R

FOR REFERENCE ONLY

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

ALGORITHMS FOR THE RECOGNITION OF HANDWRITING IN REAL-TIME

Philip Timothy Wright

This thesis has been submitted in partial fulfilment of the requirements of the Council for National Academic Awards for the degree of Doctor of Philosophy

January 1989

Trent Polytechnic in collaboration with
Plessey Research, Romsey

ABSTRACT

Algorithms for the Recognition of Handwriting in Real-Time

Philip Timothy Wright

This thesis details the work undertaken by the author from September 1984 to September 1988 into the field of dynamic script recognition. It reviews the various techniques developed since 1960 and it analyses the more popular approaches to processing the raw pen motion information. It also details the progress made in the nature of the user interface over the last 28 years.

The main emphasis of the work has been the development of algorithms capable of recognising, in a real-time user independent environment, lower case hand-printing. In particular, the design of the character shape databases provides for rapid searching and character matching and the techniques of feature reduction provide character matches to be found from previously original character encodings.

The most successful algorithm, based on the method of curve encoding by H.Freeman, forms a foundation towards the development of natural user text and data entry system. Extension of the character base is also possible with no alteration to the basic algorithm methodology. A technique of robust word segmentation has been designed that has enabled the design of a prototype cursive script recognition system. This is presently writer independent, running on a 68020 micro-processor. Initial results show a word level recognition rate of 95+ %. Development of natural editing functions provides a self contained text entry environment.

In the future, the algorithms will be ported to an 'electronic paper' environment and a user training phase will be designed as a front end to the recogniser.

CONTENTS

Table of Contents

INTRODUCTION

This thesis details the work undertaken by the author at Plessey Research over a period of four years from September 1984 to September 1988. The work is partly funded by Plessey research, and partly by the European Commission under the ESPRIT initiative. The title of the project is 'The Paper Interface -Project no. 295'. The area of research addressed by this work being the analysis and recognition of script in real time. The thesis describes the work undertaken in a roughly chronological order. The majority of the effort concentrated on the development of algorithms, initially for the recognition of real time hand-printing of the lower case alphabet (a-z), but with the ultimate intention of being able to adapt the techniques developed to be used to recognise the more natural cursive handwriting. It is also the aim to expand to a larger symbol set, including upper case characters, numerals and special characters (punctuation marks, mathematical symbols and so on).

An initial study period of three months was taken in gaining an in-depth familiarisation with the problem and the approaches taken by previous researchers into the subject. This was performed by reviewing as many previously published papers as possible. However, throughout the subsequent course of this work, any new papers published were periodically reviewed to monitor new developments, in particular with respect to the user interface, which is becoming of increasing importance in terms of gaining any degree of user acceptance from any resulting script recognition related products.

The familiarisation gained with the subject indicated that every approach to the recognition problem performed some degree of pre-processing on the raw input data. This data being a time related positional trace of the pen tip recorded as a person writes a piece of text. Therefore, analysis of the different preprocessing methods was performed in order to gain some degree of insight into the effects on character parameters of the different techniques. One particularly striking question that arose from the bulk of the papers that were studied was whether any of the methods was particularly suitable for adaptation to a real time environment. Subsequently, particular emphasis was placed on real-time implementation in the design of the two recognition algorithms described in Chapters 4 and 5.

One particularly important aspect of the recognition process is the extraction and subsequent comparison of the character features against some predefined set of rules. These rules are usually constructed by the detailed analysis of character shape and formation style as produced by a number of sampled writers. Both the algorithms developed have a database for feature comparison. The construction and accessing of these databases is of particular importance to the overall speed of the whole system. Chapter 6 describes the database construction and operation.

Once the character strokes have been recognised, it is necessary to process this sequence of characters into some recognisable sentence of words as written by the user. Both word and sentence construction, together with

some preliminary text formatting has been undertaken, although the development environment was not particularly helpful in this respect, with the disjoint writing and reading devices.

A detailed breakdown of the performance of the complete lower case hand printed script recognition system is given in Chapter 8. Each separate stage of the process is broken down in order to assess any particular strengths or weaknesses in the system. The ultimate goal of the work lies beyond the scope of the work described in this thesis. Ultimately we want to develop a system which is able to recognise a writers natural handwriting. Chapter 9 describes some of the advances we have made in investigating the feasibility of cursive script recognition. Initial results, for a system trained to the style of a specific user have shown a good deal of promise, but we are still far removed from a system which is able to recognise handwriting produced by a number of writers.

In the concluding chapter, future work is discussed, together with observations on this initial research stage.

1. STATE OF THE ART REVIEW

1.1. Introduction

A study of papers written on character recognition has led to the compilation of 95 articles and papers covering the period 1957 to the 1988. Early work tended to concentrate on a very simple subset, namely the numerals 0-9. The techniques developed and character types analysed were very much dependent on the available technology. As a result of the limitations in the acquisition of the written information, the user interface was very basic in many instances. For example, one of the more reliable techniques of character data input available during early investigation was a CRT with a light pen attached. The character shape was encoded by determination of the pen position on the surface of the CRT at set intervals in time. However, due to the scanning frequency and the accuracy in absolute determination of the pen position on the screen, each character had to be written quite slowly and the character was required to fill the screen. Within the constraints of input of the character data by the user and limitation of the data set, initial research showed promising results, Caskey [28], Teitelman [29].

The advent of the graphics tablet as a data capture device proved to be the platform into researching recognition of script as a feasible man-machine interface. The graphics tablet allows the user to construct sentences upon a piece of paper as they would normally with a pad and pen or pencil. Research rapidly evolved into the analysis of the complete range of written characters, letters a-z and A-Z, numerals 0-9, and the special characters !,@,&,+ ,-,= and so on.

A number of products appeared on the market which perform script recognition but these have so far met with limited end-user interest. The main area of interest has been found to be in form filling applications. This again is a basic limitation on flexibility of user input, in that a person must write each character within a predefined box, forming each character in one of a number of acceptable styles, the characters being limited to upper case, numerals and special characters. However, the recognition rate is usually very good (99% or more) once the user has adapted their writing style in order to eliminate any possible character ambiguities.

The natural progression is the recognition of cursive handwriting. If the recognition of script is to be the basis of a natural user interface, it must address itself to the problem of cursive script recognition. Increasingly over the last 15 years, work has moved towards cursive script recognition. However, success has so far been limited and has introduced new areas of research into techniques beyond the geometric features requires the use of dictionary look-up and n-gram analysis in order to identify possible letter sequences, and so supplement the basic recognition algorithm by processing and improving the basic recognised text.

The paper by Tappert [87] gives a good indication of the latest state-of-the-art situation regarding on-line handwriting recognition, referencing no less than 257 papers. In particular, it shows that renewed effort is being put into the problem of recognising cursive handwriting, and that much effort is now being directed towards the man-machine interface aspect of the problem. In particular this is concentrating on the development of 'electronic paper', a combined tablet and display device. Two products are already available that feature electronic paper, from Linus in the USA, and from Panasonic (Japan), the Panaword RL-450, both products limited to unconnected character analysis.

1.2. Spatial Analysis Methods

This technique, in principle, is the simplest of all the those surveyed. Much of the early work into character recognition used some form of spatial analysis. The technique is inherently limited to the analysis of individually written characters. A character is written over a platen which is divided into a number of regions.

One of the earliest methods of dynamic character recognition was devised by Dimond of Bell Labs [27] in 1957. He devised a data capture device called a Stylator. The user must write the character around two reference points. A series of wires are connected to these reference points, projecting radially outwards. As the character is being formed the pen passes over the wires to produce a path sequence around the two reference points. This technique seems to suit some characters more than others in the ability (or not) of the character to be sensibly formed around the reference points. The example below shows the character '2' being formed:-

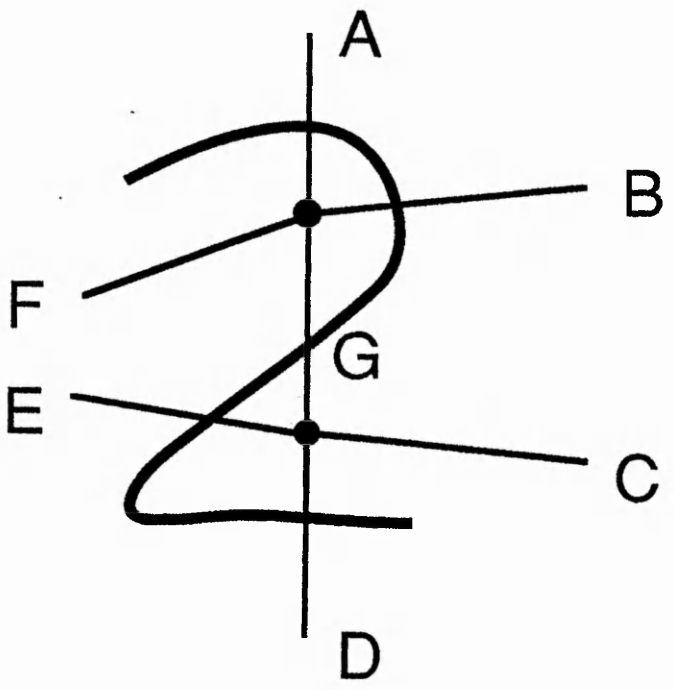


Figure 1.1 - Numeral '2' Drawn on the Stylator

Thus, the encoding for the character is:-

'2' = A B G E D

Another very early example of the approach was researched by Richard Brown in 1964 [2]. This was one of the first methods that utilised time information during data capture. A metal platen, comprising seven separate plates was written onto, as shown below:-

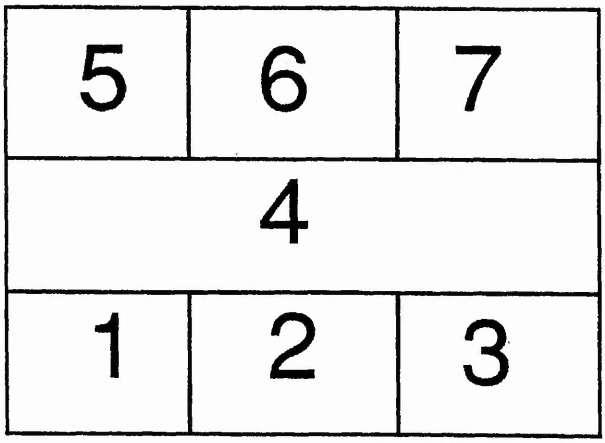


Figure 1.2 - Rectangular Pattern Matching Grid

As the pen passes over a metal plate, the produced signal is uniquely encoded in order that the host can identify which plates had been passed over and in which time sequence. Therefore, there is no character shape information. Pen-up status is also given, which is an important feature of the recognition algorithm. A user is required to train the system beforehand in order that it can store the appropriate codes for subsequent comparison. An example of such a character encoding is:-

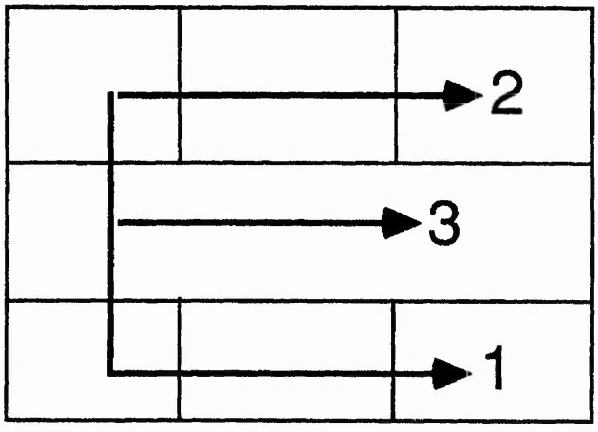


Figure 1.3 - Character 'E' written over the platen

The character 'E' produces the following grid encoding,

'E' = 541230 5670 40 (0 = Pen-up)

An error rate of between 5 and 10% was obtained on a character set which included the upper and lower case alphabet, Arabic numerals, punctuation symbols, and some mathematical symbols. The sample base for the character set was very small (400 characters), suggesting some constraint on writing style. However, this idea was picked up by future researchers, adopting the technique for use on data tablets, where the user is not so constrained by character style and size. The only major difference being the division of the grid into 9 equal rectangles, the reference grid being constructed around the character after it has been written.

Teitelman [29] extended the idea of the 9 rectangle grid by defining four overlapping regions within the character grid. This technique has the advantage that it is more flexible in the encoding of slightly different styles of writing a particular character ie. it will not require two encodings for two slightly different styles. The main problem it can overcome is character slant. It was also shown to be quite easy to extend the region areas in order to further distinguish between characters which display further ambiguities. :-

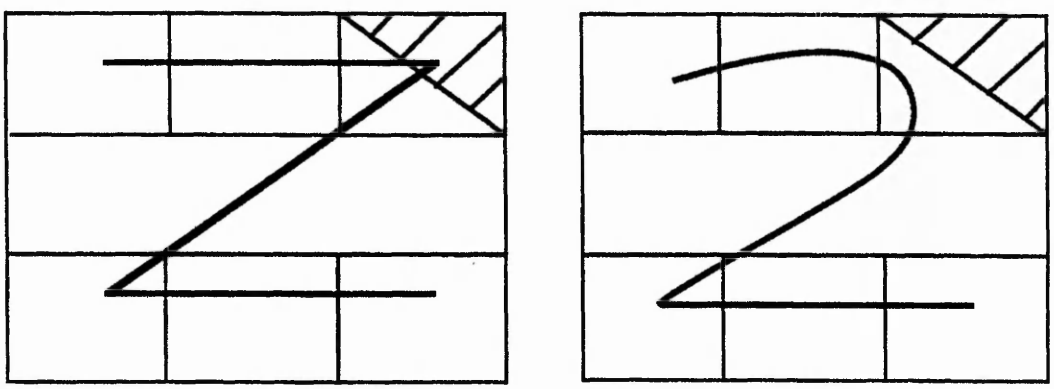


Figure 1.4 - Generation of a New Property Search

In this instance a new branch on the decision tree has been added in order to be able to distinguish the numeral '2' from the character 'Z'

Another type of spatial grid was considered by Tou and Gonzalez [3]. In this case an octagonal grid is used.

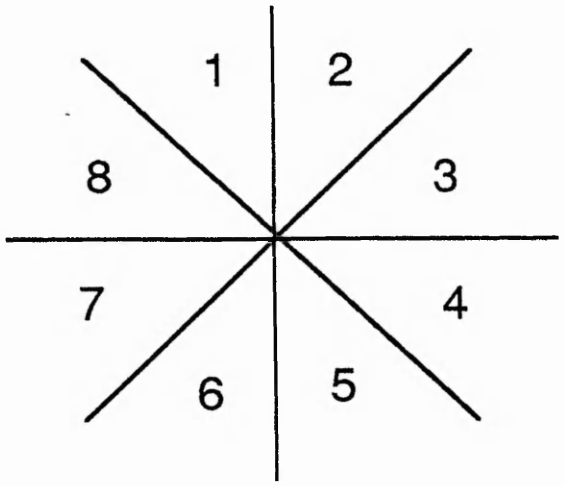


Figure 1.5 - The Octagonal Grid Representation

The character shape is input in terms of the pen position on a 60x60 matrix as it is being written. The centre of gravity of the character is determined and the origin of the grid positioned onto it. This had the advantage that the grid size did not need to be recalculated each time. Another benefit of this method was that the user was not constrained to writing a character of a certain size. The technique of encoding, however is identical to the schema adopted by Brown. If we consider the character 'e' mapped onto the octagonal grid :-

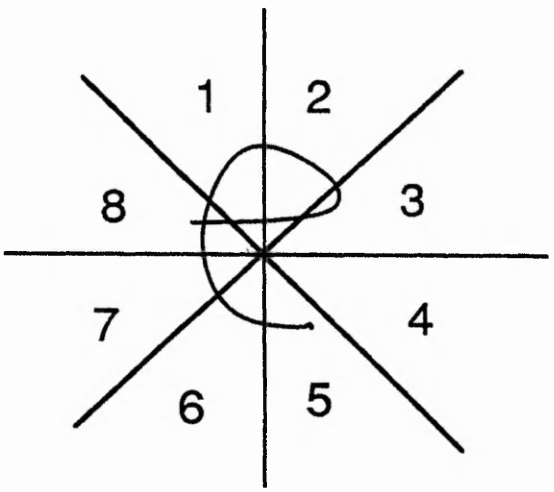


Figure 1.6 - Character 'e' on the Octagonal Grid

Producing the encoding :-

Character 'e' = 8123218765

1.2.1. Observations

Spatial analysis is extremely easy to implement, giving the capability of producing a low cost on-line recogniser. Such method was implemented by Simmons [16] as a graphics tool for a microcomputer or PC. 90% recognition is achieved on a character set consisting of the upper case alphabet. Executable code is only 2K bytes. It analyses the regions the character curve enters within the rectangular grid (0-8) and matches the region code produced with a reference table of previously trained results.

Although this technique would be quite suitable for a very simple system as described above it does not lend itself easily to an unconstrained multiuser environment. Major shortcomings are:-

1. By its very nature it is constrained to unconnected letters or numerals.
2. The training of the look up tables limit themselves to a single users character construction. Different user styles in character shape and the method of creation can produce an almost limitless number of variations on the region path which would require a disproportionately large look up table.
3. Considering only a single user system, unless the user constrains their style, it is quite feasible that each time a certain character is written it will produce a slightly different region code due to slightly different character shapes. Hence, even a user dependent system needs the support of consistency by the user.

1.3. Topological Feature Based Methods

Examination of the topological features of hand-written characters and numerals is a very popular approach to dynamic recognition. The methods adopted vary mainly in the complexity of features that they analyse. Very basic techniques include the analysis of the following features:-

- The detection of any straight line segments along the curve and their orientation, whether it be horizontal, diagonal or vertical.
- The detection of curvatures and the analysis of their direction of formation, either clockwise or anti-clockwise.
- The identification of characters comprising more than one single stroke.
- The identification of cross strokes and dots and identification to the stroke to which they are related.

This basic feature set in itself will not differentiate between all the characters in the allowable set (numerals, upper case characters or whatever).

These features do however allow the characters to be classified as belonging to a character subset. Within these subsets it is possible to investigate for additional features. These features are usually more complicated to investigate than the initial feature set as they are more specific characteristics. These tend to investigate for such features as:-

- Detection of a cusp (a point where the curve suddenly changes direction causing a tooth-like shape in the curve).
- Detection of loops.
- Detection of crossings in a character curve.
- Detection of the curve meeting itself tangentially further along.
- A more detailed analysis of any curves in the character leading to a wider classification of the curve type.

The technique has been applied to both separate characters and also to sequences of characters at word level.

1.3.1. Isolated Character Analysis

The technique described by Tou and Gonzalez [3] is a technique of spatial analysis which incorporates the beginnings of some form of feature extraction. An analysis of the regions entered by the character mapped onto the horizontal grid (Figure 1.6) enables it to be approximated by a series of horizontal and vertical strokes. Whether or not a particular stroke is curved is dependent on the manner in which the curve enters and leaves a sector.

Berthod and Maroy [4] describe an on-line character recognition system consisting of a base set of four topological features:-

a) a straight line element (T)

b) curves in the clockwise and anticlockwise directions (P), (M)

c) pen-lifts (L)

d) cusps (R)

The data points from the tablet are initially pre-processed to produce a series of vectors. These vectors are encoded into the feature set.

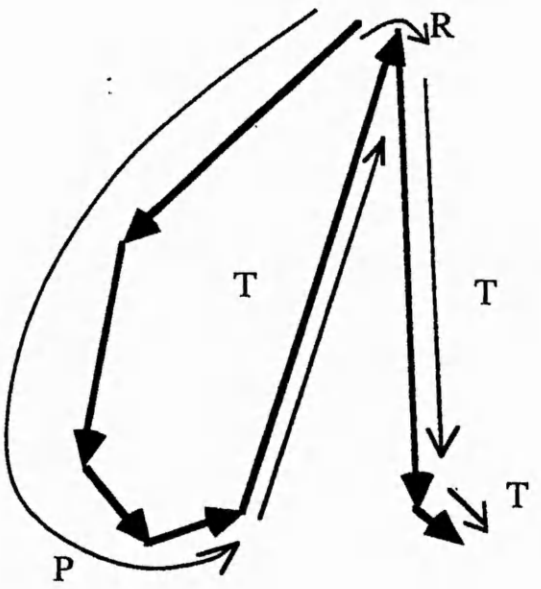


Figure 1.7 - Feature Encoding of character 'a'

If any ambiguities arise due to different characters having similar features, then some additional geometric relationships need to be added in order to make a positive decision.

A very similar technique is detailed by Guberman and Rozentsveig [25]. Again, the description of the letters is broken down into a number of standard elements. Two main types of elements are defined, elements with no intersection (arcs and straight line elements) and elements having self intersection (loops and cusps).

Tang, Tzeng and Hsu [47] describe a method which simply detects maximas and minimas in the x and y directions and use this information to recognise the numerals 0-9. This has been used as a basis for the X-Y algorithm detailed in chapter 4.

1.3.2. Cursive Word Analysis

This is potentially more difficult, whether attempting to recognise the word as a whole or attempting to segment into letters and processing the letters individually. One approach by Bozinovic and Srihari [48] segments the word by detection of the local minimas along its lower contour. Each segment is analysed in order to determine which zone(s) it resides within. Within each separate segment a number of features are searched for:-

- loops, classified by the region in which they occur (upper, mid or lower).
- large connected strokes in the upper and lower zones.
- number of peaks in the mid-region (one, two or three).
- curves and their direction in the mid-region.
- dots and dashes in the upper and mid-regions.

From this information an initial 'guess' is taken at the identity of each character in the word. However, a lexical look-up algorithm is required in order to select the most probable word from a dictionary base.

Ehrich and Koehler [40] perform a very similar analysis to the one previously described, but they extract the features with respect to the whole word. Again, significant weighting is placed on features with respect to the zone it is found in. The mid-zone is found to contain the most complicated stroke sequences.

Both the above techniques are heavily writer dependent.

Berthod and Ahyan [7] extend their theory on unconnected letters [4] to word level analysis. Their research showed that the shape of a character can change quite significantly from an isolated form to a cursive form. The factors effecting the character shape are primarily dependent on the characters immediately preceding and following. The set of primitives for the isolated character analysis is extended with the addition of both interection points and maximas and minimas in the vertical (or y) plane.

From these primitives a set of features is produced which is stated to cope with every type of curve path to be encountered in cursive script.

1.3.3. Observations

It is clear that in order for a topological feature based technique to work reliably and efficiently that:-

- all possible variants of a given letter in the letter set can be described by the same set of elements or, at most, by a small subgroup of such sets.
- an algorithm exists that detects these elements consistently.

The technique of topological feature extraction is the most popular basis for character recognition of all the papers surveyed. Some points to note on this method:-

1. None of the papers surveyed have given any indication to their suitability or not to operate in a real-time environment. A technique might recognise any character written by any writer, but if the amount of processing time and memory required are very high, the technology may not exist to realise the work in a viable product, or if it is possible, the cost may be so restrictive so as to make the product unmarketable.

2. In most instances, the basic algorithm will not uniquely define a specific character in the character set. A further level of processing is required to deal with ambiguities and in most instances this level of processing is more complex than the initial algorithm.
3. The method can be applied to both separate characters and cursive words with little or no change to the basic feature set.
4. In many instances the technique of feature extraction can rely heavily on the character shape (for example, detecting loops, curves and crossing points). This is usually very much dependent on the writer (one persons straight line is another persons curve). The majority of techniques are therefore heavily user dependent.
5. The cursive word analysis techniques tend to analyse the whole word (no definite character segmentation) and, as such, are heavily dependent on dictionary look-up. This can constrain the word set to be recognised.

1.4. Elastic Matching and Template Methods

The technique of elastic matching has been applied successfully in the areas of speech recognition, shape matching and signature verification. It is a technique which allows for the accurate comparison of two strings in which not only the contents, but also the order of the elements may vary. It gives a fast and reliable quantitative estimation of the degree of similarity between the two strings.

One of the earliest papers to consider elastic matching was written by Tappert [10]. The character 'features' are obtained directly from the raw character point information:-

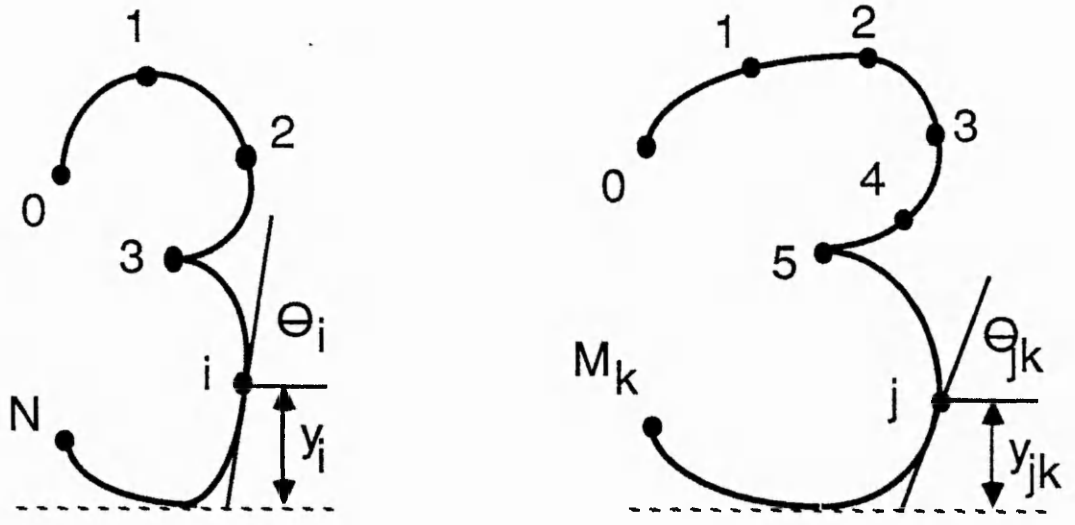


Figure 1.8 - Character Comparison

The points along the character curve are obtained at equal time intervals. The parameters selected being:-

1. a measure of the tangential angle to the curve along all the points making up that curve.
2. a measure of the corresponding vertical distances of these points from the baseline of the curve.

These two parameters were chosen because of their relative invariance with respect to character size and translation. From these parameters it is possible to construct a vector string where:-

$$S = V_0 , V_1 , V_2 , V_3 , \ldots\ldots V_N$$

$$\textit{where} \quad V_i = (\theta_i , y_i)$$

$$\text{and} \quad N = \textit{no. of points}, \quad \textit{Time}, T = \Delta t * N$$

Initially, a potential user must create their own set of prototype parameter sets. The parameter set for an unknown character is compared against the set of prototypes, and a match is said to be found with the prototype which, on comparison, yields the smallest overall distance of differences. This, however, makes the technique heavily user dependent.

Lu and Brodersen [45] designed and built a dedicated Dynamic Time Warping processor that was able to manage a template set of 500 reference symbols. They did not find it possible to handle the very high processing

overhead to run the algorithm in real time with no performance degradation without the DTW processor. In order to reduce template matching, a pre-analysis was performed to eliminate those templates which were definitely not similar to the unknown. This left only around 10 possible templates to match with. Any significantly larger number of templates would have effected the real time performance.

Elastic matching can be performed on any basic feature set, be it shapes, lengths, directions, angles or any feasible set of parameters. Szanser [13] makes a few points about the application of elastic matching to character recognition. He considered ways of reducing the template matching by making various assumptions, mainly:-

1. Ignoring upstrokes in characters, since it is the downstrokes which contain all the useful information.
2. Not to assume that all the features in a character breakdown are equiprobable.
3. Grouping sets of features to speed up the matching process.

Burr [35] describes a technique similar to Tapperts. He produced a stored set of 26 lower case reference vector arrays. An unknown vector array is compared to each of the reference arrays in turn, and a measure of similarity of shape is determined by a method of limited time-warp constraint. Figure 1.9 shows an example of matching between characters 'a' and 'd'.

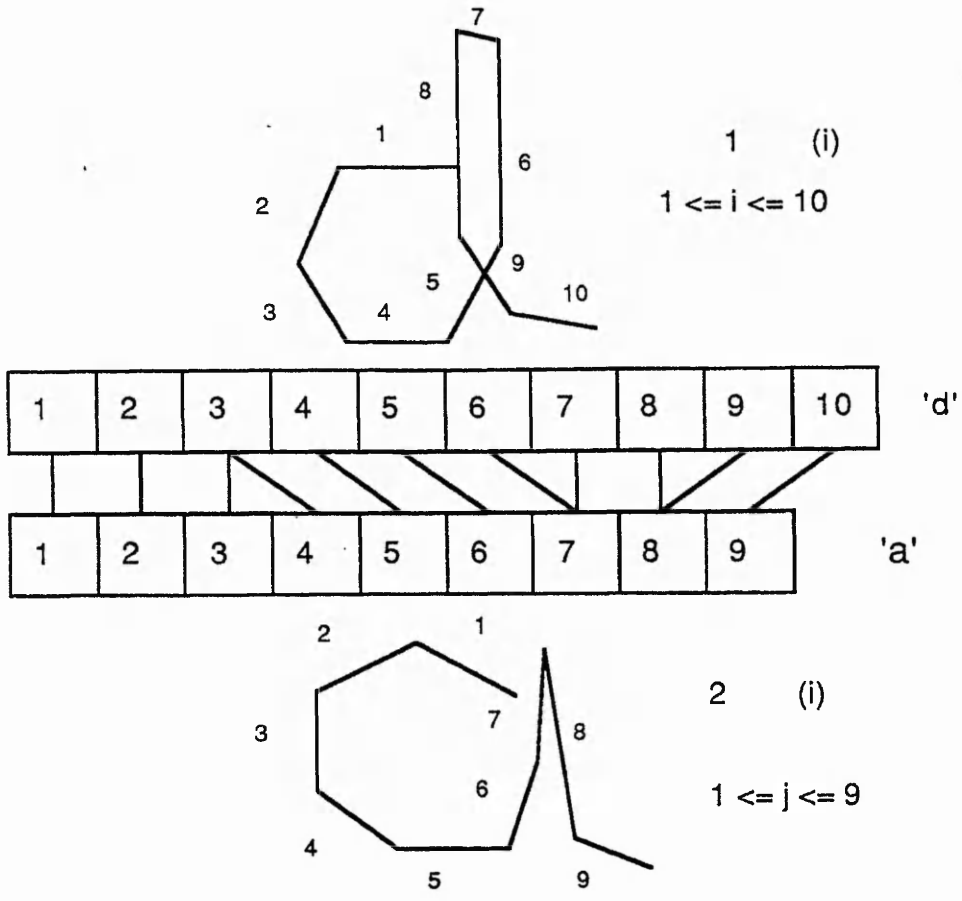


Figure 1.9 - Non-linear Stretching between curves 'a' and 'd'

An initial compression technique is used to reduce the number of samples in the unknown character to a value similar to those in the reference set. This reduces the computing time considerably. A training phase is required for each user, as with the method described by Tappert.

1.4.1. Observations

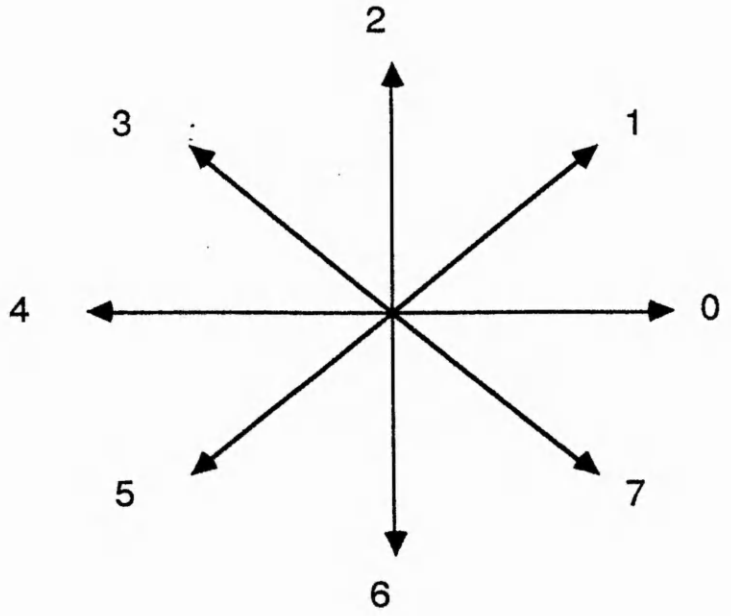
Elastic template matching is not a technique which has been originally designed with script recognition in mind. Its original application has been in the field of speech recognition. Therefore, its suitability for script recognition seems, as yet, still to be proved. Two serious drawbacks of this technique appear to be:-

1. Its limitation to being a trainable user dependent system.
2. Depending on the number of features in each feature set and the number of feature sets and the size of the character set, elastic matching can very easily become too processor dependent so as not to be feasible in real-time.

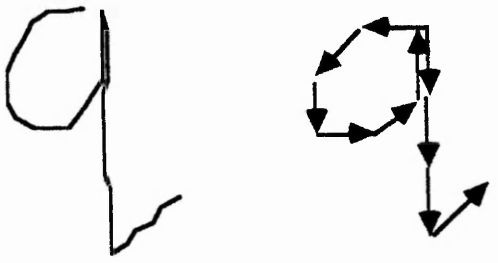
On the plus side, for a user dependent system, elastic matching provides a very reliable technique. It also requires no extra processing once having passed through the template matching process. ie. the matching process produces a definite result.

1.5. Vectoral Chain Coding Techniques

Chain coding is a means of approximating a curve by a series of straight lines. The straight lines are interconnected and follow the path of the curve so as to be a continuous approximation. The essence of the technique is that the length and direction of each line is restricted to one of a number of preset vectors, each vector being identified by a number, as in Figure 1.10 (a). Therefore, a character curve quantised into a chain of vectors may be expressed simply by a string of numbers as in Figure 1.10 (b).



(a) - Typical vector numbering scheme



4 5 6 0 1 2 6 6 6 6 1

(b) - Character curve quantisation

Figure 1.10 - Chain coding

Chain coding was introduced as early as 1960 by Herbert Freeman [81] and as a result this type of coding is often referred to as Freeman chain coding.

A large number of variations on the eight vector method have been used in the area of curve analysis and character recognition. Ikeda et al [22] used a vector set quantised into 24 alternatives for the recognition of Japanese characters. Powers [34] used an eight vector model as a basis for character recognition, however the octants were mapped such that the vectors

representing them are offset by a factor of 22.5° from the most common reference vector set shown in Figure 1.10(a). This, however, has the disadvantage that pen strokes in the important colinear and orthogonal directions (representing upstrokes, downstrokes and cross-strokes) are not explicitly represented, but can arbitrarily fall into directions 0 or 7 for right cross-strokes, 3 or 4 for left cross-strokes, 1 or 2 for upstrokes, and 5 or 6 for the downstrokes.

Berthod and Maroy [4,5] realised the importance of these colinear and orthogonal strokes and decided to segment the circle into 8 octants of unequal size. Octants in which upstrokes, downstrokes and cross-strokes should occur are made narrow in order that these strokes may be more easily identified.

