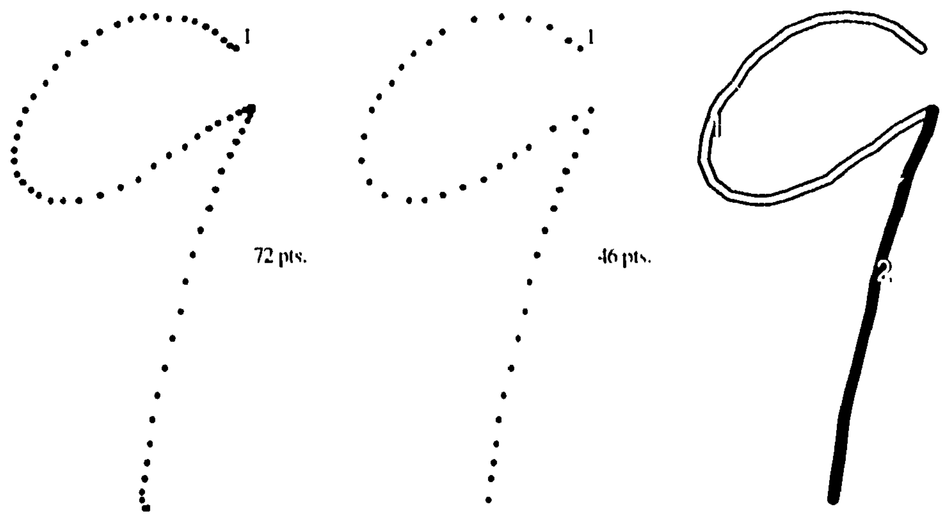
Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower south east	lower-north-east	negative	1 28	0 78	0 28	0 45	(0 57,0.96),(0 90,0 70)
2	lower north west	upper south-west	negative	0 48	0 17	0 40	0 11	(0 90,0 70),(0 29,0 94)
3	right south	lower west	positive	1 17	0.35	0 77	0 36	(0 29,0.94),(0 53,0 01)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	upper-south west	lower south-east	negative	0 67	0.35	0 44	0 20	(0 78,0 87),(0 01,0 64)
2	upper east	left north	negative	0 72	0 28	0 52	0 21	(0 01,0 64),(0 99,0 84)
3	left south	lower north-east	negative	0 74	0 01	0 73	0 05	(0 99,0 84),(0 63,0 02)

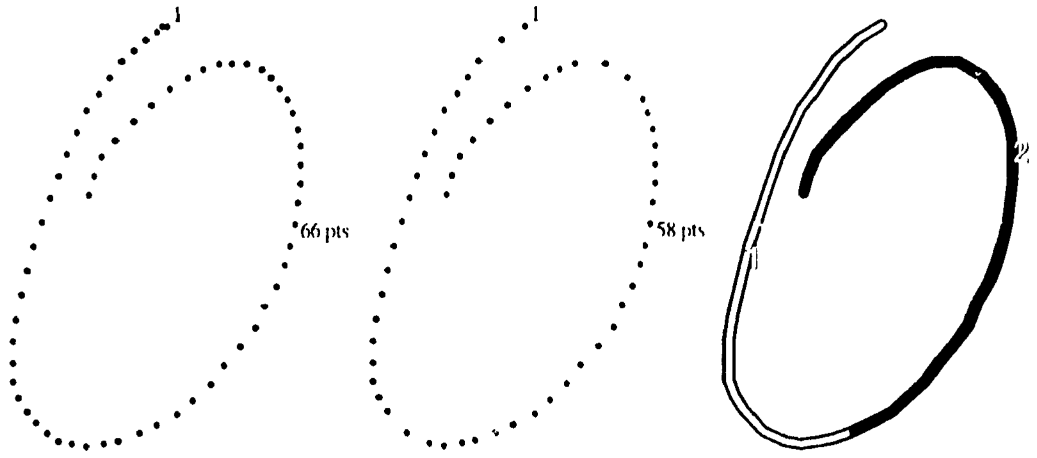


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	lower south-west	upper-south-east	negative	0 78	0 22	0 61	0 19	(0 84,0 96),(0 07,0 44)
2	lower north east	upper west	negative	0 72	0 11	0 64	0 12	(0 07,0 44),(0 99,0 90)
3	lower south west	upper north west	positive	1 07	0 15	0 91	0 26	(0 99,0 90),(0 08,0 00)

