Report Number: WUCS-93-28

1993-01-01

Segmentation/Recognition of Hand-Written Numeral Characters

Khalid Sherdil

This thesis describes a number of techniques for segmenting non-cursive handwritten digits into individual characters. It strongly emphasizes on a recognition-segmentation algorithm, which uses the linear regression method to recognize those strokes which consist of one or more straight-lined parts. A new method of sampling the pen data according to the pen speed, hence giving a more uniform points concentratino distribution, is also introduced. It is shown how several of our segmenting techniques, such as relative stroke lengths, relative stroke positions, order of stroke entry, stroke direction, stroke intersection, etc. can be combined to yield success results of about... **Read complete abstract on page 2.**

Segmentation/Recognition of Hand-Written Numeral Characters

Khalid Sherdil

Complete Abstract:

This thesis describes a number of techniques for segmenting non-cursive handwritten digits into individual characters. It strongly emphasizes on a recognition-segmentation algorithm, which uses the linear regression method to recognize those strokes which consist of one or more straight-lined parts. A new method of sampling the pen data according to the pen speed, hence giving a more uniform points concentratino distribution, is also introduced. It is shown how several of our segmenting techniques, such as relative stroke lengths, relative stroke positions, order of stroke entry, stroke direction, stroke intersection, etc. can be combined to yield success results of about 95%.

Segmentation/Recognition of Hand-Written Numeral Characters

Khalid Sherdil

WUCS-93-28

May 1993

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

An Honors thesis presented to the Sever Institute of Washington University in partial fulfillment of the requirements for the degree Bachelor of Science.

WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY

SEGEMENTATION / RECOGNITION OF HAND-WRITTEN NUMERAL CHARACTERS

by

Khalid Sherdil

Prepared under the direction of Professor T. D. Kimura

An Honors thesis (CS 499/494) presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

BACHELOR OF SCIENCE

May 1993
Saint Louis, Missouri

Abstract

This thesis describes a number of techniques for segmenting non-cursive handwritten digits into individual characters. It strongly emphasizes on a recognition-segmentation algorithm, which uses the linear regression method to recognize those strokes which consist of one or more straight-lined parts. A new method of sampling the pen data according to the pen speed, hence giving a more uniform points concentration distribution, is also introduced. It is shown how several of our segmenting techniques, such as relative stroke lengths, relative stroke positions, order of stroke entry, stroke direction, stroke intersection, etc. can be combined to yield success results of about 95%.

Table of Contents

List of Illustrations

List of Tables

1. Introduction

It is believed that within a next few years, 'Pen' will be substituting 'Mouse' extensively. The research on pen based computing can be originated back to mid 60's [1] with the development of Rand and Sylvania tablets[2] [3]. There have been considerable developments in the late 80's and into the 90's. Referred to as Silicon Paper[4], this type of computing has been studied for quite some time. This has led to the development of a new way of interacting with the computer, namely handwriting and gestures. Using pen as a convenient form of user interface, the user would be able to input commands or data via Pen in the most simple and personalized way, that is, by hand-writing them on the screen. The computer would then decipher that writing and implement the instructions received. Hence hand-writing recognition is going to be an important factor in computer industry in the near future and considerable work on this field is already under progress.

A first step towards hand-writing recognition is the recognition of the Digits. Already a lot of applications are developed which employ numerical character recognition. Examples include, mathematics tutorials for elementary school students who enter solutions to simple math quizzes on digitizing pen templates. Several public service departments have begun using pen templates for collecting Social Security numbers, phone numbers, or signatures. Hospitals and Parcel Delivery services use mobile templates as a quick and convenient form of storing and updating data.

The first phase in recognizing any number is to segmentize it into individual digits. This thesis is a preliminary study of various techniques which can be used for aiding in the segmentation of character digits. It is believed that any advanced method of segmentation would employ a combination of at least some of the methods described below.

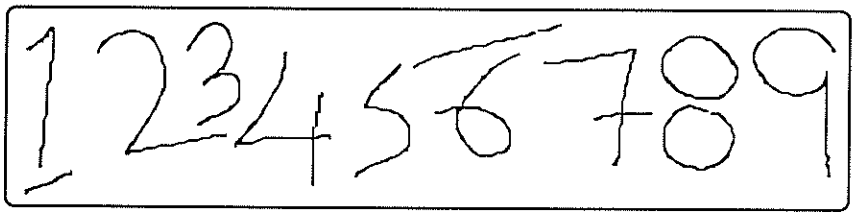
Section 2 describes the Problem Definition and states the objective. Section 3 provides a background and highlights some of the research done in this field. Section 4 gives a detailed study of the methods and techniques used in implementing a solution to the problem. Section 5 describes the testing procedure while Section 6 gives the results. These results are analyzed and evaluated in Section 7. Finally Section 8 lists some options for future work which may be carried out in this field, and Section 9 gives a conclusion to the project.

2. Problem Definition & Objective

Using pen as user-interface, users input data directly on the screen of a pen-computer or a pen-template. In our case, this data consists of numerals. The user can enter any number, such as a phone number or a social security number, which may consist of several digits. As shown in figure 1, entry of one such set of data is called a 'Scribble.' The scribble consists of various strokes. A stroke is defined as the pattern sketched between one instance of user's putting the pen on the screen and removing it. Figure 2 shows how one scribble can be divided into various strokes.

Our problem is to segmentize the data so that these strokes are grouped together as digits. For example, in the figure below, the first two strokes are grouped together as digit 5. We are presuming here that only digits 1, 4, 5, 7 and 8 can have two strokes. Rest of the digits can have only one stroke. Furthermore, our segmenter is designed only for non-cursive handwriting. This means that the entered scribble should have digits which do not intersect or overlap. Examples of non-cursive/non-overlapping strokes and cursive/overlapping strokes are given in figure 1.

ALLOWED: Non-cursive, Non-Overlapping



NOT ALLOWED: Cursive, Overlapping Strokes

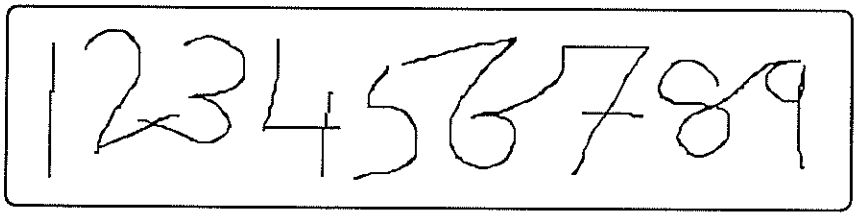


Figure 1

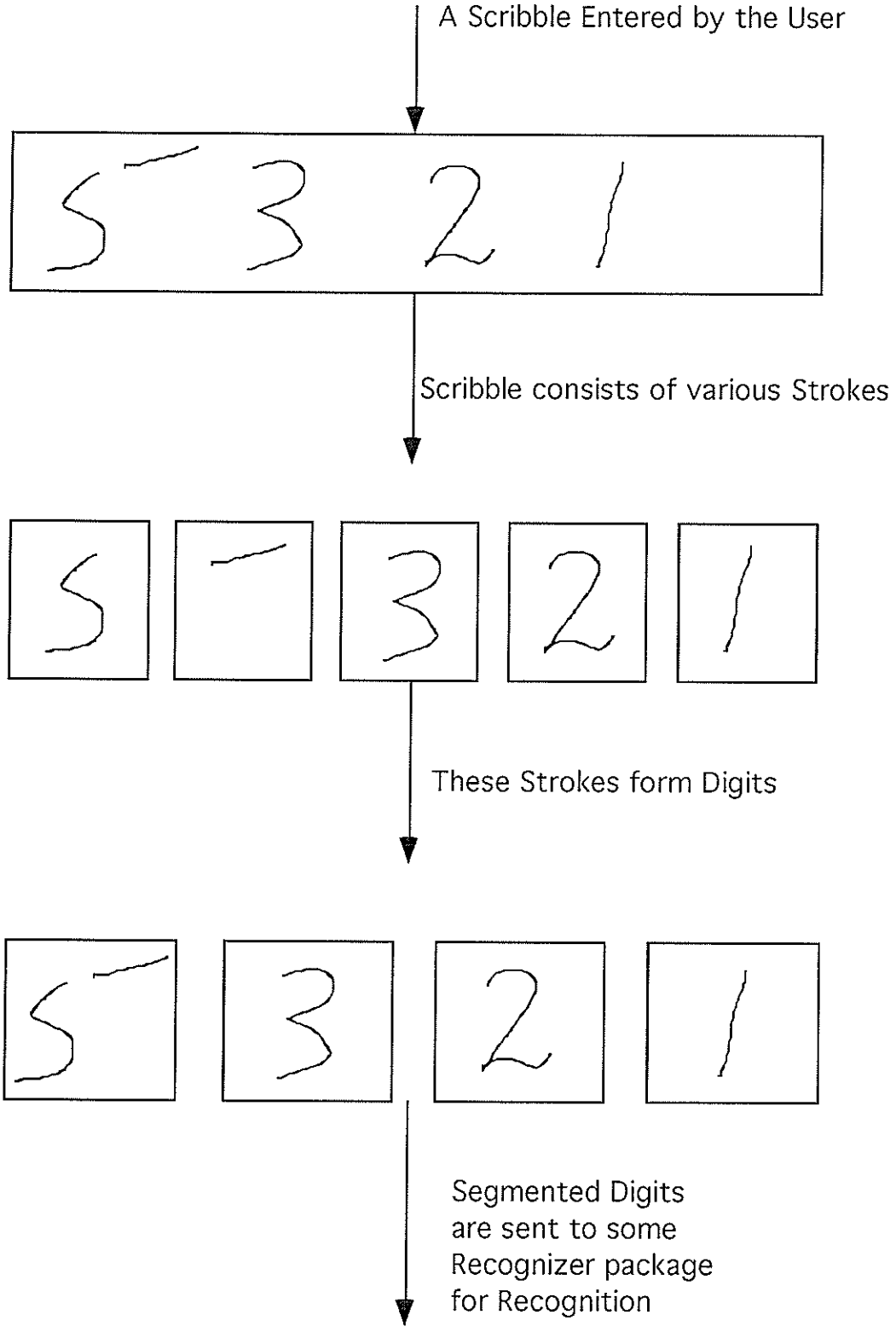

A Scribble Entered by the User
Scribble consists of various Strokes
These Strokes form Digits
Segmented Digits
are sent to some
Recognizer package
for Recognition

Figure 2

This segmented data should be of such a format that it could be sent as input to some recognizer, which would then recognize the digits (see figure 2). In our case, the recognizer is a neural net-based handwriting recognition package called Supervised Competitive Learning, SCL [5] [6]. This package has been designed by Tom H. Fuller and T, D. Kimura of Kumon Machine group.