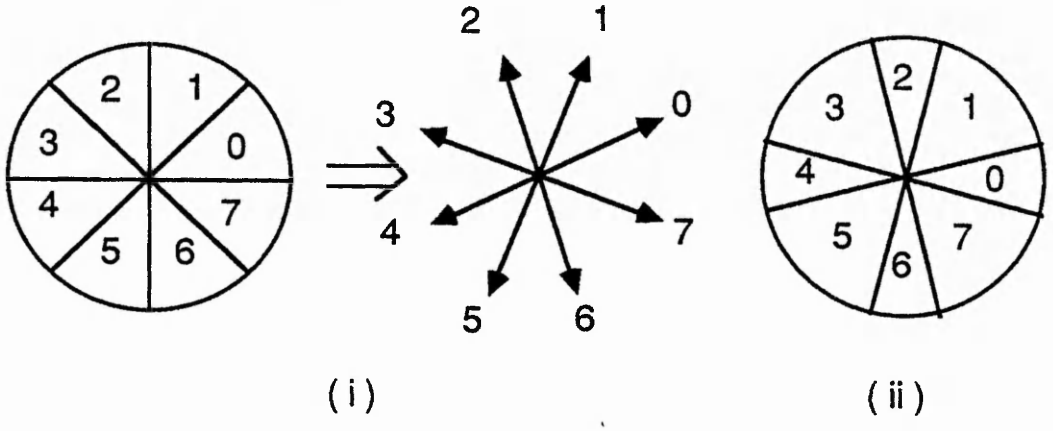


Figure 1.11 - Alternative Vector Direction Sequences

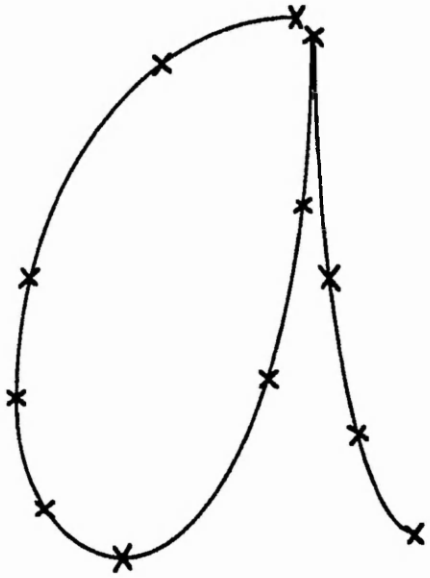
Once a curve has been encoded into the Freeman vector string, it is possible to perform a number of manipulations on it:-

- expansion
- contraction
- rotation
- path reversal (mirror imaging)

Powers [34] used Freeman encoding to vectorise character curves. However, he found that the chain code generated by a single user for a particular character can vary quite markedly. In order to overcome this problem, the chain code is processed into a number of arc and straight line sequences which is used as input to the recognition process.

Farag [11] analyses the vector string as a Markov chain. A vector set comprising eight vectors represents a Markov chain with eight states. If we consider the character 'a' shown below, a vector has a conditional dependence upon the preceding vector. For instance, a vector 6 is more likely to be followed by another 6, a 7, or a 5 than it is by a 4, 3, 2, 1, or 0. This is because there are more straight regions and gradual curves in a character than there are sharp angles or points of inflection. Each of the eight states of the Markov chain has a conditional probability, $P(y_i/y_{i-1})$, which is the probability of a particular chain code occurring, given the one before. The character 'a' will have probability equal to the product of its component vector probabilities, viz:-

$$Prob_a = P_4 \cdot P_{54} \cdot P_{65} \cdot P_{76} \cdot P_{07} \cdot P_{50} \cdot P_{55} \cdot P_{25} \cdot P_{62} \cdot P_{66} \cdot P_{76}$$



$$FR_a = 4.5.6.7.0.5.5.2.6.6.7$$

Figure 1.12 - Freeman Encoding of Character 'a'

Observations

In considering the use of chain coding as a means of representing natural handwritten input internally there are a number of arguments that weigh heavily in its favour:-

1. the grid size can be small so that the detailed deformations in a hand-drawn stroke may be captured.

2. there is a well defined set of elementary manipulations which are easy and fast to compute.
3. it can be very compact in terms of storage requirements, and its meaning easy to interpret.
4. there are a variety of simple and powerful techniques for analysing chain coded curves.

There is no other scheme for representing the Cartesian grid data from a graphics tablet. For script recognition, chain coding has been shown to be suitable for representing handwriting, and can also provide a certain amount of preprocessing.

2. TRANSDUCER REQUIREMENTS

2.1. Introduction

The role and nature of the data input device is very important to the realisation of a reliable and robust real-time data capture system for the input of dynamic hand-written script. This chapter analyses the various requirements for a suitable data capture system and pays particular attention to:-

(i) the writing surface.

(ii) the pen stylus

These aspects are particularly important for the capture of a users natural writing style. The idealised situation is where the user can treat the data input mechanism as they would do if they were writing with a pen or pencil onto a piece of paper or a pad of paper on some flat surface such as a table or desk. The pen stylus is particularly important, as it determines how easily the user can enter the script. The more familiar the user is with the pen stylus, the more representative of the users writing style will be the produced output. The ideal case will be to allow the writer to use their own personal writing implement. Also, with respect to a natural environment, it would be advantageous to allow the user to orient the tablet writing surface to suit their particular writing posture. Such a prerequisite would mean that the tablet be both light- weight and reasonably small, and that its orientation is not limited by such things as cables or peripheral hardware.

Apart from the more specific user operation parameters it should be made clear that the complete unit must conform to the normal operating specifications for such electronic devices. The major considerations are listed below:-

(i) extremes in temperatures (typical figures 40^0C operating, 55^0C stored).

(ii) humidity (90% non-condensing).

(iii) shock

(iv) vibration

(v) altitude

(vi) electrostatic discharge (such as the electrical discharges accumulated on people).

(vii) electromagnetic susceptibility (reductions in performance due to radiation from nearby equipment).

2.2. Study of Current Input Device Technology

Before producing a more detailed specification of the input device it was decided to undertake a study of devices currently available on the market. The following types of device were considered to warrant investigation:-

(i) tablet digitiser pads.

(ii) touch sensitive screens and overlays.

(iii) light pens.

(iv) analogue devices (eg. mouse, joystick).

2.2.1. Tablet Digitisers

The following technologies have been used for tablet operation:-

1. electromagnetic/ magnetorestrictive
2. electrostatic
3. pressure pad
4. quantised magnetic wave
5. sonic
6. electronic paper

2.2.1.1. Electromagnetic/ Magnetorestrictive

This is the most widely adopted technology on the market at present. The surface of the tablet is not dependent on the technique and so can be made from any hard wearing, durable non-metallic material.

Operation is by means of magnetic coupling between the pen stylus and the active surface area of the tablet. The pen contains a coil powered by an a.c. (120 KHz) source which can be regarded as the primary of an air cored transformer. The secondary being the conductors in the tablet surface. (These are usually wires or thin metal strips arranged in a grid underneath the tablet surface). The stylus induces a current in the conductors which produces a signal voltage across the surface output lines which is proportional to the distance between the conductors at the edge of the grid and the pen. The outputs at the conductors is scanned by the output control circuits giving the pen location. Refer to Figure 2.1.

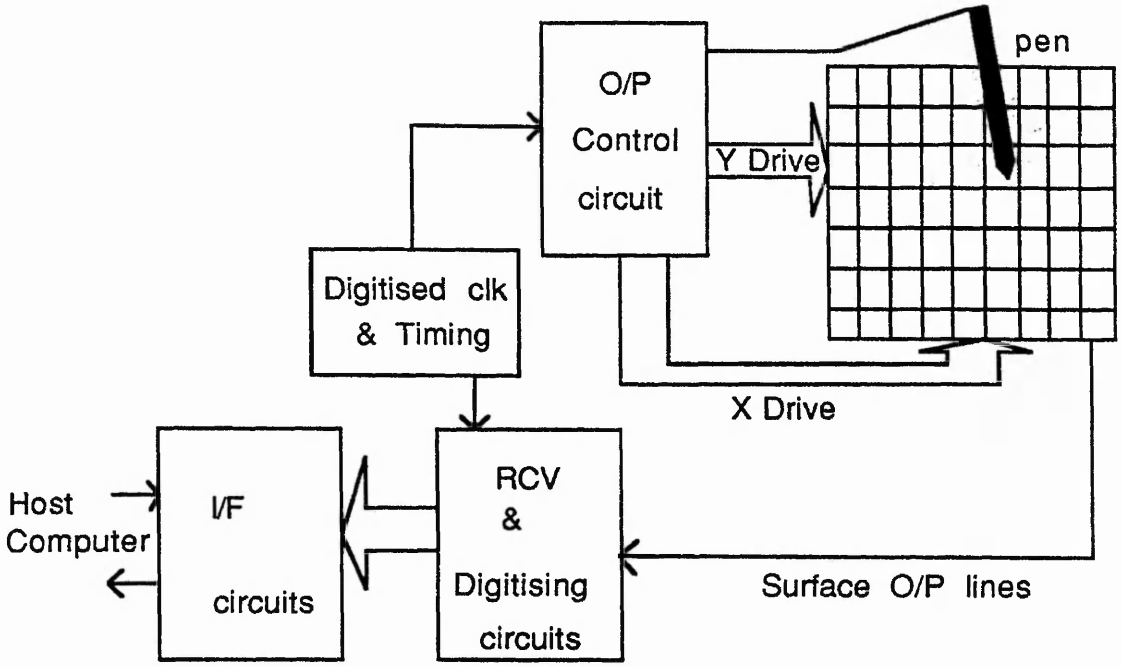


Figure 2.1 - Electromagnetic Tablet

A magnetorestrictive device reduces positional errors caused by conductive drawing materials, moisture etc.

Tablet resolution ranges from 100 to 1000 points per inch. However, tablet resolution does not have much meaning if its effect is negated by a poor sampling rate, Tappert & Kim [44]. Similarly, the benefit of such high resolutions must be tempered against the degree of accuracy that can be achieved when using a pen with a tip thickness of 1mm or so.

The technology can only detect the interference between the pen coil and the metal grid, and therefore not specifically the tablet surface. Therefore a micro-switch is usually built into the stylus tip in order to indicate that the pen is being pressed onto the tablet surface.

2.2.1.2. Electrostatic

The tablet surface is a conductive plate and the stylus picks up the voltage from the plate. The distance from a fixed reference can be determined from the voltage picked up knowing the voltage gradient. The plate must have a uniform voltage gradient and the stylus must be in contact with the tablet surface, which precludes the tracing operation. However, such devices are more expensive than an electromagnetic device of similar performance.

Tablet resolutions similar to electromagnetic devices are currently available.

2.2.1.3. Pressure Pad

Typically, the pad consists of two electrodes fabricated by flexible print wiring on a base plastic film separated by a conductive rubber sheet. When the pad is pressed onto, the rubber sheet becomes conductive, allowing a current to flow from one set of conductors (constant source current) to the other, this being picked up by operational amplifiers which determine the x-y co-ordinates.

Tablet resolution is not as good as for the previous two types of tablet, maximum figures up to 300 points per inch.

One advantage of this technology is that most types of writing implement may be used with this type of device. The major problem with this type of device is that the nature of the surface is not particularly durable and can easily break down with constant usage. These devices are mostly used for a pointing operation (eg. CAD design) and as such cannot differentiate between the hand pressure and pen pressure, both produced as a writer rests their hand on the tablet surface when writing.

2.2.1.4. Quantised Magnetic Wave

A relatively new principle. Four magnetic waves are set up in a coarse array of orthogonal conductors. There mutual interference defines the precise location of the stylus.

Resolution is not yet particularly good, up to 200 points per inch.

2.2.1.5. Sonic

No tablet as such is needed in this case. Two orthogonal metal strips are placed around the writing area. Along each of these are mounted microphones. These pick up the sound or 'sparks' generated when the stylus comes into contact with the active surface enclosed by the two strips. The delay between emission and reception of the sound can be computed to a distance measure.

So far this technique cannot produce a reasonable resolution, up to 100 points per inch. It is also particularly sensitive to changes in the ambient room temperature.

A similar approach uses sonar, where high frequency pulses are reflected by the stylus. Again, low resolution devices only are available.

2.2.2. Touch screens and Overlays

There are two types of overlay device which can be mounted on a screen such as a CRT or plasma panel. These are:-

(i) light emitter - receiver

(ii) switch matrix devices

2.2.2.1. Light Emitting Devices

These consist of a line of light emitters (infra-red LED's), one on each of the x-y planes, with photo detectors opposite them. When a stylus or finger cuts the light path at perpendicular points, the position can be determined. Resolution is very low (up to 0.25 inch at best). The cost of LED's and detectors also make this device relatively expensive.

2.2.2.2. Switch Matrix Devices

These employ two main techniques:-

1. two crossed conductive grids each containing thin film parallel conductors form the switching matrix. The x-axis is located on the convex surface of a CRT and the y-axis is located on the concave side of a flexible polyester membrane. The surfaces are separated by an air gap. Resolutions are up to 256x256 points for a 14" screen.
2. a thin clear conductive film on a glass shield is placed over the surface of the display. The shield is covered by a mylar layer with a resistive coating. When the upper layer is touched (with a pen or finger) the two layers make contact over a micro-inch separation and its position is determined by the voltage drop across the resistive coatings with the x-y co-ordinates appearing as analogue voltages which are interpreted by the systems decoder which converts them into digital signals. Refer to Figure 2.2.

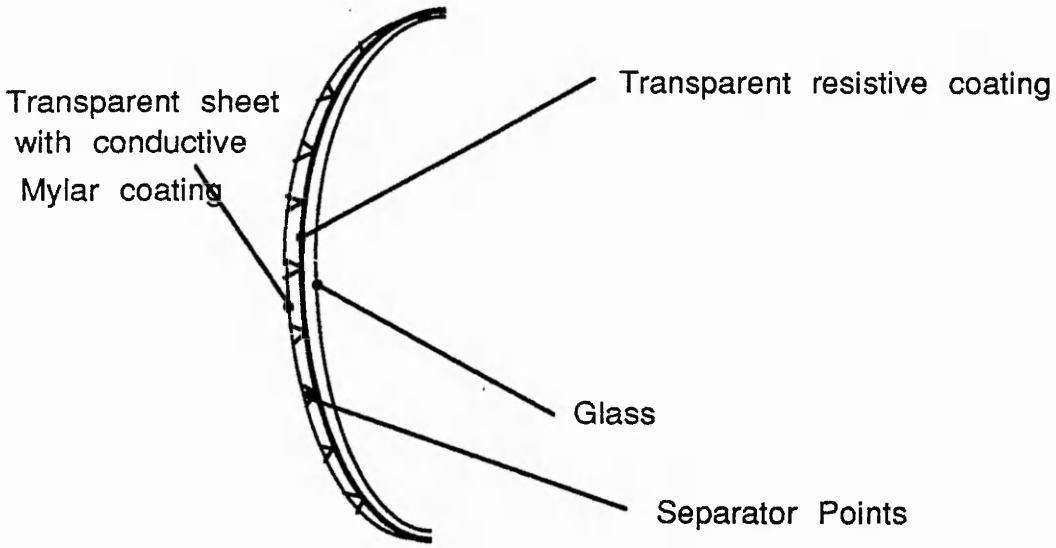


Figure 2.2 - Touch Sensitive Display

The pressure required in order to make contact between the two layers is dependent on the space between the separator points. The fewer the number of points, the less the force required to make contact. Therefore, for use with a pen or pencil, adequate separator points would be required to ensure that hand pressure does not cause contact to be made, while still ensuring a clear screen.

Resolutions up to 400 points per inch can be achieved, dependent on the d/a converter.

2.2.3. Light Pens

This method of data capture was used in some early script recognition techniques. There are two basic types of light pen, using either a photodiode or photomultiplier. The photodiode has a poor response time and, as such is, suitable for slow raster scan displays only. The photomultiplier has a response time typically 0.5 microseconds. Point positioning is relatively simple, but pen tracking is fairly complex and tracking of rapid pen movement is not possible.

Therefore a good resolution can be obtained from a high resolution screen, up to 500-1000 points per inch, but a low sampling rate makes them unsuitable for tracking handwriting movement.

2.2.4. Analogue Devices

These devices include cursor buttons, joysticks and mouse tracker balls, employing switch, potentiometer an optical systems.

They can be used as a faster and more manoeuvrable alternative keyboard cursor control, but are not suitable for handwritten input.

2.2.5. Electronic Paper

At this stage mention must be made of 'electronic paper'. This concept is the integration of digitiser and display technology. In effect, it is the perfect medium for the script recognition application. It provides a portable I/O unit that eliminates the need for a disjoint display. Hence, the user can concentrate his thoughts in the one area instead of spending effort switching between the writing and reading area.

Since the beginning of 1988 'electronic paper' products have begun to appear on the market place, mainly from Japan. So far the size of the active area on such devices has been A5. Two of the most promising products in terms of parameter requirements (described in more detail later on in the chapter) are:-

(i) The Photron FIOS-6440 [89]. It has an electromagnetic digitiser with an active area of 217mm x 140mm. A resolution of 0.1 mm (250 points per inch) and a digitising rate of up to 150 points per second. The display is LCD with a pixel size of 0.3mm x 0.3mm. Therefore the script displayed on the screen cannot faithfully reproduce the samples from the digitiser.

(ii) The WH-515 Sensor/LCD unit [90]. This has a cordless pen. The active area and resolution are the same as for the Photron device. Pixel size is similar at 0.33mm x 0.33mm. This product can also be purchased with application software, including a line-drawing mode which permits the drawing of lines, boxes and circles. Line thickness can be selected and lines and shapes can also be erased.

2.3. Survey Outcome

As a result of the initial survey two types of device were considered for further evaluation (at this early stage no 'electronic paper' product was available). A high resolution electromagnetic tablet, the Numonics 2205, costing $900 and a touchscreen device, the Elographics touchpad, costing around $1600. The tablet has a resolution of 1000 points per inch and the touchpad a resolution of 400 points per inch.

2.4. Specifications

The requirements for the tablet and stylus can be broken down into two main categories:-

(a) the quantifiable factors detailing the technical specifications of the tablet and stylus.

(b) the ergonomic requirement relating to the ease of usage.

2.4.1. Technical Specifications

The technical specifications are broken down as follows:-

2.4.1.1. Sampling Rate

The sampling rate is the rate at which the pen position can be determined by the tablet and that position transmitted to the host. The number of times this is done per second is known as the sampling rate. The vast majority of tablets available on the market today are not specifically oriented towards the accurate capture of dynamic handwritten input. CAD/CAM applications, the major user of graphics tablets, do not require large sampling rates, as they maily use the tablet as a pointing device. Therefore, many tablets have a simple serial RS232 ASCII link, maximum transfer rate 9600 or 19200 baud. The limiting factor in the sampling rate is now the number of bytes transmitted per x-y co-ordinate pair. A typical co-ordinate pair format might be:-

< STAT> < SP> < XXXXX> < SP> < YYYYY> < CR> < LF>

For our chosen tablet, the Numonics 2200, the precise serial data format per point is fifteen bytes of data, eleven bits long (1 start bit, 8 data bits and 2 stop bits) is 165 bits per x-y pair. Therefore, over a 19200 bps link, a maximum of 116 samples per second can only be obtained. A greater point status transmission rate can be obtained by sending the serial data in packed binary format, which compresses the number of bytes required to around one third of that required for simple ASCII. Another option is to transmit the data over an 8-bit parallel Centronics link.

In the final instance, the data format is not important as long as:-

(i) the link speed is not so great that the processor is required to spend too great a proportion of its time reading its input buffer and not having enough time left to process said input.

(ii) it is of such a format that it can easily be decoded to give the pen positional information, again, without substantial processor loading.

The paper 'A Sketchphone System' [82], has analysed the handwriting speeds of Japanese writers, deducing that they are dependent on character size, character types and also the type of stylus used. This could be likened to writing sentences in upper case English, since each character is usually made up of a number of separate strokes and each character is separated from its neighbours. The average writing speed determined was between 100-200 mm/second with an instantaneous speed potential of greater than 1000 mm/second.

In order to determine what was an adequate sampling rate, it was necessary to identify a 'worst case' writer. This was found to be someone who, for a given sampling rate, produces the smallest number of x-y co-ordinate pairs for a specified test sentence. This is found to be someone who writes both very quickly and writes very small letters. A study of pen writing speeds analysed from a small set of 20 writers, writing two sentences in both connected and non-connected script indicated an average writing speed of 50-75 mm/second, and an instantaneous speed potential of up to 500 mm/second. The average time taken to write a character was 0.42 seconds, while for very slow writers this figure was observed in excess of one second. Of course, these figures are very much dependent on the type of character or stroke being written. Very simple strokes or characters (eg. l,i,j,c) will not take as long to form as the more complicated characters (eg. m,g,k,w). By reconstructing the character shapes of a number of such simple and complicated characters, sampled at differing line rates it was possible to determine a sampling rate below which important character information might be lost.

Figure 2.3 below shows how the shape of the character 'k' is affected by the sampling rate.

60 x,y pairs per second

30 x,y pairs per second

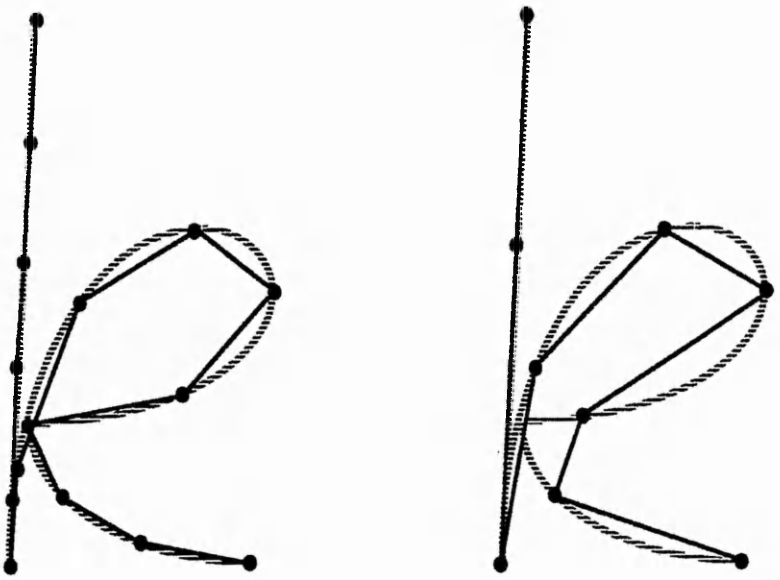


Figure 2.3 - Effect of sampling rate on character shape

Subsequently, for our 20 writer sample, a data rate of 80 co-ordinate pairs per second was determined as a figure below which it might be possible to seriously affect the shape of some characters.

2.4.1.2. Resolution and Accuracy

Resolution is a measure of the minimum distance separation on the tablet which will register as two separate points. Accuracy takes into account repeatability, i.e. the difference in successive readings obtained when the pen is placed down on the same point on the tablet surface. This measure is usually a lower value than the measure of resolution.

In this particular application the accuracy of the tablet is far less important than the resolution. The resolution of the tablet must be such that it can faithfully reproduce the character shapes for writers who form particularly small letters. We preclude all script so small that its identity cannot be determined by the human eye. Therefore a lower limit for script size is that having a mid-zone width of no less than 1mm. Refer to Figure 2.4 :-

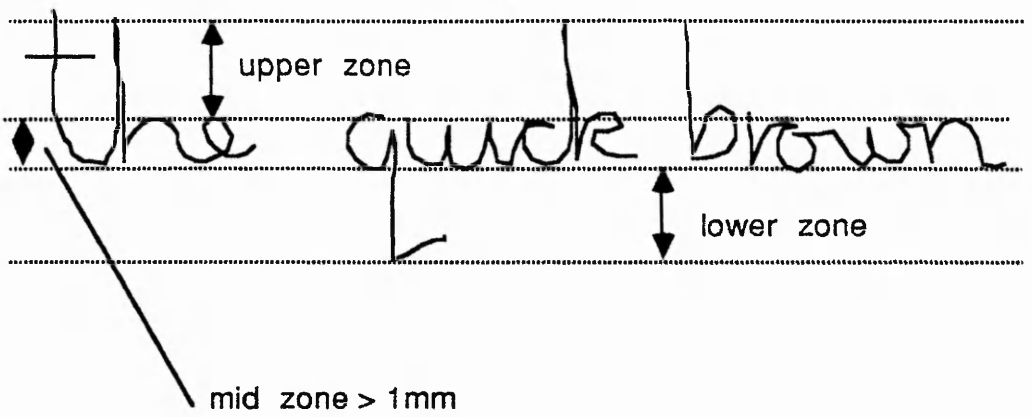


Figure 2.4 - Minimum Character Sizes

A resolution of 20 divisions/mm (500 divisions/inch) gives an adequate clarification of the shape of mid-bound characters, most important being the arcs and loops in this region, however the Elographics tablet with a resolution of 10 divisions/mm does begin to show some shape deterioration.

2.4.2. Tablet Requirement Considerations

The are certain features of the data input mechanism which must be carefully considered in order to allow a writer to enter their handwriting data into the computer as naturally as possible. The ultimate aim is 'electronic paper' and technology is developing rapidly so that in the next few years an integrated screen and tablet can emulate the process of the writer working on a sheet of paper. At present the set up of separate tablet and display is quite disconcerting for an untrained user and annoying for a familiar user. Recently, a great deal of research has been directed towards the user interface and one aspect of this is the realisation of 'electronic paper'. Tappert et al [83] suggest how powerful a handwriting system could be when the writing is directly above the display. They have developed a prototype 'electronic paper' system but have exposed problems due to parallax between the tablet surface and the display surface. Also, because they have an integral stylus, the stylus shape, its tip characteristics and the pen-down sensing mechanism have all caused problems similar to those experienced in evaluating electromagnetic tablets.

Until a suitable 'electronic paper' system can be proven, a number of guidelines are suggested in order to facilitate ease of use at the human interface. These have been noted from personal tablet usage and from feedback obtained from various people asked to write test sentences onto the tablet.

2.4.2.1. Tablet Surface Material

The most important features of the tablet is that it be hardwearing and durable. Most of the pressure type tablets on the market at present have a very limited lifetime and the components directly below the writing surface do tend to be very susceptible to breakdown, even after a small amount of usage. However, they do have an advantage over the electromagnetic type of tablets in that the surface properties are very similar to those of a pad of paper, whereas the electromagnetic tablets have a hard, unreceptive surface (particularly for writing) and in some instances they have a coarse surface, making writing very difficult, since this tends to cause the writer to produce a very angular and unnatural style of writing and these angularities will cause the recognition algorithms great difficulties.

2.4.2.2. Stylus Considerations

A pressure type tablet has the benefit of allowing the user to write into the data capture system using whatever type of writing device they prefer to use. This ensures that the writing being captured is the users normal style. Tablets which have an integral stylus are not normally suitable for capturing handwriting because the pens are bulky and awkward to use. They are also limiting by their nature of attachment to the tablet. The reason for this is that, to date, tablet devices have predominantly been designed as pointing devices or simple graphical input devices and not as a text entry device, and as such, they are not ergonomically suited as a means of inputting handwriting information into the computer.

2.4.2.3. Tablet Size and Active Area

Typically, office documents are produced on A4 size paper. The active area of the tablet should be such as to encompass an A4 size area (11.7" x 8.3" or 300mm x 210mm). The majority of tablet manufacturers today produce tablets with active areas of 12" x 12" as a standard part of their range. This area is ideally suited to also allow a part of the tablet area to be configured for command mode operations, for example, switch to script/sketch mode, start document creation, end document creation, clear display and so on. In addition, a region around the active area would also be very useful. This would allow anyone using the device a comfortable "dead

zone", at the bottom edge of the device and the left or right hand side of the active area, would allow someone to comfortably write on the tablet without having their hand drop off the tablet edge. This is particularly of increasing importance with the height of the tablet surface above the normal user working area. Assuming a tablet height such that a dead zone is required, a plan view of the tablet would be as shown below:-

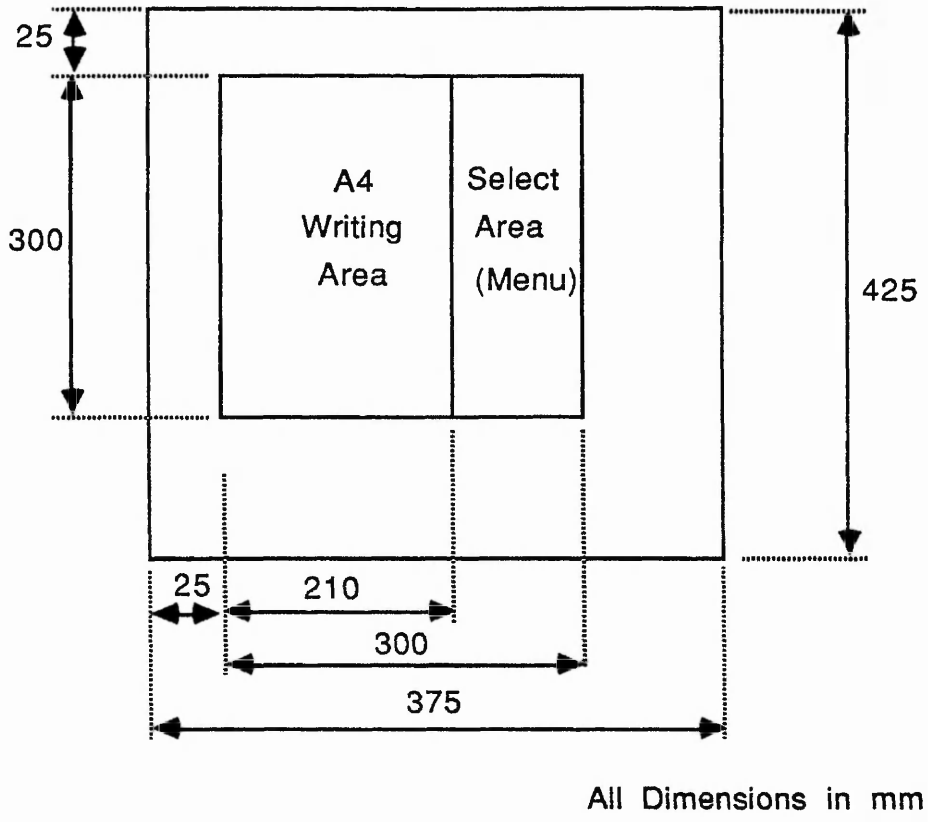


Figure 2.5 - Tablet Area Dimensions

2.4.2.4. Hard-copy Considerations

If the user is writing onto a piece of paper on the tablet surface, it would be particularly necessary to keep the paper static on its surface during the course of the writing session, since any dislocation of the paper would result in the invalidation of further text input with respect to that already produced and located. Therefore, some kind of restraint (perhaps as for a clip-board) would be necessary as an integral part of the tablet.

3. PREPROCESSING OF THE RAW DATA INPUT

3.1. Introduction

As a result of the state-of-the-art review undertaken it was noted that a large number of authors performed 'preprocessing' of the raw data received from the data tablet. This could mean anything from filtering out unwanted points, to changing the character shape and/or altering the character size. In particular, it is a 'normalisation' of the character shape. This was found to involve one or more of the following:-

(i) Data thinning or angular variation analysis, whereby the number of data points received for any one character are filtered in order that a roughly similar number of co-ordinates received for a particular character are processed irrespective of the speed of the writer.

(ii) Angular variation analysis or curve smoothing, performed in most instances to filter out the 'jagged' appearance of a character introduced by a poor resolution tablet.

(iii) Slant analysis. A method for normalising character shapes by either shearing or rotating characters that exhibit left or right slant in order that they are similar in feature to the same character written without slant.

(iv) Size normalisation. All written characters are either enlarged or reduced to provide a 'standard' data block as input to the feature extraction algorithms.

These techniques feature strongly in many papers in order to reduce the complexity of the succeeding feature extraction and recognition algorithms. For this reason it was felt helpful to investigate these techniques and gain some familiarisation in case they may prove necessary or useful at some later stage.

Some preprocessing techniques perform analysis necessary to remove certain adverse features which may confuse the following encoding algorithm. In general the preprocessing was performed for two reasons:-

1. To provide some degree of uniformity in the amount of data input to the recognition algorithms irrespective of the type of tablet capturing the data. Each tablet has slightly different attributes (see Chapter 2, Technical Specifications section). We surveyed a total of 15 digitising tablets and purchased four, the Numonics and Elographics previously mentioned and also the Calcomp 2000 data tablet and the Penpad device supplied by Pencept Inc. in the USA.

2. As a character standardisation mechanism. Many characters exhibit features related to the user writing style as well as character specific features. In some instances the detection and neutralisation of these features after the character encoding can prove to be a major task. Removal of the user dependent features limits the range of alternatives

quite markedly, more so as the user base is expanded.

3.2. Background

The main reason for preprocessing is the need to normalise unconstrained hand-writing. Unconstrained in terms of size, pen speed, character formation style. Unconstrained writing will inevitably produce a large variation in the input data collected for different users. These features can be classed as being related to the the two preprocessing functions mentioned above. Case (1):-

- different amounts of input data will be captured during the construction of characters by different users
- different users produce characters with a wide range of sizes

Case (2) is mainly as a result of:-

- variation in character shape. Usually people who write characters very quickly produce much more angular characters than people who write more slowly.
- character slant. The most common feature of user writing style.

Several techniques for data thinning of the tablet co-ordinate points have been described, generally by means of a simple input filter. However, Brown and Ganapathy [19] then performed interpolation on the data points in order to generate a stream of equidistant points. The other method for data thinning is by angular variation analysis, as used by M. Berthod and S. Ahyan [7]. Points may be removed from the character curve if the angular variation of the curve is small. Therefore generally far fewer points are required to describe a straight line section of the curve than to describe a loop or cusp.

Curve smoothing is found to be a necessary preprocessing step for a number of real time algorithms. Burr [43] performs curve smoothing by initially performing a sine fitting algorithm to the quantised data. The reconstructed curve then is resampled at a higher data rate. Another technique is performed by some authors, Burr [43] and Brown and Ganapathy [56], which is to perform some form of normalisation on the character size. This usually takes the form of character size translation into a specific character box area, required by the encoding and recognition algorithms.

Consideration of character slant receives mixed attention. Some authors simply state that heavily slanted characters cannot be processed by the algorithm. D. Burr [43], Brown and Ganapathy [56] and Higgins and Whitrow [53] describe techniques for character slant detection and removal. It is interesting that the consideration of character slant and its removal has only recently been addressed, basically with the move towards the more unconstrained user input. In the very comprehensive state of the art survey into the recognition of handwritten characters by C.Suen, M. Berthod and S. Mori [42] there is no explicit reference to character slant, only the fact that

distortion and style variations are produced by the writer and the fact the all user independent techniques are highly sensitive to these variations.

3.3. Techniques Evaluated

3.3.1. Data Filtering

Raw co-ordinate data filtering is by far the most common preprocessing technique used. The main reason for this is the wide variation in the amount of data during the quantisation of characters by the data tablet. The graphics tablet must be capable of faithfully encoding characters which have been written either very quickly and/or very small.(See Chapter 2). However, in order to ensure sufficient data for the worst case (or fastest) writer, we obtain the adverse result of collecting much more data than we required from a very slow writer. In most recognition techniques the recognition time per character or word is roughly proportional to the amount of data input to the encoding algorithm. In a data set of around 100 writers we noted the slowest writer produced 2.6 times as many co-ordinate points in the production of two test sentences than did the fastest writer.

It was not the intention to use the co-ordinate data filter in an attempt to standardise the number of points processed per character, since we required to retain the raw character shapes from every type of writer (fast and slow). We did not wish to remove shape information from characters which could be of use further along the processing, in identifying the character.

It was found, however, that some writers would pause with the pen resting on the paper during the formation of a word or character. This lead to the capture of unwanted data, in the form of noise and not at all related to the character shape. The data filter was therefore implemented so as not to remove character shape information.