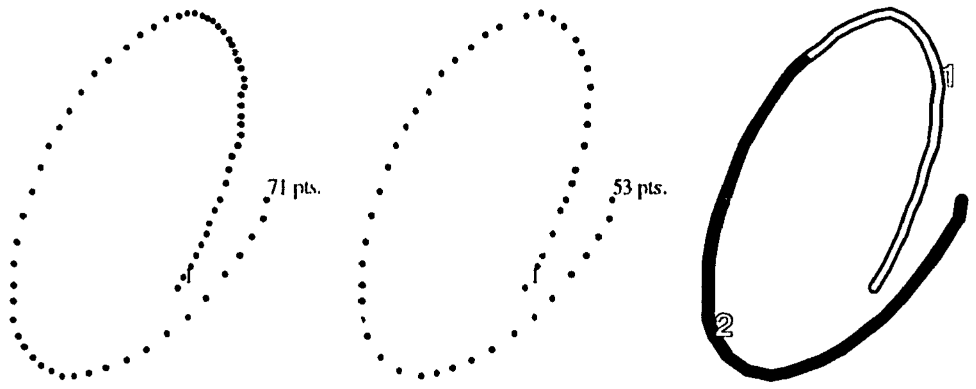


Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	right-south	upper-east	negative	1 07	0 90	0 11	0 43	(0 94,0 93),(0 98,0 81)
2	left-south	right-south	negative	0 75	0 00	0 74	0 02	(0 98,0 81),(0 54,0 02)

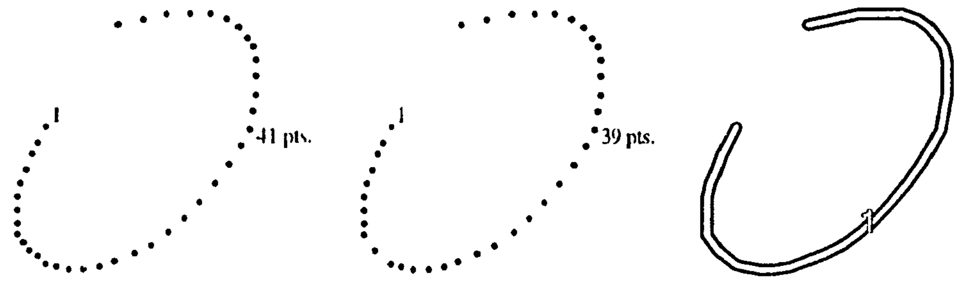
5.1.10 Samples of the digit '0'



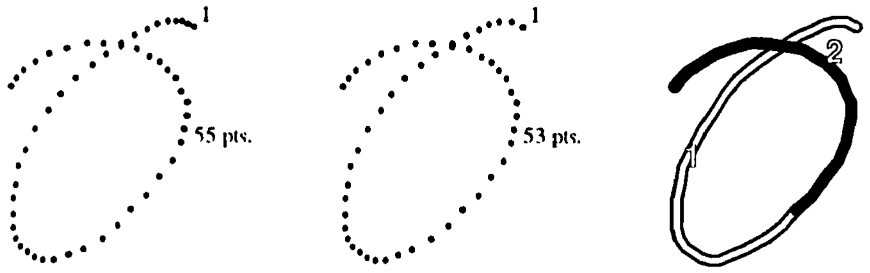
Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left south	lower north-east	negative	1 10	0 27	0 81	0 26	(0 53,1 00),(0 44,0 03)
2	left-north	lower-south-west	negative	1 32	0 63	0 48	0 42	(0 44,0 03),(0 27,0 60)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left north	lower-south west	negative	0 93	0 42	0 54	0 28	(0.66,0 24),(0.38,0 87)
2	upper south east	upper north east	negative	1 46	0 68	0 47	0 62	(0 38,0 87),(1 00,0 48)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	upper north-east	lower-north-west	negative	1 86	0 81	0 36	0 50	(0.11,0 55),(0 42,0 96)



Feature	Direction	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	left south	upper east	negative	1 51	0 57	0 64	0 40	(1.00,0 98),(0 66,0 21)
2	upper north west	upper south-west	negative	1 14	0 50	0 58	0 42	(0 66,0 21),(0 00,0 73)

5.1.11 Observations

As can be observed from these images, the feature extraction system performed quite well with respect to our goal of identifying major structural components. The filtering stage significantly reduced the effective number of points required to extract the features while maintaining a good structural representation of the pattern. Desirable straight and curved segments were identified with appropriate boundaries even though the data were not smoothed. The feature vector elements, namely *trajectory, concavity, rotation, length, curvature, spread, depth, start point* and *end point*, were found to be sufficiently expressive for on-line recognition of numerals. It is felt that they are sufficiently general to be applicable to the on-line recognition of other characters as well, such as the 26 letters of the alphabet.