We would be using a WACOM digitizing pen template for data collection. The Segmenter/Recognizer package is implemented on NeXT machines using C programming language.

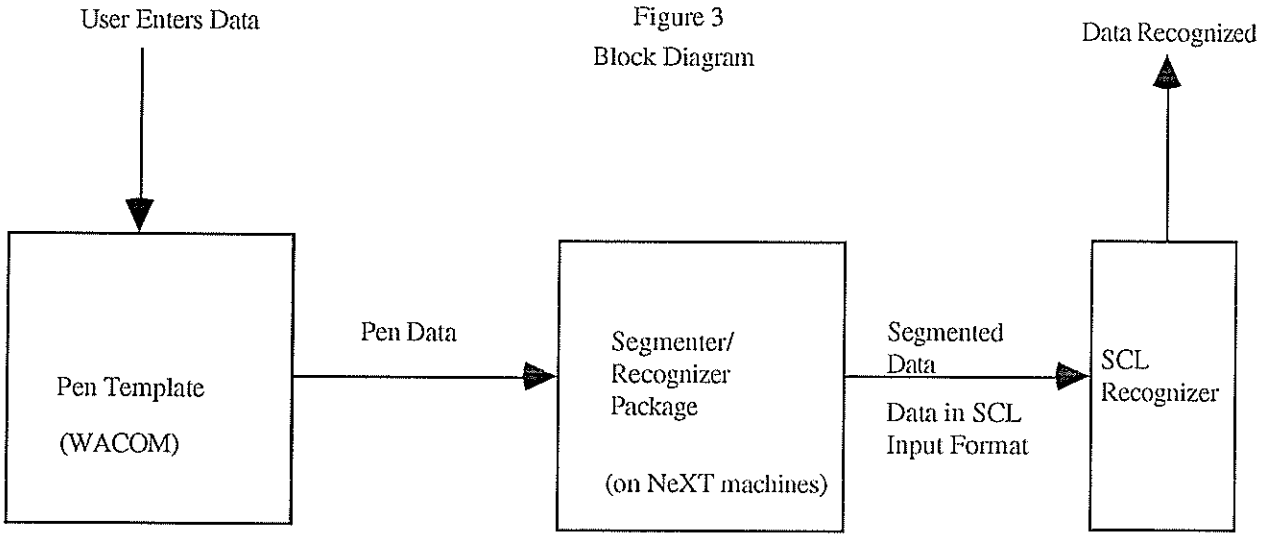
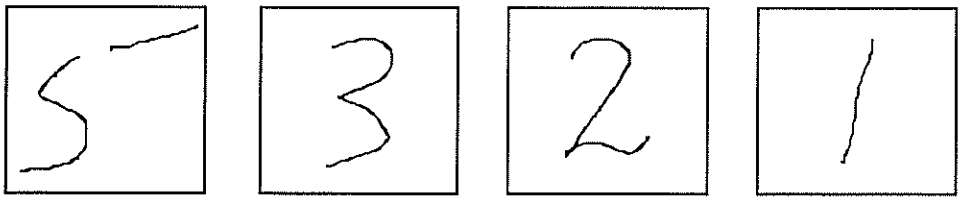


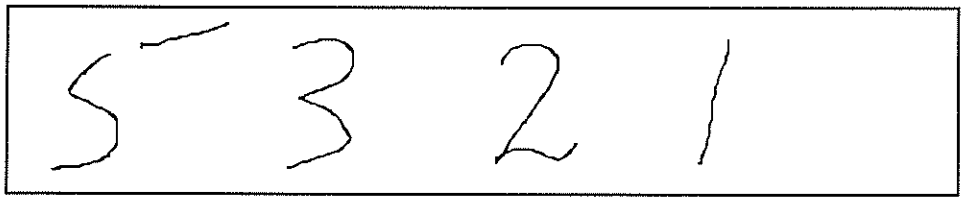
Figure 3
Block Diagram

Various algorithms have been developed to improve the efficiency of recognizing handwriting. In most cases, the user is presented with a grid containing square blocks, as shown in figure 4. Each block takes in one character, which is then recognized. At present, such a procedure can be carried out quite successfully, with success results of above 95% [6] [7]. However, the restrain of entering only one character at a time in different blocks, not only decreases the speed of handwriting, but also affects the flow of thinking, hence causing a lot of frustration for the user. Presently some work has already been done to eradicate segmentation. Further work is also being done to interpret continuous joined cursive handwriting. Taking a step forward towards continuous writing recognition, our algorithm would attempt to decipher long numbers with several digits on a grid-free area. Hence the user would no longer need a grid consisting of blocks or segments, in which each digit has to be entered separately.

Formerly Users had to enter data in separate blocks



This limitation is now removed, and users are free to enter numbers in a grid-free area



However, this number needs to be segmentized into its digits

Figure 4

3. Background

As described in the Introduction section, need for handwriting recognition is growing. Hospitals and Parcel Delivery services are already using mobile templates as a quick and convenient form of storing and updating data. The main motivation behind our designing a segmenter/recognizer for handwritten numeral characters is the Kumon Machine Project [34]. This project includes work on various applications which require pen-computing. One main purpose of this project is to design Mathematics Exercise sheets for school children. The students would be asked to solve various math problems and enter answers on a Pen Template. These answers then need to be recognized for grading. The Supervised Competitive Learning package, SCL [5] [6], was also designed for this project, and this is why we use it as the recognizer where our segmented data should be fed.

Segmenting and recognizing handwriting of junior and elementary school children is not a trivial task, even though it may be non-cursive writing. In deciding how to segment, it is often quite difficult to decide which strokes need to be grouped together to form a separate digit. Often a stroke of one digit is written very close to some stroke of a second digit. Also most of the strokes usually have no fixed shape or size. In nearly all the cases, the pen data is accompanied by a lot of noise. This noise occurs due to vibrations in the pen, so that a straight line is usually never perfectly straight. Moreover, different users have different pen velocities and enter digits with different pen directions (e.g, digit '1' can be written from bottom to top

rather than top to bottom). All these inconsistencies increase the level of difficulty of our problem. This is why as of now, there is no handwriting recognizer which can give perfect results for handwriting recognition from any user.

Research has been going on in the field of hand-writing recognition since the mid 60's. A Dialog search [8] (which accesses a database of all the papers published in almost all the registered periodicals) of the last 10 years was carried out which listed virtually all the methods employed for recognition. Some of the work involves recognition of scripts from cursive languages, such as Arabic (Goraine & Usher) [9] in which segmentation plays an important role. Others involve recognition of scripts which are highly segmented, such as Chinese or Japanese characters (Ling Chen) [10]. Most of the work is on English script recognition, though not all of it is on on-line data entry through pen. Considerable research is under progress on optical character recognition, which can scan either printed data (Amano & Yamashita) [11] or hand-written data (Cesar & Shinghal)[12] [13], such as in Zip Codes, and then segmentize it.

Since segmentation is the first phase in most of the recognition techniques, different researchers have come up with different ideas to tackle this problem. One idea, as presented by Fujisaki and Chefalas, is to recognize the digits first and then segmentize them [14]. In such a case, segmentation can be done quite easily once the tedious and difficult process of recognition is completed. A similar idea, as implemented by Balestri and Masera, is to evaluate the goodness of the entire set of segmentation points of a word, instead of separately considering the segmentation of a single character of the word [15]. Another approach, as used by Shridhar and

Badrelelin, is to use a Knowledge based tree which tests various hypotheses ranging from the case where the numerals are completely isolated to that where the numerals may be connected or overlapping [16]. Or one can also use a method in which various strokes have been stored previously in a vocabulary directory, and are compared with the data to come up with the best match (Tampi and Chetlur)[17].

In our paper we have used the concept of recognition using best curve fit. The dialog search of last 10 years in the field of 'Segmentation' showed that no author has mentioned Linear Regression in his or her abstract. However, there is a good possibility that some of them might have used this concept at some lower level of their broader area of research. Dunn and Wang have summarized some such segmentation techniques in a survey report [33]. Some similar work has been done in this field in Chinese strokes segmentation by. C. W. Liao. In his work, Liao used a Bernstein-Bezier curve method to fit the data with a maximum circle technique, and hence obtain the radius of curvature [18]. Inflection points with very small radius of curvature are found. All possible pairs of stroke segments are considered and finally the curve fitting is used to decide which pair belongs to the same stroke.

Although this paper is not directly on Sampling, we have introduced an original way on how to sample data points based on pen velocity and points concentration. We did not come across any past work which uses this method of sampling.