A number of techniques process the input co-ordinates in such a way so as to produce co-ordinate pairs at roughly equal distances along the character curve. However, it was decided at the early analysis stage that this process would discard timing information, which might be of later use. Therefore we decided to adopt the very simplistic technique of a simple data filter. A lower threshold value was set . If we consider a series of encoded points along the character curve:-

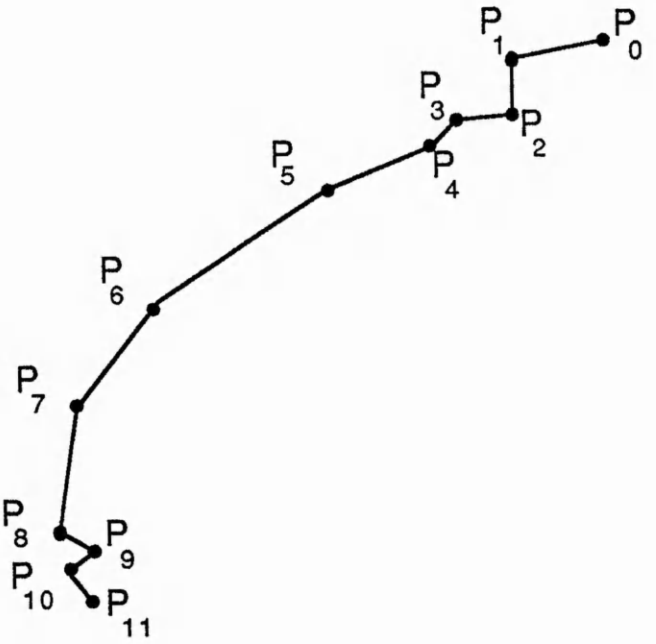


Figure 3.1 - Data Thinning

Setting our minimum distance threshold, d_t,

if $d_{(P_0-P_1)} > d_t$
    accept P_1
else
    discard P_1

If point P_1 is accepted, we restart from P_1 to determine whether we should accept point P_2.

However, if point P_1 is discarded, we remain at point P_0 in order to determine whether we should accept point P_2.

This process continues over the entire length of the character curve. The minimum threshold distance is a fixed value. Therefore a large character 'a' written at the same speed as a small character 'a' by the same writer should produce x times more points, where:-

$$x = \frac{D_{a_L}}{D_{a_S}}$$

where D_{a_L} = *total travel of large a*

and D_{a_S} = *total travel of small a*

Great care must be taken, therefore, when determining a lower threshold value, to ensure that no important point data is removed during data filtering. This means setting a smallest character size which may be written and subsequently recognised as being a character or part character and not a dot. This is usually limited by the resolution of the tablet.

It was decided not to attempt to filter out too much of the data produced while the pen was in motion. The main source of unwanted data arose as a result of the user pausing after bringing the pen down onto the paper, before writing the character or pausing with the pen on the tablet surface on the completion of a character. Character pauses in a string of x-y co-ordinates were quite easy to detect. Tablet accuracy ensures that it will not result in the production of a sequential string of identical spatial points. However, it will result in the production of a string of points which vary by a very small distance. The minimum diameter character size allowed was set at 2*mm*. Below 2*mm* it is difficult to read individual characters comfortably. Analysis of pause periods showed that they could last up to 0.5 seconds, which, at a sampling rate of 80 points per second, is the capture of 40 redundant data points. For an average of 19 data points per character, this is not an insignificant amount of redundant data.

Initially it was decided to retain the timing information supplied by the tablet by only removing the redundant data caused as a result of user hesitation. This ensures that the character shape is not degraded further. All data produced as the pen is in motion along the curve was retained by selecting a threshold of 0.25*mm* for the Numonics tablet. This value is proportional to the tablet resolution. This ensured that points would only be removed if the pen was stationary. At this stage it is of the utmost importance that no possibly useful information is discarded before it is analysed. It was felt that the concentration of points along a character curve may provide pen speed information which could be attributed to certain character features. For example, a sharp point of inflection is bounded by a large reduction in pen speed. Also, upstrokes and downstrokes also exhibit the greatest pen speeds along the character curve.

3.3.2. Angular Variation

Angular variation analysis may be used as an alternative to data thinning. By examining the angular variation between successive straight lines produced as a result of connecting points in sequence it is possible to remove 'redundant' points. If the angular difference between two successive lines falls below a threshold angle (θ_T) the mid point of the three can be removed, as it contributes little or no information to the overall character shape.

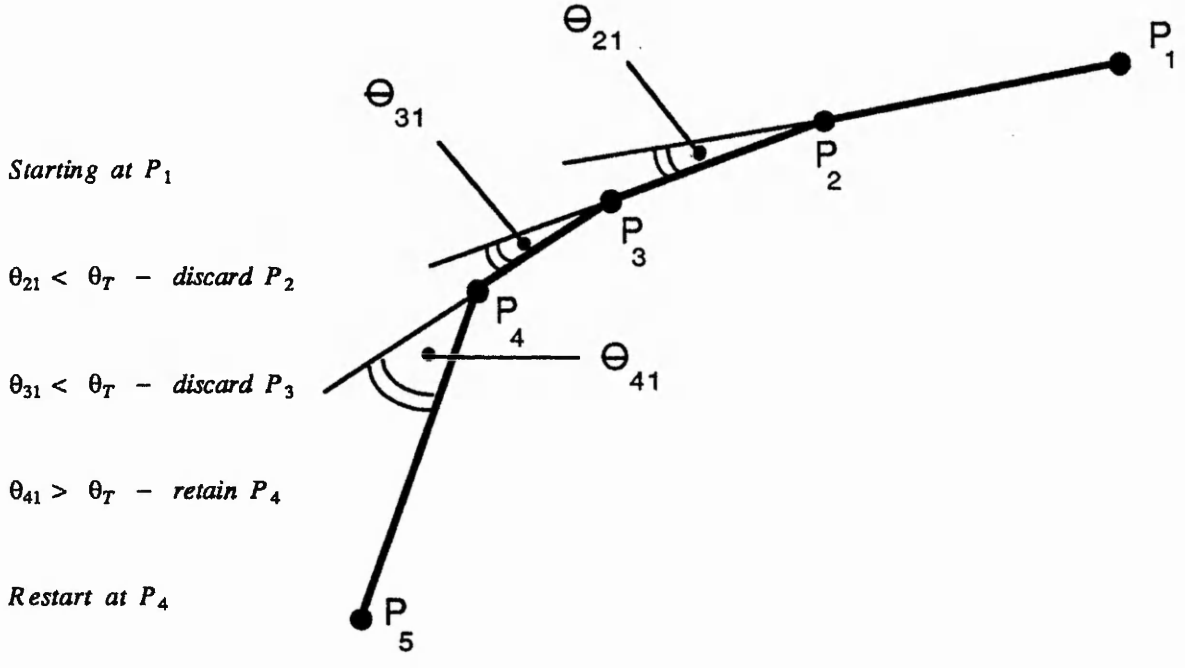


Figure 3.2 - Angular Variation Analysis

An example of the effect on character shape for varying degrees of angular variation analysis can be seen in Figure 3.3. The original character 'a' is composed of 19 co-ordinate points. Seven degrees of data thinning are shown for threshold angles of 10°, 20°, 30°, 40°, 50°, 60° and 70°. In the final case, at $\theta_T = 70°$, only 7 co-ordinate points remain, a reduction to only 36.8% of the original data size. At $\theta_T = 70°$ the character still resembles the letter 'a' and although it has become severely angular in shape, it cannot be confused by any other character in the lower case alphabet.

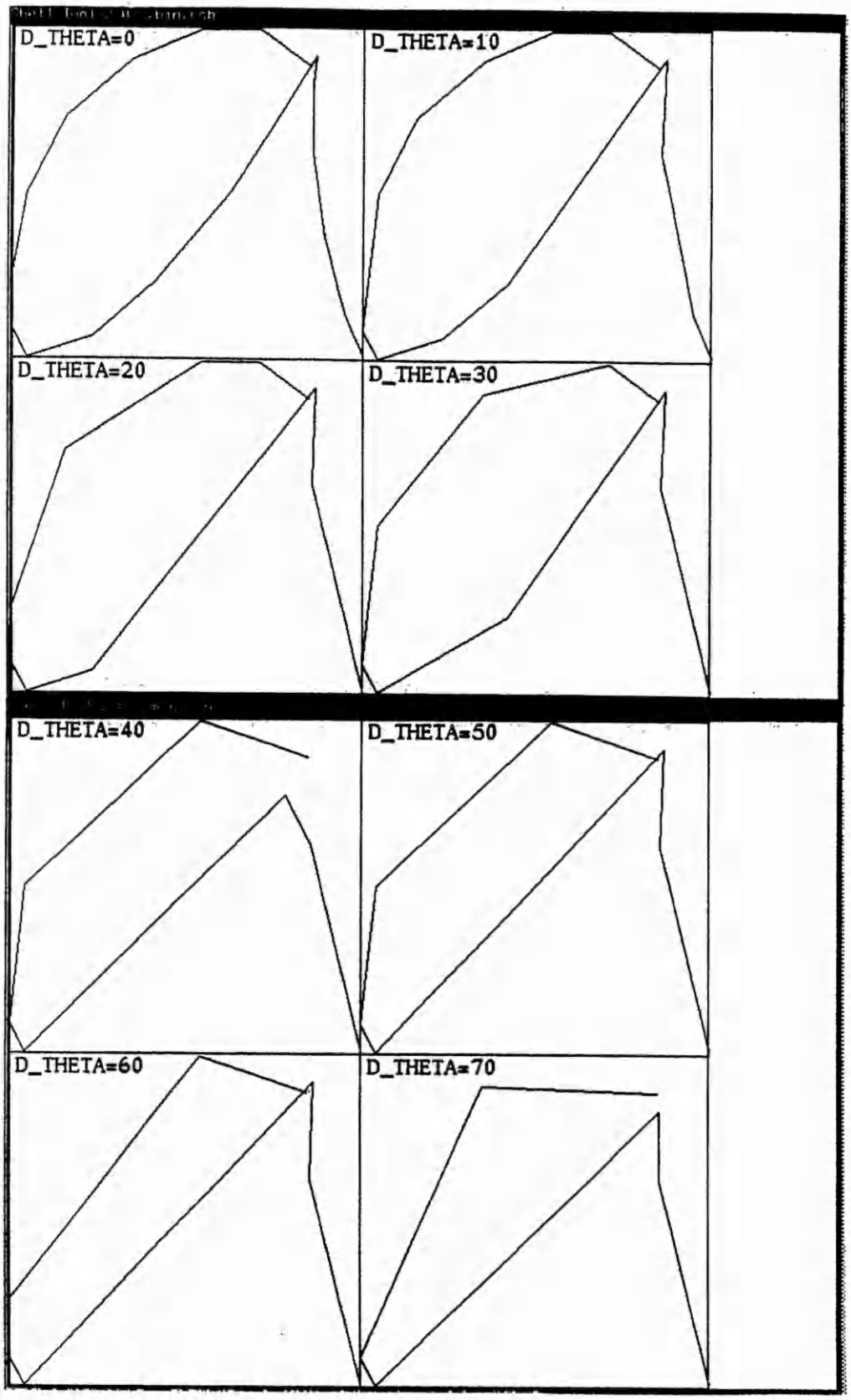


Figure 3.3 - Degrees of Thinning by Angular Variation Analysis

At $\theta_T = 40°$ the shape of the letter 'a' begins to deteriorate, in particular the shape of the anti-clockwise arc is lost. However, at $\theta_T = 30°$ the shape is still retained with only 10 data points, 52.6% of the original data.

One particularly useful feature highlighted by angular variation analysis is the identification of upstrokes/downstrokes in a character, as long as the stroke is a reasonably straight line.

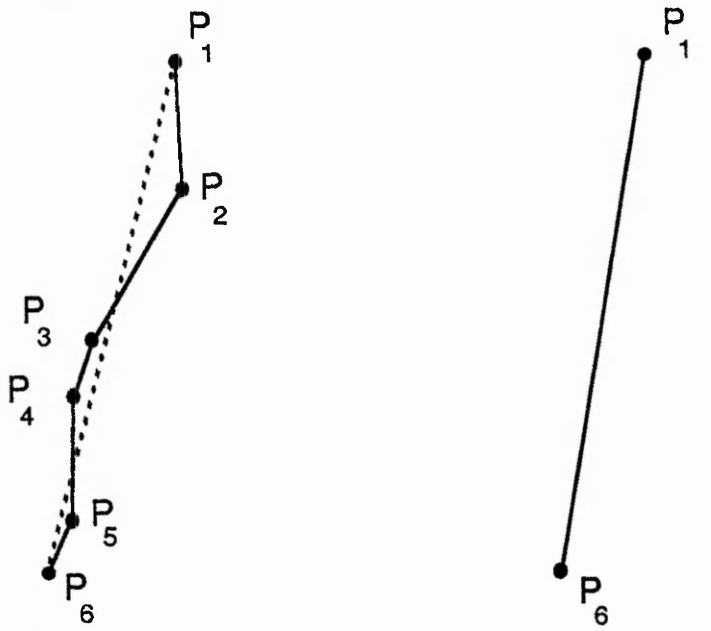


Figure 3.4 - Upstroke/downstroke Detection

This feature is highlighted along the character curve by a large individual distance measure compared to all other retained points along the curve.

Although the technique is good for this particular character example, it was found that it would not operate reliably for all characters. It cannot handle the noise introduced by some writers produced by pen rest, because the random nature of the cluster of points produced also leads to a random path connecting these points, with the result that the angular variation technique is ineffectual. Therefore, it is necessary to perform data thinning by lower distance threshold measure beforehand.

3.3.3. Curve Smoothing

Curve smoothing is particularly useful as a preprocessing stage to character recognition. It can remove erroneous points in the character curve introduced as a result of:-

- quantisation error (poor tablet resolution)
- jitter points (transmission noise or inadequate pen position detection)

- user hand shaking (hesitation)

In general it is not necessary to reconstruct the characters original curve perfectly. However, the method presented by Burr (sine fitting) might be of use on sampled data from a particularly poor tablet. The sampled points could be fitted to a series of sine curves. These curves are then sampled at a higher rate to produce a more accurate representation of the original character curve. It would not, however, be particularly useful for real time preprocessing due to its large processor overhead.

Of the range of tablets evaluated, all had an accuracy greater than or equal to 0.005*in* and a sampling rate of at least 60 points per second. This was found to encode a curve with sufficient accuracy. Quantisation and jitter noise are usually not severe enough to distort the character shape so as to be too far removed from the original character shape.

An example of particularly severe jitter could often be obtained from a Calcomp 2000 tablet, especially when writing a character quite quickly. Some data tablets detect the next character point by searching an area around the previously detected character point. This is much faster than interrogating the entire tablet surface each time and as a result, a much higher sampling rate can be achieved. However, problems can arise if the pen speed takes the pen outside the bounds of the next search area, leading to the production of a completely random data point.

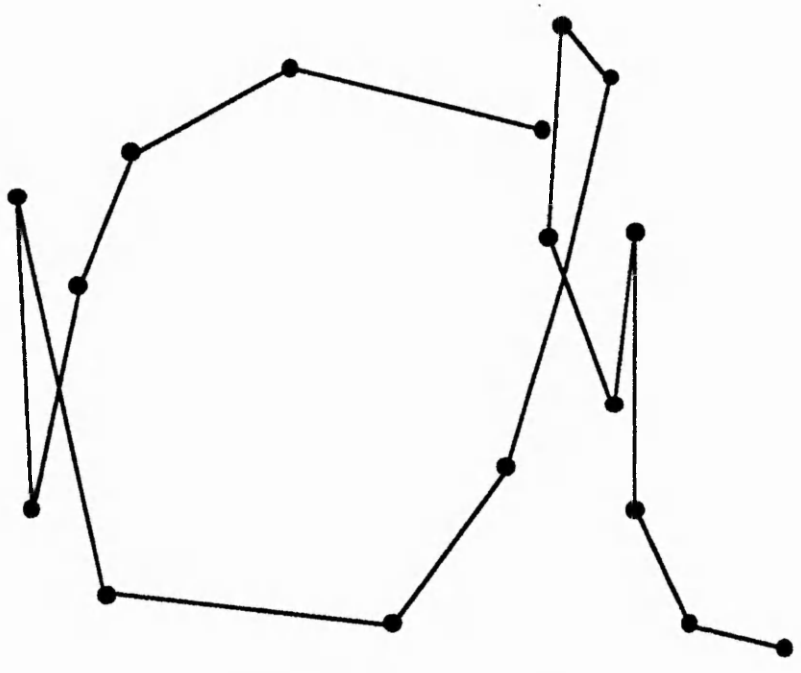


Figure 3.5 - Encoding Noise Due to Jitter

The noise points have introduced extra cusps into the character curve. These extraneous features tend to mask the curves natural characteristics. However, this is a particularly rare case of noise now that the tablets have a much better degree of resolution and accuracy. A more common problem is quantisation noise on a particularly small character, which can seriously distort the character curve. Higgins and Whitrow [53] do not perform any curve smoothing because they feel it to be too time consuming and can lead to a loss of detail which might prove useful at a later stage.

The curve smoothing technique is simply a series of point averagings between the start and end points of the curve. Basically, a new mid-point is calculated as being positioned centrally between the two end points. The process can be repeated a number of times, each pass refining the curve shape further.

1st Iteration:

$$x_1' = x_1; \qquad y_1' = y_1$$

$$x_2' = \frac{x_1' + x_3}{2}; \qquad y_2' = \frac{y_1' + y_3}{2}$$

$$x_3' = \frac{x_2' + x_4}{2}; \qquad y_3' = \frac{y_2' + y_4}{2}$$

.

.

$$x_n' = x_n; \qquad y_n' = y_n$$

2nd Iteration:

$$x_1'' = x_1'; \qquad y_1'' = y_1'$$

$$x_2'' = \frac{x_1'' + x_3'}{2}; \qquad y_2'' = \frac{y_1'' + y_3'}{2}$$

$$x_3'' = \frac{x_2'' + x_4'}{2}; \qquad y_3'' = \frac{y_2'' + y_4'}{2}$$

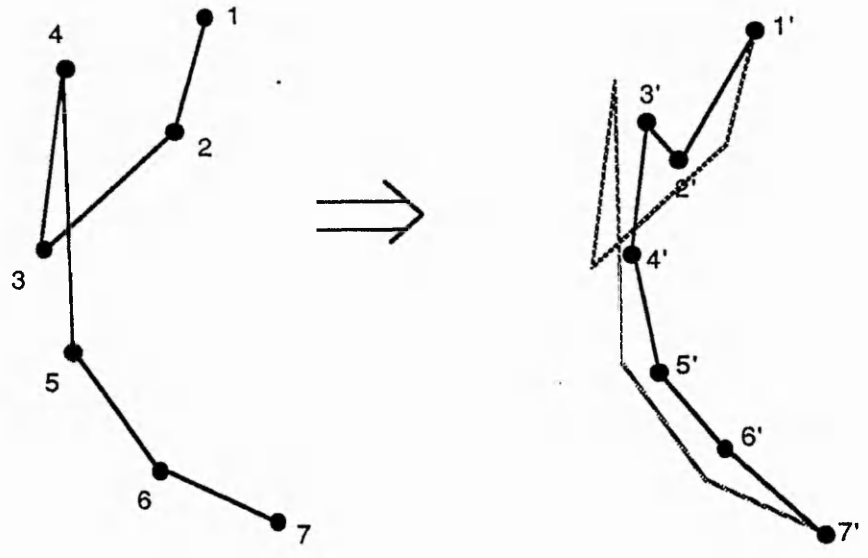
.

.

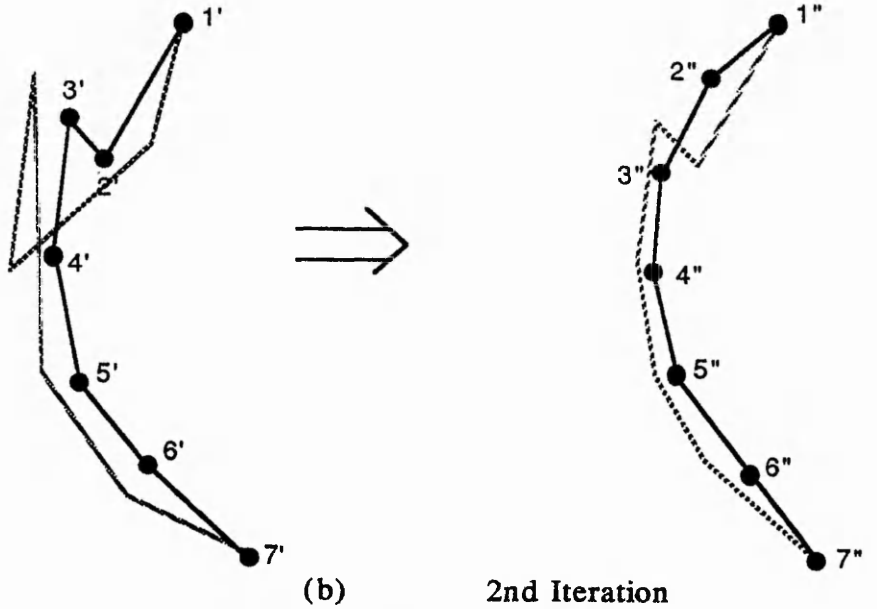
$$x_n'' = x_n'; \qquad y_n'' = y_n'$$

If we take a section of a character curve that is observed to exhibit severe noise problems and pass it through only two iterations of the algorithm we can see (Figure 3.6) that the noise has been eliminated. The character curve

smoothing can only be effectively performed once the curve has been finished (while the data thinning can be performed as the points are received from the data tablet). This is necessary in order that the algorithm does not smooth out legitimate character curve features, especially sharp turning points.



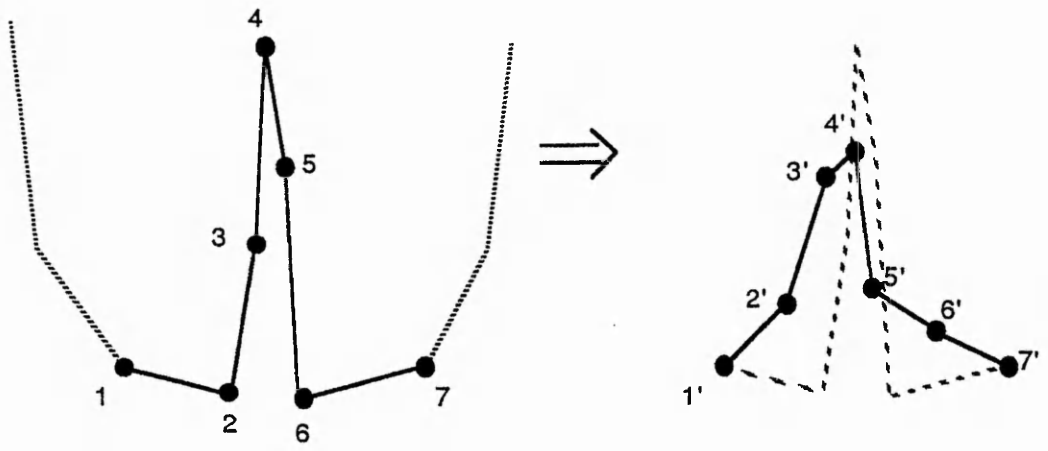
(a) 1st Iteration



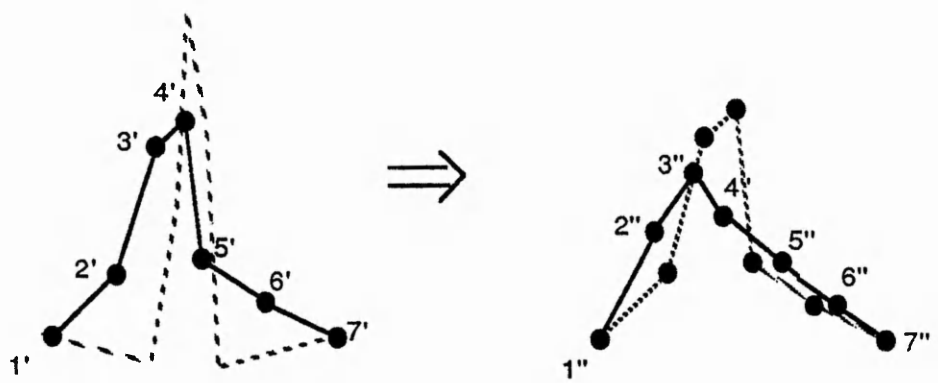
(b) 2nd Iteration

Figure 3.6 - Curve Smoothing

If the algorithm were allowed to smooth the whole curve unconditionally we would observe such detrimental results:-



(a) 1st Iteration



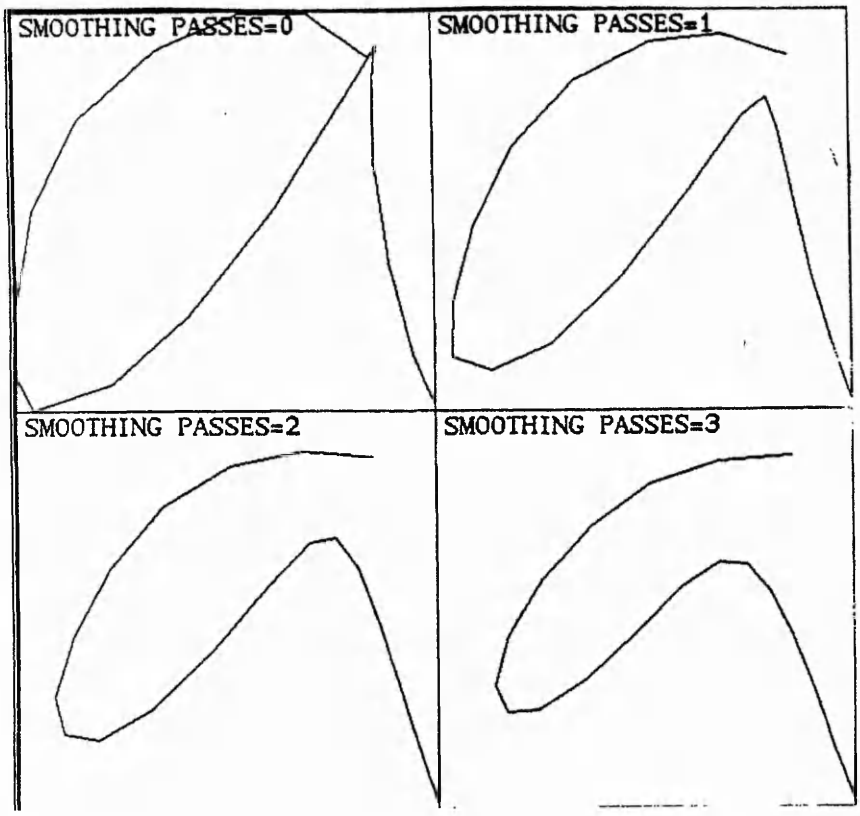
(b) 2nd Iteration

Figure 3.7 - Smoothing of Legitimate Curve Features

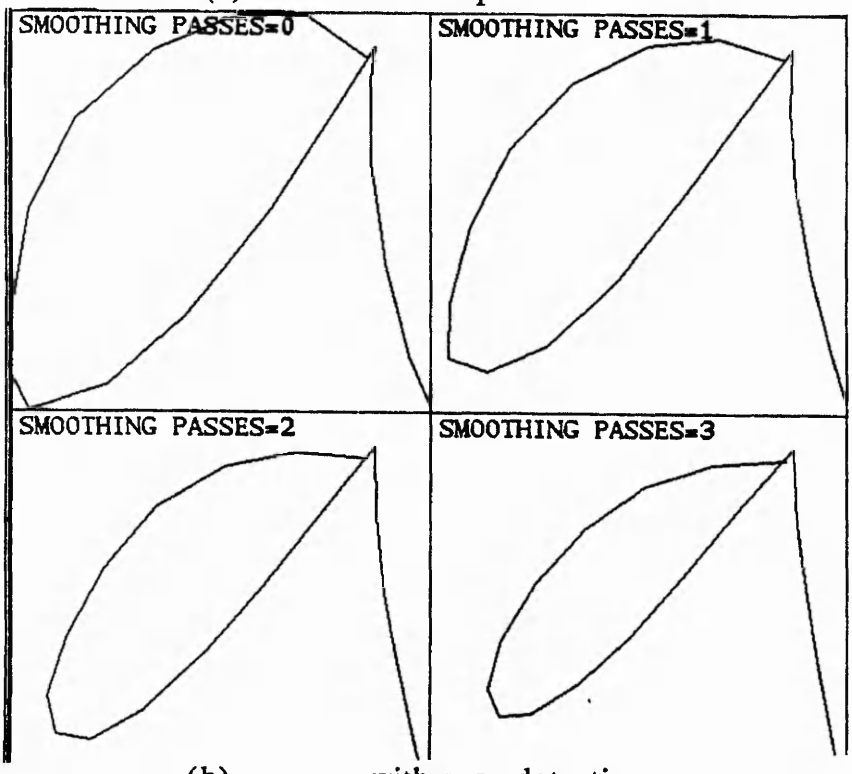
The curve above is part of the character 'w'. Smoothing of this section of the curve removes the feature of the character 'w' which distinguishes it from the characters 'u' or 'v'. In order to detect and retain such features it is necessary to analyse the curve two points ahead of the point to be smoothed. This enables one to decide whether the point is indeed a spurious noise point, in which case the two points before and after it will not

suggest the presence of a local maxima or minima. Therefore the point should be smoothed. Otherwise, we have a genuine maxima or minima in the curve which we do not want to degrade by smoothing.

Figure 3.8 shows the algorithm applied to our character 'a'. Figure 3.8 (a) indicates how the character shape is distorted if any cusps are not identified. The character shape begins to seriously degrade after only three iterations. If we detect cusps we can eliminate these points from the smoothing algorithm, so preserving the upstroke and downstroke information. The character 'a' in this example does not exhibit very serious jitter, and so is not requiring smoothing. One adverse affect of the smoothing is the reduction in the overall size of the character, notably loops as can be seen in Figure 3.8 (b).



(a) no cusp detection

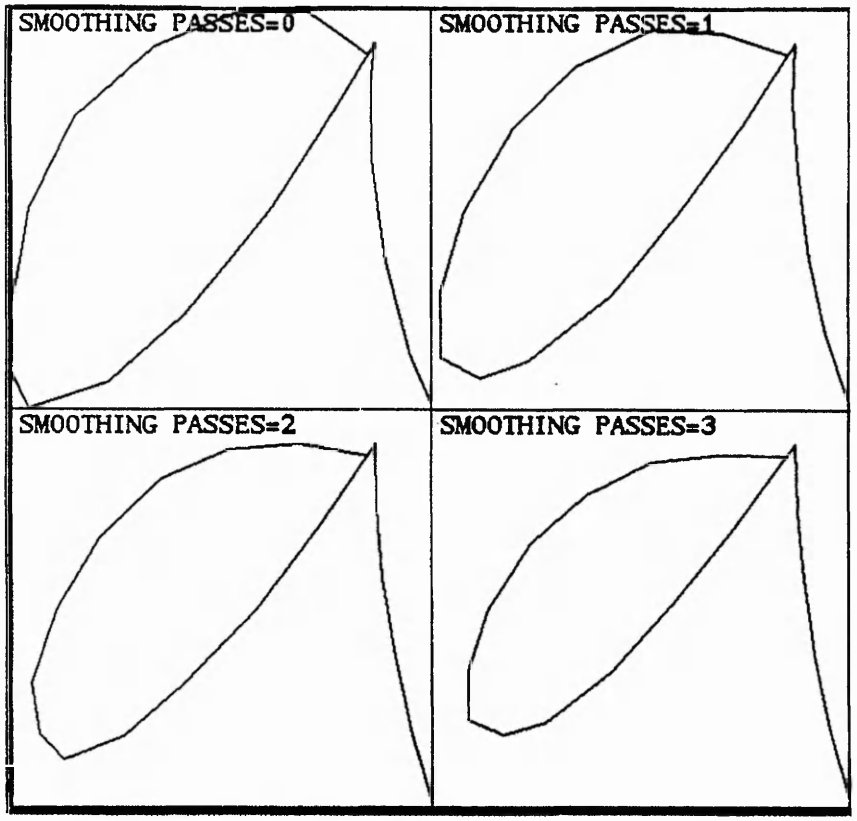


(b) with cusp detection

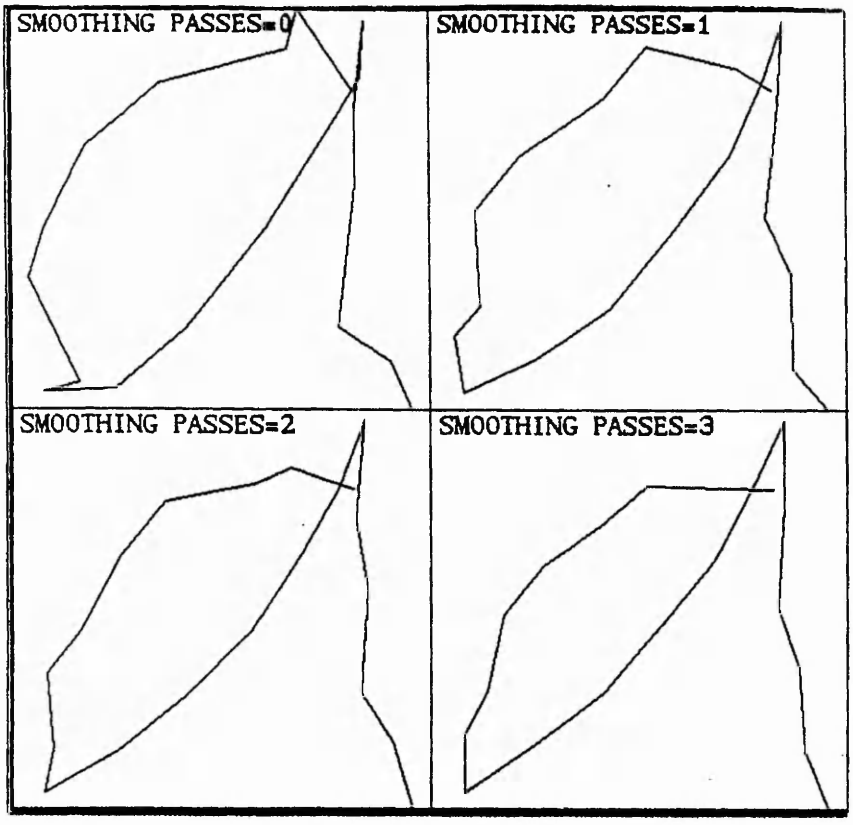
Figure 3.8 - First Smoothing Algorithm

In order to reduce the shape degradation due to the smoothing algorithm, a modification to the original algorithm will smooth the character curve while preserving the character shape more effectively. The modification was fairly minor, still performing the averaging process, but not incorporating the previously averaged point in the next stage of the same iteration. The character smoothing is still as effective as before, while maintaining the loop size. However, it is found to degrade quite seriously after only three iterations of the smoothing algorithm. Figure 3.9 (a) shows the modified smoothing on the character 'a'.

It is not particularly useful to observe the effectiveness of the smoothing algorithm on our character 'a'. Figure 3.9 (b) shows a seriously deformed character 'a' which has been passed to the smoothing algorithm. After three passes the jitter points have been ironed out. These jitter points must be removed in order to successfully analyse the 'true' character features.



(a) Modified Smoothing



(b) Noisy Character

Figure 3.9 - Smoothing Algorithm

3.3.4. Slant Analysis

Slant removal is performed by a number of authors in order to minimise the complexity of the subsequent feature extraction and recognition algorithms. As it is a user dependent feature, slant is considered by most authors as a trait which should be removed before character/word processing. Slant removal has the benefit that it standardises the basic shape of the character set (a-z for example). It has a more severe effect on certain types of character, those which have large aspect ratios ,ie. characters occurring in the upper and mid zones and the mid and lower zones:-

b,d,f,g,h,k,l,p,q,t,y,z all can experience severe slant distortion

a,c,e,m,n,o,r,s,u,v,x do not suffer from the same amount of distortion

It is important that slant removal should be totally divorced from the recognition algorithms. The technique for slant identification and removal described below is not particularly involved with the retention of the character shape with deskewing as long as the only feature seriously altered is the elimination of the slant. Brown and Ganapathy [56] deskew cursive words by slope analysis on only the mid-zone letters.Usually the initial problem in most slant removal algorithms is the determination of the angle of slant (θ_S). One technique by Burr [43] investigated initially set the centre of the character to (0,0). The centre was defined as:-

$$O = \left[\frac{(x_{max} + x_{min})}{2,} \frac{(y_{max} + y_{min})}{2} \right]$$

The angle of slant is calculated by computing the centres of gravity of the curve portions above and below the x-axis, T and B. The angle of the line joining these points to the horizontal is deemed to be the angle of slant, θ_S. Figure 3.10 shows this performed on the unslanted character 'r'.