5.2 Rule-Based Classification

	System Classification										
Identity	0	1	2	3	4	5	6	7	8	9	Reject
0	98	0	0	0	0	0	0	0	0	0	2
1	0	99	0	0	0	0	0	0	0	0	1
2	0	0	99	0	0	0	0	0	0	0	1
3	0	0	0	96	0	0	0	0	0	0	4
4	0	0	0	0	95	0	0	0	0	0	5
5	0	0	0	0	0	96	0	0	0	0	4
6	0	0	0	0	0	0	98	0	0	0	1
7	0	0	0	0	0	0	0	96	0	0	5
8	0	0	0	0	0	0	0	0	95	0	5
9	0	0	0	0	0	0	0	0	0	94	6

Table 3: Confusion table for the 1000-sample training set

Table 3 shows the system performance on the training set in the form of a *confusion table*. For each class of character i in the left-most column (labelled "Identity"), the confusion table lists the number of cases where character i is classified as character j, where possible values of j are listed across the top row. (labelled "System Classification"). The number of rejected samples for each character class is also shown in the right-most column. The training was successful as indicated by the high values along the diagonal of the table corresponding to correct classifications, and the zero entries elsewhere indicating that no incorrect classifications were made. A modest number of rejected cases for each class of character was experienced. Samples of the digit '9' had the highest number of rejections at six, while only one sample was rejected in each of the '1', '2', and '6' character classes. These rejected cases are those which were marked as *bad samples* during the training phase. Included in the bad sample category were several cases of ambiguous characters where it was decided to be cautious and allow the character to be rejected rather than training the system to conclude it's *intended* identity. Doing otherwise would likely increase the number of misclassifications made by the system. There were also some patterns which yielded unsatisfactory sets of features and thus were marked as bad samples. This was due to noisy or rough input data as a result of strokes drawn at high pen velocities.

Digit	Recognition (%)	Substitution (%)	Rejection (%)	Reliability (%)	Time (msec.)	Samples
0	98.00	0.00	2.00	100	49.60	100
1	99.00	0.00	1.00	100	59.70	100
2	99.00	0.00	1.00	100	68.60	100
3	96.00	0.00	4.00	100	62.50	100
4	95.00	0.00	5.00	100	184.40	100
5	96.00	0.00	4.00	100	166.10	100
6	98.99	0.00	1.01	100	48.18	99
7	95.05	0.00	4.95	100	114.75	101
8	95.00	0.00	5.00	100	129.30	100
9	94.00	0.00	6.00	100	88.60	100
Overall	96.60	0.00	3.40	100	97.24	1000

Table 4: Overall system performance on the training set

Table 4 presents some system performance measurements obtained for the training set. The metrics used are the percentage rates for recognition, substitution, rejection, and reliability, as well as processing speed. The recognition rate is the percentage of all cases which are correctly classified; substitution is the percentage of all cases which are incorrectly classified; rejection is the percentage of all cases which are rejected; reliability is the percentage of all classifications (i.e. excluding rejected cases) which are correct. Rows of the table list the calculated averages for each individual character class, in addition to the overall averages in the bottom row. On the training set, we observe an overall recognition rate of 96.6%, no substitutions, and a 3.4% rejection rate, yielding 100% reliability. Processing speed (Time) measures the total execution time for both the feature extraction and classification computations for a single sample. These values were obtained from the ITIMER_PROF[1] processes interval timer under SunOS 4.1.1 on a Sun SPARCstation 2 with 32 Mb RAM. The overall average processing time of 97 msec. per sample corresponds to a rate of approximately 10 characters per second, which easily accommodates normal human writing speeds.