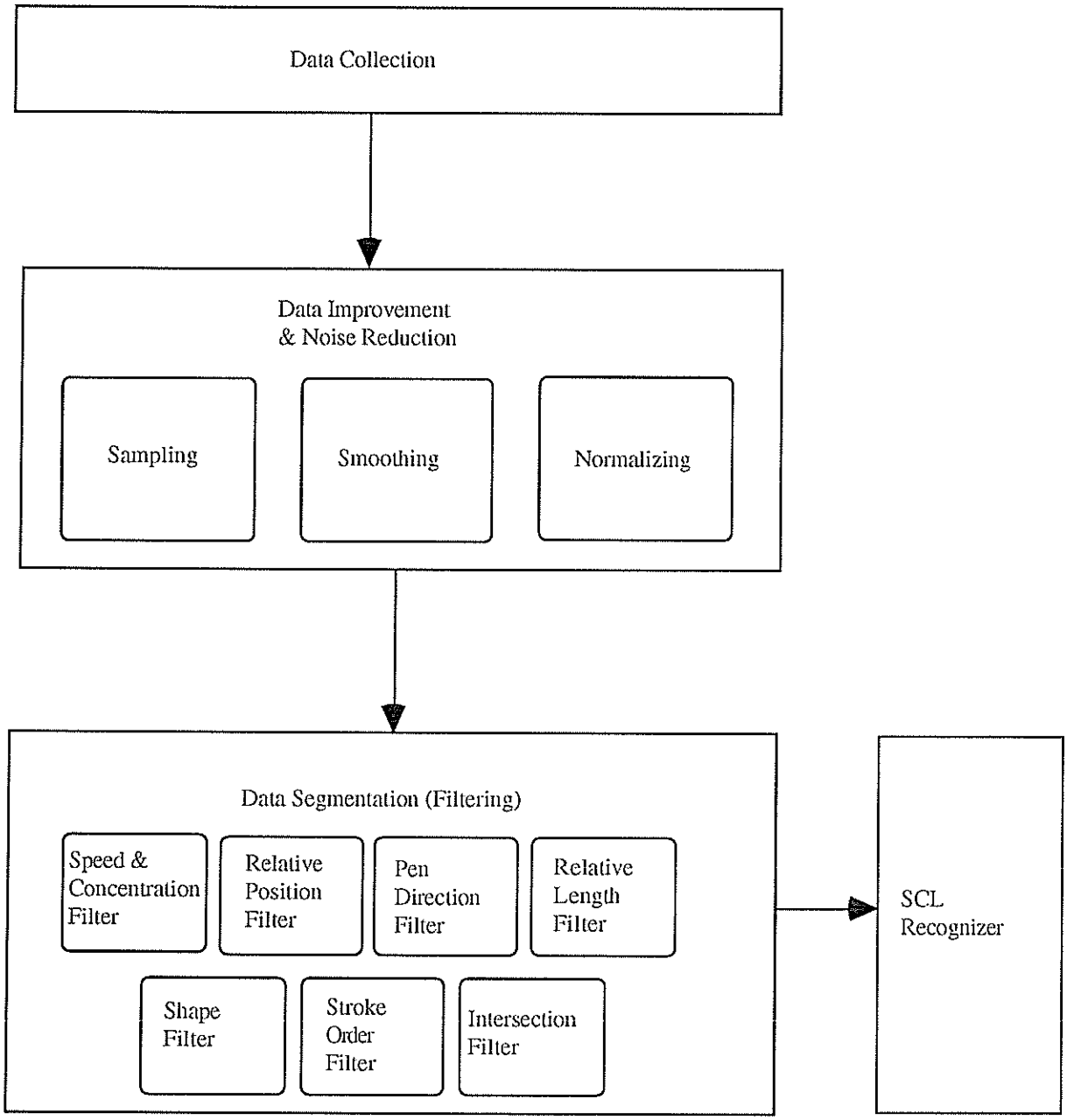
4. Design Method

The method used to implement the problem of segmenting/recognizing the handwritten data can be summarized as follows (see figure 5):

1. Collect the Data (about 10 strokes per file) on Kumon machine using Pen interface.
2. Improve the Data by reducing noise.
3. Segment the data by using any relavant combination of the following seven filtering techniques. a) Strokes Intersection
   b) Stroke Shapes
   c) Relative Stroke Lengths
   d) Relative Stroke Positions
   e) Order of Stroke Writing
   f) Pen Speed and Points Concentration
   g) Pen Direction
4. Feed the segmented data into Supervised Competitive Learning, SCL, package for complete recognition.

The most important filtering concept of all the above ones is the 'stroke shapes.' This thesis extensively uses the statistical theory of linear regression to recognize the shapes of those digits which contain straight-components. Once recognized, these strokes can then be segmented easily.

Figure 5


Data Collection
Data Improvement
& Noise Reduction
Sampling
Smoothing
Normalizing
Data Segmentation (Filtering)
Speed &
Concentration
Filter
Relative
Position
Filter
Pen
Direction
Filter
Relative
Length
Filter
Shape
Filter
Stroke
Order
Filter
Intersection
Filter
SCL
Recognizer

This section describes in detail the above procedure used in implementing the solution to our problem. Section 4.1 lists the hardware and software resources needed for the implementation. The first step in tackling this problem is to collect data from user's pen and store it in some standard form, which can be transported from one machine to another. Section 4.2 describes how the data is collected and Section 4.3 describes how it is represented. The collected data usually contains significant noise. Before the data can be utilized for problem solution, it needs to be modified or improved, so that the noise is reduced. Section 4.4 describes how Data Improvement is carried out for noise reduction. Finally in Section 4.5, we describe the various filtering techniques used for the segmentation process.

4.1 Hardware & Software

The system was implemented in C language. ANSI C was used on GNU compiler. The program was run on a NeXT station. NeXT computers use Mach operating system (System Release 3.0) which is a UNIX compatible operating system. Our machine was linked up to a network of several NeXT Stations and NeXT Cubes. This network included an external disk with 660 MBytes of memory. The network file-server was a NeXT Cube. Also connected to the network was our data acquisition computer. This computer is a Motorola 68030 processor based machine, built by the Kumon Group at Washington University. It runs on OS/9 operating system. It can be linked to a WACOM digitizing pen template, which is used to get data input. Pen computers such as those made by 'GO' can also be used instead. Figure 6 shows our set-up.

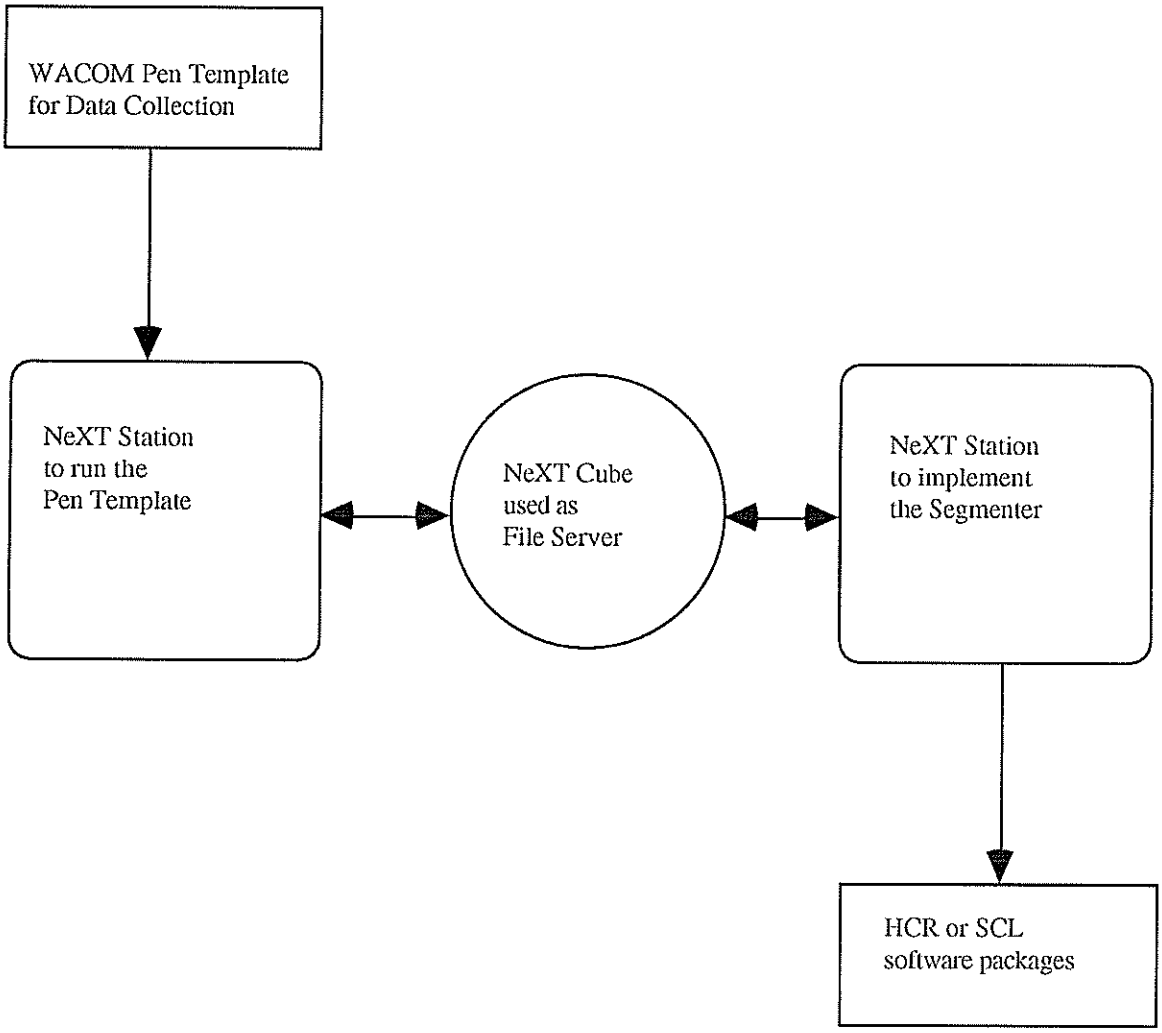

WACOM Pen Template
for Data Collection
NeXT Station
to run the
Pen Template
NeXT Cube
used as
File Server
NeXT Station
to implement
the Segmenter
HCR or SCL
software packages

Figure 6

The data was collected using a demonstration program, which is written by Julie Chan of Kumon Machine Group, Washington University. The collected data was transferred from one machine to another using File Transfer Program (ftp). It was viewed using a software package 'Handwritten Character Recognition, HCR,' which was designed by Tom H. Fuller at Washington University using NeXT's application builder. Appendix C shows samples of Handwriting obtained using HCR. Also used for linear regression data analysis were the software packages 'Cricket Graph' and 'Microsoft Excel' which run on Apple Macintosh computers. The output of our program, which is the segmented data, is eventually fed in Supervised Competitive Learning, SCL, package, for recognition. This software package is designed by Tom H. Fuller/Dr. T. D. Kimura of Washington University.