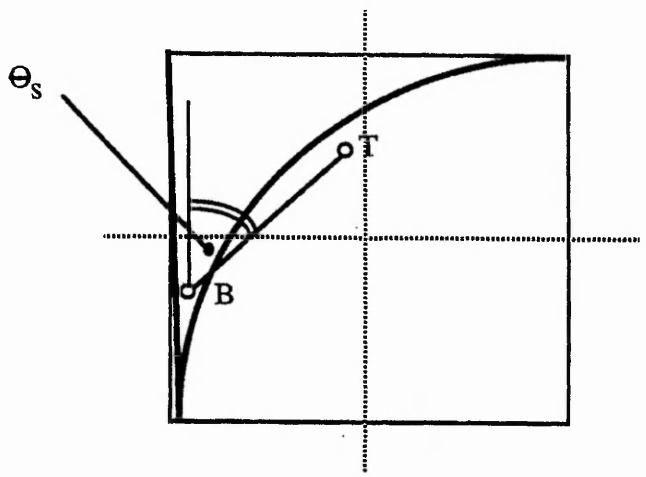


Figure 3.10 - Using Centres of Gravity to Determine Character Slant

Burr [35] adopts a different approach for a connected letter string. We define character slant as the angle to the vertical of any major downstroke detected in the character curve. Slant removal is broken down into the following processes:-

(i) detection of all downward (negative y) travels along the character.

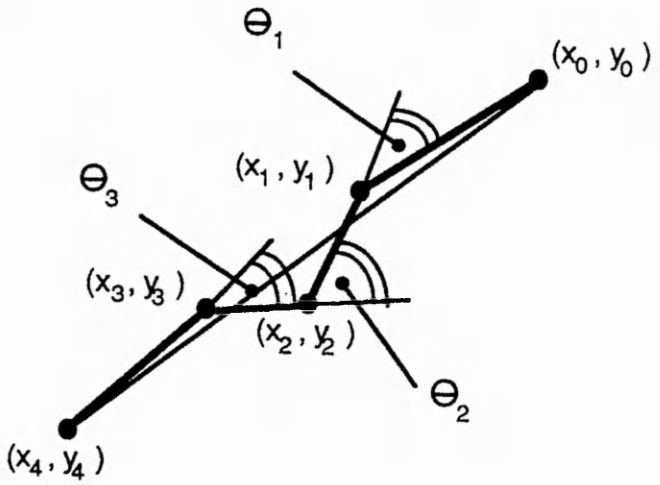
(ii) identification of which, if any, is the major downstroke.

(iii) determination of the angle of the major downstroke to the vertical.

(iv) slant removal by shear transformation

The first stage is quite easily performed. The second stage involves the investigation of the local angular variations along each negative y curve

element. The angular variations along a straight line portion will be small compared to the angular variations of a negative y portion which is an arc. Thus negative arc portions can be eliminated by selecting a minimum threshold angle. Any local angular variation exceeding this threshold disqualifies the curve portion from being a valid downstroke.



If $((\theta_1 < \theta_M) \&\& (\theta_2 < \theta_M) \&\& (\theta_3 < \theta_M))$

Element E_{0-4} = *straight line*

Figure 3.11 - Curve Downstroke Detection

It is possible to find a number of elemental curves which could be classed as straight line elements (for example in the character 'm'). If more than one downstroke element is detected, each separate element, if found to be a valid downstroke is deskewed individually.

After calculating the angles of the these elements to the vertical we must decide whether they are valid downstrokes or some other straight line element.

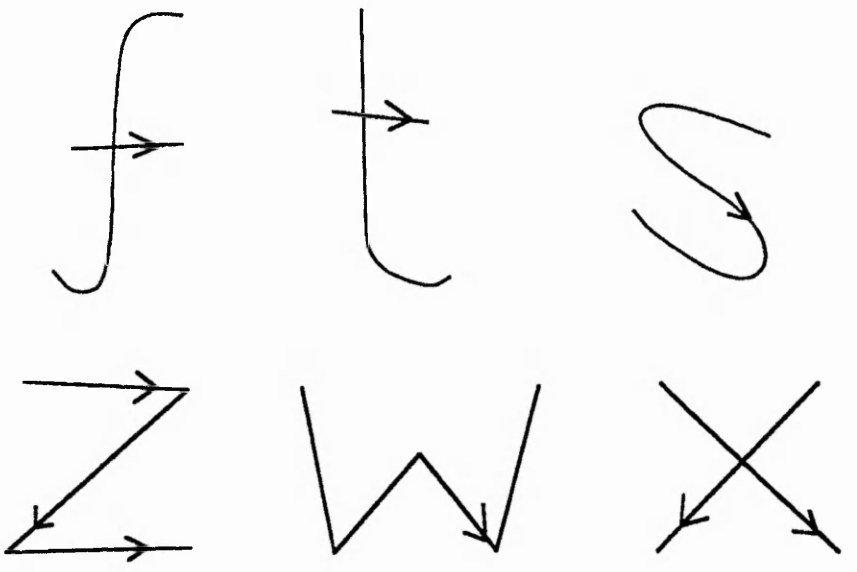


Figure 3.12 - Non downstroke Straight Line Elements

These straight line elements do not require alignment as they are not meant to be vertical. Therefore we must select a lower threshold angle (θ_L) of 60°. Any straight line detected whose angle to the vertical is θ_L or less is not processed further. In this way we do not attempt to deskew diagonal straight line elements as might be produced in the subset above.

It was decided to perform a shear transformation to remove the slant from the character. Although this may distort the shape of the character somewhat, it will not alter the shape to the extent that it alters particular character features. Consider the slanted character 'a' transformed by rotation.

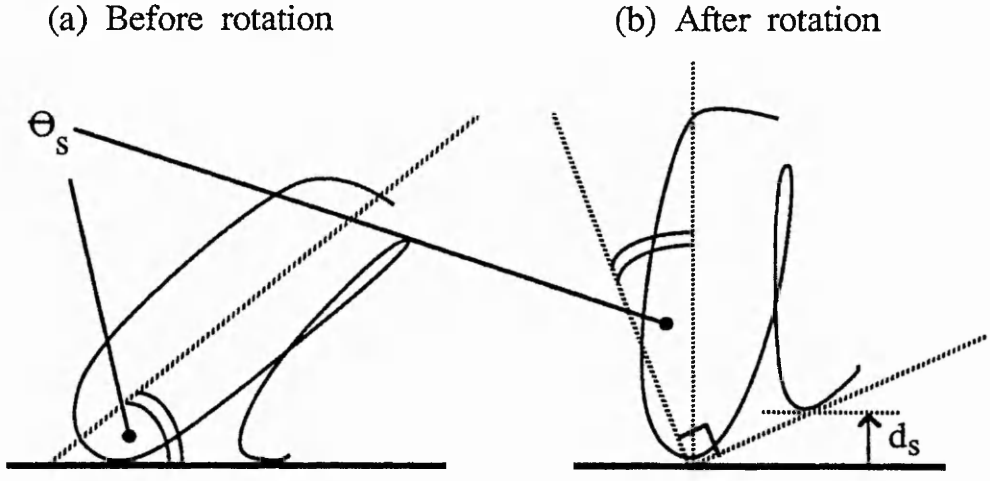


Figure 3.13 - Rotational Transformation

Shear transformation has two advantages over the technique of rotation:-

(i) it is far simpler to perform than rotation of the entire character.

(ii) it is possible to retain the original size and position along the baseline more easily than rotation.

Consider a slanted character 'a':-

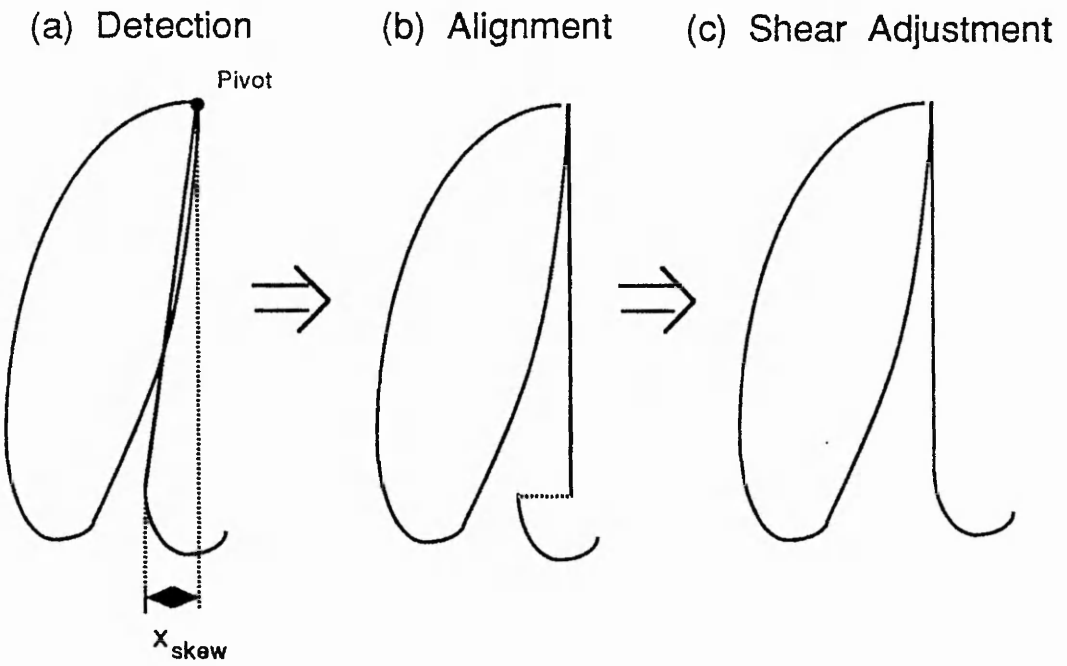


Figure 3.14 - Deskewing Procedure

By determining where along the character curve a particular downstroke is to be found it is possible to decide whether to use the top of the downstroke or the bottom as the pivot point. If the downstroke is in the second half of the character, the pivot is the top of the downstroke, otherwise the pivot is the bottom of the downstroke. This ensures that we only need adjust the minimum number of x positions after downstroke alignment.

If we detect a number of valid downstrokes, the operation must be repeated:-

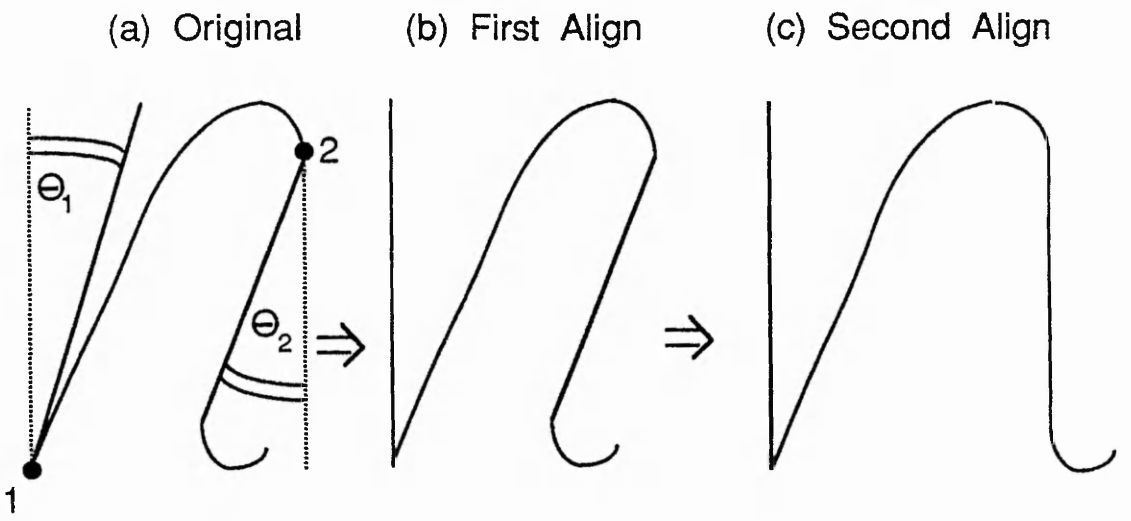


Figure 3.15 - Multiple Downstroke Alignment

This technique is very simple to implement and proved effective on most slanted characters. It did, however, occasionally fail in instances where:-

(i) the downstroke was particularly curved so as to fail the test for straight line detection.

(ii) the downstroke was so slanted that it exceeded the lower threshold θ_L, to the extent that it could not be distinguished from a diagonal.

3.4. Conclusions

It appears that in many papers reviewed a great deal of emphasis is placed on pre-processing in one guise or another. Indeed in some cases the complexity of the pre-processing matches the complexity of the recognition algorithms. The familiarisation gained has shown that, while in many cases, this processing of the raw data is highly beneficial to the success rate of the recognition algorithms using them, a number of worrying points did arise:-

- it is not possible to be able to preprocess every single character perfectly. For example, curve smoothing sometimes removes a vital feature, while slant removal might introduce an erroneous feature. In such instances the recognition algorithms will certainly fail to recognise the character correctly.
- the preprocessing algorithms ideally should be fast and simple to implement. Techniques involving complicated iterations sometimes appear

to have only limited benefit to the subsequent recogniser, their usefulness being outweighed by their size and speed.

For these reasons the only technique which seemed both beneficial and easy to implement was the simple data filter that removed pen pauses. This is the only preprocessing performed at present on the raw data before being input to the recognition algorithms described.

4. X - Y TREND ANALYSIS

4.1. Introduction

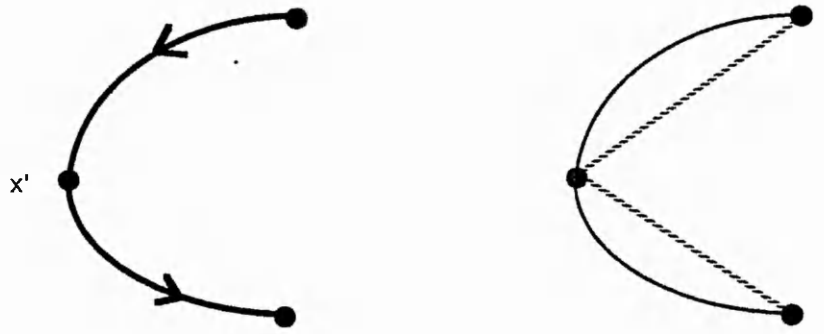
One of the major factors in developing an algorithm for the real time analysis and display of handwriting is that it be sufficiently efficient so as to give the recognition level desired without requiring so much computing time that eventual migration to a real time environment is precluded from the outset. Therefore , while not producing code which will run in real time from the outset, there must be the capability that the algorithms can be optimised for real time operation. This is the initial algorithm used for the encoding of the raw data points output from the tablet in a form suitable for subsequent recognition procedures. This section describes its development for the analysis of unconnected script and numerals. However, its applicability is not limited to unconnected character recognition , and by its nature of operation, the algorithm will migrate in some form to become a basis for the analysis of connected script

4.1.1. Background

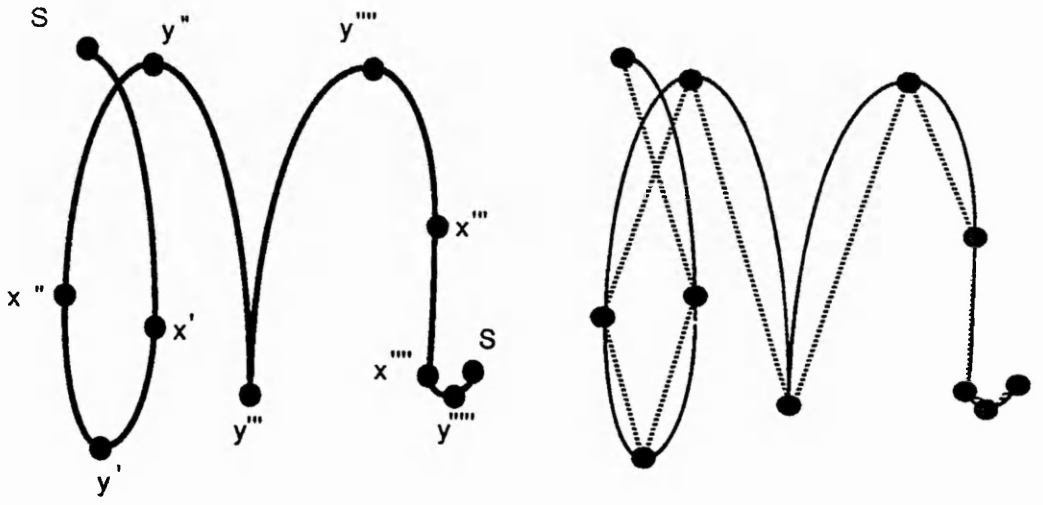
The x-y trend algorithm was based on the work of Tang, Tzeng and Hsu [47]. The paper describes the application of the technique to the recognition of the numerals 0-9. The x and y turning points are extracted from a curve and from these are extracted a series of primitive shapes. Six primitives are extracted, four of which indicate the maximum and minimum x and y turning points, a fifth indicating a straight line and the last being the stroke start / stop delimiter. This basic idea was applied to the lower case character alphabet. Tang claimed that with only 10 turning point sequences they could encode over 70% of all input patterns for numerals 0-9 for a sample set of 30 students writing the numerals 0 to 9 once only.

If we consider the lower case alphabet set a-z, many of the letters were found to be formed by writers with only a single pen to paper stroke. However, in some instances it is necessary to form the character with more than one single pen down motion (most notably the diacritical marks in letters i,j,t,f). We can deduce that the simpler strokes will have far fewer turning points than the more complex strokes. The Cartesian (x-y) travel of the pen is tracked as the pen traces the character outline. Therefore a point on the curve (x_i,y_i) is a turning point in the x direction if $(x_{i+1}-x_i)\leq 0$ and $(x_i-x_{i-1})\leq 0$ and is a turning point in the y direction if $(y_{i+1}-y_i)\leq 0$ and $(y_i-y_{i-1})\leq 0$.

If we consider two characters, 'c' a simple character and 'm' a complex character. The very simply constructed 'c' has only one turning point (an x minima) between its end points whereas the 'm' has a total of 9 turning points between its end points.



Character 'c' – 1 turning point



Character 'm' – 9 turning points

Figure 4.1 - Comparison of character turning points

The one major constraint about designing a character recognition algorithm is that it should be able to perform in a real time environment (i.e. the user should be able to see the result of writing a character appear instantaneously on the screen). Any amount of delay between writing and displaying degrades the effectiveness and naturalness of the system. In the study undertaken of algorithms for the recognition of script [84] the most promising papers in terms of achieved recognition rate were as a result of complex algorithms suitable for a large mainframe, but not so suitable for a real-time stand-alone recognition system if the recognised output does not appear instantaneously. Methods based on matching an unknown character or word template against a database of pre-formed templates gave very good recognition results for a user dependent system on which the user had already performed an initial training session. However, for a user independent system to be undertaken using this technique a very much larger number of reference templates is required and the manner of the template

matching would appear to degrade the system performance so as to exclude real-time operation. Topological feature extraction is another popular approach , some algorithms incorporating x & y turning points as just one of a number of different features to be extracted.

4.1.2. Initial X Y Algorithm

This method is a very simple extension of the work described earlier by Tang. It was not envisaged at the outset that it would be able to sufficiently differentiate between all the letters in the alphabet, but rather to be able to classify letters into some subset of alternatives related to the similarity between certain letter shapes in the alphabet. Analysis of an initial set of 25 writers led to an initial broad alphabet sub-classification:-

f t
y g
a d q
p b
n h u
r v
o c e
m w
k x z
i s j l

Therefore the main purpose of this initial algorithm was some form of verification that letter shape information could be extracted very easily, and more importantly, very quickly. It is hoped that this could be used as input to more advanced stages, reducing the computation required at these later stages by focusing the decision procedure.

4.1.2.1. Theory

The first approach to the analysis of the turning points was to analyse the x and the y travel separately as the co-ordinate points trace out the path of the character. The x-travel is effectively split into the relative x-travel from one x-turning point to another. The reason for analysing the relative movements rather than the absolute movements was to be able to classify letters by their shapes irrespective of their sizes, i.e. two letters of similar shape but different sizes would have identical travel moments.

In this respect we also classify the start and stop points of the character as turning points. The same process is performed independently for the y-travel over the path of the character. When the pen is lifted from the paper, the incremental travels in both the x and y direction are normalised. This involves simply expressing each incremental travel as a fraction of the total travel in that particular plane. Therefore an encoded character will be of the form:-

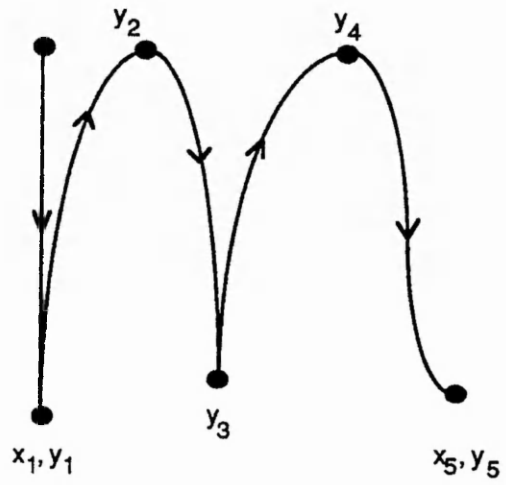
$$x\text{-}travel = x_1 , x_2 , x_3 , \ldots\ldots x_n$$

$$y\text{-}travel = y_1 , y_2 , y_3 , \ldots\ldots y_m$$

$$where \quad |x_1| + |x_2| + |x_3| + \ldots\ldots + |x_n| = 1$$

$$and \quad |y_1| + |y_2| + |y_3| + \ldots\ldots + |y_m| = 1$$

Alternate x values will be negative. Similarly for the y-trends. If we take our character 'm' shown previously and encode it we will produce the result shown below (Figure 4.2):-



$$x\text{-}travel = -0.10 \;\; 0.90$$

$$y\text{-}travel = -0.23 \;\; 0.22 \;\; -0.20 \;\; 0.18 \;\; -0.17$$

$$where \quad |-0.10| + |0.90| = 1.00$$

$$and \quad |-0.23| + |0.22| + |-0.20| + |0.18| + |-0.17| = 1.00$$

Figure 4.2 - Character Encoding by the X-Y trend algorithm

Analysis of encoding the lower case alphabet for a number of writers showed that a number of characters produced a quite unique x and y trend representation, for example, only the character 'm' produces such a regular sequence of y-travel characteristics, five turning points commencing with a minimum. These characters were found to be the more complicated characters to form (e.g. m,w,g,k). However, for the majority of the alphabet set it was noted that different characters would produce similar encoded representations (i.e. identical in terms of the number of the x and y trends and their

sign sequence). If we consider the sign sequence in Figure 4.3.

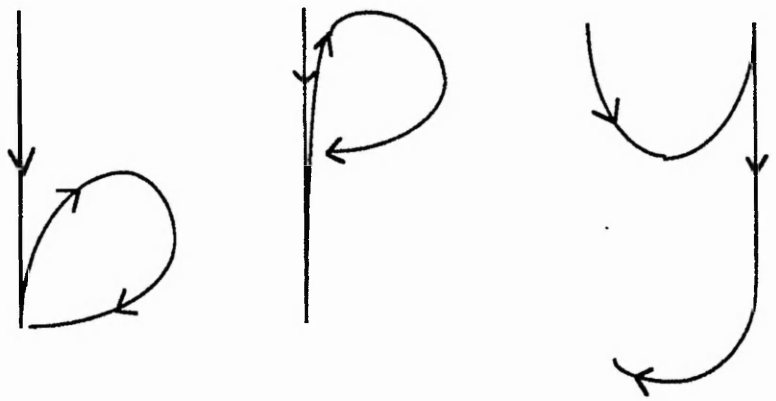


Figure 4.3 - Characters with similar X - Y trend encodings

The x and y trends in these cases all have the same tendency. Encoding of the above characters produces

b:- $x-trends = +0.50 -0.50$
$y-trends = -0.50 +0.25 -0.25$

p:- $x-trends = +0.50 -0.50$
$y-trends = -0.40 +0.40 -0.20$

y:- $x-trends = +0.50 -0.50$
$y-trends = -0.20 +0.20 -0.60$

An analysis of the relative in the y-direction alone allows quite a simple means of differentiating between these three alternatives. Hence, an unknown character which exhibits the same x and y tendency would simply be interrogated in order to determine which of the above sequence of y trends it matched the closest. The technique for correlation is described in Chapter 6. It fits both Freeman vectors and X-Y trends produced by encoding the unknown character and performs an elemental fit to likely candidates extracted from the corresponding database. The measure of fit is expressed as a percentage. It was determined early on in the analysis of both algorithms that it would not be enough simply to try and identify a character by identifying its basic features. Tang et al attempted to recognise the numeral by simply identifying the turning points but problems arose if two numerals have an identical turning point sequence as observed with the numbers 0 and 6. Hence by comparing the relative trends between the turning points we can obtain a percentage fit figure. 100% indicates a perfect fit and 0% a very poor fit.

4.1.2.2. The X-Y Database

A database was originally constructed from a user set of over 100 writers. Each writer was asked to write two sentences in lower case unconnected script. However, the only other constraint was that they should write from left to right across the paper. Letter size, speed of writing, character formation, and neatness of writing were left to the discretion of the writer. By analysis of the recorded raw data from the graphics tablet on a graphics terminal it was possible to identify the actual character and stroke sequences produced. It is subsequently possible to encode the stroke/character from its raw data form into the x-y trend encoding. Hence each encoded string along with its intended identity is written to a file for subsequent analysis. Therefore this file will contain the encoded character strings for all the tested writers, a total of 112 people. This gave a total of over 8000 encodings for the 79 characters making up the two test sentences produced. These test sentences contain every letter of the alphabet.

"pack my bags with five dozen extra liquor jugs"

"both wizened men quickly judged four sharp vixens"

The aim is to produce a representational database, which can be used for the recognition of any person who wants to use the system, in other words a user independent system. Therefore the aim was to produce a database which contained a representational cross-section of all the character styles produced by the writers. (a detailed description of all aspects of the database is described later on in Chapter 6). One immediate concern arose for the recognition of user independent script. In many instances one particular writer will form a character in exactly the same way as another writer forms a different character. This was observed to occur especially between characters r and v, u and v, g and q, b and f, u and n. However, in most instances a writer would form characters in a unique manner (i.e. in a way that would allow another person to view that character, in isolation and to be able to identify it. Some people have analysed exactly how good people are at recognising text out of context. Results vary between 90-94%, Suen et al [42]. Most characters were found to be constructed in the same manner by the majority of writers, which would produce very similar, if not identical x-y trend encodings. A breakdown of the deviation of x-y trends produced for a character 'a' is shown in Table 4.1.

Number of trends		Signs of start trends			
X plane	Y plane	--	-+	+ -	+ +
2	2	0	0	0	0
2	3	229	13	0	0
2	4	27	246	0	0
2	5	0	30	0	0
3	2	0	1	0	0
3	3	24	0	3	2
3	4	0	15	1	78
3	5	0	0	6	27
4	2	0	0	0	0
4	3	150	0	5	0
4	4	50	118	0	17
4	5	0	27	2	0
5	2	0	0	0	0
5	3	0	0	1	0
5	4	0	0	1	55
5	5	0	1	2	27

Table 4.1 - X Y Trend distribution for character 'a'

From the test writers samples a total of 1158 a's were formed and the table shows a distribution of their encodings. There are a total of 27 different ways the letter 'a' has been encoded. However its is observed that a great number of a's fall into one of four encodings. These represent a figure of 64% of the total and are represented by the shapes given below in figure 4.4. Deviations occur due in the main to the pen being lowered onto the paper and moving to the start point of the 'a' and/or the pen remaining down after the 'a' has been completed.

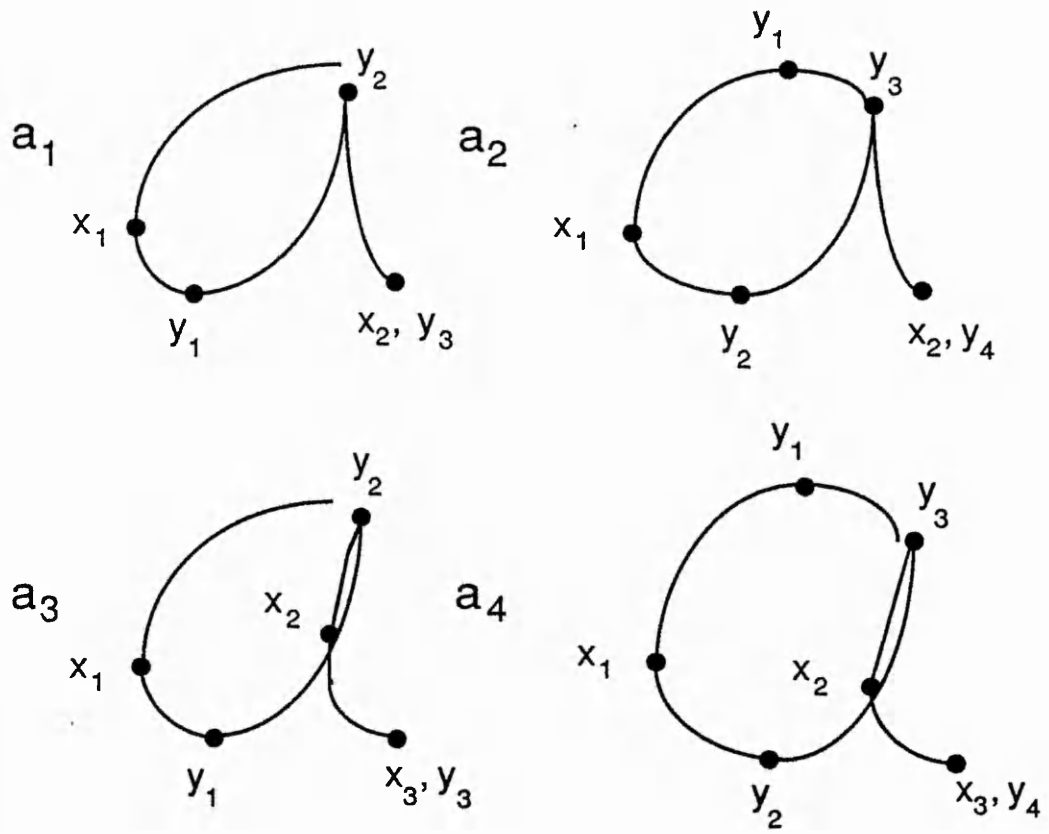


Figure 4.4 - The four most common encodings of the character 'a'

In order to produce a representative single encoding for each of the 27 different encodings observed in the table a single averaged version is produced by averaging the trends of all the members. If we consider the most popular encoding, a_2 in the above figure, (X= -+ , Y = + -+ -), and show how four such examples of this specific shape are averaged:-

a_1 , $X = -0.42+0.58$, $Y = +0.10-0.33+0.25-0.32$

a_2 , $X = -0.41+0.59$, $Y = +0.02-0.30+0.29-0.39$

a_3 , $X = -0.38+0.62$, $Y = +0.15-0.40+0.19-0.26$

a_4 , $X = -0.36+0.64$, $Y = +0.09-0.39+0.26-0.26$

Averaged 'a'

$$X = -\frac{(0.42+0.41+0.38+0.36)}{4} + \frac{(0.58+0.59+0.62+0.64)}{4},$$

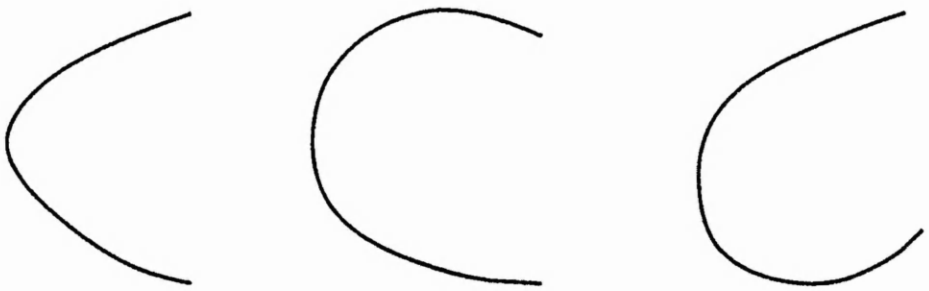
$$= +\frac{(0.10+0.02+0.15+0.09)}{4} - \frac{(0.33+0.30+0.40+0.39)}{4} + \frac{(0.25+0.29+0.19+0.26)}{4} - \frac{(0.32+0.39+0.26+0.26)}{4}$$

Giving,

a_{av} , X = −0.335+0.665, Y = +0.09−0.355+0.247−0.308

This method of averaging was performed on the characters produced by the 112 writers in the test sentences. A total of around 16000 unique character encodings was produced. After averaging the number of averaged trend encodings was reduced to just 700. This was for the lower case character set a-z and part characters [,],\ and /. This is an average of 25 unique encodings per character or part-character.

When the database had been constructed each characters variability of formation could be assessed by a direct comparison of its relative occurrence in the test sentences against the relative occurrence of the encodings for that character in the database. In theory, if all characters were of the same level of complexity to form, the ratio of relative occurrence in the database to relative occurrence in the test sentences should equal 1. For example, if we consider the letter 'l' as being formed as a single down-stroke, only two possible encodings could result, depending on whether it was slanted to the left or right, + 1.00/-1.00 or -1.00/-1.00. Therefore, however many times it were written, it should only result in one of these two XY encodings. Now consider the character 'c', this is slightly more complex in its means of creation and as such might produce the following alternatives.



c_1 = −0.50+0.50/−1.00 c_2 = −0.50+0.50/+0.10−0.90 c_3 = −0.50+0.50/−0.90+0.10

Figure 4.5 - Different Encodings of the Character 'c'

It should follow that a very simple character will produce fewer unique encodings and therefore give a ratio below 1, while the more complex characters will give a ratio in excess of 1. The results are shown below in table 4.2.

Analysis of X Y trend database			
Character	%age in ref sentences 10352 characters	%age in database 732 trends	Ratio
a	4.16	3.73	0.897
b	1.78	5.67	3.185
c	3.93	2.49	0.634
d	3.79	3.60	0.950
e	8.23	3.18	0.386
f	0.55	4.15	7.545
g	3.06	5.39	1.761
h	3.07	3.18	1.036
j	3.80	3.73	0.982
k	0.85	3.04	3.576
l	13.89	4.98	0.356
m	2.11	2.90	1.374
n	4.21	3.60	0.855
o	4.66	4.56	0.978
p	1.39	3.60	2.590
q	2.00	3.73	1.865
r	4.39	4.29	0.977
s	4.23	4.01	0.948
t	0.14	1.66	11.857
u	5.27	4.84	0.918
v	2.29	3.73	1.629
w	2.12	4.15	1.957
x	0.07	0.43	6.143
y	1.95	4.56	2.338
z	2.21	4.98	2.253

Table 4.2 - Distribution of character occurrence to database representation

A visual breakdown of the results (Figure 4.6) does indicate a general trend from the simpler strokes (those with the lowest ratios) to the most complex strokes (those with the the largest ratios). Character 'f' has a very large ratio simply due to the fact that people tend to form it in a large variety of ways. However, the results are not totally consistent. They do indicate that the simplest character is the letter 'l', however, they show the character 'e' as a simpler letter than the character 'c'. It also shows the character 'v' as a complex stroke, in fact above character 'm'. This leads us to believe that the database is not truly representative. Also, the fact that the database contained so few unique encodings for the 10,000 or so separate characters written (only 732) did give some indication that the algorithm was too simplified and was removing too much useful information from the raw data character representation.