The training set of the digit '6' contains only 99 samples due to a human error which occurred at the time of data acquisition. One of the writers drew a seven when they were actually prompted for a six.

	System Classification										
Identity	0	1	2	3	4	5	6	7	8	9	Reject
0	198	0	0	0	0	0	1	0	0	1	30
1	0	215	0	0	0	0	0	1	0	0	14
2	1	0	186	0	0	0	0	1	0	0	42
3	0	0	0	186	0	0	0	0	1	0	43
4	0	0	0	0	194	0	0	0	0	1	36
5	0	0	0	0	0	190	0	0	0	0	40
6	0	0	0	0	0	1	186	0	6	0	37
7	0	2	0	0	0	0	0	200	0	0	27
8	1	0	0	0	0	0	0	0	145	0	84
9	0	0	0	0	2	6	0	0	2	170	50

Table 5: Confusion table for 2300-sample testing set

Tables 5 and 6 show the corresponding confusion and performance tables obtained for the 2300 member test set. In the confusion table, we observe that the most significant error is that of misclassifying '9's as '5's. Six of the 230 samples of the

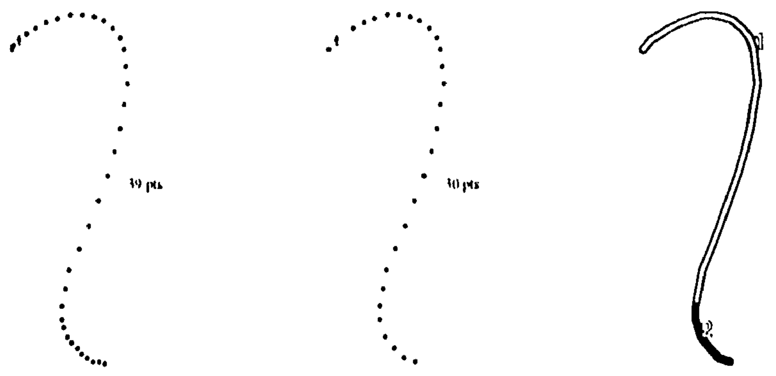
[1]This timer measures the elapsed process virtual time plus the time the system spends running on behalf of the process.

Digit	Recognition (%)	Substitution (%)	Rejection (%)	Reliability (%)	Time (msec.)	Samples
0	86.09	0.87	13.04	99.00	63.39	230
1	93.48	0.43	6.09	99.54	33.78	230
2	80.87	0.87	18.26	98.94	56.22	230
3	80.87	0.43	18.70	99.47	56.83	230
4	83.98	0.43	15.58	99.49	172.34	231
5	82.61	0.00	17.39	100.00	171.26	230
6	80.87	3.04	16.09	96.37	44.52	230
7	87.37	0.87	11.79	99.01	74.89	229
8	63.04	0.43	36.52	99.32	110.30	230
9	73.91	4.35	21.74	94.44	73.43	230
Overall	81.30	1.17	17.52	98.58	85.74	2300

Table 6: Overall system performance on the testing set

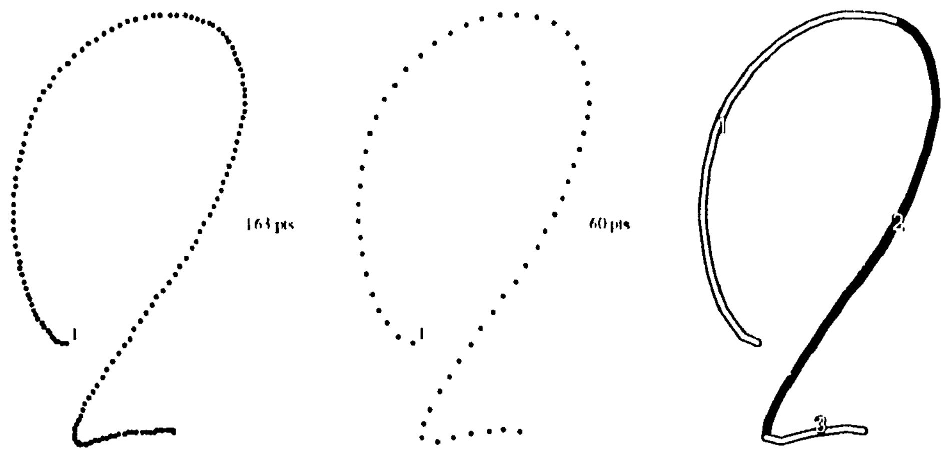
character '9' were confused in this way. Observation of these samples shows that they are border line cases which *could* be interpreted as sloppy five's.