4.2 Data Collection

The data was collected by running a demonstration program on a pen template. This program displays two boxes on screen, as shown in figure 7. These boxes are labeled 'Home Phone' and 'Work Phone.' These boxes are large enough to hold a 10 digit phone number. The box size can be changed, without affecting our segmentation program. The user then enters the data through the pen, which is stored in Cartesian representation (see next section 4.3). The demonstration program also displays icons for saving the data, clearing the data, stopping the acquisition, etc., which can be selected using the pen. Details of test data collected from various users are given in Section 5.

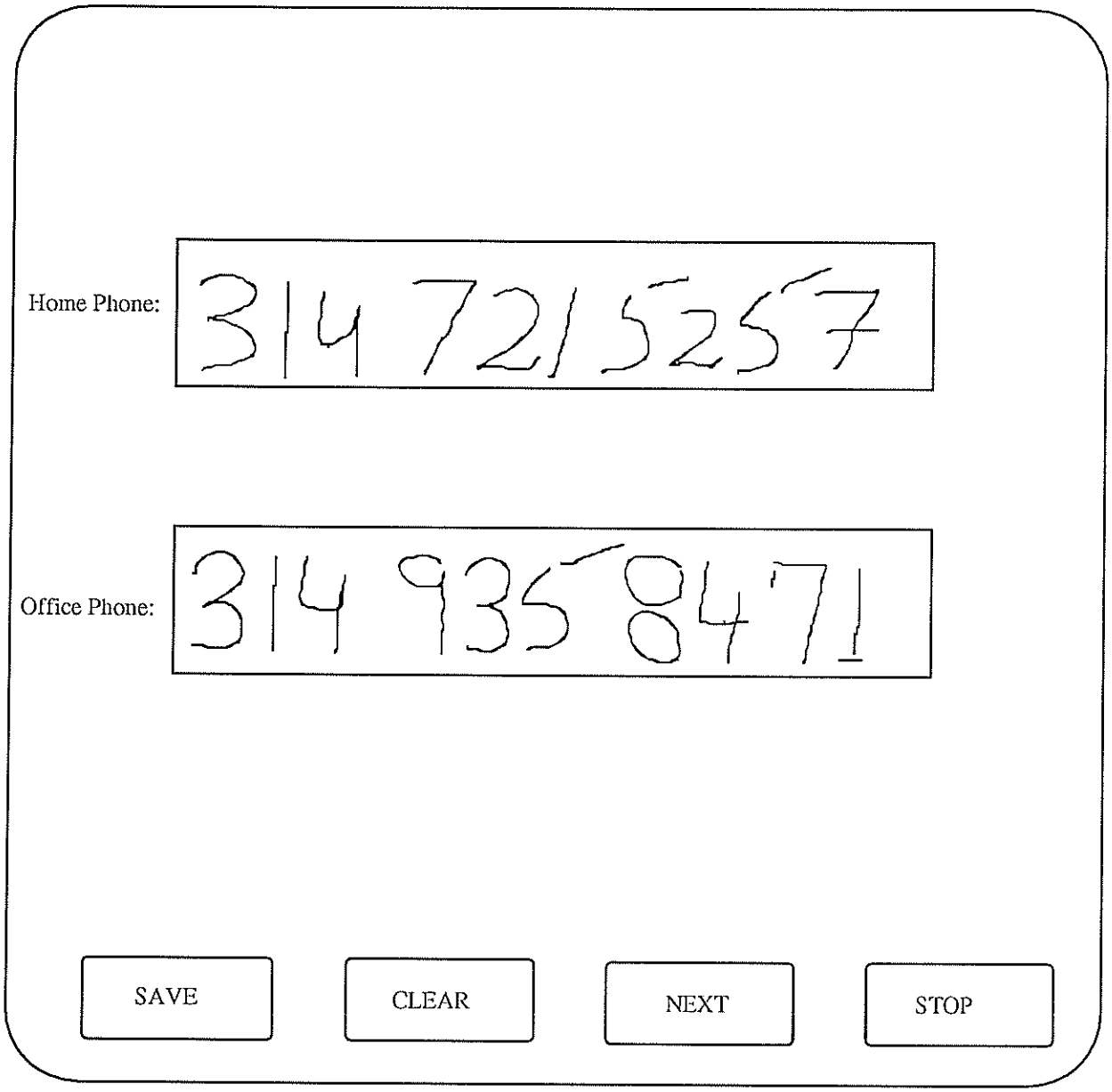

Home Phone:
314 7215257
Office Phone:
314 935 8471
SAVE
CLEAR
NEXT
STOP

Figure 7

4.3 Data Representation

The output of our program has the following format which is compatible with the SCL input format:

2 32 2
82 1277,2263 1273,2264 1273,2265 1271,2267 1271,2268 1271,2270 1273,2270 1274,2271 1278,2275 1283,2278
1290,2282 1291,2282 1301,2282 1306,2282
1312,2282 1319,2282 1334,2280 1340,2278 1347,2274 1356,2271 1361,2268 1368,2265 1372,2262 1373,2260
1377,2258 1377,2254 1377,2250 1375,2244 1373,2239 1367,2233 1364,2228 1355,2221 1347,2214 1343,2207
1333,2201 1328,2198 1324,2194 1316,2192 1312,2191 1311,2191 1310,2191 1310,2192 1311,2192 1314,2192
1317,2192 1322,2192 1333,2191 1338,2191 1345,2190 1349,2189 1357,2187 1363,2185 1366,2182 1372,2179
1378,2176 1385,2173 1391,2169 1394,2163 1396,2158 1398,2156 1399,2155 1399,2153 1399,2150 1399,2147
1399,2142 1397,2140 1395,2139 1395,2136 1394,2134 1387,2128 1386,2124 1381,2121 1375,2119 1367,2117
1363,2115 1358,2113 1357,2111 1354,2110 1350,2109 1348,2108 1346,2108 1345,2108
34 1456,2293 1454,2296 1453,2296 1453,2295 1453,2294 1453,2290 1454,2284 1455,2278 1456,2273 1457,2267
1457,2259 1458,2252 1460,2245 1461,2229 1461,2220 1462,2210 1462,2200 1464,2191 1464,2185 1464,2177
1464,2169 1464,2164 1465,2159 1465,2153 1465,2146 1465,2139 1465,2135 1465,2132 1465,2130 1468,2128
1469,2128 1473,2128 1474,2128 1475,2126

The above data represents one complete scribble. Observe the first line (2 32 2). The first digit '2' and the second digit '32' are significant only for the SCL package, and play no role in our segmentation process. The last digit '2' means that there are two strokes in this digit. It is followed by 82, which is the number of points in the first stroke. It is followed by all the points (x,y) separated by a space. Then we have number '34,' which gives the number of points in the 2nd stroke, followed by all the 34 points in standard Cartesian coordinates. A file can have more than one such scribble. Each scribble is separated from another scribble by a blank line.

The input to our segmenter package, therefore, is a file with one scribble and multiple strokes. This package would then group those strokes together which form a digit. A new scribble file is then made for each digit. This method is pictorially shown below in figure 8.