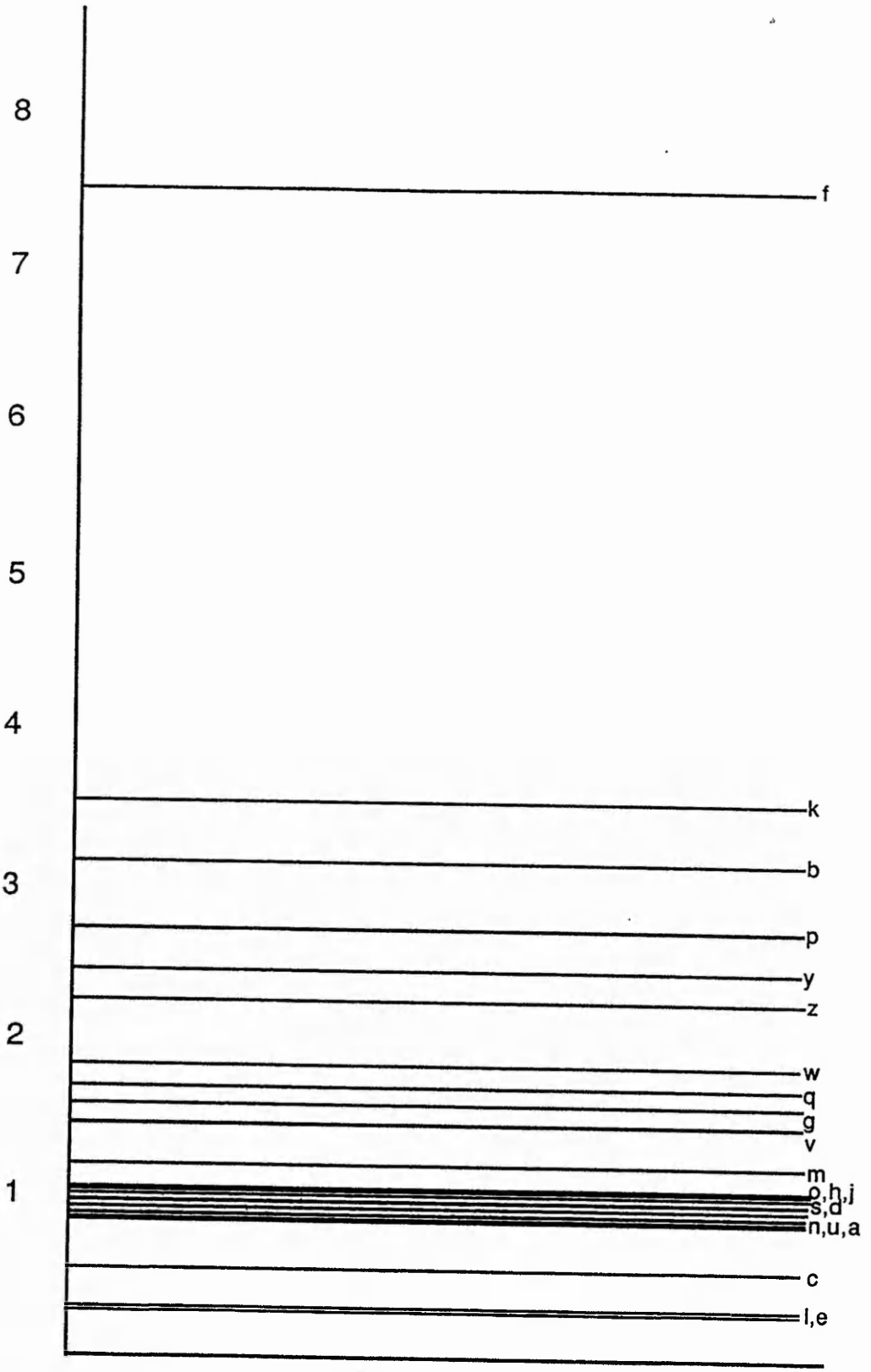
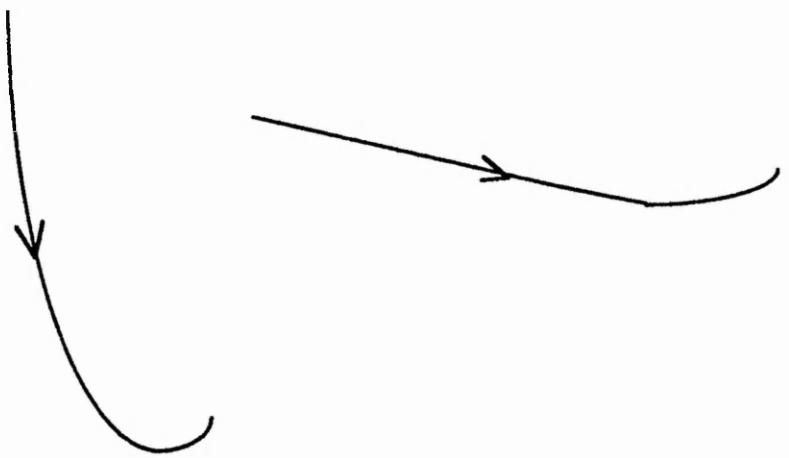


Figure 4.6 - % database occurence / % character occurence

4.1.2.3. Initial Results

Analysis of the recognition rate on the test users data did confirm the doubts about the recognition performance. The overall recognition rate was less than 65%. A character breakdown of the mis-recognised or non-recognised characters showed a reasonable performance for the more complex characters (i.e. f,g,m,k,q,w,z 80-90% recognition). However, it did not perform very well on the rest of the alphabet (i.e. a,b,d,e,h,n,o,p,s,u,y 60-75% recognition) and was particularly poor on the very simple characters or strokes (i.e. c,l,[,],\,/,- ~50% recognition). It was apparent that the algorithm was far too coarse in its encoding. As an example, if we consider the case of two very simple strokes 'l' and '-'. It is quite a simple matter for a human to differentiate between these strokes. The former exhibits a predominantly vertical motion and the latter a predominantly horizontal motion - Figure 4.7.



Character 'l' *Character '–'*

Figure 4.7 - X Y Trend Encoding deficiencies

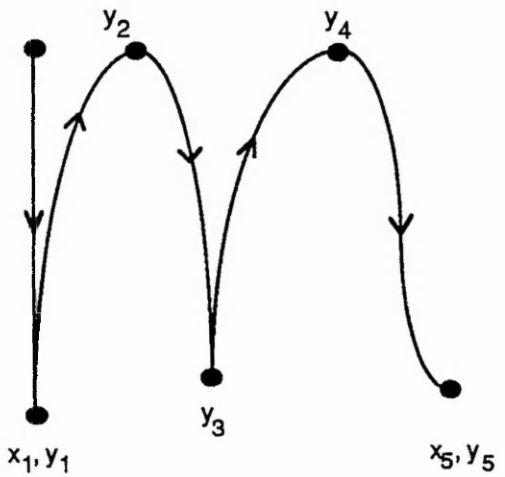
When we come to encode these two strokes we observe that they both produce the same XY trends :-

$$x\text{-}trends = +1.0$$
$$y\text{-}trends = -0.90\ +0.10$$

Two totally unalike strokes have produced identical encodings. This is an extreme example of the algorithms shortcomings, but it is also highlighted in a number of separate instances. Confusions between u/n, r/v, v/u, c/l are common. Thus a more reliable algorithm was required which would be able to cope with the confusions appearing in the algorithm in its present form.

4.1.3. Modified X-Y Algorithm

The approach adopted is a simple extension of the original algorithm. Instead of simply recording a single incremental travel in the x or y direction whenever an appropriate turning point is detected, the incremental travel in both directions was recorded each time either an x or y turning point was detected. This produces trend strings with equal numbers of x and y trends. Therefore the accumulated trend string is longer. We now also cannot assume that we will have an x or y trend string with alternate positive and negative travels. If we consider our character 'm' encoded using the old and new techniques:-



Old algorithm, $x-travel = -0.05+0.95$

$y-travel = -0.22+0.20-0.20+0.18-0.20$

New algorithm, $x-travel = -0.05+0.10+0.15+0.20+0.45$

$y-travel = -0.22+0.20-0.20+0.18-0.20$

Figure 4.8 - Modified X Y trend algorithm

Using the old algorithm, there is no immediate correlation between the x- and y- turning points. The relative travel trends are completely divorced. However, the new technique gives an indication as to how the x and y travel is altering in relation to one another. As a result we can actually reconstruct a quantised version of the character shape from the encoded trend strings. This is not possible with the old algorithm since the sequence information has been discarded.

Original quantised shape *XY trend quantised shape*

Figure 4.9 - Character shape regeneration

This feature allowed the design of a tool which, given an XY trend string would reconstruct and display the quantised character. (An assumption has to be made on the aspect ratio when displaying on the graphics screen in order produce a sensible shape.)

It is extremely difficult to visualise a character shape from its encoded trend string, and with the old algorithm, it is actually impossible to do so. It was found to be very important to verify that the representations in the database should be reasonable representations of the characters that they are meant to portray. Hence the tool allows us to verify that the database representations are reasonable. Any particularly bad or misleading representations can therefore be removed. This could not be performed on the old database.

4.1.3.1. The X-Y Database

A new X-Y database was constructed using the new algorithm. The size of the X-Y database increased substantially. Not only were the X-Y strings themselves longer, a greater number of unique trends was produced. We have now around 2500 entries in the database. We performed a similar comparison of character occurrence in the test sentences against the relative occurrences of character unique encodings in the database. Table 4.3 gives a similar breakdown of the new database as did Table 4.2 for the old database. We can now say that the more complex a character, the greater the number of unique strokes will be produced due to the larger deviation possible away from the idealised shape. Again, the larger the ratio of database entries to character occurrence in the test sentences, the more complex the character. The results (Figure 4.10) now appear to be somewhat more consistent with

the theory. The most complex strokes all have larger ratios (> 2.0). These being f,k,b,g,m,q,w,z,p,y. Also, the very simple characters have the lowest ratios (< 0.5). The simplest character now is the letter 'c', which we would expect. Also, we now see the letter 'v' has a much smaller ratio. These results were quite encouraging in leading us to believe that we may have an algorithm which is more representative of character shape.

Analysis of the modified X Y trend database			
Character	%age in ref sentences 10352 characters	%age in database 2571 trends	Ratio
a	4.16	4.08	0.98
b	1.78	6.34	3.56
c	3.93	1.24	0.32
d	3.79	4.78	1.26
e	8.23	2.99	0.36
f	0.55	3.38	6.15
g	3.06	7.78	2.54
h	3.07	2.84	0.92
j	3.80	2.88	0.76
k	0.85	3.07	3.62
l	13.89	5.13	0.37
m	2.11	5.13	2.43
n	4.21	4.51	1.07
o	4.66	3.31	0.71
p	1.39	2.88	2.07
q	2.00	4.90	2.45
r	4.39	5.52	1.26
s	4.23	3.31	0.78
t	0.14	0.93	6.64
u	5.27	5.10	0.97
v	2.29	1.98	0.86
w	2.12	4.47	2.10
x	0.07	0.43	6.143
y	1.95	3.93	2.01
z	2.21	4.75	2.15

Table 4.3 - Distribution of character occurrence to new database representation

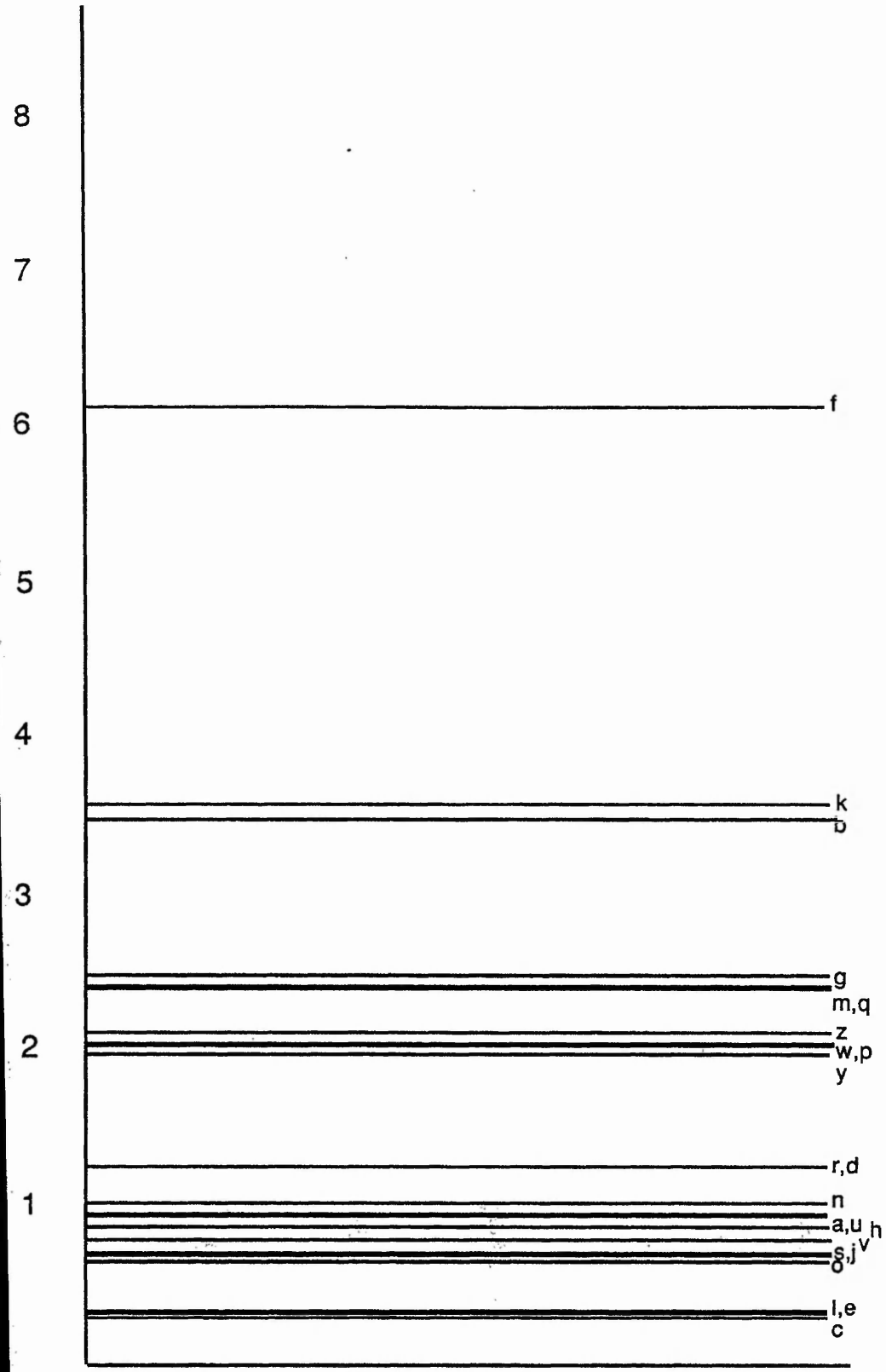


Figure 4.10 - % database occurrence / % character occurence

(Modified x-y algortihm)

4.1.3.2. X-Y Trend Processing

4.1.3.2.1. Pen-down Problems

An investigation of the XY trend encodings for our test characters produced some inordinately long strings of trends. These long trends were usually found to be a result of some minor perturbations at the beginning or end of a character (at either pen-up or pen-down). In the main they occurred at the time of the writer placing the pen down onto the paper prior to forming the stroke. Two causes of these perturbations were determined.

1. Dithering by the writer where the pen is rested on the paper for some time before they form the character.

2. More commonly it is due to pen-switch bounce or as a result of drift in the threshold detection circuitry causing the pen to be detected somewhat sooner than it is brought down onto the paper.

This tended to lead to the detection and subsequent encoding of spurious pen-down points totally unrelated to the character itself. An example of the problem will show how these trends can be eliminated.

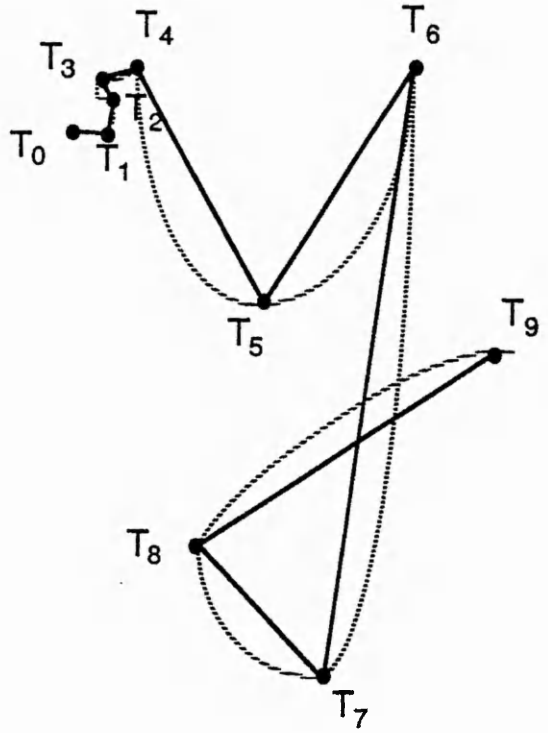


Figure 4.11 - Detection and elimination of spurious pen-down perturbations

The first 4 X-Y trend elements are of no significant importance to the overall character shape. In fact, they should not be associated with the shape of a character 'y'. As such they should be detected and removed. Raw data

preprocessing is not able to remove such superfluous points, since they represent a degree of pen movement while the writer is resting the pen on the paper. This will therefore pass through the data filter which does not attempt to remove any degree of pen movement. Therefore, in order to detect and remove these elements the moduli of pairs of x and y trends are added together in order to give an estimation of the contribution of the trend pair to the overall character travel. These perturbations represent very small relative travels compared to the significant x,y pairs of trends. Hence, any combination which is less than some pre-determined threshold value is deemed to be a superfluous trend combination and both the x and y trends are removed. Therefore, in our example case we have:-

$$x\text{-}trend = +0.03-0.04+0.04+0.10+0.15+0.10-0.25-0.08+0.20$$

$$y\text{-}trend = +0.02+0.03+0.01-0.15+0.14-0.20-0.20+0.05+0.20$$

$$Combined = 0.05\ 0.07\ \ 0.05\ 0.25\ \ 0.29\ 0.30\ \ 0.45\ 0.13\ \ 0.40$$

The combined trends falling below the threshold are removed and the reconstituted encoding becomes:-

$$x\text{-}trend = +0.10\ +0.15\ +0.10\ -0.25\ -0.08\ +0.20$$

$$y\text{-}trend = -0.15\ +0.14\ -0.20\ -0.20\ +0.05\ +0.20$$

Because so many occurrences of this pre-character pen travel is observed, it is important that they be removed before entry into the database. If all such superfluous trends were ignored and allowed to be entered into the database,

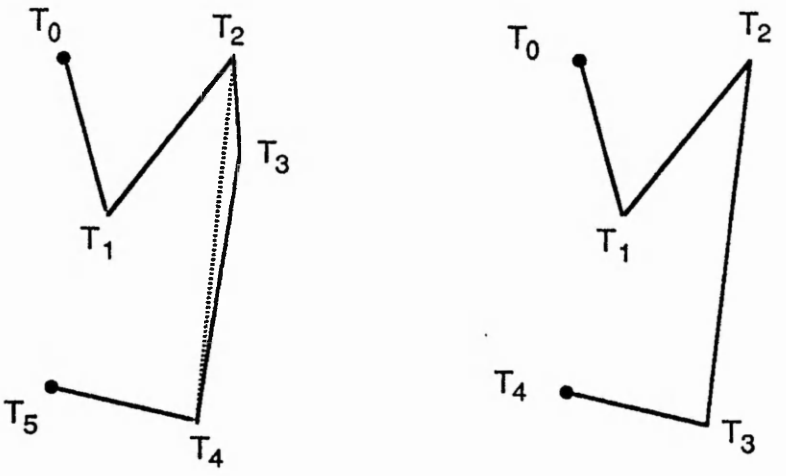
1. The database size would be disproportionately larger to accommodate all the extra 'unique' trends which would be produced.

2. As a direct result of 1. the searching time and correlating would be increased

4.1.3.2.2. X-Y Trend Reduction

The algorithm gave a very good reproduction of characters which have a large number of turning points, but is still observed to somewhat over simplify the less complex strokes as before, but not quite to the extent as did the first algorithm. The recognition rate achieved from our test set of 112 writers did improve somewhat, but not significantly. Only a matter of 2-3 %. This was found to be due to the fact that now we have many more unique trends per character, so that a particular encoding for a character did not appear in the database and consequently that character was either not recognised or misrecognised. The reason for this is illustrated below. Figure

4.12 is of an encoded character 'y' which has five turning points along the curve. However, it was observed that some turning points are due to a minor perturbation in the x or y direction. In our example (Figure 4.12 (i)) we have a turning point T3 produced by a small x-change over the down-stroke part of the y. Removal of this x-trend would produce a reduced (and more common) encoding (Figure 4.12 (ii))

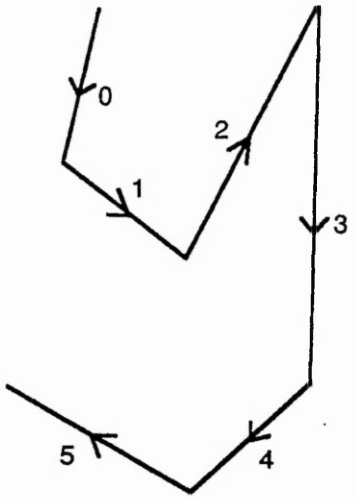


(i) Original *(ii) Reduced*

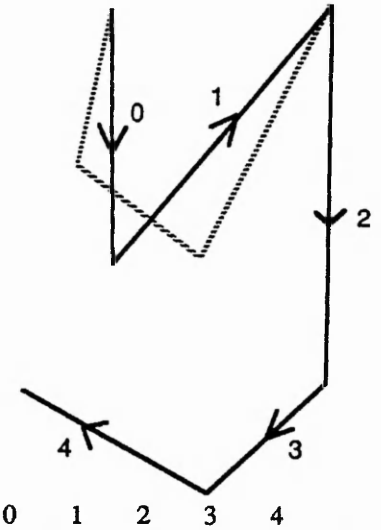
Figure 4.12 - X Y Trend Reduction

Therefore, in order to maximise the probability of finding a correct match a number of reductions are performed and the database searched to find any matches. It was observed that the likelihood of finding a match with the database increased as the size of the trend string decreased. However, it was also found that the number of alternative character possibilities increased.

Therefore, a mechanism was required to detect the smallest combined trend pair (as in the initial trend filtering). However, in this case we do not want to simply discard the x,y pair since in this case the contribution of its relative travel is important to the overall character. Therefore we add the component to the neighbour which exhibits the same travel direction. Figure 4.13 shows an example of such a reduction.



Original 'y'	0	1	2	3	4	5
x – travel	−0.05	+0.10	+0.25	+0.00	−0.35	−0.25
y – travel	−0.15	−0.05	+0.20	−0.35	−0.15	+0.10
Combined	0.20	0.15	0.45	0.35	0.50	0.35



Reduced 'y'	0	1	2	3	4
x – travel	−0.05	+0.35	+0.00	−0.35	−0.25
y – travel	−0.20	+0.20	−0.35	−0.15	+0.10

Figure 4.13 - X Y Trend Reduction

Therefore, even though the actual trend is removed, its contribution to the total travel is retained. In our example the component of the x-trend removed ($x1$) is added to the $x2$ direction to produce the combined $x1$ trend. However, the $y1$ trend removed has its component added to the old $y0$ direction. Therefore, for a particular encoding a series of reduced encodings can be produced, each of which can be searched against the entries in the database to find a match. Using this technique, if an initial fit does not identify the correct character, then one or more of the reductions could find

the correct match.

One obvious pitfall to be avoided when reducing is to remove a trend pair which changes a character shape so that the new encoding resembles a different character (Figure 4.14).

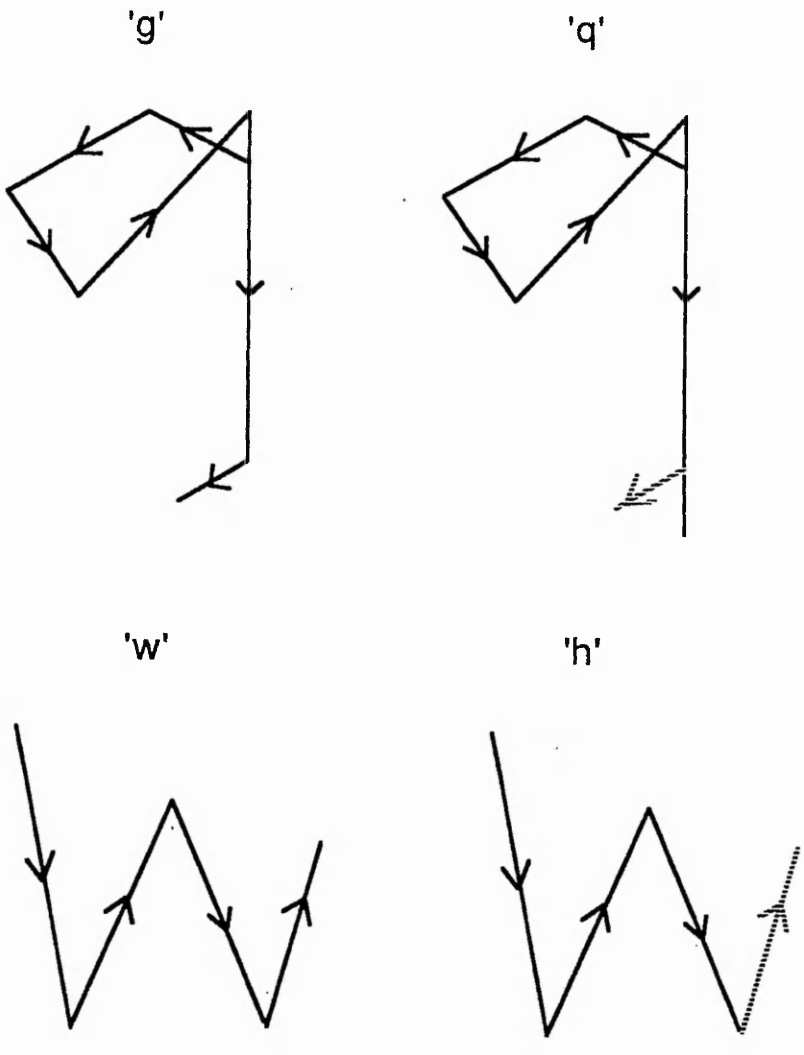


Figure 4.14 - X Y Trend Reductions to be Avoided

4.2. Conclusions

The XY algorithm work has evolved from the initial idea (the analysis of the x and y turning points) which proved to be far too simplistic when attempting to differentiate between letters from the subsequent extracted turning points. This was rectified by the simple modification whereby the

incremental travel in both the x and y direction was analysed whenever a turning point was detected, irrespective of whether it was an x or y turning point. This method was intuitively superior, since the character shape could now be realised from the encoding. As expected, this technique improved the recognition rate, by around 15-20%. A full breakdown of results is given in Chapter 8.

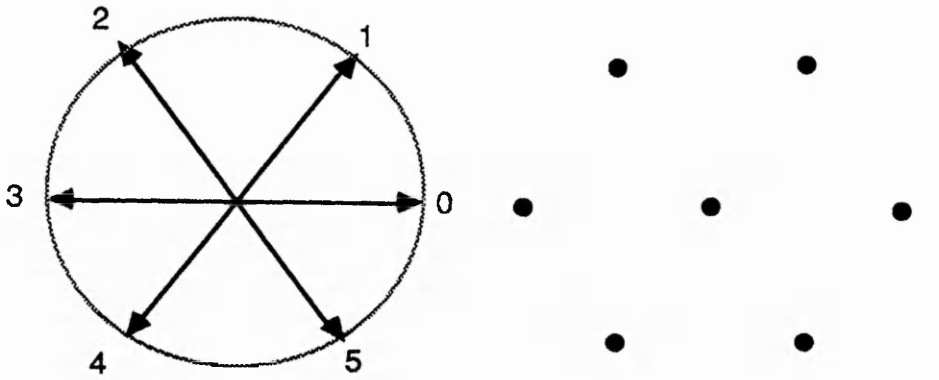
However, a particularly worrying feature of the algorithm was its inability to reliably differentiate between the identity of very simple characters (those with few turning points) as highlighted earlier in this chapter. Over-reduction, described above in Figure 4.14 is easily eliminated by a user training session which will ensure a database match is found before the reduction begins to alter the character shape.

The simplicity of the technique, coupled with the promising results, encouraged us to investigate methods of similar ease which might be able to resolve the problem of more reliably recognising the simpler characters.

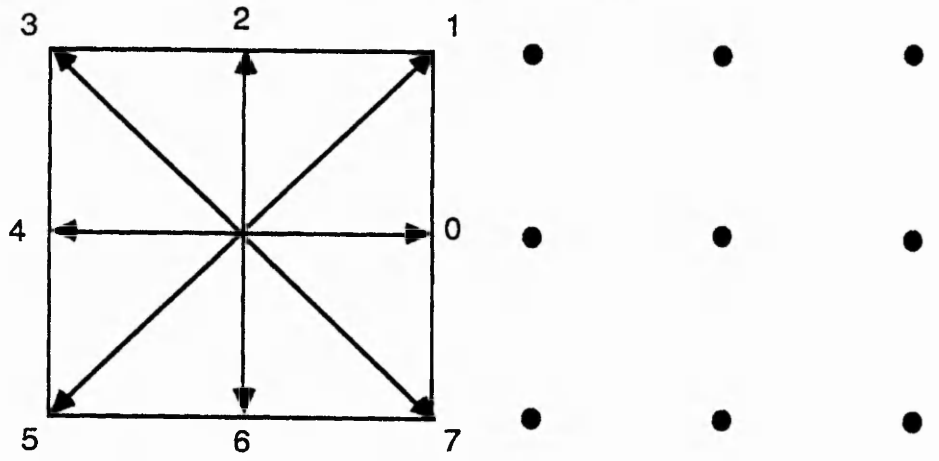
5. FREEMAN VECTOR ANALYSIS

5.1. Introduction

This technique is based on an approach used in a number of papers by Herbert Freeman on curve analysis. It was selected because it is simple to implement and showed promise in memory usage and processing times when considering some future real time implementation. It was also felt that it could also overcome the shortcomings found in the X-Y algorithm (discussed in the conclusion to the previous chapter). There are two alternative techniques [81] for the chain coding of arbitrary plane curves. The two encoding mechanisms are described below, one based on a hexagonal grid configuration, and the other on a square grid configuration.



(a) Hexagonal Grid



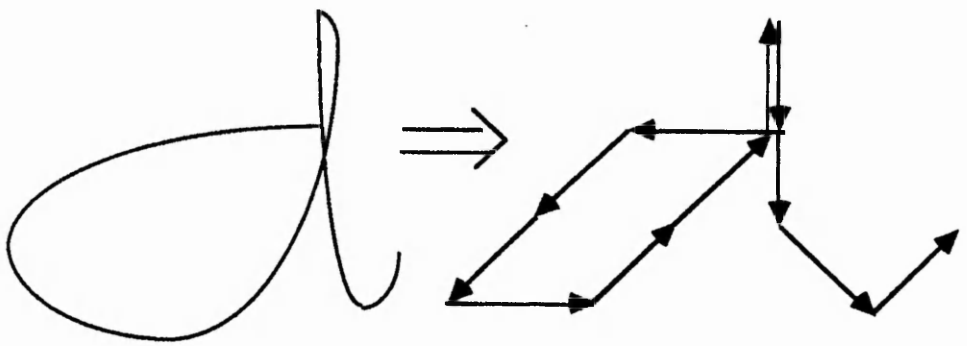
(b) Square Grid

Figure 5.1 - Chain Coding Processes

The hexagonal grid has the advantage that the vectors are of equal length, making manipulation simpler. Rotating the curve through 60° does not distort the curve shape. However, the square grid has the advantage that it is compatible with the co-ordinate grid adopted for the majority of graphics input devices, including the data tablet used for the capture of user writing. This makes the square grid encoding technique the obvious choice for encoding our characters.

5.1.1. Theory

Given a point on a continuous curve, the next point can assume one of eight possible adjacent positions. Assigning digits 0 to 7 to represent these eight positions, and starting with the one horizontally to the right as 0, the others are numbered sequentially in an anticlockwise direction. Vector directions 0,2,4,6 are of a unit length, while vector directions 1,3,5,7 are of length $\sqrt{2}$. If we take an example of a character 'a' encoded using the basic Freeman approach:-



'45501126671'

Figure 5.2 - Basic Freeman Coding Approach

5.2. Modified Freeman Algorithm

5.2.1. Theory

By only encoding the curve with set length vector elements another degree of quantisation is introduced. This is probably acceptable for curves and lines produced in, say, sketching. However, in order to retain the character features for a writer who produces very small characters, we would have to either have very small unit vectors or variable length unit vectors, related to the size of a particular writers script. The approach adopted was to use the eight vector directions to determine the character path, but to allow variable length travel in any one direction. The eight vector directions, 0 to

7 represent a 360/8=45° octant.

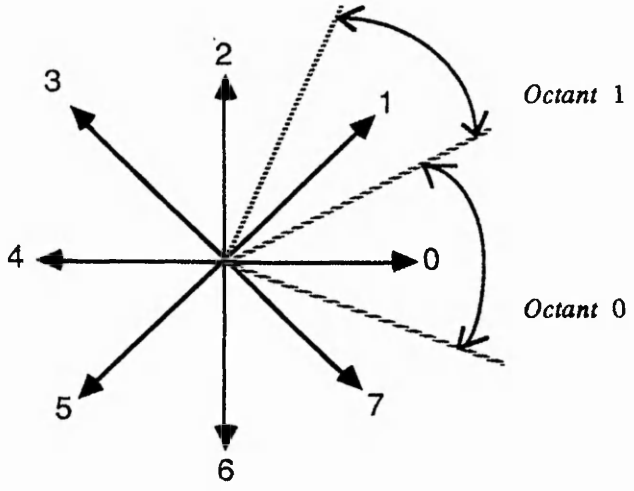


Figure 5.3 - Octant Boundaries

Therefore, a particular portion of the curve is said to be travelling in a particular quantised direction if its incremental direction keeps within the bounds of the associated octant boundary. For example, it will be deemed to be traveling in direction '1' as long as its incremental directional angle is between 22.5° and 67.5° (45° ambient).

The character curve is initially quantised by the graphics tablet into a series of (x,y) co-ordinates. Starting from the first co-ordinate pair, the incremental distance to the next point is calculated:-

$$(x_0,y_0),(x_1,y_1),(x_2,y_2),\ \ldots\ldots\ldots(x_n,y_n).$$

$$d_1=[(x_1-x_0)^2+(y_1-y_0)^2]^{1/2} \qquad (5.1)$$

and the direction of travel from the first to the second point is:-

$$\theta_1=|arctan[(y_1-y_0)/(x_1-x_0)]| \qquad (5.2)$$

However, in order to determine the exact octant, we must recalculate θ_1 as an angle over the complete range 0 to 360°, ie:-

if $[(y_1 - y_0) \geq 0]$ and $[(x_1 - x_0) \geq 0]$ $\theta_1 = \theta_1$ (5.3)

if $[(y_1 - y_0) \geq 0]$ and $[(x_1 - x_0) \leq 0]$ $\theta_1 = 180 - \theta_1$ (5.4)

if $[(y_1 - y_0) \leq 0]$ and $[(x_1 - x_0) \leq 0]$ $\theta_1 = 180 + \theta_1$ (5.5)

if $[(y_1 - y_0) \leq 0]$ and $[(x_1 - x_0) \geq 0]$ $\theta_1 = 360 - \theta_1$ (5.6)

Having calculated the angle, we can thus determine the quantised direction of travel (digit 0 to 7). Therefore, between each point on the character curve we can calculate:-

- the linear distance
- the direction of travel

5.2.2. Encoding Mechanism

Therefore, if we take our character 'a' of Figure 5.2, we can see how this modified algorithm encodes the curve.