On the test set, the system achieves an overall average recognition rate of 81.3%, substitution rate of 1.17%, and rejection rate of 17.52%. The drop in recognition rate, and its corresponding increased rejection rate compared to the training set suggests that the training set is not representative of the writing styles experienced in the test data. Upon comparison of samples from the training set with samples from the test set which were rejected, we can conclude that this is in fact the case in many instances. For example, the test set contained numerous samples of the character '0' written with a slanted straight line stroke crossing the circle, whereas these were absent in the training data. This is not surprising since only ten writers were used to formulate the training set [42]. Moreover, we can conclude that the training set should be larger. Another observation is that many of the testing samples which the system rejected, barely missed being classified correctly due to a feature vector value which was just beyond a threshold set in one of the labeling rules. Also, there are a fair number of very confusing samples in the test set. This is of course due to certain writers being asked to input difficult and confusing patterns when the data were acquired.



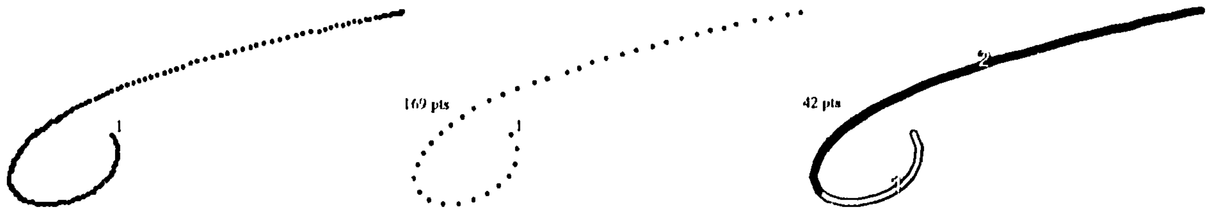
Feature	Trajectory	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	right-south	lower-south-west	positive	1.11	0.36	0.71	0.31	(0.00,0.90),(0.45,0.16)
2	lower-south-east	upper-east	negative	0.20	0.08	0.18	0.03	(0.45,0.16),(0.75,0.00)

(a)



Feature	Trajectory	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	upper-north-east	upper-south east	positive	0.96	0.25	0.72	0.25	(0.23,0.23),(0.84,0.98)
2	left-south	upper-south-west	positive	0.97	0.08	0.90	0.13	(0.84,0.98),(0.26,0.01)
3	upper-east	upper-east	undefined	0.21	0.01	0.20	0.01	(0.26,0.01),(0.69,0.03)

(b)



Feature	Trajectory	Concavity	Rotation	Length	Curvature	Spread	Depth	End Points
1	upper-south-west	upper-north west	positive	0.34	0.25	0.25	0.09	(0.26,0.36),(0.01,0.06)
2	lower-north-east	lower-east	positive	1.04	0.06	0.97	0.12	(0.01,0.08),(0.99,0.99)

(c)

Figure 22: Three misclassified samples.

Figure 22 shows three typical misrecognized samples. Figure 22 (a) is an example of an ambiguous, or confusing pattern. The intended identity of this sample is '2', but the system concluded '7' with a confidence level of 84%. The length of feature 1 is 84% of the total length of the stroke, and feature 1 was the only feature used in this particular classification, hence the confidence value. The actual identity of the sample shown in Figure 22 (b) is also '2', but the system classified it as '0' with a 90% confidence level. Here features 1 and 2 form a large dominant loop, comprising 90% of the total length of the stroke. Feature 3 makes up the other 10% of the total stroke length but did not contribute toward this classification. This sample is definitely not a typical well-formed '2'. The sample in Figure 22 (c) is a very skewed '6', however it was classified as an '8' with 100% confidence since all (both) features were used in the classification. In this case, some rather lenient feature end-point threshold values enabled feature 1 to be labeled as "S-curve-of-8". In some instances it is desirable for a left-facing concavity to be labeled as "S-curve-of-8". However, this is clearly an undesirable label when considered in conjunction with feature two, which merits the label "top-left-curve-of-8", and thus results in the false positive conclusion.