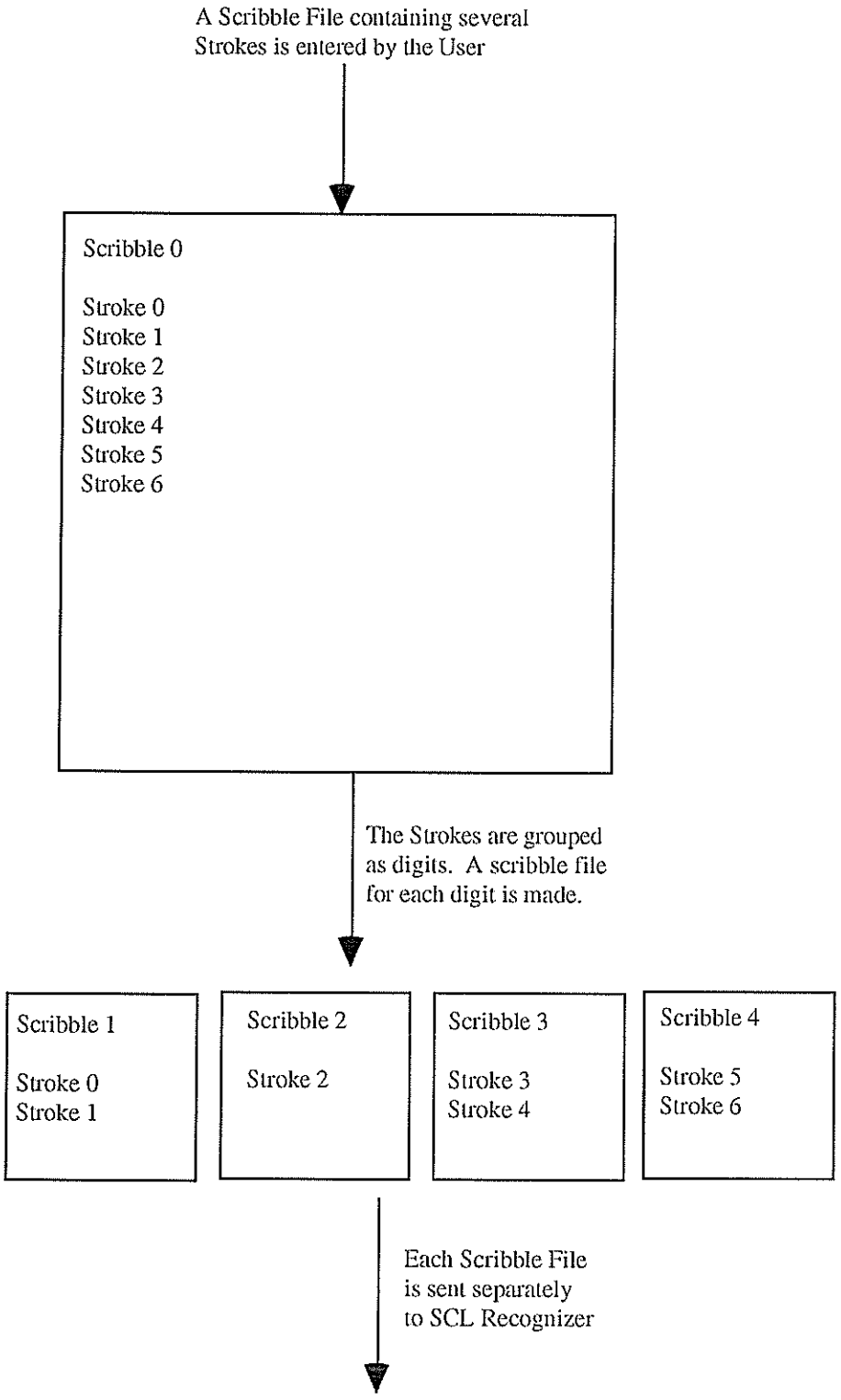

A Scribble File containing several Strokes is entered by the User
Scribble 0
Stroke 0
Stroke 1
Stroke 2
Stroke 3
Stroke 4
Stroke 5
Stroke 6
The Strokes are grouped as digits. A scribble file for each digit is made.
Scribble 1
Stroke 0
Stroke 1
Scribble 2
Stroke 2
Scribble 3
Stroke 3
Stroke 4
Scribble 4
Stroke 5
Stroke 6
Each Scribble File is sent separately to SCL Recognizer

Figure 8

4.4 Data Improvement

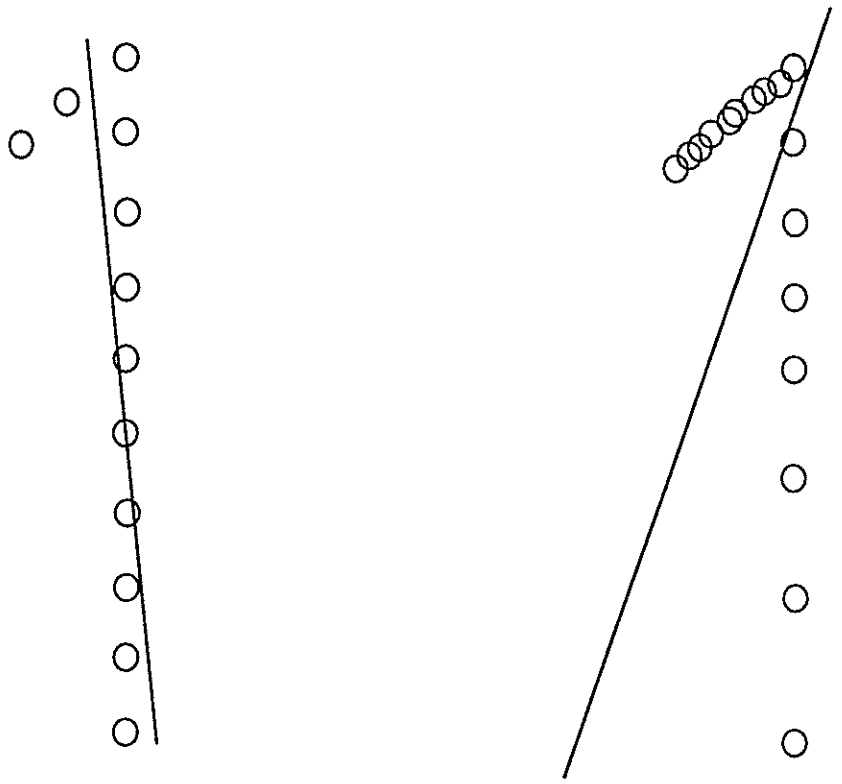
The collected data usually contains significant noise. Before the data can be utilized for problem solution, it needs to be modified or improved, so that the noise is reduced. We employed three ways of improving our data: sampling, smoothing and normalizing, each of which is described below.

4.4.1 Sampling

This Sampling algorithm uses the concept of Pen Velocity and Energy-related Feature Abstraction [7]. The algorithm basically selects and discards about 30% of the data points (roughly one-third) where the pen velocity is low. It does not discard data points where the pen velocity is fast (and hence the distance between the points is large). Such an algorithm is important in noise reduction because generally there is more noise distribution due to pen vibrations when the speed of the pen is slow. Furthermore, when doing linear regression on the data, if due to slow speed there are more points per unit length, then they weigh more and hence pull the slope of the best straight line towards them, as shown in figure 9. Therefore it is better to have a uniform point density throughout the stroke, which can be obtained by sampling the data.

Figure 9

The Best Straight Fits using Linear Regression are given below:



Digit '1' written with
Uniform Pen Velocity

Digit '1' with same shape
written with slower velocity
at the begining and faster
velocity at the end

The speed of the pen is approximated as the first derivative of distance. Hence, velocity = distance / time. Since we collect data after every certain fixed period of time, which can be referred to as unity, velocity is then simply the difference in the distance traveled by the pen between two successive samples. Therefore,

velocity = (d2 - d1) / dt = d2 -d1.

A seemingly simple way of sampling the data might be to calculate the velocities of pen at all the points, and then chop off the slowest 1/3rd points. However, this would not maintain 'Shape Preservation.' For example, consider a vertical stroke, say digit 3. What if the first 1/3rd set of the points are really slow and the last 1/3rd are really fast (relative to each other). One simply cannot discard all the first 1/3rd points (and keep all the last 1/3rd points) since this will change the shape of the stroke.

To implement a more suitable sampling algorithm, we had to work with 'relative velocities.' Let us define the Average Pen Velocty as the average of the pen velocities at each data point for a given stroke. Since each stroke has a different Average Pen Velocity, there are no absolute cut-off points for determining which velocities are low and which are high. Statistical theory of Normal Gaussian Distribution can help us here in categorizing the data points with respect to their speeds [20]. Fig 10 shows how all the point-velocities in a stroke can be represented by Normal Gaussian distribution. These points can then be divided into four equal categories: i) Fastest, ii) Fast, iii) Slow and iv) Slowest. Each category would then have 25% of the points. We need to find the cut-off velocity values which divide these categories. This can be done by first finding the mean and the standard deviation of all the velocities, which are given by [21]

$$Mean = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} x_i^2 - n\left(\bar{x}\right)^2}{n-1}}$$

The 25% boundary values can then be found using the Standard Normal Distribution equation [22]

$$P(0 < Z < z) = \int_0^z \frac{1}{\sqrt{2\pi}}\, e^{-x^2/2}\, dx$$

where z = 0.25. From the Statistical Tables [22], for this value of z, the cut-off points are at velocities u +/- 0.6075 (as shown in figure 10).