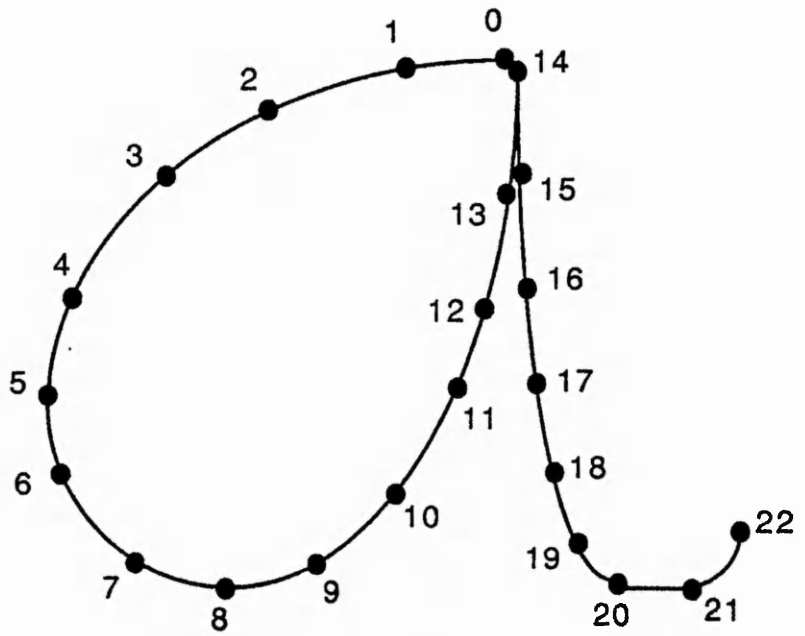


Figure 5.4 - Character 'a' as Output from Graphics Tablet

Table 5.1 gives a breakdown of exactly how the character curve is quantised into its Freeman vectors.

Points	Distance	Angular Direction	Quadrant
0 - 1	5	187	4
1 - 2	10	204	5
2 - 3	8	225	5
3 - 4	7	237	5
4 - 5	6	246	5
5 - 6	3	265	6
6 - 7	4	297	7
7 - 8	4	355	0
8 - 9	5	20	0
9 - 10	7	41	1
10 - 11	7	46	1
11 - 12	6	52	1
12 - 13	6	75	2
13 - 14	5	84	2
14 - 15	6	268	6
15 - 16	6	270	6
16 - 17	6	272	6
17 - 18	5	275	6
18 - 19	5	287	6
19 - 20	4	325	7
20 - 21	4	10	0
21 - 22	3	43	1

Table 5.1 - Incremental Travel of Character 'a'

The travel of the character curve is now grouped into successive similar vector directions, each having a cumulative distance traveled equal to the sum of all its elements, as shown in Table 5.2

Points	Quadrant	Cumulative Distance	Normalised Distance
0 - 1	4	5	0.0397
1 - 5	5	31	0.2460
5 - 6	6	6	0.0476
6 - 7	7	4	0.0317
7 - 9	0	9	0.0714
9 - 12	1	20	0.1587
12 - 14	2	11	0.0873
14 - 19	6	28	0.2222
19 - 20	7	4	0.0317
20 - 21	0	4	0.0317
21 - 22	1	3	0.0238

Table 5.2 - Vector Groupings for the Character 'a'

Each cumulative distance is divided into the overall curve travel in order to produced a normalised travel. By doing this, we are able to encode the vector strings in such a form so as to be able to process characters independently of the size they were written. Thus we produce the final Freeman encoded vector string '45670126701'. This contains 11 vectors, and associated with each vector we have a relative travel in the range 0.0 to 1.0. This weighting allows one to quantise the character curve with a high degree of accuracy. The character is represented in an ASCII string as shown below:-

$$char \quad V_0.L_0 \quad V_1.L_1 \quad V_2.L_2 \quad \ldots\ldots\ldots\ldots \quad V_n.L_n \quad < CR > \qquad (5.7)$$

$$where \qquad L_0 + L_1 + L_2 \ldots + L_n = 1 \qquad (5.8)$$

Therefore, our character 'a' is represented by the string:-

a 4.04 5.25 6.05 7.03 0.07 1.16 2.09 6.22 7.03 0.03 1.02 < *CR* >

From the encoded string we can reconstruct the character shape.

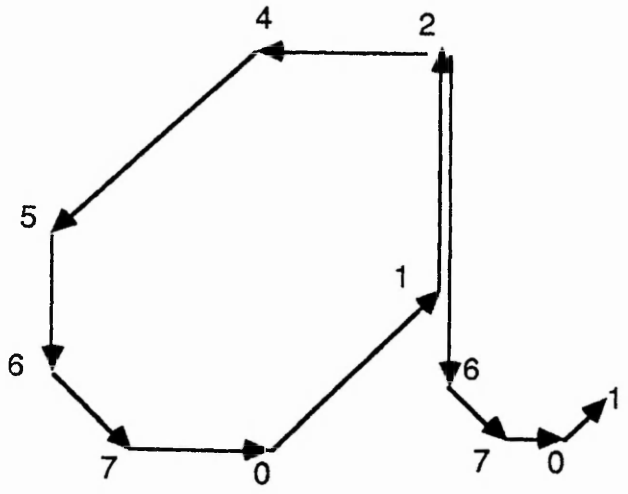
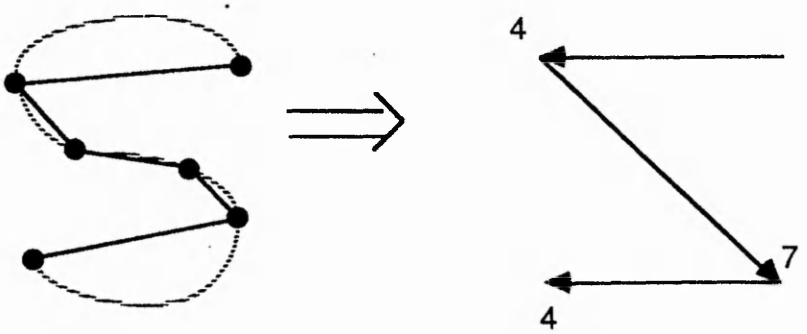


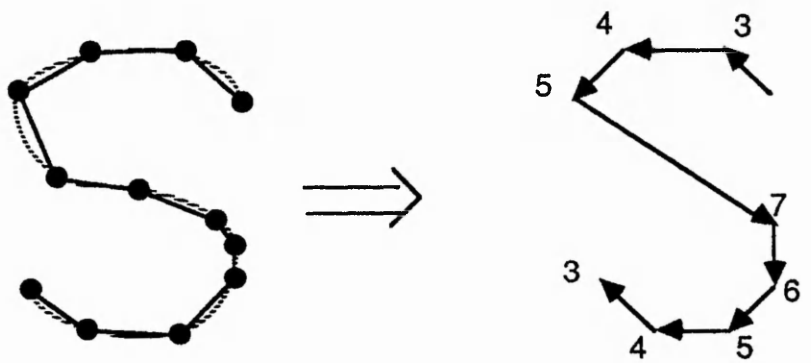
Figure 5.5 - Reconstruction of Character 'a' from Freeman Encoding

5.2.3. Vector String Distribution

An initial analysis of character encodings produced by a small number of writers showed that the size of the vector string produced varied greatly in size from as few as 1 vector to as many as 20 or so vectors. In general, the fewer the points describing the character, the less the amount of curvature information that can be extracted. As a result, the fewer the number of vectors in the encoded string. Consider the cases of the two characters 's' given below:-



(a) 's' described by 6 tablet co-ordinates



(b) 's' described by 14 tablet co-ordinates

Figure 5.6 - Character Curvature Variability

The number of encoded vectors produced from a character curve has been found to depend on three main factors:-

- character complexity
- character size
- number of points in the character (proportional to speed of formation)

The main factor is the character complexity. The more complex the character is to form, the greater the number of inter-octant transitions, and thus the greater the number of vectors needed to describe the character path.

The other two points do have some co-relation. As can be seen from Figure 5.6 a more slowly written character will retain the character shape better than a very quickly written character. Our example shows only 3 vectors

describing the character 's' which has been written quickly, while it is described by 8 vectors when written more slowly. Note that the pen traces the same path in both examples. Therefore the detail of the character, as described by its Freeman vector string is dependent on the data rate of (x,y) co-ordinates from the tablet and the tablet accuracy. However, the reduction procedures described later in the chapter attempt to filter out both user and tablet dependencies.

Character size is, as has been mentioned, related to speed of writing. Generally small characters are formed more quickly than larger characters. However, very small character shapes do tend to be influenced by the resolution of the graphics tablet also. In extreme cases the accuracy of the tablet (or rather lack of it) will distort the character shape and in so doing create an encoding with more vectors than would be expected.

If we consider the length of the vector strings produced for a particular character 's' we can determine the mean vector length and compare it with the mean vector length for the complete character set.

Number of vectors	*Number of combinations*
3	7
4	103
5	256
6	173
7	89
8	21
9	9
10	3

Producing a length distribution:-

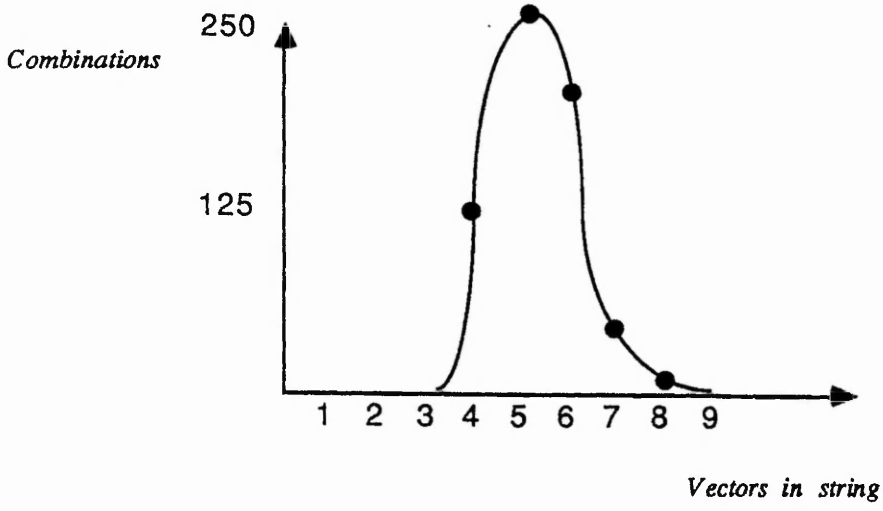


Figure 5.7 - Vector Length Distribution for the Character 's'

The mean vector length for the 's' is 4.80. For a more complex character it would be much higher.

The questions which must be resolved now are:-

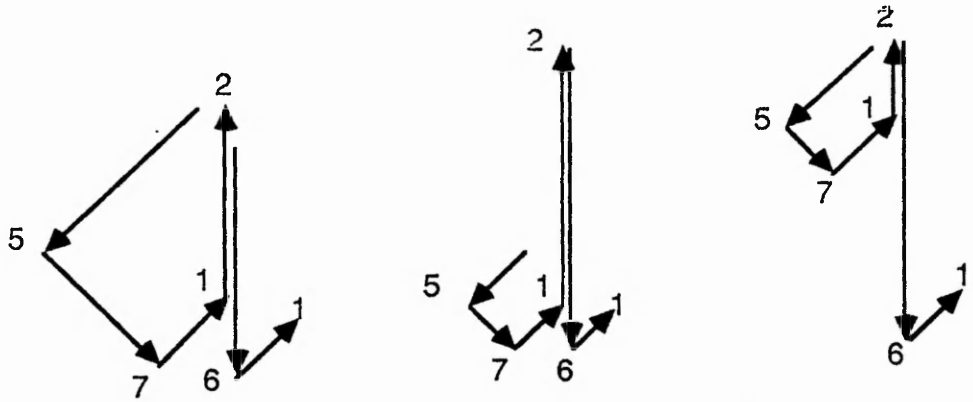
- For the lower case alphabet (a to z) how many different vector strings will be produced for a single writer, and ultimately how many more different vector strings will be additionally produced as more users writing is encoded and added to the database. Too many alternatives may ultimately be too much to handle for real time operation.
- Can a particular vector string for one character be sufficiently unique so as to be able to distinguish it from all the other vector strings in the database. Too much ambiguity between different character vector strings will negate the effectiveness of the algorithm.
- Related to both these points, will the algorithm extract sufficient unique character information.

The following sections describe the evolutionary steps leading to the algorithm in its present form.

5.3. Original Freeman Analysis

Initially it was decided to attempt recognition of a character by the analysis of the path it describes alone. In other words to analyse the vector string in isolation. Possible character identities would arise when a match of identical vector paths was found between the unknown character and some previously analysed known character. The advantage here over the dynamic time warping method favoured by some authors in Chapter 1 is the speed of search and detection. It was hypothesised that the character matches would be found to divide into neat subgroups which could be further processed in order to determine the character identity within the subgroup. If we show an example to illustrate the reasoning behind this assumption. Characters 'a',

'd' and 'q' are candidates for a subset as they trace very similar character curves as drawn by the majority of writers. Therefore we would expect to produce a number of identical vector strings.

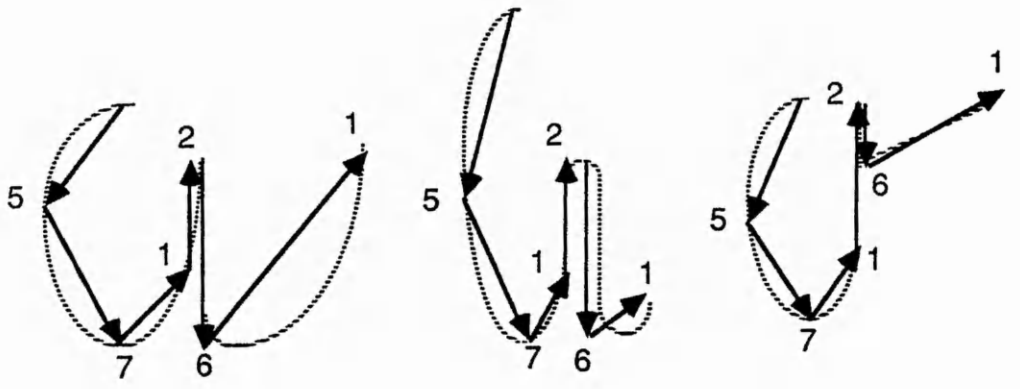


'a' 571261 'd' 571261 'q' 571261

Figure 5.8 - Freeman Character Subgroups

One such vector string which describes all three characters was found to be downstroke. Therefore, by calculating its relationship to the start point of the character and its size to the size of the curved portion of the character a decision can be made as to its most probable identity, 'a' 'd' or 'q'. Other subgroups having similar shapes can also be identified, 'h' and 'n',

However, it was soon shown, by analysis of the vector strings produced for the character set, that the assumption was not at all valid. The vector string we show as an example ('571261') was found also to describe other characters whose basic shape is nothing like the shape of 'a', 'd' or 'q'. Figure 5.9 shows three such examples of characters 'w', 'h' and 'r' which could also be encoded to produce the vector string '571261'.



'w' 571261 'h' 571261 'r' 571261

Figure 5.9 - Dissimilar Characters Exhibiting the same Vector Strings

It was apparent that a particular character string, taken in isolation, could describe quite a number of characters in the alphabet set. This diverse representation was found to be mainly due to two factors:-

- Character slant - for example, a slanted 'u' often has the same vector string as an 'a'
- Tops and tails - often produced by writers. These have no bearing on describing the character shape. They are actually part ligatures, there shape and position relating to the characters produced immediately before and after the present one in the word.

Hence it was decided that to attempt to properly disambiguate characters it would be necessary to take into consideration the relative sizes of the vectors in the encoded string. (As in the case of the XY algorithm, relative vector size analysis allows us to compare characters directly, regardless of their size).

5.3.1. Modified Freeman Analysis

The whole essence of this technique for character encoding is that the unknown character is compared against a database of alternatives and subsequent entries from the database, with matching vector paths, used for further analysis. The chances of finding two or more matching vectors strings from a database search would be highly unlikely unless the database contained a large number of writer examples. Hence, some rationalisation of vector string length was required. In our example encoding in Figure 5.5, the vector string is 11 vectors long. In fact, for many character encodings, 10 or more vectors is quite normal. However, quite a few of the vectors

produced are as a result of a single data point quantisation. These vectors contribute very little to the overall character shape. By applying a lower threshold we can eliminate the very small vectors and so rationalise the string length. Initially, a threshold of 0.04 was chosen, 1/25*th* of the total travel. All vector contributions falling below this value being discarded. Therefore our example character would be reduced as follows

ORIGINAL STRING

a 4.04 5.25 6.05 7.03 0.07 1.16 2.09 6.22 7.03 0.03 1.02 < *CR* >

AFTER REDUCTION

a 4.04 5.25 6.05 --- 0.07 1.16 2.09 6.22 --- --- --- < *CR* >

i.e. four vectors have been eliminated. However, now the total quantised travel no longer adds up to unity, but to 0.88. Therefore, it is necessary to re-normalise the component travels to return to a value of 1.0.

a 4.(04/0.88) 5.(25/0.88) 6.(05/0.88) 0.(07/0.88) 1.(16/0.88) 2.(09/0.88) 6.(22/0.88) < *CR* >

Giving

a 4.05 5.28 6.06 0.08 1.18 2.10 6.25 < *CR* >

This 7 vector string has not lost any significant information contained in the 11 vector string. The overall character shape is not lost, in fact, the tail on the end of the character has been eliminated, and tops and tails are redundant information in the analysis of lower case unconnected script. In some instances they have been shown to be misleading. Therefore, in this instance, the removal of the very small vectors has enhanced the character shape by removal of the unwanted tail.

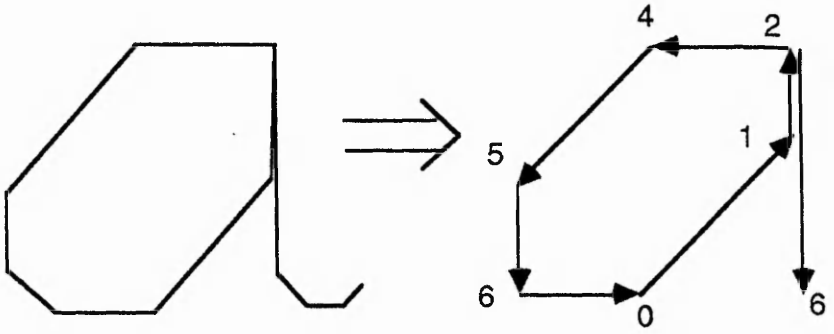


Figure 5.10 - Low Pass Filtering of The Freeman Vector String

A breakdown of the vector strings produced for the complete data set of 112 users, having been passed through the filter show how the strings are distributed. (Table 5.3).

Vectors in String	Number of Strings	Cumulative String No.
2	164	164
3	455	619
4	837	1456
5	965	2421
6	1002	3423
7	1027	4450
8	1084	5535
9	1095	6630
10	781	7411
11	539	7950
12	323	8273
13	23	8296

Table 5.3 - Freeman Vector String Distribution.

The highest string length incidence is for strings containing 9 vectors. The mean vector string length is calculated at 6.70. As with the X-Y Trend algorithm (Chapter 4), a particular representative code for a character will be produced as a result of averaging out the relative trend distances for all similar characters with identical vector strings produced as a result of encoding the test data sets. The technique is the same as that described in section

4.1.2.2. Therefore, the unknown character can be compared against the elements in the Freeman database in order to find all such elements which have the same vector string as the unknown character. Any such discoveries in the database will be fitted against the unknown character in order to calculate a measure of fit which is given as a percentage figure 0% - 100%. 0% indicates a very poor fit, and 100% a perfect fit. Database construction, searching and character fitting are described in chapter 6.

Results of the initial recognition performance using this technique were quite promising (~70%) on an initial user test set of 25 writers, as used before on the XY algorithm. However, the number of Freeman vector strings which could represent a single character did appear to be quite large (an average of over 100 representations per character). In fact 97 different representations of character 'a' in the database were produced as a result of encoding only 150 character 'a's. Even a single writer, producing consistently shaped characters would produce quite different Freeman strings (varying in both vector size and vector string path) from one character to another. The maximum length of these filtered encodings was 13 vectors. The database constructed from the 25 user set had over 2700 vector string entries stored in over 100 Kbytes of ASCII codes.

Apart from the large size of the database, the main concern was the amount of computation required on the larger vector strings as encoded, and the amount of time that would be required to search the database for any matches. With an average filtered vector length of around 7, it did appear that in the ultimate aim for a user independent system:-

- database size would soon exceed any manageable proportions
- vector string size would mean a heavy load on vector string manipulation in database searching and subsequent processing required to fit the alternatives.

If we were to assume that a database could conceivably contain every single possible vector combination in all strings from length 1 to 13 we would soon reach serious size problems. Total number of possibilities is:-

$$No.\ of\ possible\ vectors = \sum_{n=1}^{13} 8*(7)^{(n-1)}$$

Giving,

Vectors	*Combinations*	*Cumulative*
1	8	8
2	56	64
3	392	456
4	2744	3200
5	19208	22408
6	134456	156864
.	.	
.	.	
13	$1.1073*10^{11}$	

After five vector combinations the database size can be seen to be becoming quite large for representation of a mere 30 or so character shapes.

5.3.2. Reduced Freeman Vector Algorithm

Our problem is that, in allowing all the shape possibilities to be present in the database so as to be able to recognise characters independent of a particular writer we would have to allow a very large number of possible vector combinations, adding new vector combinations every time a totally new writer would require to use the system. In order to approach writer independence the size of the database would approach the maximum value calculated above for 13 vectors.

Therefore it was decided to approach the problem of vector string length from another direction. Instead of working around vector lengths up to 13 vectors long, the question was posed - How few vectors could a particular character be constructed from which would still uniquely differentiate that particular character from the others in the alphabet set?

The reason for the question is the underlying and absolutely vital necessity for a system at the end of the day which will operate in real time. If we were to allow any number of vectors in a character string up to the maximum of 13 the size of the database would tend to $1.1073*10^{11}$. If we can limit the number of string options to search through, there is a far better chance of transposition into a real-time environment.

If we consider the lower case alphabet (a-z) we can construct the following vector profile for each character derived from the most common style of character formation as written by the 112 user test set.

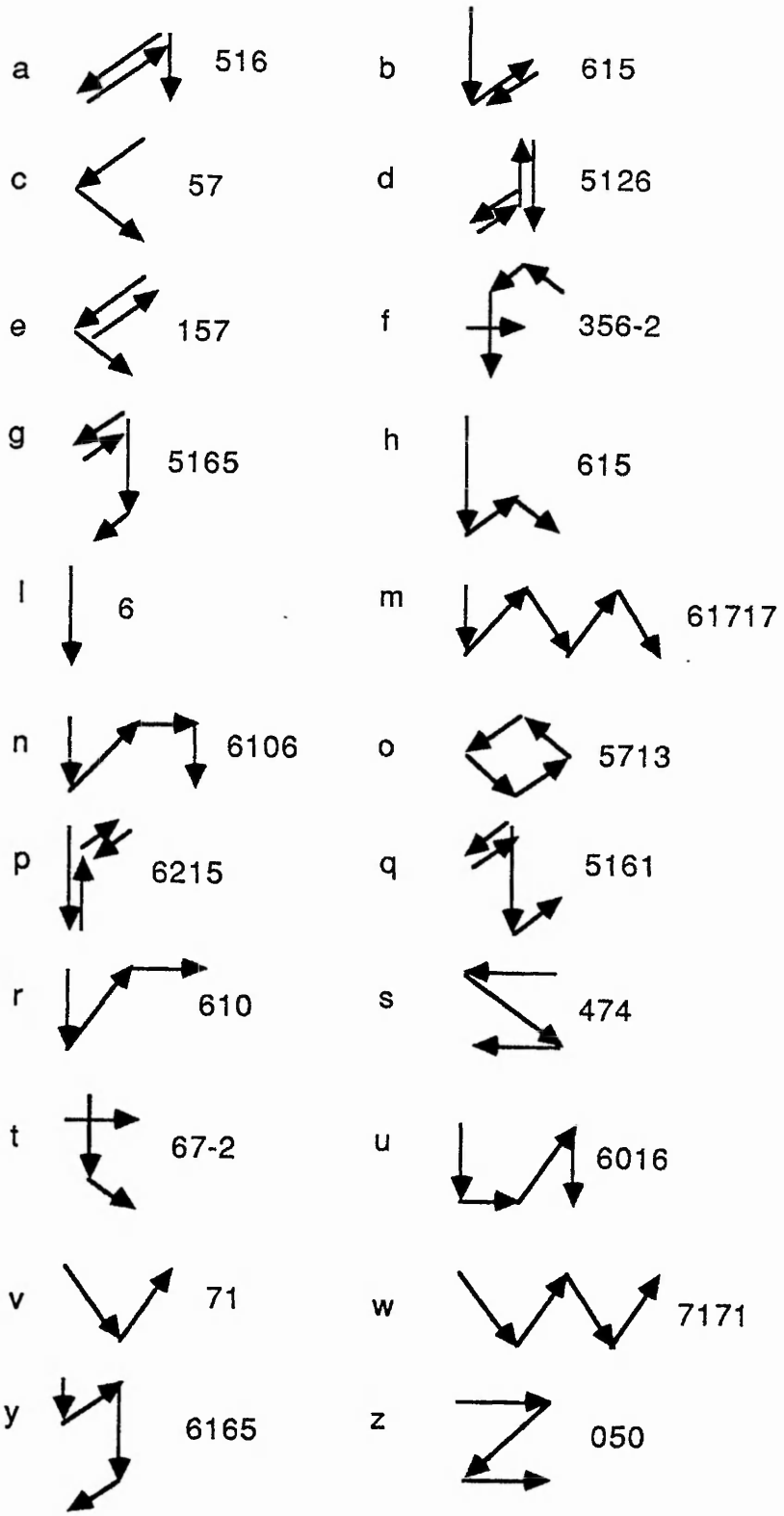


Figure 5.11 - Idealised Freeman Encoding For Lower Case Alphabet

If every writer were to produce the vector strings describing this idealised sub-set we would have no problem whatsoever. No one vector string is the same as another, so we would not even need to consider the relative sizes of the vectors in order to decide the character identity. This is not the case, even for a single writer. A large variety of writing styles has been observed. However, by limiting the maximum size of the vector string for any one character to a maximum of 5 vectors we could still safely describe all the characters (in our idealised set only one character, 'm', has 5 vectors). At present the database contains only 2421 vector strings of 5 vectors or less out of a total of 8296 vector strings. This represents only 29% of the total number of vector strings.

By so doing we

1. limit the size of the database to a sensible number of vector strings
2. reduce database searching times and speed up the vector string manipulation and comparison procedures which account for a significant amount of algorithm time.

The problem, however, is the reduction of a large vector string (up to 13 vectors long) to a five vector string without the loss of any information vital to the unique identity of a particular character.

5.3.2.1. Initial Vector Reduction Technique

If we consider our character 'a' of Figure 5.10. This has already had the very small vectors eliminated, producing a filtered result:-

$$a\ 4.05\ 5.28\ 6.06\ 0.08\ 1.18\ 2.10\ 6.25 < CR >$$

Hence we have a seven vector string to be reduced to a five vector string in order to attempt a match against entries in the database, maximum size five vectors. Initially the vector string is searched in order to determine which vector is the smallest. Vector reduction by eliminating the smallest element in the string will ensure that distortion of the original character shape is minimised:-

V_S = *smallest vector*
V_i = i^{th} *vector*
L_S = *smallest vector length*
L_i = *length of* i^{th} *vector*

for (i=0 *upto* i < *no of vectors*)

if ($L_i < L_S$) $V_S = V_i$
$L_S = L_i$ (5.9)

Taking our example character 'a' we find that the smallest contributory vector is the first vector, in direction '4'. However, on occasions two vectors may be found to have equal smallest contributory travels. In such cases, the vector appearing first in the string is taken as the reducing vector. This is an arbitrary choice. Unless the initial vector string is only 6 vectors in length, the other smallest vector will be the next to be reduced anyway.

The elimination procedure is to remove the vector with the smallest travel from the string and to add its contributory travel to one of the vectors either side of it. If the smallest vector is either the first or the last vector we have no problem in choosing which vector travel will have the smallest vectors travel added to it. It will be the second or penultimate vector respectively. Otherwise we have to choose between the two neighbours. In an attempt to preserve the overall shape as long as possible it was decided that the smallest vector should be incorporated into whichever of its neighbours has has the least angular difference to it. In the event that it has an equal angular difference between both neighbours, its contribution is added to the neigbour which has the lower relative vector length. If both neighbours have the same angular difference and the same relative vector length it is arbitrary as to which vector the smallest should be added. This again is an attempt at preserving the character shape. Consider the case of identical angular differences and relative lengths below.

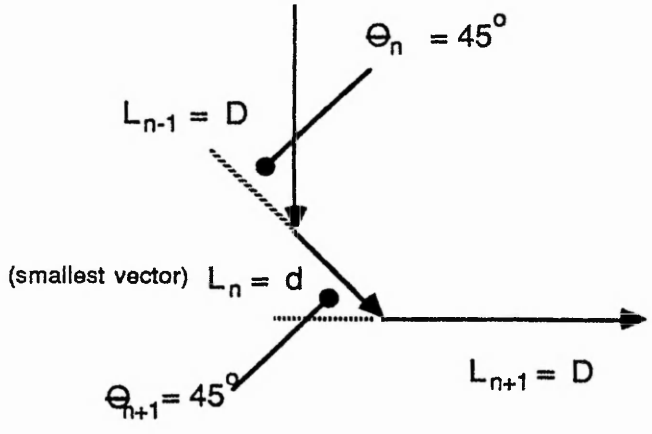


Figure 5.12 - Arbitrary Reduction Decision

Therefore we produce a series of conditions for a single vector reduction:-

vector string = $V_0, V_1, V_2, \ldots\ldots V_n$

vector lengths = $L_0, L_1, L_2, \ldots\ldots L_n$

if $(V_S = V_0)$

$R = 0$

$L_1 = L_1 + L_0$ (5.10)

else if $(V_S = V_n)$

$R = n - 1$

$L_{n-1} = L_{n-1} + L_n$ (5.11)

else if $(|V_S - V_{S-1}| \leq |V_S - V_{S+1}|)$

$R = S$

if $(L_{S-1} < L_{S+1})$

$L_{S-1} = L_{S-1} + L_S$

else

$L_{S+1} = L_{S+1} + L_S$ (5.12)

else if $(|V_S - V_{S-1}| < |V_S - V_{S+1}|)$

$R = S$

$L_{S-1} = L_{S-1} + L_S$

else

$L_{S+1} = L_{S+1} + L_S$ (5.13)

Having added the smallest vectors weighting to the appropriate neighbour we must now remove the vector from the string completely:-

vector for *removal* = V_R

for $(i = R$ *upto* $i < n)$

$V_i = V_{i+1}$

$L_i = L_{i+1}$

$n = n - 1$ (5.14)

In our example $V_R = V_0$, therefore a first vector reduction (using equations 5.10 and 5.14) produces:-

a' 5.33 6.06 0.08 1.18 2.10 6.25 < *CR* >

This six vector string must be reduced once more. The new smallest vector is the new second vector in direction six. Therefore we must decide which of its neighbouring vectors will incorporate its contributory relative length. From equation 5.12 we must determine the angular difference between vectors '5' & '6' and vectors '6' & '0'. The difference '5'-> '6' is 45°, while the

difference '6'-> '0' is 90°. Therefore we choose vector 5 to add the length to, having given the smallest angular difference. Therefore, a second reduction produces:-

$$a''\ 5.39\ 0.08\ 1.18\ 2.10\ 6.25 < CR >$$

Therefore we have our reduced 5 vector string. If we reconstruct the shape from the string we produce the character 'a' as shown below:-

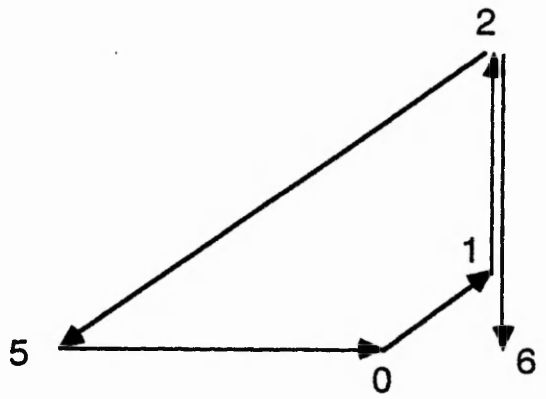


Figure 5.13 - Reduced 5 Vector Character 'a'

The character 'a' has retained its unique features over the course of the reductions from the initial 11 vector encoding, although the shape has become somewhat more angular than the original quantisation (Figure 5.5). Performing this process of vector string reduction on the set of 25 writers results in only 1900 encodings held in 59000 ASCII bytes. This is a reduction of 30% in the number of vector strings and a reduction of 45% on actual physical size. This produced very encouraging results on the 25 writer set (> 90% recognition). However, attempting recognition on the script of a new writer, not represented in the database did tend to give variable results (ranging from 50-95% recognition). The reason for the poor recognition of some script was observed to be a result of not having the particular encoding for a certain character in the database. This would produce either a non-recognition or sometimes a mis-recognition if the particular vector string was present in the database, but representing another character. In such cases the character fitting produced a very poor fit.

We have already seen that around 22000 unique vector combinations can be produced for vector strings up to 5 in length. Also, a number of characters can produce identical vector strings. In order to be able to adequately recognise characters as written by a variety of writers, one approach would be to

keep adding to the database every time new vector string representations are identified for a particular character. This did not appeal particularly, since it appeared that a completely user-independent database may take a large amount of users and time to achieve and would again tend to an unwieldy amount of vector entries.

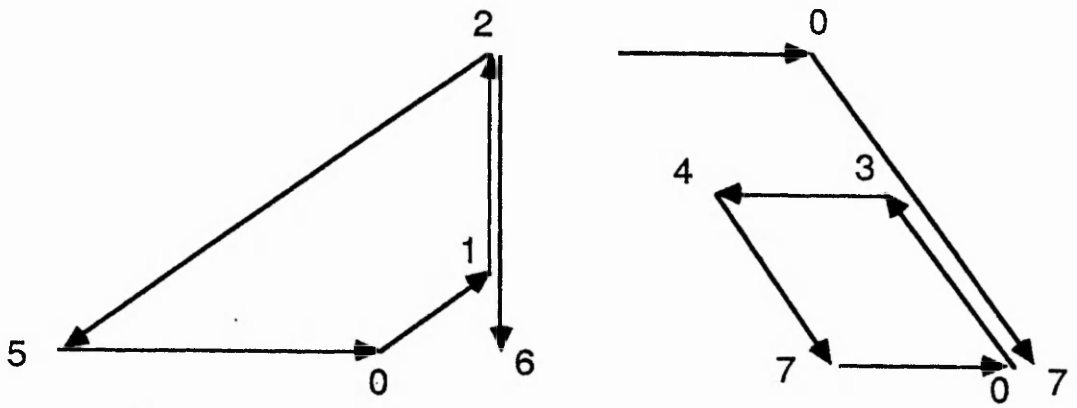
It was observed that vector reduction tended to stylise the character shapes towards the basic encodings shown in Figure 5.11. It was also noted that where a mis-recognition or non-recognition occurred due to the encoding not being in the database, that a further reduction (from 5 to 4, or from 4 to 3) would produce a correct match. If we consider our character 'a':-

$$a'' \; 5.39 \; 0.08 \; 1.18 \; 2.10 \; 6.25 < CR >$$

No vector string '50126' may be present in the database for the character 'a'. However, a further reduction to four vectors would produce:-

$$a''' \; 5.39 \; 1.26 \; 2.10 \; 6.25 < CR >$$

There is more likelihood of finding this four vector representation of the character 'a' ('5126'). This was seen to be a better alternative to keep updating the database. It would be necessary if a completely new style of character formation was found.