The overall substitution rate of 1.17% is quite low, which translates directly into the high reliability obtained: 98.58%. This is a benefit of the cautious approach to training which was adopted. As is evident from table 6, the average execution times vary substantially among the character classes. This is because certain digits are structurally more complex than others, and therefore are in general composed of more points and yield more features. However, the average execution times for the training and the testing data sets are comparable, as expected. The 84 rejected cases of the test set for the numeral '8' present the highest rejection class. Examination of these cases revealed that this character class demonstrated the highest structural variability as can be seen in section 5.1.8. This is due to the lack of vertices and because of the complexity of strokes for the numeral class '8'. The character class '1' has the fewest rejected cases at only 14 out of 230, or 6.09%. As happened during data collection for the training set, a human error occurred where one of the subjects drew a '4' when they were actually prompted to draw a '7'.

The system performance is good overall, and is quite robust with respect to not misclassifying digits of styles which differ from those which it was trained on.

5.3 Future Work

The experimental results obtained suggest the need to expand the training set to improve the recognition rate with a corresponding lowering of the rejection rate. This expansion of training will necessarily involve further refinements to the knowledge base, specifically with the addition and modification of primitive labelling rules (Sec. 4.4.2. Alternatively, a rule-induction approach is suggested to automate the training of the classifier [36]. Other classification strategies such as neural networks could be used to process the structural features obtained. A very useful extension would be to handle both upper and lower cases of the 26 character alphabet, a formidable undertaking.

Chapter 6

Conclusion

To sighted humans, the process of character recognition, and pattern recognition in general, is an automatic reflex not generally considered to be a skill. However, it remains a complex and challenging problem for computers. In an effort to develop new methods to approach this problem, graphical user interface and rule-based expert systems technology can be applied.

This thesis has described the design and implementation of OLDRES, a graphical system for recognizing isolated samples of on-line digits. An overview of the basic principles and relevant literature on character recognition, expert systems, and man-machine communication was presented. The system automatically segments the stroke data from which geometric feature vectors are computed. A hand-crafted rule base classifies the patterns based on the feature vectors to recognize the digits. The implementation offers a graphical development environment for further research in on-line character recognition. The system was developed on the Sun SPARCstation platform running the SunOS 4.1.1 (UNIX) operating system and the X Window System, Version 11, Release 4. A training data set of 1000 digit samples, and a testing data set of 2300 digit samples obtained from a digitizer tablet were used. Some experimental results were presented and discussed. Suggestion for further improvements were given.

References

[1] S. Sheppard A. Harbert, W. Lively. A graphical specification system for user interface design. *IEEE Software*, 7(4):12–20, July 1990.

[2] Adobe Systems Incorporated. *PostScript Language Reference Manual*, November 1988. Addison-Wesley Publishing Company, Inc.

[3] Adobe Systems Incorporated. *PostScript Language Tutorial and Cookbook*, May 1989. Addison-Wesley Publishing Company, Inc.

[4] Artificial Intelligence Section, NASA/JSC, COSMIC, University of Georgia, 382 East Broad Street, Athens, GA 30602. *CLIPS Reference Manual*, May 1989. Version 4.3 of CLIPS.

[5] V. Barker and D. O'Connor. Expert systems for configuration at digital: Xcon and beyond. *Communications of the ACM*, 32(4):298–318, March 1989.

[6] A. Batarekh, A.D. Preece, A. Bennett, and P. Grogono. Specification of expert systems. In A. Dollas, W.T. Tsai, and N.G. Bourbakis, editors, *Proc. 2nd International Conference on Tools for Artificial Intelligence (TAI-90), Herndon, VA.*, pages 103–109, Herndon, VA., November 6–9, 1990.

[7] M.K. Brown and S. Ganapathy. Preprocessing techniques for cursive script word recognition. *Pattern Recognition*, 16:447–458, 1983.

[8] Virtual reality. *Business Week*, (3286):96–105, October 5 1992.

[9] Digital multimedia systems. *Communications of the ACM*, 34(4), April 1991.

[10] C. Cruz-Neira, D.J. Sandin, T.A. DeFanti, R.B. Kenyon, and J.C. Hart. The cave, audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):64–72, June 1992.