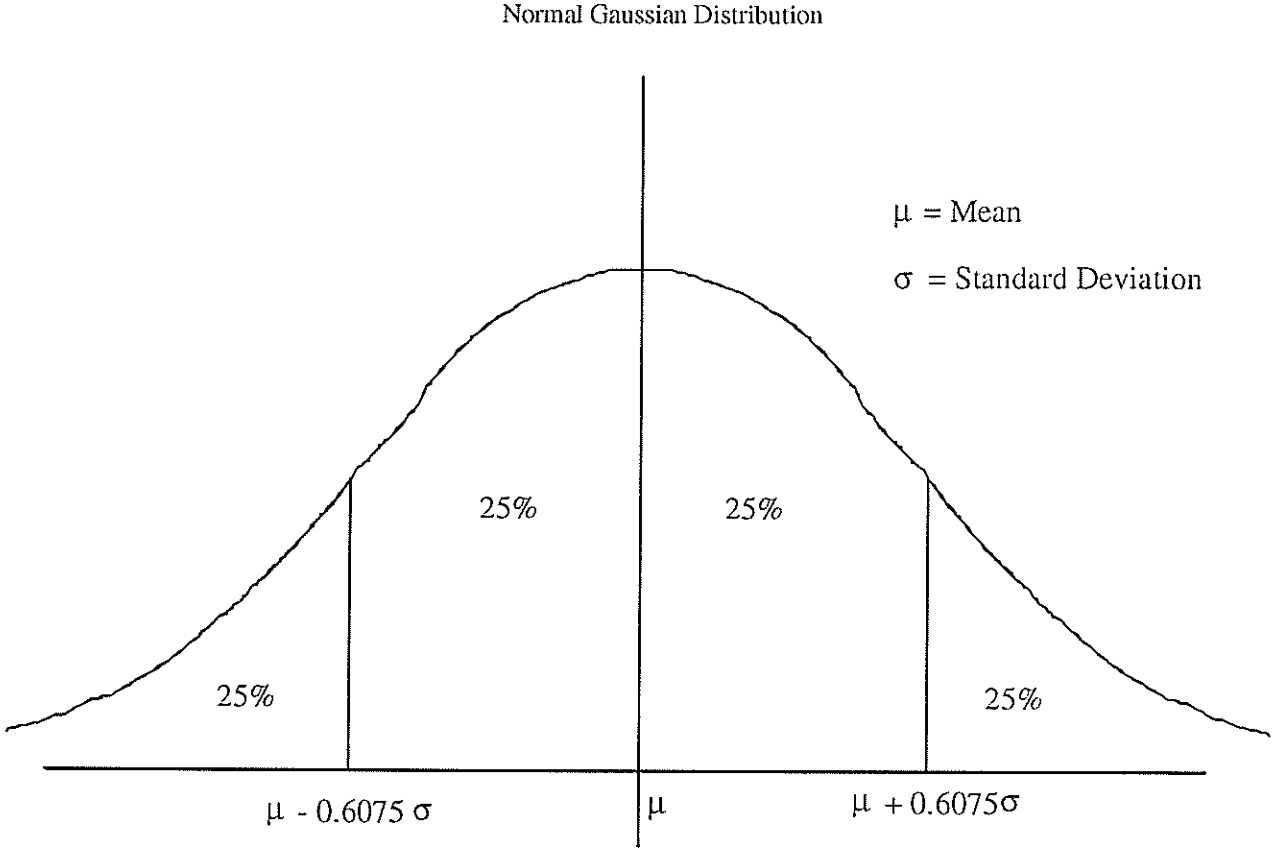


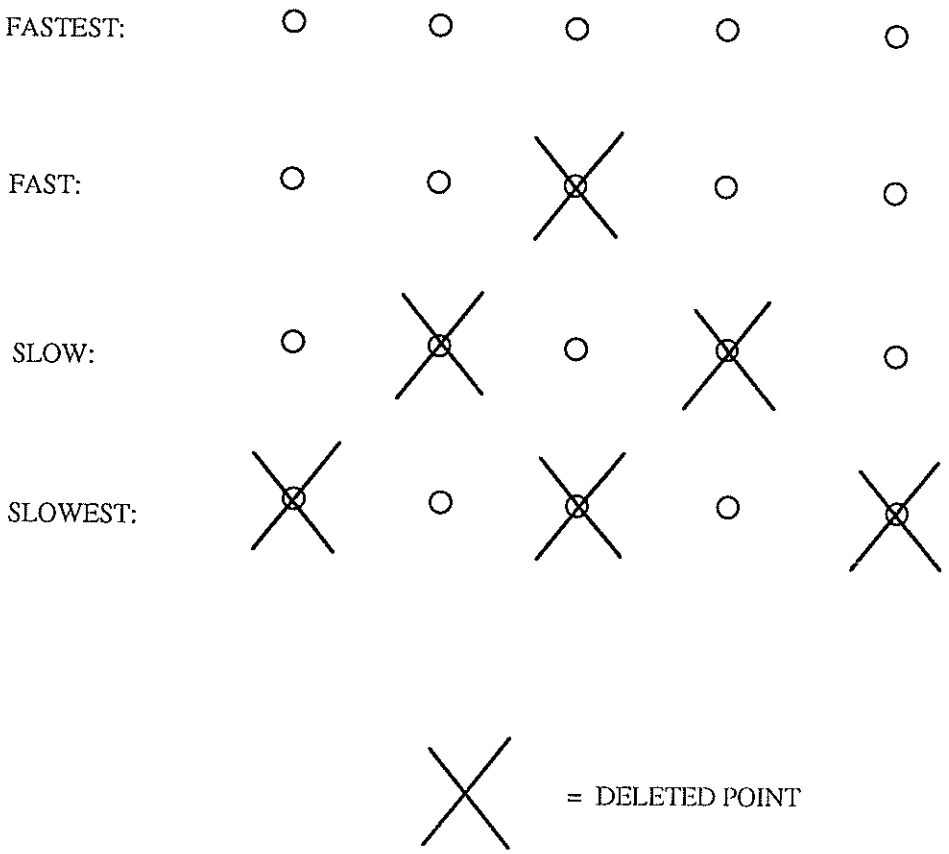
Figure 10

Now we divide all the velocity-points in groups of five, grouping them in the order in which they were collected. The mean velocity value for each group is calculated. We then check under which of the four categories this average group velocity falls. If it is 'Fastest,' we keep all the points. However, if it is slowest, then we chop off three of the five points. The deleted points are shown in figure 11, and the following table shows the number of deleted points in each category:

Category	Points Deleted
Fastest	0 of 5
Fast	1 of 5
Slow	2 of 5
Slowest	3 of 5
Total	6 of 20 (30 %)

Table I

Figure 11



Since on average, we would have an equal number of groups under each of the four categories, for every 20 points we are removing 6 of them, giving us a 30% drop rate. Actual runs on various different strokes showed that the percentage rate varies from 26% to 34%, usually giving an average rate of 30% for a scribble of 10 strokes.

Some overhead calculations to this algorithm need to be made for the end points. This is important when the total number of points in some stroke is not a multiple of 5. The extra left-over points, whether they be 1, 2, 3 or 4, have to be dealt with individually. There are several ways of implementing an algorithm for dropping the end points. In our case, we decided that if there are 1 or 2 extra end points, then they should not be dropped. However, if there are 3 or 4 end points, then one or two of them might be dropped depending on which of the four categories the mean velocity of these points lies.

4.4.2 Smoothing

Sampling discards some unnecessary points, but does not change the points position. Hence any noise due to a shaky or vibrating pen continues to exist. This noise can be reduced by smoothing the data. Several algorithms exists for 'smoothing.' Gunst and Mason [23] give a good smoothing algorithm which uses medians of the adjacent points and then swaps the points. However, their algorithm does not work on data where 'spatial preservation' has to be maintained. Also, a week point in their algorithm is that one loses some starting and finishing points (end points).

We implemented an algorithm which is simple and smoothes the points significantly while maintaining the spatial preservation. The idea behind this algorithm is to replace each point by the mean of its neighbors. Various different versions were tested. The criteria for judging improvement was to visually observe the original and the resultant scribbles on screen, and then to judge the degree of smoothness. After trials of at least four different methods, it was found that the most appropriate results were obtained by taking the mean of five points. These other methods included assigning weights to the points, such that the closer neighbors have more weight than the further ones, or re-smoothing repeatedly the resultant data. These five points whose mean was taken comprised of two neighbors on either sides of the point, and the point itself. Decreasing five points to three (only one neighbor at each side) decreased the level of smoothness while increasing it to seven (3 neighbors on each side) did not show much improvement.

Again, the smoothing algorithm has to encompass the end-points, such as those which have no neighbors, or less than two neighbors. This introduces some overhead calculations, but they are worth it. Our algorithm checks as many neighbors as it can up to a maximum of two on either end. If less than two neighbors are present, the algorithm will take the mean of only those which are present. This works out quite well and gives good results. The first two handwriting specimens in Appendix C show data before and after smoothing; the improvement in data is clearly visible.

4.4.3 Normalizing

Normalizing the data means scaling it with respect to the minimum x and y coordinates. Help was taken from past work done on Normalization [6] [24] This is especially useful in viewing the data, since it has the effect of expanding the data so that it fills up whole of the screen. This is why software such as HCR [25] which displays data on screen has in-built normalizing functions. It is important to note that since we would be dealing with relative lengths of adjacent strokes, we cannot normalize the strokes individually; instead, the whole scribble has to be normalized with one minimum x coordinate and one minimum y coordinate.

4.5 Methods for Segmentation

Once the data has been collected and improved, we begin with the segmentation process. The key component of this process is our seven-level filter. The system can be represented by the following algorithm, which is also shown in the flow chart in figure 12.

1. Collect and Improve Data.
2. Starting from the first stroke, use the 7-level filter to determine if any particular stroke (nth stroke) can be grouped together with its neighboring stroke (n+1th stroke) to form a digit.
3. If Yes, then make a new Scribble file for this particular double-stroked digit, containing the nth and n+1th strokes.
4. If No, then make a new Scribble file for a digit containing only this particular nth stroke.
5. Repeat the above procedure for all the strokes.

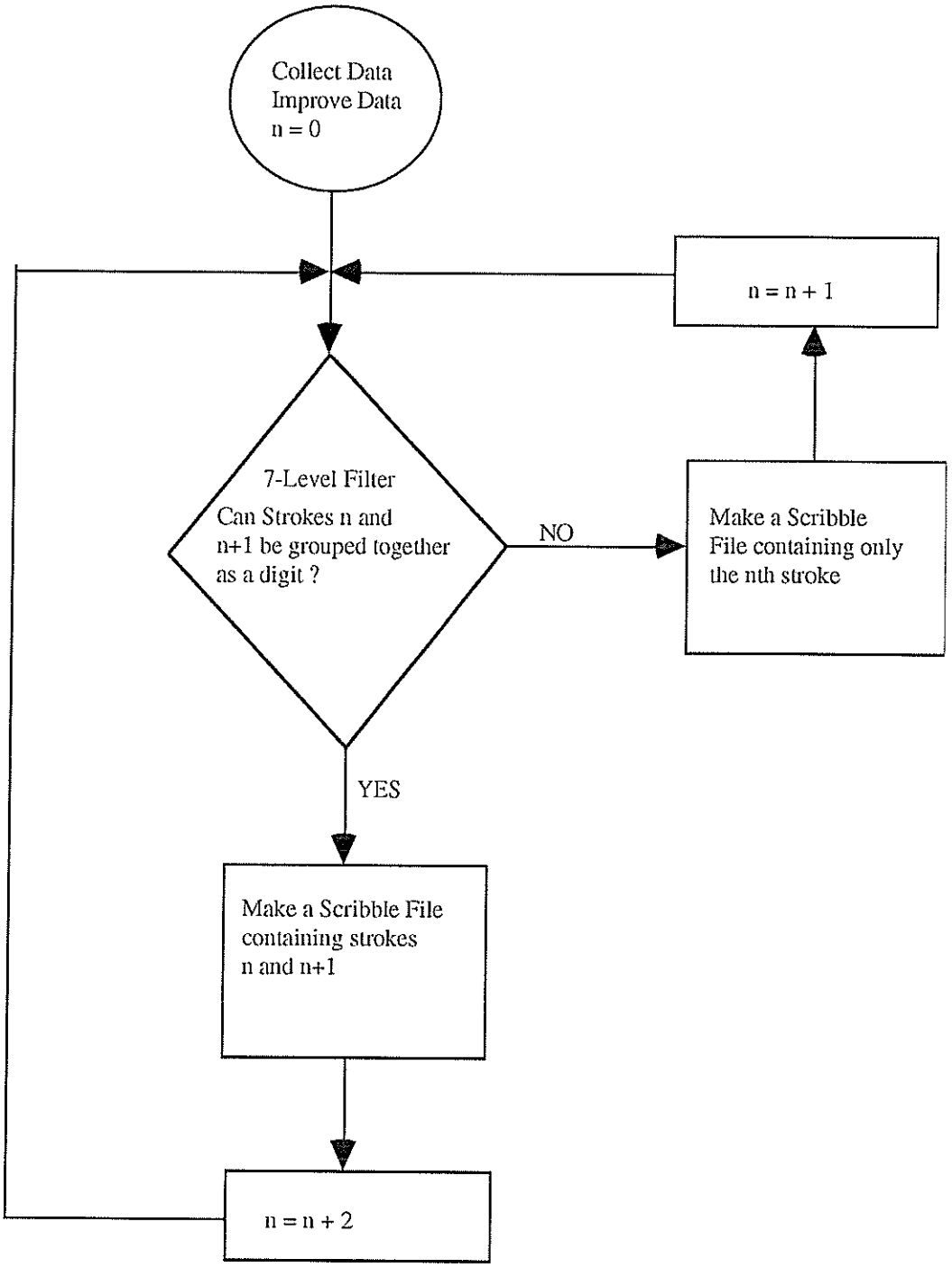
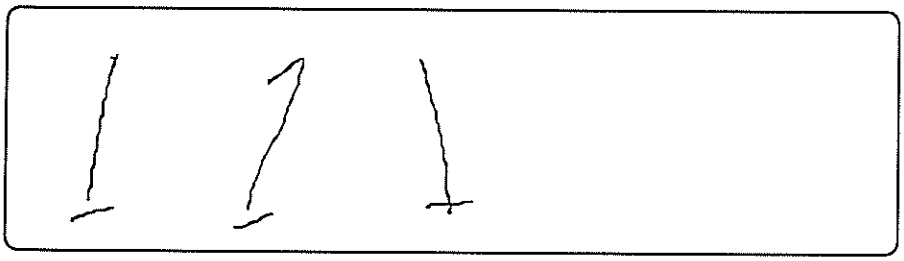