(a) Usual character 'a' (b) Alternative character 'a'

Figure 5.14 - Inclusion of a new character formation style

However, continued vector reduction does mean that we are not required to find every unique vector encoding for every writer. We can control the size of the database, but will this produce the desired recognition rate for a writer who has not been incorporated into the database?

Obviously, by continually reducing the vector string we will ultimately reach the stage where we will be removing vectors from the string which would be vital to the overall unique identity of the character. For our character 'a' this would not be so until we have reduced to two vectors:-

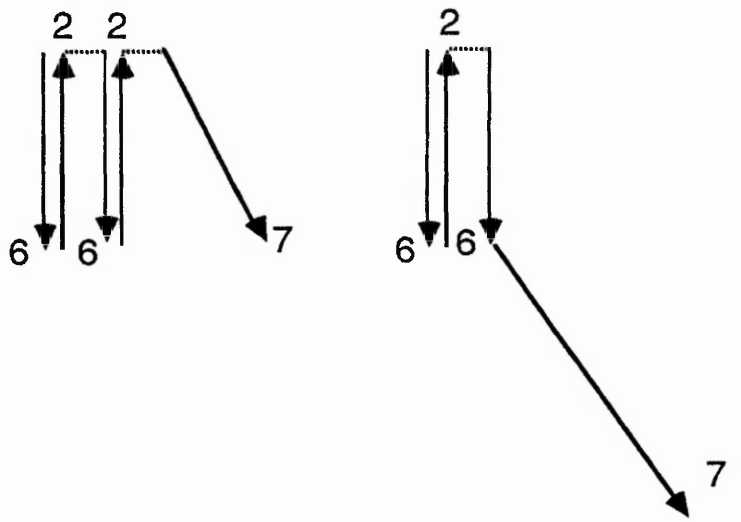
a''' 5.39 1.36 6.25 < *CR* > - shape still present

a'' 5.39 1.61 < *CR* > - shape is lost

However, with other characters the shape would be lost well before reduction to only 2 vectors. Consider the character 'm':-

m 6.19 2.23 6.19 2.18 7.21 < *CR* >

m' 6.19 2.23 6.19 7.39 < *CR* >



(a) Original 'm' (b) Reduced 'm'

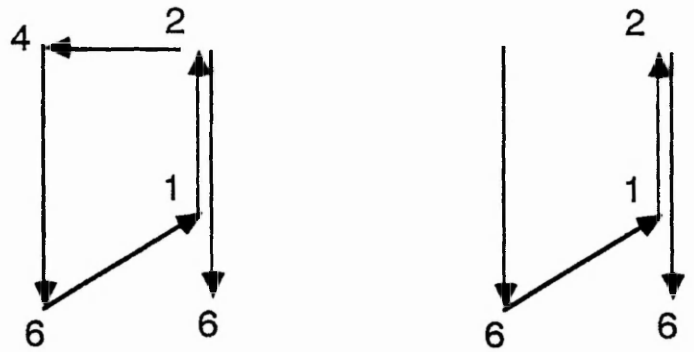
Figure 5.15 - Invalid Vector Reduction

The vector reduction has produced a very distorted result. In many instances it was found that where vector reduction did destroy the integrity of the character shape, the resulting vector string, when matched against the entries in the database would either find no match, or would match with a very poor confidence of fit (due to its distorted shape).

Results of this continued reduction technique were promising. When applied to the whole 112 test writers, around 5000 vector entries were produced, held in 130Kbytes of ASCII strings. A recognition of around 85% was achieved. However, it was observed that a number of character mis-

recognitions were due to a flaw in the reduction technique. Results showed that less than 0.5% of the characters were not recognised compared to 4-5% before continued reduction. This problem, however, was not difficult to rectify and is corrected in the next section.

A large number of mis-recognition errors were found to be due to a common sub-set of mis-recognitions. For example, a's recognised as u's, g's recognised as q's, g's recognised as y's, e's recognised as c's, k's recognised as b's. In all such cases it was found that the reduction technique had reduced the vector string down to such a level that it had changed the shape of the character so as to look like another character. Hence a further refinement needed to be investigated.



(a) Original 'a' (b) Reduced 'a'

Figure 5.16 - Vector Reduction Flaw

The character 'a' in Figure 5.16(a) has the following vector string:-

$$a \ 4.13 \ 6.19 \ 1.20 \ 2.18 \ 6.30 < CR >$$

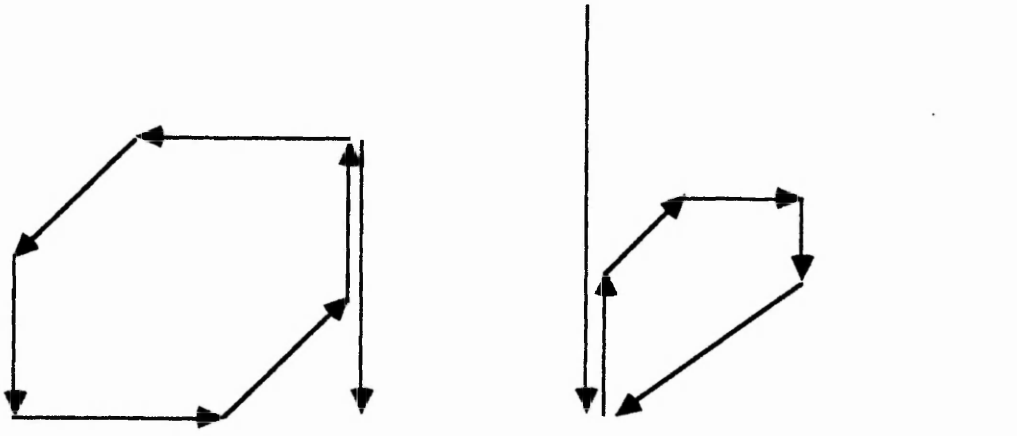
The first reduction produces:-

$$a' \ 6.32 \ 1.20 \ 2.18 \ 6.30 < CR >$$

The shape of the vector string no longer looks like the shape of a character character 'u' on the strength of the reduced vector string '6126'. Similar situations are found to arise for the other mis-recognitions quoted above. In such instances it is found that even though it is the smallest vector which is removed, it is not the least significant because it is actually important in preserving the integrity of the character shape.

5.3.2.2. Modified Vector Reduction Technique

It was decided to resolve this reduction problem by analysis of the angular variation of the vectors in the string as a means of choosing the vector to remove. A large change of direction from one vector direction to another is an indication of an important area on the character curve. Small angular differences, say + or -45^o are usually found to indicate a gradual change over the character shape as would be observed along a clockwise or anti-clockwise loop. The largest angular difference (+ or -180^o) is usually an indication of an upstroke/downstroke reversal or vice-versa.



Angular change
$-45^o, -45^o, -90^o, -45^o, -45^o, -180^o$

(i) Character 'a'

Angular change
$+180^o, +45^o, +45^o, +90^o, +45^o$

(b) Character 'b'

Figure 5.17 - Angular Variation considerations

Therefore, if we perform vector reduction between vectors exhibiting the smallest amount of angular variation, we should be able to hold the character shape longer. Hence the technique was modified so as to initially search for the two vectors which exhibited the least amount of angular variation. In many cases it will be found that more than one pair of vectors have equal lowest angular variation. In such cases the pair which also has the smallest vector among all the pairs is chosen. This smallest vector being the vector for removal. The reduction technique is the same as that described in equations 5.10 through 5.14. Therefore our character 'a' vector representation of Figure 5.16(a) will exhibit the following angular variations:-

$$a \; 4.13 \; 6.19 \; 1.20 \; 2.18 \; 6.30 < CR >$$

$+90^{o}$ $+135^{o}$ $+45^{o}$ -180^{o}

The smallest angular variation is between the 3rd and 4th vectors (1-> 2 transition) with the vector direction '2' exhibiting the smallest magnitude. Therefore we take it as the reducing vector to produce:-

a' 4.13 6.19 1.38 6.30 < *CR* >

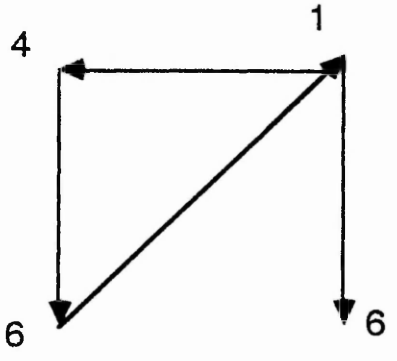


Figure 5.18 - Modified Reduction Technique

We have now retained the shape of the character 'a' through this method of angular variation analysis. The recognition rate for the 112 writer set was increased to 95% with the new technique now resolving many of the the erroneous vector strings produced by the original reduction method. However, this technique of angular difference analysis, when used in isolation can also cause problems. The problem is usually caused by the writer.

It is difficult for a writer to produce lower case unconnected script without producing a 'top' or 'tail' on a number of characters. This usually manifests itself as a small upward tick produced as the pen is brought down onto the paper and then moved (upwards and slightly to the right) to a position from which they start forming the character. The encoding of such a character usually produces an initial vector which is not part of the character shape.

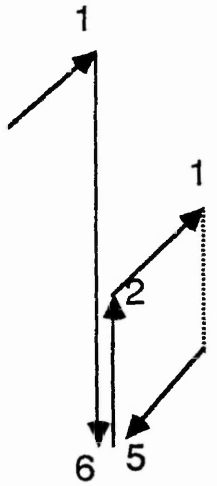


Figure 5.19 - Character tick problems

Usually the tick is only represented by a very small vector. But since the angular difference between the tick and the first contributory vector direction is so large (usually 135° or 180°) the tick will be retained in preference to the removal of valid character vectors. This, too, has been found to cause serious shape loss. Hence an additional test needed to be introduced in order to remove these ticks.

The ticks were mainly shown to occur in characters which should start with a downstroke (ie. b,h,i,j,k,l,m,n,p,r,t,u,v,w,y) encoded as vector '6'. The ticks themselves were encoded as either vector '1' or '2'. Hence an initial test was made to determine if the first vector in the string was a vector '1' or '2' whose length was small and it was followed by a much larger vector in direction angular variation analysis is performed.

5.4. Conclusions

The results obtained for the Freeman recognition technique (given in Chapter 8) have proved the method to be particularly robust. One particular advantage over the X-Y technique was found to be in instances where the character is not found as the best choice. In a number of cases where the X-Y algorithm mis-recognises a character, the alternative list does not give a clear indication of what the letter shape might be. For example, for an 'a' written, the alternatives might be:-

w:67 *h*:60 *a*:55 *g*:45 *d*:40

However, for the Freeman technique, the letter subset is usually quite apparent even though the correct letter is not the best alternative:-

d:65 *q*:60 *a*:58

This will more likely assist future word constructions from these letter string alternatives. More detail of this is given in Chapter 8, where alternative substitution is described.

This encoding mechanism also tends to retain the original character shape better through the reduction process. Hence giving better recognition where reduction is necessary.

6. CORRELATION AND DATABASE TECHNIQUES

6.1. Introduction

Once a character has been encoded into a set of basic parameters it must be checked against some reference model set in order to determine which model in the set it resembles the closest. Usually it is not possible to encode a particular character in such a manner so as to uniquely distinguish it from all other character models in the data set, unless either:-

- a particular character is clearly distinguishable in its formation than any of the others (eg. 'l'), or
- the model set is small (eg. numerals 0-9)

The level of complexity of the correlation depends on the amount and variation in the feature parameters extracted from a character curve.

The actual process of correlation is not entirely dependent on the encoding technique used, although some methods are better at detecting, say, curved elements rather than straight line elements of characters. One type of encoding where correlation is very evident is in the 'elastic matching' type recognition techniques typified by Tappert [10]. He describes a technique of feature extraction by calculating the tangential angles of various points along the character curve, together with the vertical distance of these points from the baseline of the character. These features are matched against similar features extracted from a reference set previously produced by a writer in a learning phase. An overall 'smallest difference' measure is calculated between the unknown character and each of the models in the test set by a series of recursive and dynamic programming equations. The model in the test set which produces the lowest 'smallest difference' gives the most probable identity of the unknown character.

Burr [43] also describes a similar technique of correlation known as 'warp based shape matching'. In this instance the only character encoding performed is that produced by the digitiser quantising the character curve into a series of (x,y) co-ordinates. The character is constructed by joining the co-ordinates with a series of line segments. The string of line segments produced by the unknown character are processed to produce a 'smallest difference' measure, calculated as:-

$$S(i,j)=D(i,j)+min(S(i-1,j),S(i,j-1),S(i-1,j-1)) \quad (6.1)$$

where

$S(i,j)$= *smallest accumulated difference*

$D(i,j)$= *distance measure between* i^{th} *element of curve* 1 and j^{th} *element of curve* 2

As with the previous example, recursion is applied. In both cases the character curve must be normalised before applying the 'smallest difference' measure, otherwise the subsequent processing would be invalid.

Lu and Brodersen [45] designed a Dynamic Time Warping Processor in order to allow the recognition to run in real-time, because the technique is so processor-intensive. The reference set contains 500 symbols for comparison with an unknown character. Even so, pre-matching is performed in order that only 10 or so templates are picked out for matching.

In all the above instances the 'correlation' or 'shape matching' is explicitly defined in the recognition process. However, some techniques do perform a kind of correlation but it is not explicitly defined. Badie and Shimura [18] encode a letter into a series of characteristic curves, namely arc, loop, and corner (ie. the topological features are extracted). Correlation is used to identify characteristic curves in a written word.

Other techniques perform no kind of correlation whatsoever, namely tree structure database analysis. Teitelman [29] extracts character spatial information and performs a tree search which produces a single recognition result. No alternatives are produced. In such cases where only a single result is possible, no correlation is necessary. However, such techniques have the disadvantage that they result in a black and white decision, rather than having a choice of possibilities. Correlation introduces the aspect of 'most likely' result followed by a number of alternatives. Associated with the alternative string is usually a numerical indication of the relative recognition certainty.

Closely related to correlation, and a very important factor in the overall recognition process is the database against which the unknown character encodings must be compared in order to produce a match. It was quite noticeable in the state-of-the-art study that there was very little detail given as to the construction of the database and its inter-relation with the recognition procedure. In most instances there is only a brief description of the nature of the elements in the database. Yoshida and Sakoe [23] have a reference memory pattern area containing patterns of character categories, but no indication is given to the methodology of database matching or its size. In many other instances, especially the topological feature extraction, an explicit database is not used. Recognition is performed as a series of decisions based upon the features extracted, and knowing that only a certain subset of characters exhibit certain features. Hence the decision rules

gradually eliminate elements from each subset until a final single possibility is reached. Suen et al [41] performed a state-of-the-art report on hand printed characters (mainly upper case A-Z and numerals). In it they studied databases used to represent these characters. Database size will be dependent on the type of script being input:-

- numerals
- upper case (A-Z)
- lower case (a-z)
- connected script
- any combination of the above

Database sizes quoted for hand-printed characters ranged from 8Kytes (numerals 0-9) to 64Kbytes (letters A-Z). Only 35% of the papers studied by Suen actually mentioned the size of the database. It was also noted that many of the techniques would require a rethink on the database in order to implement the algorithms on a mini- or micro-computer (ie. database searching would be too slow as it was). In no paper was the database structure mentioned.

6.2. Correlation

The techniques evaluated in deciding in the final correlation procedure are described in the following sections.

6.2.1. Theory

The method of correlation needs to be designed so that it can be applied to the analysis of both the Freeman encodings and the XY encodings transparently. This is important because we need to be able to relate the recognition alternatives produced from one algorithm with those produced by the other in order that the correlation results of each recognition algorithm can be used to produce a final alternative string. This is a result of combining the various outcomes of the separate algorithms for XY encodings (Chapter 4) and Freeman encoding (Chapter 5). The result of searching the XY and Freeman databases is such that only strings with either:-

- identical XY trend strings
- identical Freeman vector strings

are extracted from the appropriate database against which to perform correlation with the unknown character encoding. The database methodology is described later. However, assuming that we have picked out a number of alternatives against which to match the unknown character, the technique is to measure the difference between individual features in each string. The total difference calculated for all the feature elements is taken as a measure of 'goodness of fit' of one string to the other. A very small overall difference is an indication of a very good correspondence, while a very large measured difference is an indication of a very poor correspondence. The

algorithm has been amended and evolved over a number of stages.

6.2.1.1. Initial Correlation Measure - The Chi-Square Test

The initial 'goodness of fit' measure considered was the chi-square distribution. The idea of 'goodness of fit' is to compare a sample measure obtained with the type of sample one would expect from a hypothesised distribution in order to see if the hypothesised distribution function "fits" the data in the sample. In our case the hypothesised distribution function is the encoding found from the database to which is "fitted" the unknown sample. Formally, the test is given as:-

$$\chi^2 = \sum_{i=1}^{m} \frac{(O_i - E_i)^2}{E_i} \qquad (6.2)$$

where m = *no. of samples*
E = *expected result*
O = *observed result*

In this case, the expected result will relate to the the encoding found in the database, while the observed result is the encoding of the unknown character.

The smaller the computed value of χ^2 the better the "fit" between the sets of results. We want to use the equation to compare an unknown character encoding with a number of possible alternatives obtained form the appropriate database. We wish to know which of the alternatives gives the best fit to the unknown character. If we consider an unknown character producing a Freeman vector string as below:-

$F(?)$ = 4.12 5.18 1.27 6.39 0.04

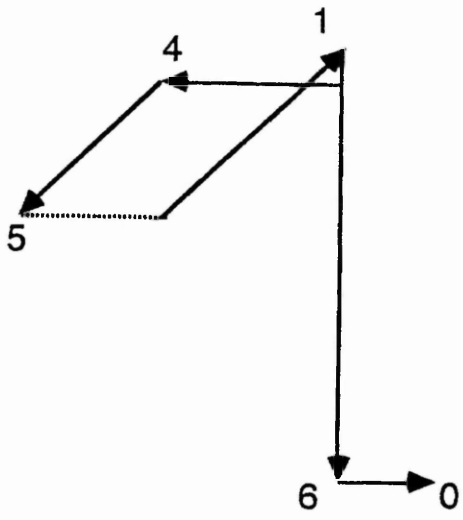


Figure 6.1 - Freeman Encoding of Unknown Character

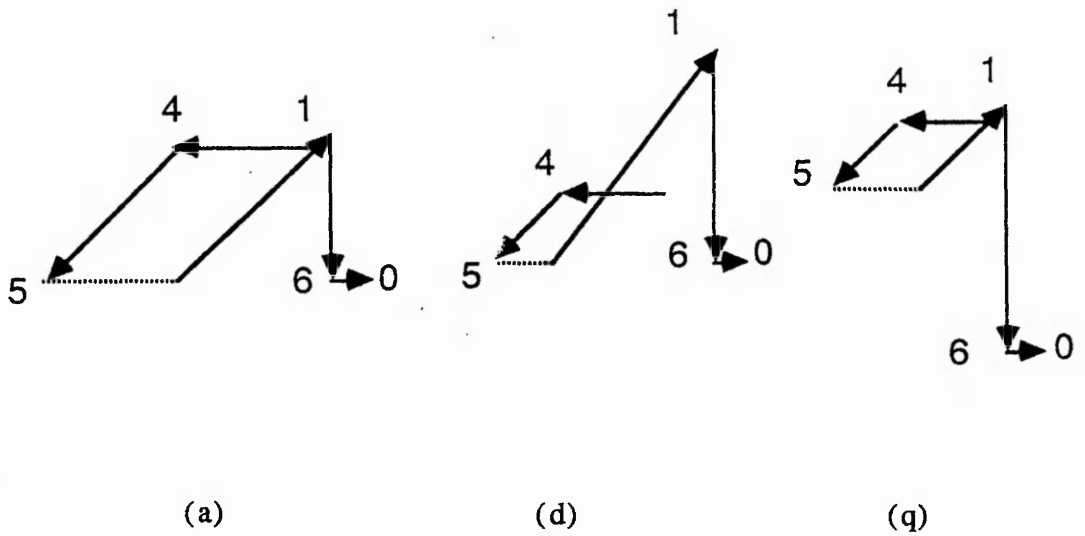
A search of the Freeman database produces three possible characters that the unknown string might represent:-

$F(a)$= 4.14 5.24 1.35 6.16 0.12

$F(d)$= 4.19 5.20 1.36 6.20 0.04

$F(q)$= 4.10 5.16 1.30 6.35 0.09

Reconstruction of the character shape for these three cases produces:-



(a) (d) (q)

Figure 6.2 - Database Alternatives

Applying the χ^2 test gives:-

$$\chi^2(?/a)=\frac{(0.14-0.12)^2}{0.12}+\frac{(0.24-0.18)^2}{0.18}+\frac{(0.35-0.27)^2}{0.27}+\frac{(0.16-0.39)^2}{0.39}+\frac{(0.12-0.04)^2}{0.04}=0.3427$$

$$\chi^2(?/d)=\frac{(0.19-0.12)^2}{0.12}+\frac{(0.20-0.18)^2}{0.18}+\frac{(0.36-0.27)^2}{0.27}+\frac{(0.20-0.39)^2}{0.39}+\frac{(0.04-0.04)^2}{0.04}=0.1938$$

$$\chi^2(?/q)=\frac{(0.10-0.12)^2}{0.12}+\frac{(0.16-0.18)^2}{0.18}+\frac{(0.30-0.27)^2}{0.27}+\frac{(0.35-0.39)^2}{0.39}+\frac{(0.09-0.04)^2}{0.04}=0.1555$$

The lowest value of χ^2 indicates the best fit. In this case the actual percentage figures are not of as much relevance as the order. Hence the 'q' encoding extracted from the database gives the best match to our unknown character as it has best fit. Fortunately, this also corresponds with a visual human analysis. The Freeman encoding of the character does indeed look most like the character 'q' from the database. However, a serious drawback was found with the χ^2 method when analysing an unknown vector which had a very small vector contained in its vector string. Consider our character to have the slightly different vector string:-

$$F(?)= 4.12\ 5.18\ 1.27\ 6.41\ 0.02$$

ie. the tick on the end of the downstroke is now only half the size as the vector string of Figure 6.1. However, the overall shape of the unknown

character is hardly changed, and it still most closely resembles the character 'q'. But if we perform the χ^2 test we get the results:-

$$\chi^2(?/a)=\frac{(0.14-0.12)^2}{0.12}+\frac{(0.24-0.18)^2}{0.18}+\frac{(0.35-0.27)^2}{0.27}+\frac{(0.16-0.41)^2}{0.41}+\frac{(0.12-0.02)^2}{0.02}=0.6995$$

$$\chi^2(?/d)=\frac{(0.19-0.12)^2}{0.12}+\frac{(0.20-0.18)^2}{0.18}+\frac{(0.36-0.27)^2}{0.27}+\frac{(0.20-0.41)^2}{0.41}+\frac{(0.04-0.02)^2}{0.02}=0.2006$$

$$\chi^2(?/q)=\frac{(0.10-0.12)^2}{0.12}+\frac{(0.16-0.18)^2}{0.18}+\frac{(0.30-0.27)^2}{0.27}+\frac{(0.35-0.41)^2}{0.41}+\frac{(0.09-0.02)^2}{0.02}=0.2638$$

The χ^2 test now indicates that the unknown character now matches the character 'd' the closest. However, visually character 'q' should still be the best match. This is due solely the the large measure of deviation produced between the last vector in the two strings. The measure of deviation is magnified by the division of the very small relative travel of the end vector. Hence the χ^2 test was discounted due to this oversensitivity when analysing very small vector deviations. The example and fit problem have been shown using the Freeman vector string, but the same problem occurs when performing the fit on XY trend strings (in fact it can produce even more alarming results due to the fact that a particular x- or y-trend can actually be zero as long as its complimentary trend is sufficiently large that the trend pair are not removed before analysis).

An important point to make here is that the measures of fit must be such that they can be compared not only between a set of fits between alternatives for a particular Freeman encoding, but also between those alternatives and the alternatives produced from its reduced vector string, and, more importantly, between the Freeman alternatives and the alternatives produced from "fitting" the XY results. Therefore, for a particular unknown character (Φ) we will produce:-

$$\begin{aligned}
&\Phi(F)= M_1(\alpha),\ M_2(\beta),\ M_3(\gamma),\ M_4(\delta),\ .....\\
&\Phi'(F)= M_1'(\alpha'),\ M_2'(\beta'),\ M_3'(\gamma'),\ M_4'(\delta'),\ .....\\
&\\
&\Phi(XY)= N_1(\alpha),\ N_2(\beta),\ N_3(\gamma),\ N_4(\delta),\ .....\\
&\Phi'(XY)= N_1'(\alpha'),\ N_2'(\beta'),\ N_3'(\gamma'),\ N_4'(\delta'),\ .....
\end{aligned} \qquad (6.3)$$

where $\{\alpha,\beta,\gamma,\delta,....\}$ = *possible character id's*
$[M],[N]$ = *correlation measure*

The measures of fit $[M]$, $[M']$, $[N]$, $[N']$ must have a linear correspondence in order that an overall result can easily be determined. Relating to this, the measures of fit are required as a percentage figure 0% to 100% for a perfect fit. The measures of fit are required to indicate the relative fit and not merely a sequential relationship to indicate the order of goodness of fit. Although the chi-square method was found to be unsuitable, it did produce

an indication of the amount of deviation for a number of alternatives from an observed result. A goodness of fit was produced by subtracting the deviation measure from 100% (which is a perfect fit).

$$Fit(?/a) = 100.0 - 34.27 = 65.73\%$$

$$Fit(?/d) = 100.0 - 19.38 = 80.62\%$$

$$Fit(?/q) = 100.0 - 15.55 = 84.45\%$$

6.2.2. Kolmogorov-Smirnov Test

This is another type of goodness of fit measure. It is preferred by some people over the chi-square test as it is found to be more sensitive and more reliable over a small sample set, where the standard deviation may not be truly representative of the the actual. Basically it determines the largest difference along the sample set between the expected and observed results and uses this as a measure of goodness of fit. Again, the smaller the difference, the better the goodness of fit. This has the advantage over the chi-square test that it is insensitive to the variations between very small vectors that makes the previous test unsuitable.

$$T = \sup_x | F(x) - S(x) | \qquad (6.4)$$

Using this technique on our second chi-square example produces the result:-

$$T(?/a) = (0.41-0.16) = 0.25 \rightarrow Fit(?/a) = 75\%$$

$$T(?/d) = (0.41-0.20) = 0.21 \rightarrow Fit(?/d) = 79\%$$

$$T(?/q) = (0.09-0.02) = 0.07 \rightarrow Fit(?/q) = 93\%$$

The problem of the small vector weighting has been overcome using this technique, the fits are ordered in the correct sequence, but the range of fits will not span the range 0% to 100% because of the nature of the algorithm. An example of the problem shows how insensitive the difference measure is. Take the unknown character:-

$F(?)$ = 1.07 6.16 0.18 2.23 4.29

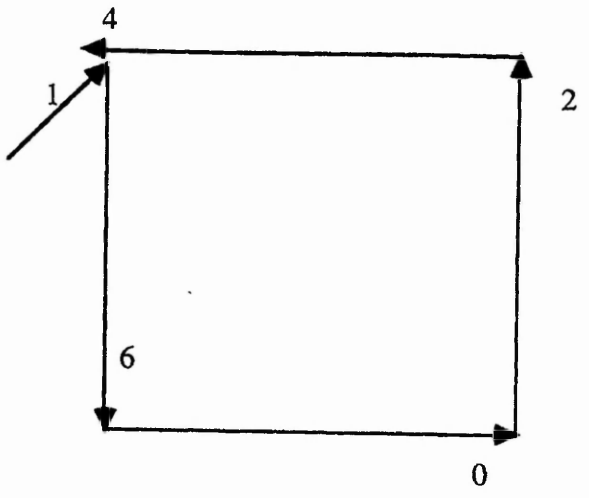


Figure 6.3 - Freeman Unknown character

The following two alternatives are found from the database:-

$F(b)$ 1.21 6.43 0.13 2.11 4.10 $F(o)$ 1.03 6.29 0.17 2.26 4.25

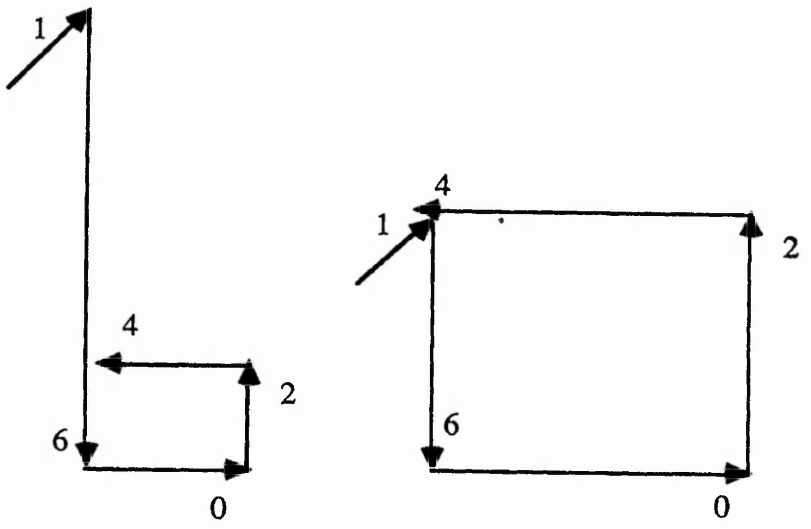


Figure 6.4 - Freeman Database Alternatives

Application of the goodness of fit measure gives the result:-

$$Fit(?/b) = (1 - |0.16 - 0.43|) \times 100\% = 73\%$$

$$Fit(?/o) = (1 - |0.16 - 0.29|) \times 100\% = 87\%$$

Although the 'b' gives the worst fit, the relative fit of the two characters to the unknown does not properly indicate how bad the 'b' is compared to the 'o'. Visually the 'b' is a very poor match to the unknown character. Therefore we would like the goodness of fit measure to be indicative of this.

6.2.3. Correlation Technique

As the established techniques given above were tried and rejected, a goodness of fit measure was devised which is really an extension of the Kolmogorov-Smirnov test. The corresponding vectors in each vector string were compared to find a difference measure. All the separate difference measures were totaled to produce an overall difference measure. As before, the smaller the difference measure the better the goodness of fit. A relative difference measure is not produced by division of the expected result as in the chi-square test in order to avoid the sensitivity problem previously encountered. Formally:-

$$D = \sum_{i=1}^{m} |O_i - E_i| \qquad (6.5)$$

where m = *no. of samples*
E = *expected result*
O = *observed result*
D = *total difference measure*

And in order to produce a percentage fit measure we get:-

$$\%\, fit = [\, 1 - \sum_{i=1}^{m} |O_i - E_i| \,] \times 100\% \qquad (6.6)$$

The benefits of the technique can best be shown by correlating the two problem cases for the χ^2 test and the Kolmogorov-Smirnov test,

- χ^2 test case (ii)

$$Fit(?/a)=(1-[|0.14-0.12|+|0.24-0.18|+|0.35-0.27|+|0.16-0.41|+|0.12-0.02|])\times 100\% = 49\%$$

$$Fit(?/d)=(1-[|0.19-0.12|+|0.20-0.18|+|0.36-0.27|+|0.20-0.41|+|0.04-0.02|])\times 100\% = 59\%$$

$$Fit(?/q)=(1-[|0.10-0.12|+|0.16-0.18|+|0.30-0.27|+|0.35-0.41|+|0.09-0.02|])\times 100\% = 80\%$$

- Kolmogorov-Smirnov test case (ii)

$$Fit(?/b)=(1-[|0.07-0.21|+|0.16-0.43|+|0.18-0.13|+|0.23-0.11|+|0.29-0.10|])\times 100\% = 23\%$$

$$Fit(?/o)=(1-[|0.07-0.03|+|0.16-0.29|+|0.18-0.17|+|0.23-0.26|+|0.29-0.25|])\times 100\% = 75\%$$

The percentage measures now also give an indication of how well or how badly an unknown character matches an entry in the database.
This goodness of fit was found to be identical to the Cramer-Von Mises test for goodness of fit dating from 1930.

6.2.4. Algorithm Result Cross-Correlation

Once we have determined the correlation results for the Freeman and XY algorithm and their respective reductions, we will be left with a series of correlation measures as shown in equation 6.3. In order to produce some means of analysing the outcome of both algorithms , the relative correlations for multiply represented characters are summed to produce an overall correlation result. This is best illustrated by means of an example. Consider the results of correlation of a character 'p':-

$\Phi(F)= 86(p)\ 81(b)$
$\Phi(F')= 88(p)\ 75(b)\ 45(g)\ 42(y)$

$\Phi(XY)= 81(p)$
$\Phi(XY')= 80(p)\ 62(b)\ 26(z)\ 15(g)$
$\Phi(XY'')= 80(p)\ 60(b)\ 47(s)\ 36(y)\ 31(g)$
$\Phi(XY''')= 78(p)\ 66(b)\ 61(m)\ 56(r)\ 53(n)\ 45(g)\ 40(h)$

Summing the correlation measures produces:-

$\Phi(F+XY)= 493(p)\ 344(b)\ 136(g)\ 78(y)\ 61(m)\ 56(r)\ 53(n)\ 47(s)\ 40(h)\ 26(z)$

The results are normalised to produce an averaged fit measure between 0% and 100%, by dividing each accumulated fit by the integer which allows the results to be as high as possible (up to 100%). In this case, this is achieved by division by 5 to give a final averaged correlation result:-

$\Phi(F+XY)$= 95(p) 69(b) 27(g) 16(y) 12(m) 11(r) 11(n) 9(s) 8(h) 5(z)

Although this method of cross-correlation is not directly comparable using the oversimplified approach detailed above, analysis of the outcome of this technique does find the alternative string simply and quickly.

6.3. Databases

Some aspects of the database size have already been mentioned in chapters 4 and 5. It is important to determine the size of the respective Freeman and XY databases which will allow script recognition for a single writer with a good degree of accuracy (95-100%). However, it is more important to be able to estimate the size of the databases which will be required in a user independent system. The recognition rate for a particular user will most likely be lower in a user independent system due to the fact that the need to cater for a much larger degree of character variability will lead to the introduction of a greater amount of ambiguous character formations in the database. Therefore, we need to show that for a user independent database:-

- (i) the size will not grow to become a limiting factor (in terms of both memory requirements and speed of recognition).
- (ii) the recognition rate will not degrade to an undesirable level in attempting to achieve user independence.

6.3.1. Analysis of Captured Data

Software tools have been designed which will allow for the automatic construction of the Freeman and XY databases from any amount of data collected from the graphics tablet. Character strokes input from the graphics tablet are saved to file as they arrive in the form of raw (x,y) ASCII coordinates. Each stroke drawn on the graphics tablet is individually reconstructed on a graphics terminal (using the GKS software package) in order for the user to key in the identity of the character as written on the tablet. These keystrokes are inserted into a datablock constructed at the start of the file. Although the subjects write two set test sentences, it is not possible to simply insert a preformatted header at the beginning of each file containing the text strings for these two sentences. This is because the great majority of lower case characters, written in isolation can be formed from more than one single stroke. Therefore each element of a particular multi-stroke character must be saved separately in the header block, because they will be recognised separately to begin with.