[11] C.Y.Suen, M. Berthod, and S. Mori. Automatic recognition of handprinted characters - the state of the art. *Proceedings of the IEEE*, 68:469 483, April 1980.

[12] C.Y.Suen, C. Nadal, R. Legault, T.A. Mai, and L. Lam. Computer recognition of unconstrained handwritten numerals. *Proceedings of the IEEE*, 80(7):1162 1180, July 1992.

[13] R.O. Duda, J. Gaschnig, and P. Hart. Model design in the prospector consultant system for mineral exploration. In D. Michie, editor, *Expert Systems in the Microelectronic Age*, pages 153-167. Edinburgh University Press, Edinburgh, 1979.

[14] C. Forgy. Rete: A fast algorithm for the many-pattern/many-object pattern match problem. *Artificial Intelligence*, 19(1):17–37, September 1982.

[15] J. Gettys, R.W. Scheifler, and R. Newman. *Xlib - C Language X Interface*. MIT X Consortium Standard, May 1989. X Version 11, Release 4.

[16] J.C. Giarratano and G. Riley. *Expert Systems: Principles and Programming*. PWS-KENT, 20 Park Plaza, Boston, Massachusettes 02116, 1989.

[17] V. Gurbaxani and S. Whang. The impact of information systems on organizations and markets. *Communications of the ACM*, 34(1):59-73, January 1991.

[18] D. Hearn and M.P. Baker. *Computer Graphics*. Prentice-Hall, 1986.

[19] D. Heller. *XView Programming Manual*, volume 7 of *The X Window System*. O'Reilly & Associates, Inc., 632 Petaluma Ave., Sebastopol, CA 95472, October 1990. Fourth Printing.

[20] Visualization in computing. *IEEE Computer*, 22(10), October 1989.

[21] Computer generated music. *IEEE Computer*, 24(7), July 1991.

[22] Interactive multimedia getting the whole picture. *IEEE Spectrum*, 30(3):22–39, March 1993.

[23] S. Impedovo. Plane curve classification through fourier descriptors: 'an application to arabic hand-written numeral recognition'. In *Proc. 7th Int. Conf. Pattern Recognition*, pages 1069–1072, 1984.

[24] A. Irgon, J. Zolnowski, K.J. Murray, and M. Gersho. Expert system development: A retrospective view of five systems. *IEEE Expert*, 5(3):25–40, June 1990.

[25] K. Ishigaki and T. Morishita. A top-down online handwritten character recognition method via the denotation of variation. In *Proc. 1988 Int. Conf. Comput. Processing of Chinese and Oriental Languages*, pages 141–145, Aug-Sept 1988.

[26] Y. Ishii. Stroke order free online handwritten kanji character recognition method by means of stroke representation points. *Trans. Inst. Electron. Commun. Eng. Japan*, pages 1069-1072, 1986.

[27] P. Jackson. *Introduction to Expert Systems.* Addison Wesley Publishing Company, second edition, 1990.

[28] J. Johnson, T.L.Roberts, W. Verplank, D.C. Smith, C.H. Irby, M. Beard, and K. Mackey. The xerox star: A retrospective. *IEEE Computer*, 22(9):11–26, September 1989.

[29] L. Lam and C.Y. Suen. Structural classification and relaxation matching of totally unconstrained handwritten zip-code numbers. *Pattern Recognition*, 21(1):19–31, 1988.

[30] L. Lamport. *LaTeX: A Document Preparation System.* Addison-Wesley Publishing Company, Inc., 1986.

[31] R. Legault, C.Y. Suen, and C. Nadal. Classification of confusing handwritten numerals by human subjects. In *Proc. Int. Workshop on Frontiers in Handwriting*

Recognition, pages 181-193. CENPARMI, Concordia University, Montreal, April 1990.

[32] R. Lindsay, B.G. Buchanan, E.A. Fiegenbaum, and R. Lederberg. *DENDRAL*. McGraw-Hill, New York, 1980.

[33] P.J. Mercurio, T.T. Elvins, S.J. Young, P.S. Cohen, K.R. Fall, and M.H. Ellisman. The distributed laboratory. *Communications of the ACM*, 35(6):54-63, June 1992.

[34] W. Mettrey. A comparative evaluation of expert system tools. *IEEE Computer*, 24(2):19–31, February 1991.