Collect Data
Improve Data
n = 0
n = n + 1
7-Level Filter
Can Strokes n and
n+1 be grouped together
as a digit ?
NO
Make a Scribble
File containing only
the nth stroke
YES
Make a Scribble File
containing strokes
n and n+1
n = n + 2

Figure 12

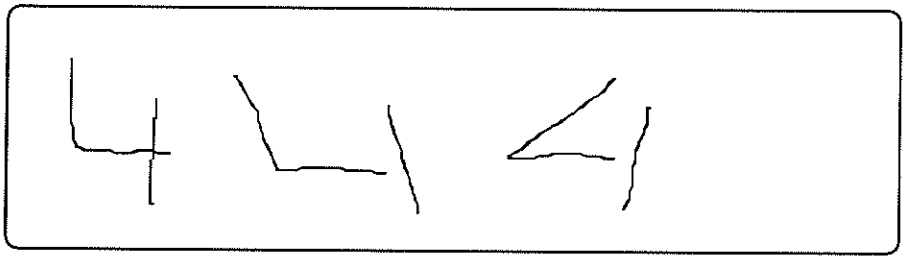
Observe that we are making an important presumption here, that if two neighboring strokes intersect, then they form one double-stroked digit. For a non-cursive writing, this is generally true; several people tend to write digits separately without intersection, though exceptions do exist. Therefore, the first step is to check for intersection of neighboring strokes. However, the difficulty arises when we have to check if a stroke is part of a double-stroked digit. Basically every digit can be written using 2 or more strokes. However, a survey of 15 users writing a total of 650 strokes showed that the digits 0, 2, 3, 6 and 9 are almost always written with one stroke only. Therefore, as mentioned in the problem definition section 2, our problem can be simplified greatly if we presume that the only double-stroked digits can be 1, 4, 5, 7 or 8. This is a very discrete and fair presumption, which is supported not only by sample data but also by common observational knowledge. Of course there are exceptions, but it was found that finding a correct solution for the exceptions can actually spoil the results for proper (non-exceptional) data. Figure 13 shows some different ways with which various people write these 5 digits using 2 strokes per digit.

Figure 13

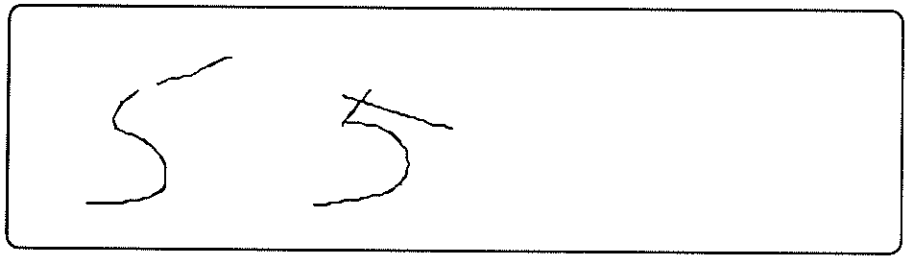
Digit '1':



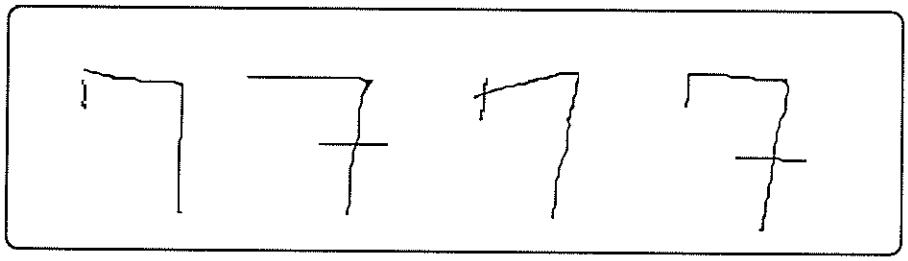
Digit '4':



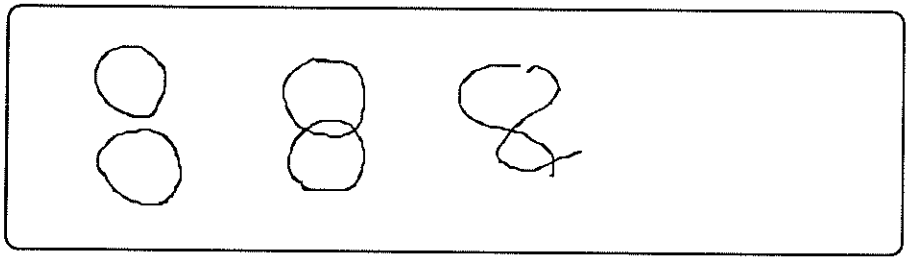
Digit '5':



Digit '7':



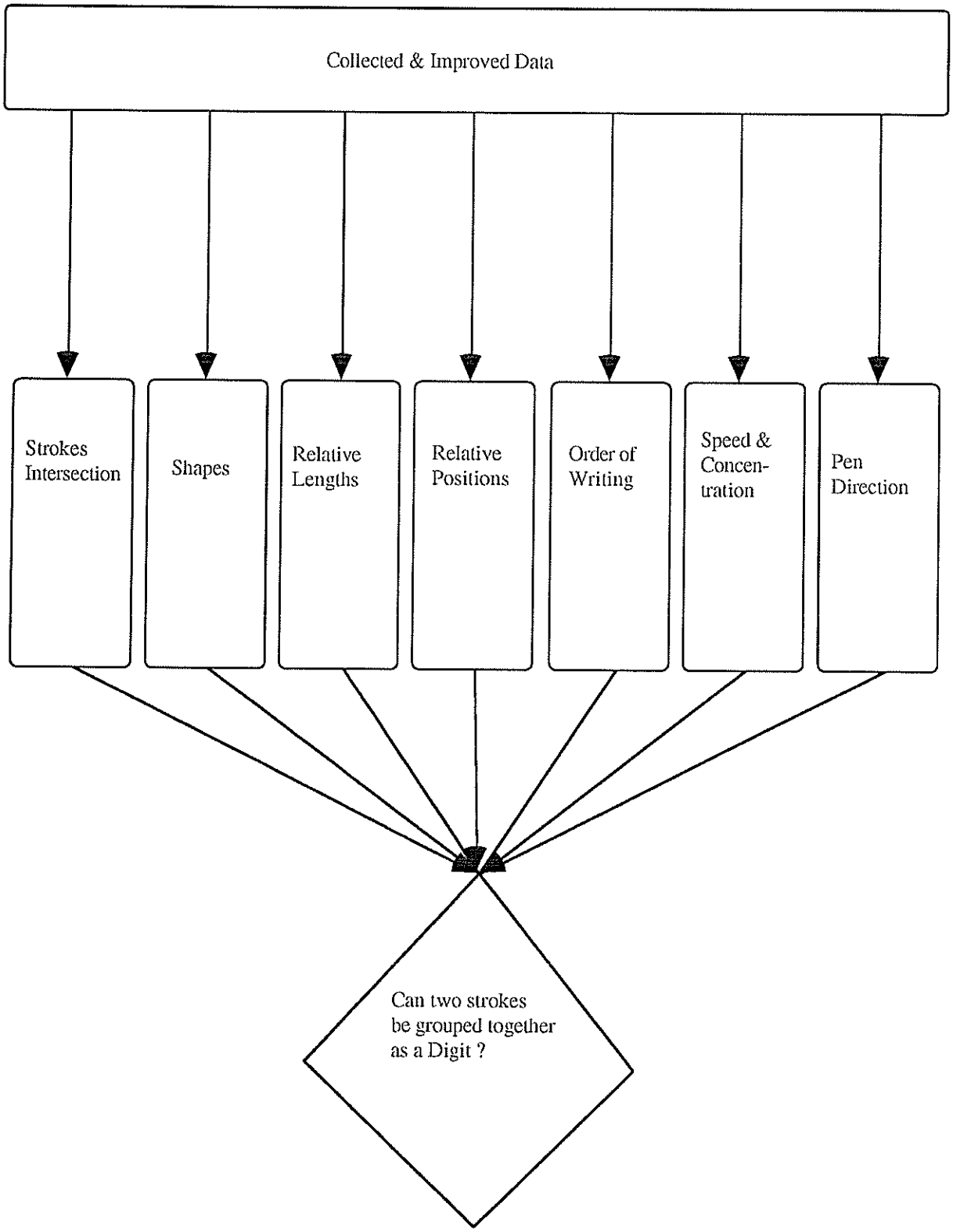
Digit '8':



The main problem, therefore, is to use the 7-level filter to check if every stroke is part of a double-stroked digit or not. Each filter uses some stroke characteristic to give us certain feature abstraction information related to that stroke. Although each filter helps in the segmentation process, probably none of these can achieve 100% success on its own. These filters are listed below, shown pictorially in figure 14, and then described in detail:

1. Strokes Intersection
2. Stroke Shapes
3. Relative Stroke Lengths
4. Relative Stroke Positions
5. Order of Stroke Writing
6. Pen Speed and Points Concentration
7. Pen Direction

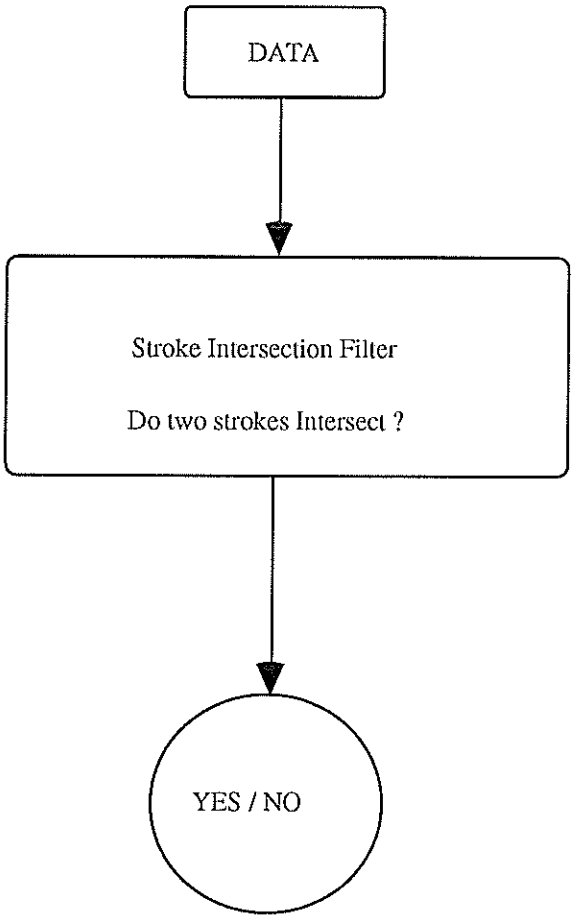
Figure 14


Collected & Improved Data
Strokes Intersection
Shapes
Relative Lengths
Relative Positions
Order of Writing
Speed & Concen-tration
Pen Direction
Can two strokes be grouped together as a Digit ?

4.5.1 Strokes Intersection

We are presuming that the digits don't overlap, i.e., if two neighboring strokes intersect, then they must belong to the same digit. Therefore checking for intersection of strokes plays a key role in our algorithm. This is why we paid considerable attention to the intersection algorithm, and came up with one which does not fail. Had we also presumed that the strokes of double-stroked digit MUST intersect, then we would have achieved a 100% segmentation success rate. Unfortunately, in practice, this assumption cannot be made.

Figure 15



The intersection algorithm is very fundamental. The block diagram of this filter is given in figure 15. Lets first define a segment as the line joining two successive points within a stroke. In order to check if two strokes intersect, we have to check if any of the segments of first stroke intersects with any segment of the second stroke. Research was carried out on past work done and it was found that Ben Waily [26] had written such a 'segmentIntersect' procedure in Objective C and C++, which worked in an efficient and fast way, giving 100% success rate for integer points. This code was modified to work with our version of plain C, and then expanded from segmentIntersect to work for strokeIntersect procedure as well.

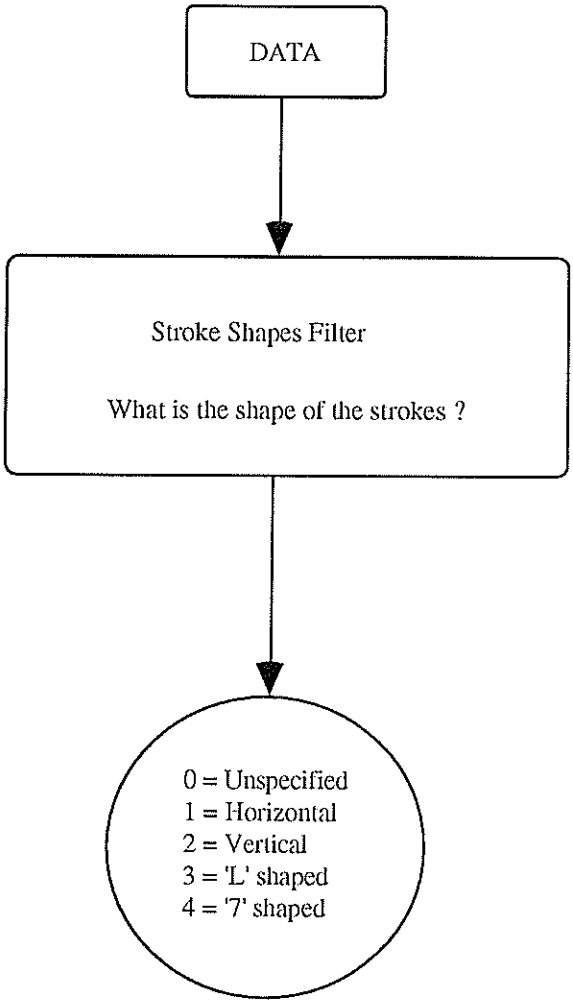
4.4.2 Stroke Shapes

In order to carry out segmentation, some recognition of shapes of digits would definitely be helpful. In particular, shapes of those strokes which are composed of straight lines can be easily recognized. Recall that the only double-stroked digits we are concerned with are 1, 4, 5, 7 and 8. Of these, 1, 4 and 7 have both the strokes in the form of straight lines, while 5 has one such straight stroke. The first stroke of digit 4 is 'L' shaped, while that of digit 7 is reverse 'L' shaped. Furthermore, an 'L' shaped stroke can also be represented by two straight lines, roughly perpendicular to each other.

We, therefore, implemented a recognizer package which recognizes the shapes of any straight lines present amongst our strokes. Figure 16 shows the block diagram of the Stroke Shapes Filter. Such strokes would then fall under one of the following five categories:

0) Not straight
1) Horizontal
2) Vertical
3) 'L' shaped
4) Reverse 'L' ('7') shaped

Figure 16



Two different methods were employed, which are termed as i) Orthodox Method and ii) Linear Regression Method. The final solution uses both the methods. These methods are described below in detail.

4.5.2.1 Orthodox Method

Our orthodox algorithms use an easy way of recognizing a vertical stroke: a vertical stroke is the one in which all the y coordinates decrease. Moreover, it should not have curves in it; hence if x coordinates are increasing, they should continue to increase or if they are decreasing, they should continue to decrease. The slope of the points can also tell whether we have a vertical or a horizontal line. To summarize, we can detect horizontal and vertical strokes as follows:

Horizontal: x coordinates continuously increase, so x2 >= x1 for all x, while y coordinates either only increase or only decrease.

Vertical: y coordinates continuously decrease, so y2 <= y1 for all y, while x coordinates either only increase or only decrease.

Ideally, the orthodox method can give 100% results. However, no one writes perfectly straight strokes or digits and also the noise distorts the actual shape. When data was analyzed, it was found that a sample digit '1' contained several minute curves in it, and that due to the tiny distances between the points, there was no guarantee that the y coordinates would decrease all the time. The results depend on the speed of the pen, and how much the user's hand shakes. We were able to achieve (in best cases) a success rate of only 70% in recognizing horizontal and vertical strokes. Such a rate is completely unacceptable, especially when one is only at the most fundamental level (later levels would include recognizing digits 4 and 7). The orthodox approach also has the problem with the directionality. If someone writes a 1 from bottom up, or sketches the 2nd stroke of 5 from right to left, then our algorithms of decreasing y coordinates and/or increasing x coordinates simply fail.

We came up with various modifications for the above orthodox algorithm to yield slightly more accurate results. Not all of them were employed in the final version though. For example, we found that in a lot of cases a vertical stroke had proper decreasing values of y except for at the starting point and the ending point, where the pen is initially placed and then removed. So one method was to ignore the 10% points on either ends while checking for the increase and/or decrease in x or y coordinates. Another approach was that while checking the increase or decrease pattern, instead of comparing every set of adjacent points (say x2 >= x1), compare pairs which are 5 points away from each other (so check for x5 >= x1). Hence minor noise variations could be eliminated to quite some extent.

Since the orthodox method fails to recognize straight strokes unless they are perfectly ideal, it can only be used as a backup method which can be used to counter-check the results of the Linear Regression method. Nevertheless, the techniques employed in this method give considerable insight towards comprehending stroke shapes. It was entirely these techniques which led us to develop the Linear Regression method.

4.5.2.2 Linear Regression

Noise and other problems were encountered through the use of coordinate geometry and statistical theory of linear regression [27]. Linear Regression is the method which gives us the best possible straight line fit through a set of points. Mathematically speaking, there can be no other straight line (besides the best fit line) which can pass through the points with a smaller average distance from them. Hence linear regression is the most accurate way of finding the equation of the best