[35] W. Mettrey. Expert systems and tools: Myths and realities. *IEEE Expert*, 7(1):4–12, February 1992.

[36] R.S. Michalski. A theory and methodology of inductive learning. *Artificial Intelligence*, 20:111-161, 1983.

[37] D. Miranker. Treat: A better match algorithm for ai production systems. In *Proc. Sixth Nat'l Conf. Artificial Intelligence*, pages 42-47. Cambridge, Mass., MIT Press, 1987.

[38] J.F. Muratore, T.A. Heindel, T.B. Murphy, A.N. Rasmussen, and R.Z. McFarland. Real-time data acquisition at mission control. *Communications of the ACM*, 33(12):18–31, December 1990.

[39] B.A. Myers and M.B. Rosson. Survey on user interface programming. In *CHI '92 Conference Proceedings*, pages 195-202. ACM Conference on Human Factors in Computing Systems, Monterey, CA., May 3-7, 1992.

[40] M. Parizeau and R. Plamondon. A comparative analysis of regional correlation, dynamic timewarping, and skeletal tree matching for signature verification. *IEEE Trans. Patt. Anal. Machine Intell.*, 12, 1990.

[41] R. Pausch, G.G. Robertson, S.K. Card, J.D. Mackinlay, and M. Moshell. Three views of virtual reality. *IEEE Computer*, 26(2):79–83, February 1993.

[42] S.J. Raudys and A.K. Jain. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(3):252–264, March 1991.

[43] A. Reinhardt. Momenta points to the future. *Byte*, pages 48–49, November 1991.

[44] M. Usher S. Al-Emami. On-line recognition of handwritten arabic characters. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(7):704–710, July 1990.

[45] S.L. Shiau, J.W. Chen, A.J. Hsieh, and S.J. Kung. On-line handwritten chinese character recognition by string matching. In *Proc. 1988 Int. Conf. Comput. Processing of Chinese and Oriental Languages*, pages 76-80, Aug-Sept 1988.

[46] E.H. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* Elsevier, New York, 1976.

[47] Sun Microsystems, Inc. *OPEN LOOK Graphical User Interface Application Style Guidelines*, June 1990. Addison-Wesley Publishing Company, Inc.

[48] Sun Microsystems, Inc. *OPEN LOOK Graphical User Interface Functional Specification*, July 1990. Addison-Wesley Publishing Company, Inc.

[49] I.E. Sutherland. Sketchpad, a man-machine graphical communication system. In *Proc. Spring Joint Computer Conference*, pages 329-345, 1963.

[50] W. Swartout, C. Paris, and J. Moore. Design for explainable expert systems. *IEEE Expert*, 6(3):58-64, June 1991.

[51] C.C. Tappert. Speed, accuracy, flexibility trade-offs in on-line character recognition. Research Report RC13228, IBM, October 1987.

[52] C.C. Tappert, C.Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Trans. Pattern Anal. Machine Intell.*, 12(8):787 808, August 1990.

[53] R.J. Vetter and D.H.C. Du. Distributed computing with high-speed optical networks. *IEEE Computer*, 26(2):8 18, February 1993.

[54] Visual programming. *IEEE Computer*, 18(8):6 94, August 1985.

[55] T. Wakahara. On-line cursive script recognition using local affine transformation. In *Proc. 9th Int. Conf. Pattern Recognition*, pages 1133 1137, November 1988.

[56] J.R. Ward and M.J. Phillips. Digitizer technology: Performance characteristics and the effects on the user interface. *IEEE Comput. Graphics Appl.*, pages 31 44, April 1987.

[57] Y. Watanabe, J. Gyoba, T. Hirata, and K. Maruyama. A psychological approach to the human recognition of ambiguous characters. *J. Inst. TV Eng. Japan*, 39:509–515, 1985.

[58] Donald A. Waterman. *A Guide to Expert Systems*. The Teknowledge Series in Knowledge Engineering. Addison-Wesley Publishing Company, Inc., Don Mills, Ontario, 1986.

[59] Patrick Henry Winston. *Artificial Intelligence*. Addison Wesley Publishing Company Inc., second edition, July 1984.

[60] P.H. Winston and B.K.P. Horn. *LISP*. Addison-Wesley, second edition, 1984.

[61] D.A. Young. *X Window Systems: Programming and Applications with Xt*. Prentice-Hall, 1989.