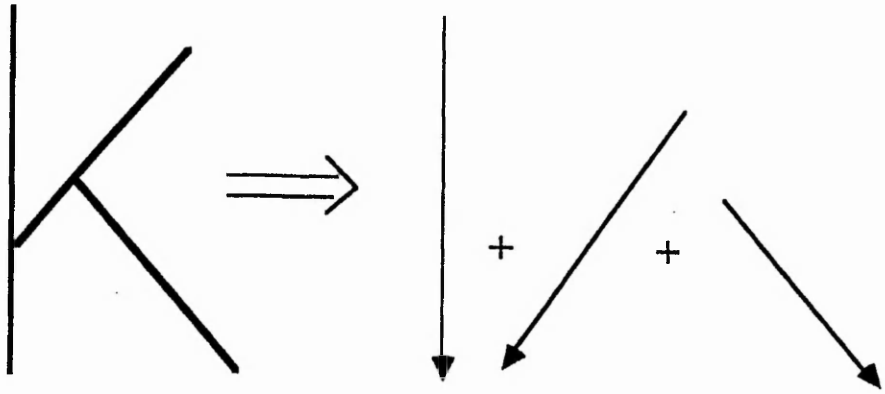
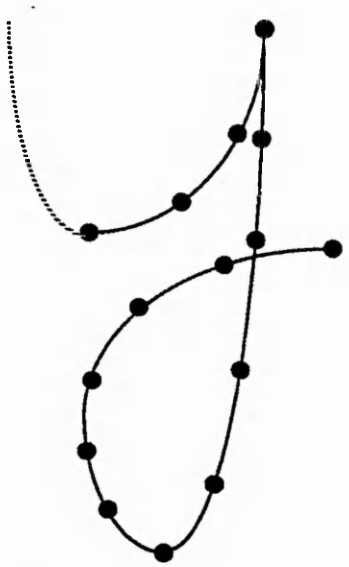
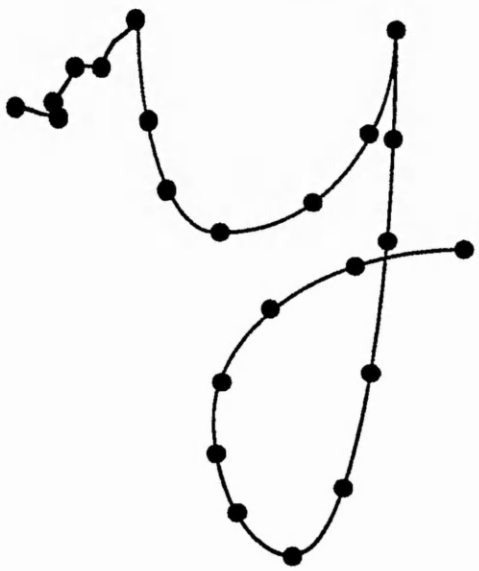


Figure 6.5 - Typical Multi-stroke Character (k)

This process of header creation is the most important part of the database construction. It is the only part that must be performed manually. We must be careful to ensure that the header stroke sequence is a faithful representation of the raw data strokes following it. Any mistakes in the header stroke sequence will cause invalid entries to be formed in the databases. The most important reason for the need for character verification is that, in some instances, the curve traced out by the pen on the paper does not correspond to the curve captured by the tablet. Two types of inconsistency may occur, both of which are usually caused by erroneous pen down detection in the graphics tablet circuitry.



(i) incomplete character capture



(ii) additional pen movement capture

Figure 6.6 - Invalid Character Curve Capture

In such occurrences the data is invalidated and does not contribute to the database construction.

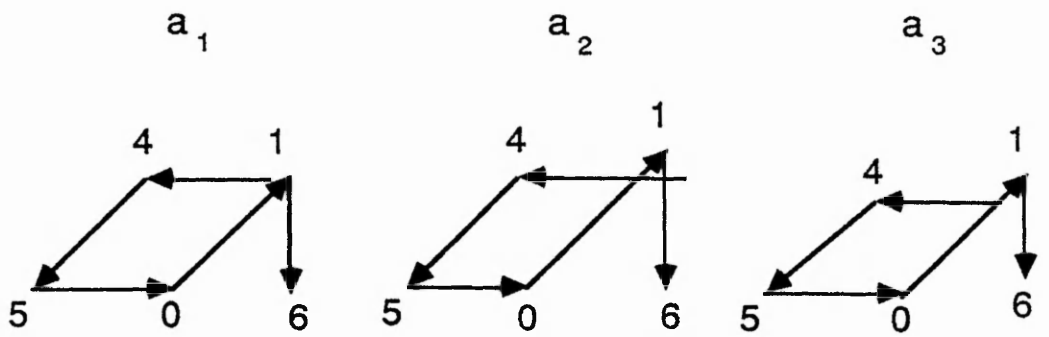
6.3.2. Database Construction

All valid stroke sequences collected from the users test sentence set are passed through the Freeman and XY algorithms to produce the encoded and reduced strings as detailed in Chapters 4 and 5. Each encoded string is assigned a stroke identity (a-z,\,/,> ,- for lower case script), the identity being the corresponding element of the header stroke sequence. The Freeman strings, together with their intended identities are filtered to one output

file and the XY strings, along with their intended identities are filtered to another output file. This may be for one particular writer or for any number of writers.

We now need to perform averaging on the Freeman and XY strings. Many of the XY and Freeman strings will not be unique for a particular character. Section 4.1.2.2 shows how similar XY strings are averaged to produce one single string which will be stored in the XY database. Experiments conducted showed that this averaging process reduces the number of Freeman strings by a factor of around 3 and the number of XY strings by a factor of around 6. These factors are seen to be larger for a particularly neat writer and smaller for an untidy writer, as would be predicted. The act of averaging should produce an averaged encoding which should produce a high measure of goodness of fit when correlated against its composite encodings.

If we have a number of composite Freeman encodings for a character 'a' for the vector string '45016' we can produce an averaged Freeman string as shown in Figure 6.7 below:-



$F(a_2)$= 4.15 5.20 0.14 5.24 6.27

$F(a_1)$= 4.17 5.21 0.16 1.24 6.20 $F(a_3)$= 4.20 5.19 0.15 1.24 6.22

$$F(a_{av}) = 4.\frac{(17+20+15)}{3}\ 5.\frac{(21+19+20)}{3}\ 0.\frac{(16+15+14)}{3}\ 1.\frac{(24+24+24)}{3}\ 6.\frac{(20+22+27)}{3}$$

$F(a_{av})$= 4.17 5.20 0.15 1.24 6.23

Figure 6.7 - Vector Averaging

Producing the goodness of fit measures (using the Cramer-Von Mises test):-

$Fit(a_1/a_{av}) = (1 - [0.00+0.01+0.01+0.00+0.03]) \times 100\% = 95\%$

$Fit(a_2/a_{av}) = (1 - [0.02+0.00+0.01+0.00+0.04]) \times 100\% = 93\%$

$Fit(a_3/a_{av}) = (1 - [0.03+0.01+0.00+0.00+0.01]) \times 100\% = 95\%$

The two test sentences written are composed of between 80 to 100 separate strokes. After producing the original Freeman and XY encoding for each stroke, the reductions are also produced. This results in around 150 Freeman encodings and around 220 XY encodings per writer. However, averaging reduces this figure to an average of 123 unique Freeman encodings and 108 unique XY encodings. The results of the script encoding and averaging procedures are given in Tables 6.1 and 6.2 for the Freeman and XY encodings respectively.

No. of Users	Cumulative Encodings (Isolated)	Cumulative Encodings (Additive)
1	118	118
2	238	232
3	368	344
4	499	458
5	640	570
6	776	663
7	913	770
8	1041	858
9	1110	902
10	1228	970
20	2456*	1678
30	3684*	2310
40	4912*	2871
50	6140*	3401
60	7368*	3833
70	8596*	4246
80	9824*	4745
90	11052*	5137
100	12280*	5430
112	13754*	5483

* (extrapolated values)

Table 6.1 - Freeman Database Construction

No. of Users	Cumulative Encodings (Isolated)	Cumulative Encodings (Additive)
1	85	85
2	239	221
3	344	302
4	450	370
5	586	456
6	681	497
7	825	579
8	925	612
9	985	630
10	1076	656
20	2152*	970
30	3228*	1235
40	4304*	1444
50	5380*	1646
60	6456*	1853
70	7532*	1990
80	8608*	2233
90	9684*	2410
100	10760*	2526
112	12050*	2571

* (extrapolated values)

Table 6.2 - XY Database Construction

If we extrapolate these results for the first 10 writers up to the full 112 test set, then at most we would produce 13754 unique Freeman vectors and 12050 XY trends. However, this is an analysis of the scripts in isolation. It would be a fair assumption to theorise that writers will produce many encodings for certain characters which would be identical to encodings produced by other writers. Therefore, we would expect databases representing the full 112 users to be much smaller than the maximum figures we have calculated. Figure 6.8 and 6.9 show how the combining of peoples unique encodings begins to show a marked tailing off over an initial trial of 10 writers. A reduction of 21% for the Freeman database and 39% for the XY database. Already the combined total of unique encodings for the XY database is beginning to show a limiting tendency. Figures 6.10 and 6.11 show the results of combining writer data up to the 112 user set. The final total of Freeman unique vector encodings is now 5483 strings held in 132 Kbytes of ASCII text. This represents a total reduction on the maximum possible number of strings of 60%. The XY database contains only 2571 unique strings held in 157 Kbytes of ASCII text, a total reduction of 79%. The XY database shows a much more marked tailing of of unique vector entries to

number of users. Extrapolation of the graphs shows that if we add another 112 users the Freeman database will increase by another 20% (around 1000 extra unique encodings) and the XY database will increase in size by a similar figure (adding around 500 extra encodings).

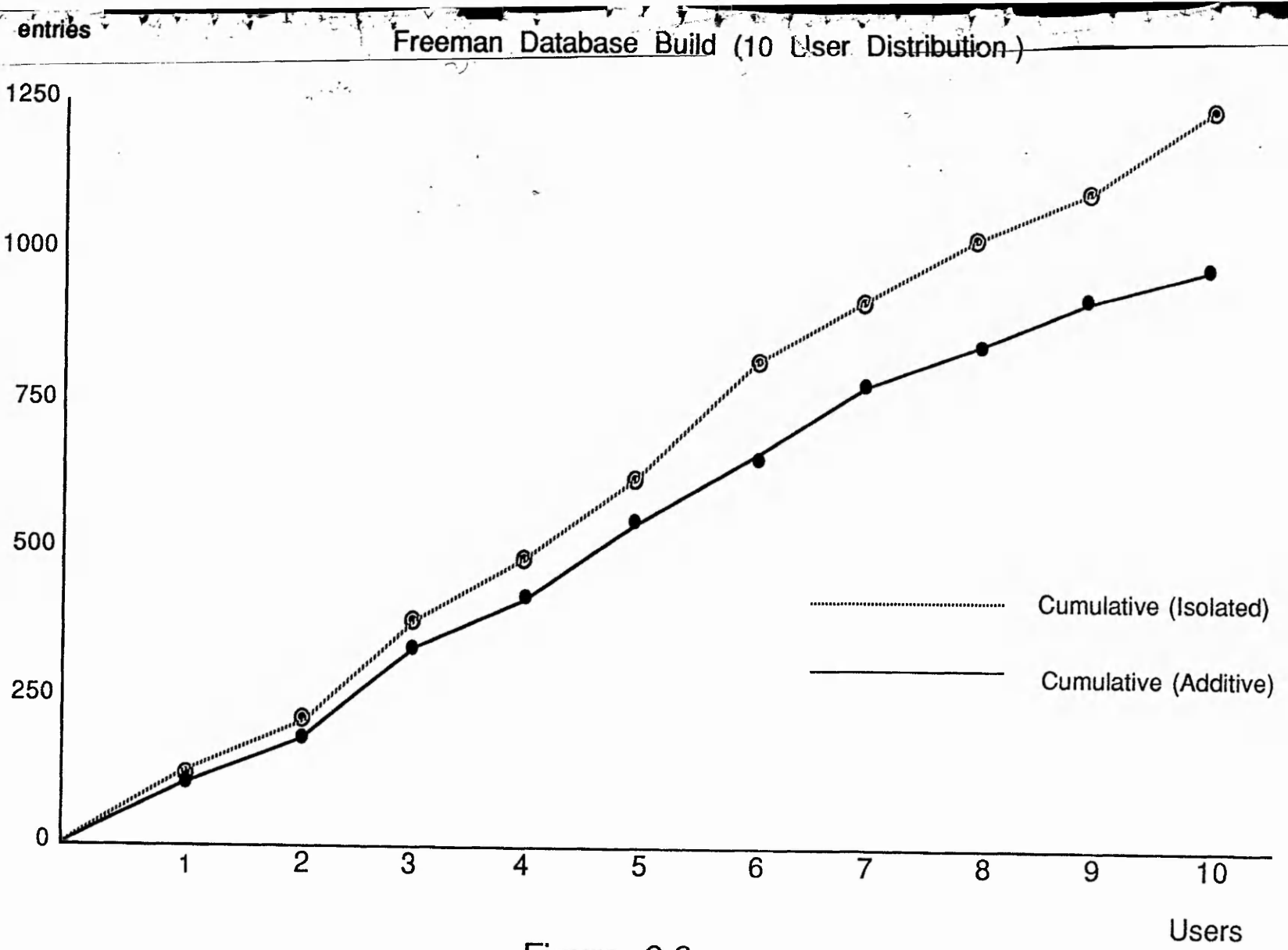

entries
Freeman Database Build (10 User Distribution)
1250
1000
750
500
250
0
1
2
3
4
5
6
7
8
9
10
Users
Cumulative (Isolated)
Cumulative (Additive)

Figure 6.8

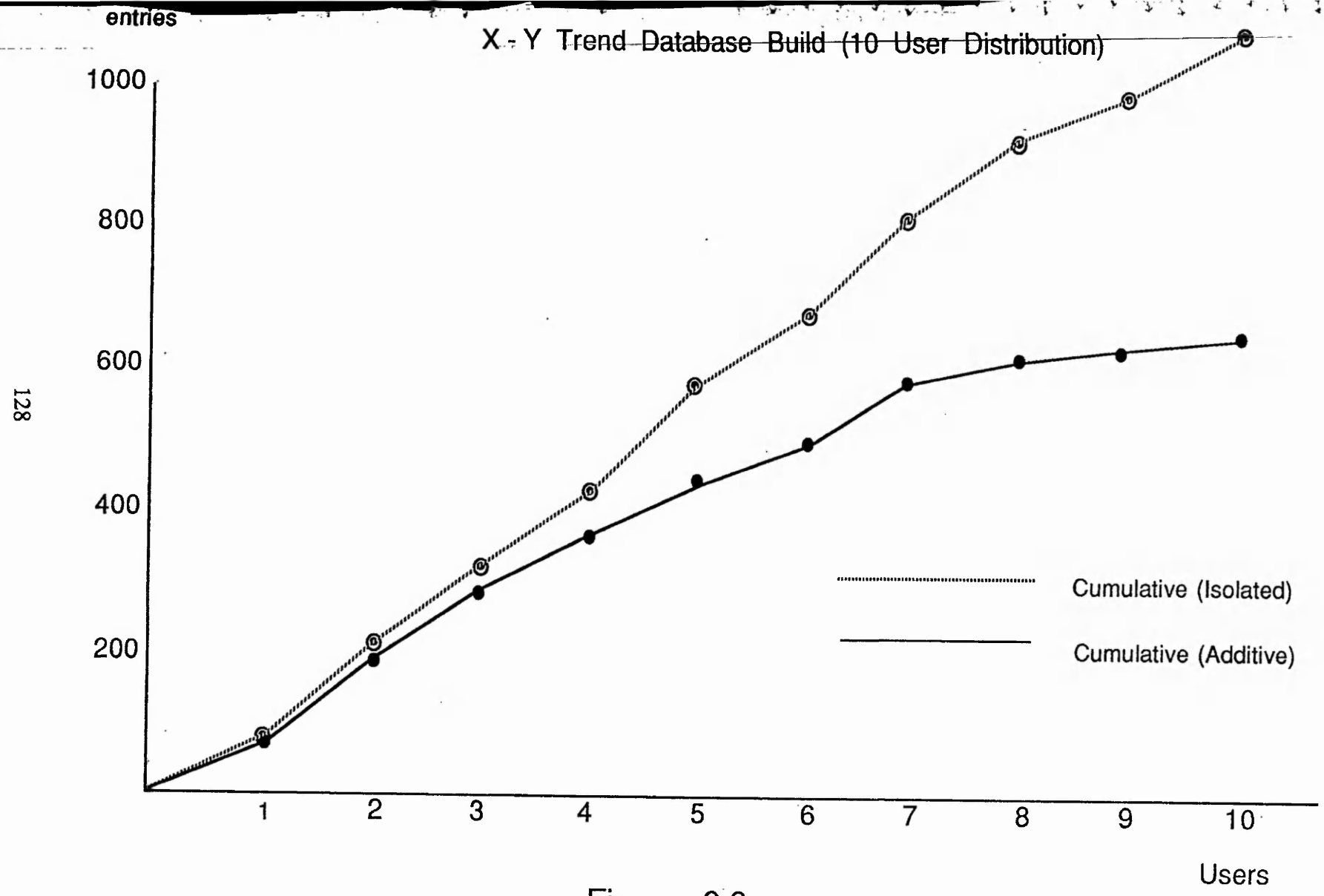

entries
X - Y Trend Database Build (10 User Distribution)
1000
800
600
400
200
Cumulative (Isolated)
Cumulative (Additive)
1
2
3
4
5
6
7
8
9
10
Users

Figure 6.9

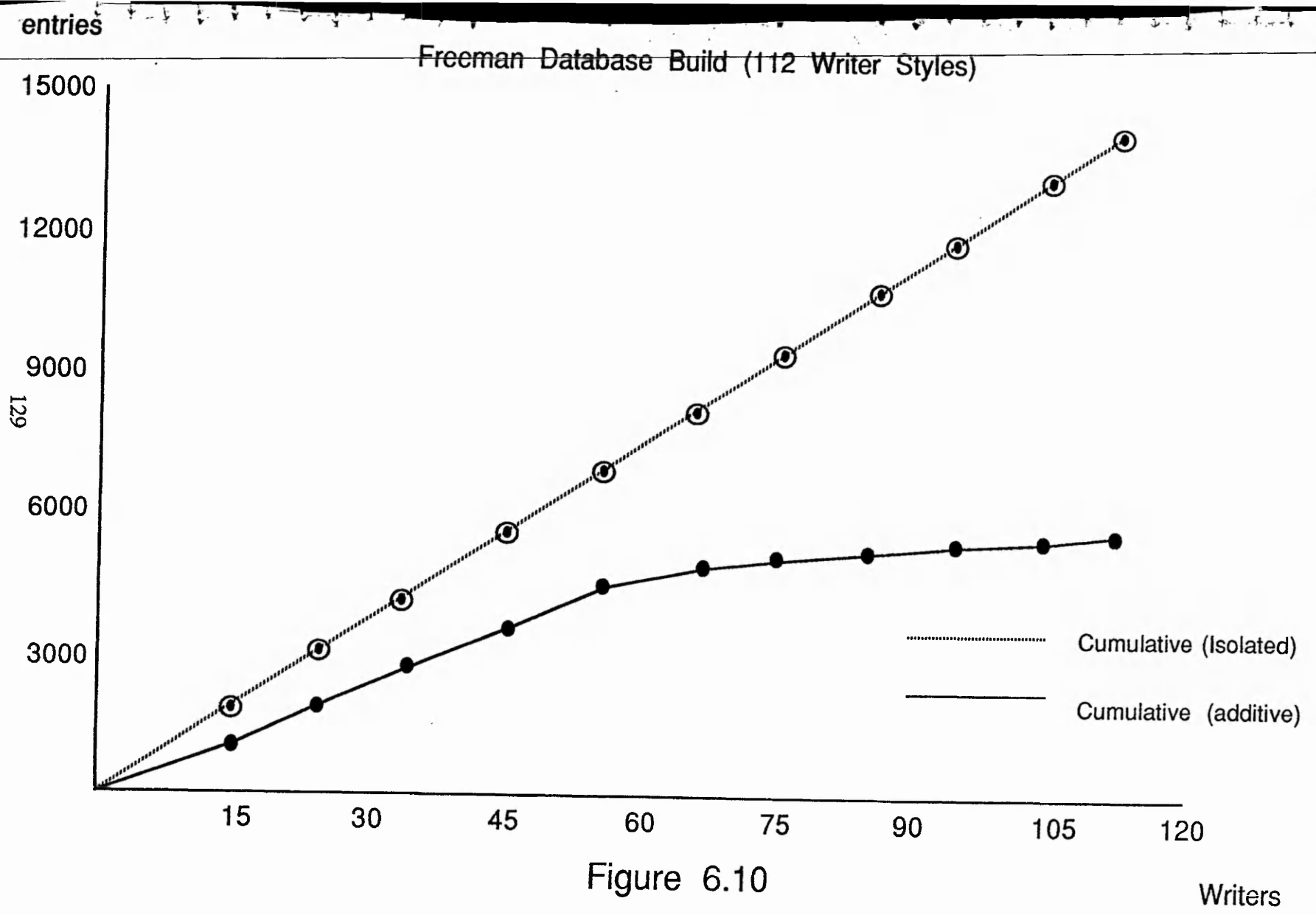

entries
Freeman Database Build (112 Writer Styles)
15000
12000
9000
6000
3000
15
30
45
60
75
90
105
120
Writers
Cumulative (Isolated)
Cumulative (additive)

Figure 6.10

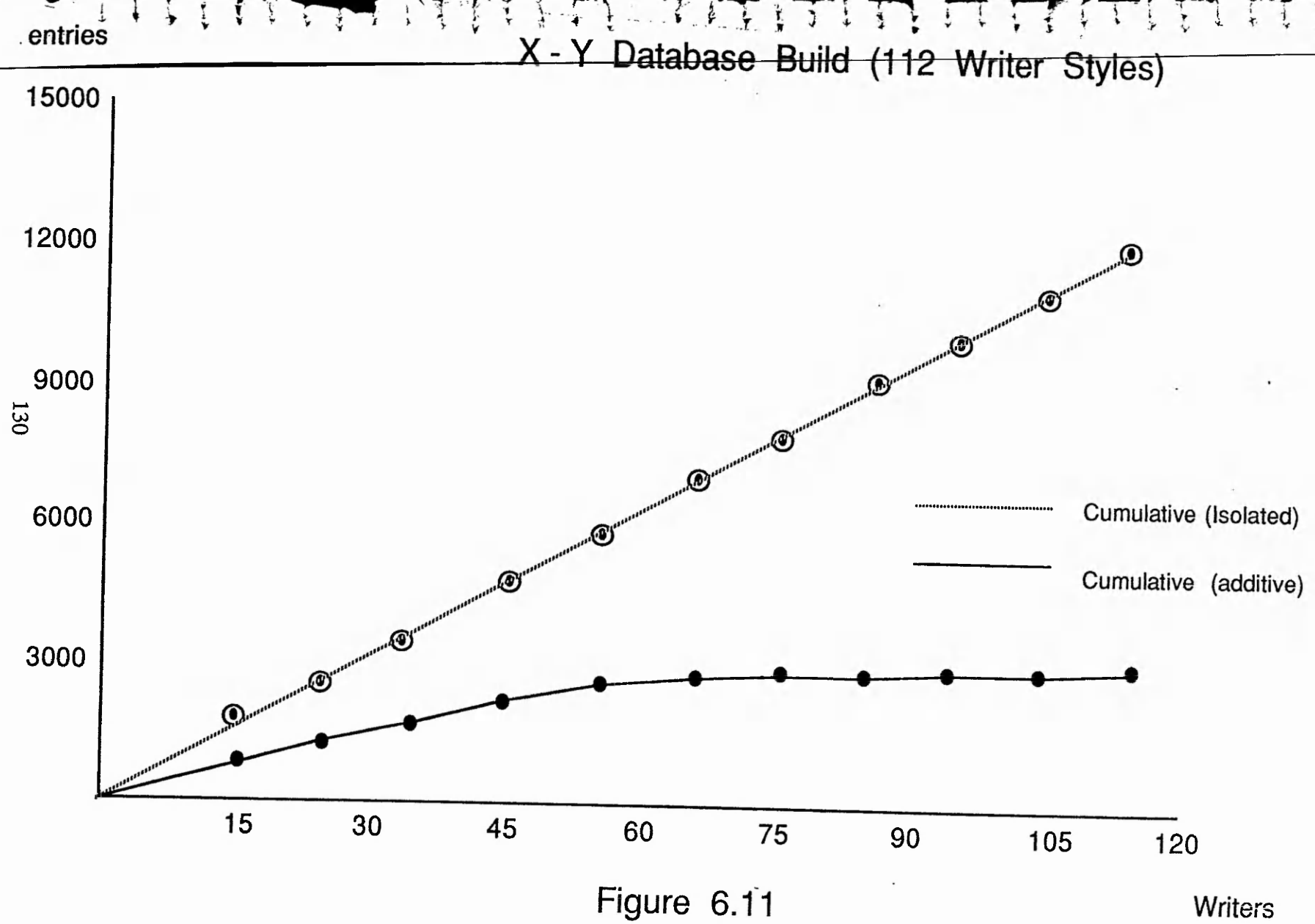

entries
X - Y Database Build (112 Writer Styles)
15000
12000
9000
6000
3000
130
Cumulative (Isolated)
Cumulative (additive)
15
30
45
60
75
90
105
120
Writers

Figure 6.11

6.3.3. Database Searching

In order for the recognition algorithms to perform in a real time environment it is vital that the database search strategy is as efficient as possible. We have shown that we can produce reasonably constrained databases for a user independent system. However searching of the databases should not take up an inordinate amount of processor time. In many papers on script recognition the algorithms have not been developed with ultimate real time operation in mind and, as such, the database searching sometimes accounts for the bulk of the processing time. In many instances this is because a simple linear search of the database from start to finish is performed in order to find a correspondence. We set out to devise a more economical search strategy. During the database development working on the VAX 11/750 the databases are stored in ASCII text files. This is necessary in order to visually check particular database entries to check their authenticity. Some bad entries tend to find their way into the databases during the automatic database generation phase.

Loading of the databases from their respective ASCII files (5483+ 2571= 8054 lines of ASCII text) into the database structures took a long time (around 10 minutes). Therefore a program was devised which would load the databases into an area of memory and perform a binary dump of the database structures to a file. This allowed us to simply read in this file of structures into memory instead of the two ASCII files on program initialisation. This took only a matter of 5-10 seconds. Of course, once transferred to the real time hardware the databases will be stored in ROM and no loading will be necessary.

6.3.3.1. Freeman Database

The Freeman database was ordered so as to minimise search time. If we consider a Freeman encoding, we can interrogate the vector string to determine a number of distinct features. The obvious feature is that of number of vectors in the string. For our database this will be 1,2,3,4 or 5. Hence we can categorise the Freeman database into vector strings of equal length. Another feature which can be used to categorise a vector string is the direction of the first vector. This will be in one of the eight quantised directions (0,1,2,....7). The Freeman database is ordered in both vector length and initial vector direction as shown in Figure 6.12

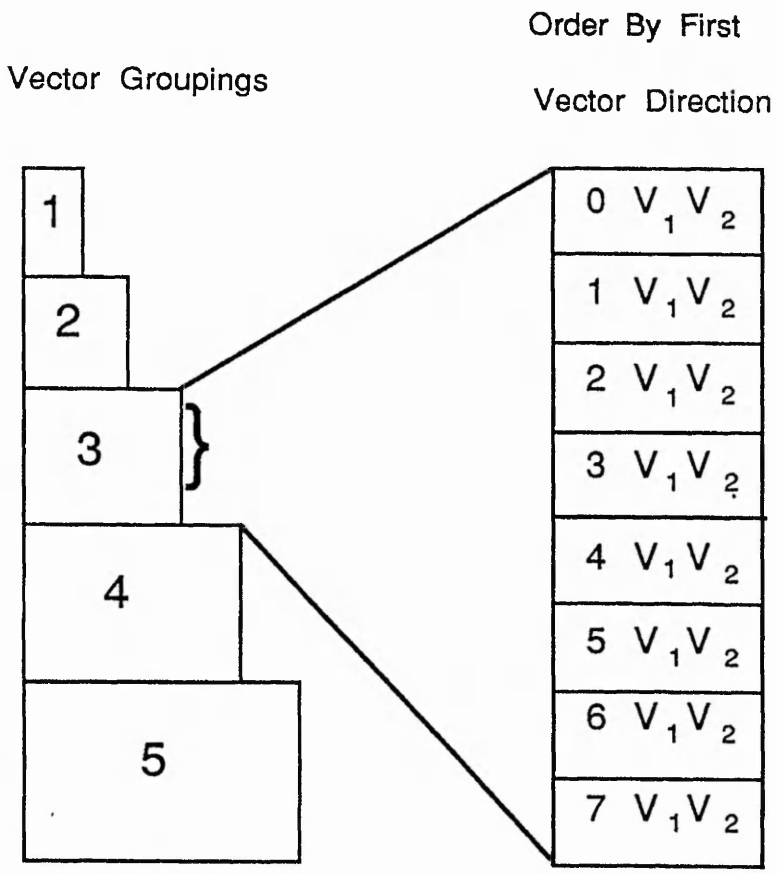


Figure 6.12 - Freeman Database Ordering

On program initialisation, after the database has been loaded, a two dimensional matrix is constructed of dimension $O[5,8]$. The row index relates to the number of vectors in the string (1→5) and the column index relates to the initial vector direction (0→7). Contained in this array are the addresses of the boundaries we have created in the ordering of the database as in Figure 6.12. If we consider the vector string:-

$$F(\alpha) = v_0.l_0\ v_1.l_1\ \ldots\ldots\ldots\ v_n.l_n$$

We can find the start address of all vector strings of length n which have initial vector direction v_0 by accessing element $O[n,v_0]$ of the array. This will give us the address of the first structure from which we should start our search of the database. We should stop searching the database when we get to the Freeman structure whose address is obtained by accessing the element which indicates the next boundary. This address is obtained from element $O[n,v_0+1]$ or element $O[n+1,0]$ if vector direction v_0 is 7. We have

effectively restricted our search area by construction of this search array, reducing search time greatly. The largest search area in the present Freeman database is for vector strings 5 vectors long, beginning in direction 6. There are 542 such entries in the database. This represents 9.9% of the total database size. If we wanted to speed up search time further in the future we could produce a three dimensional search array $O[5,8,8]$ and order in terms of the direction of the first two vectors in the string.

6.3.3.2. XY Database

The XY database is ordered in a very similar manner to the Freeman database. In this case, the two features extracted from the XY trend encoding are:-

- (i) x - y trend count (1,2,3,4,5,6), six maximum.
- initial x and y travel directions (either x -ve/y -ve, x -ve/y + ve, x + ve/y -ve, x + ve/y + ve).

Therefor the XY database is ordered as shown in Figure 6.13.

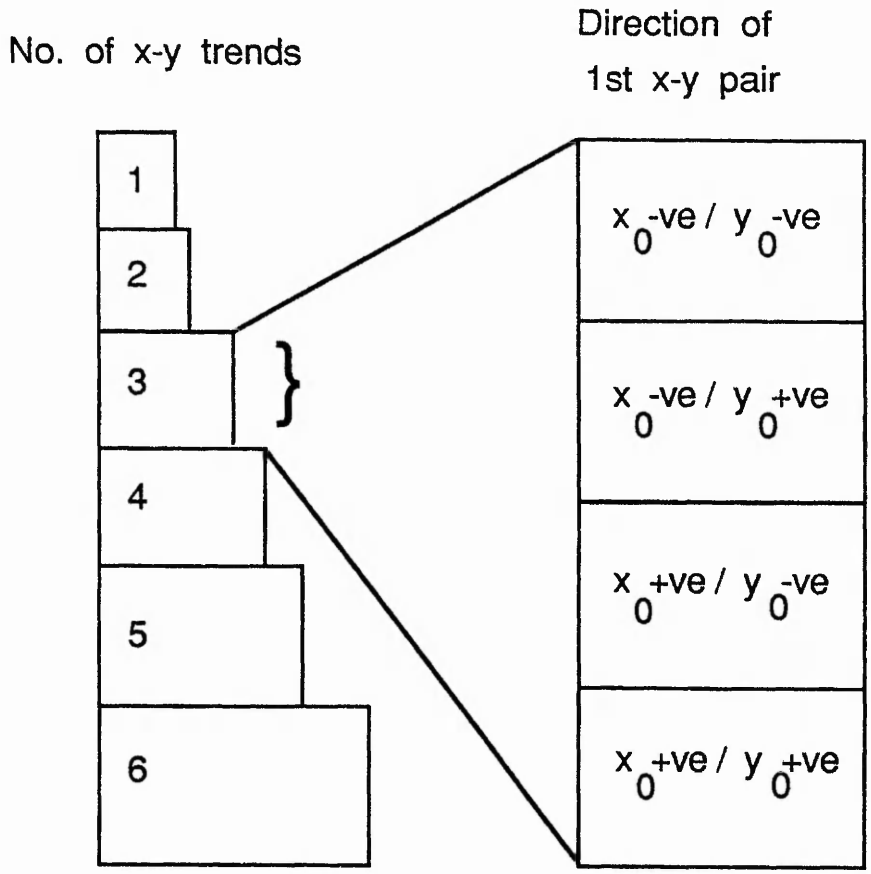


Figure 6.13 - XY Database Ordering

A similar method of access of the search pointer array *O*[6,4] allows us to define our limited search area as for the Freeman database. The largest search area for the XY database is for strings which have 5 x and 5 y trends and commence in direction x + ve/y -ve. This area is represented by 184 entries, 7.1% of the total database size.

7. ANALYSIS AND DISPLAY OF THE RECOGNISED OUTPUT

7.1. Introduction

Once each single pen stroke has been identified by the Freeman and XY algorithm combination it is necessary to interpret the sequence of recognised pen strokes into sensible sentences of text as meant by the writer. This takes us into the area of the human interface. How and when should we show the recognised output to the user and how will they react to the way that the interpretation to their written text is displayed back to them. This must be done in such a way that is sensible and clear to the writer. The ideas presented in this chapter are given as a basic initial approach more as an aid in analysing the algorithms developed in a real-time environment. Research into the complexities of the human interface is probably as big a task, if not bigger than the problem of dynamic script recognition. A number of stages of processing of the individual pen strokes are undertaken during the display phase,

- pen stroke combinations
- word boundary detection
- line detection

7.2. Pen Stroke Combination

Many of the lower case characters written by our initial user base of 112 writers were found to be formed by using more than one single pen stroke (the exact figure being 1537 out of 10352 characters, or 14.85%). The most common occurrence being the characters with diacritical marks - i,j,f,t. In such instances it is necessary to reconstitute a character by analysis of these part shapes as produced in the separate pen strokes.

At this point it would be helpful to describe the manner in which the test writers were allowed to create their test sentences. As long as the sentences were written from left to right along the A4 sheet in a relatively straight line the user was allowed to write using their natural writing style (speed, size, style of formation) as long as the text was legible to someone reading the sentences. However, this does introduce extra pen stroke combination considerations. In most instances a character formed from two separate pen strokes is usually completed sequentially in time. However, this was found not to be the complete story. In some instances, multi-pen stroke characters are only completed at the end of a word, namely the characters i,j,f,t. Hence we must keep a record of the absolute position of every previously written pen stroke in order to determine whether two pen strokes written apart in time are proximate. In many instances it is quite a simple task to determine whether two separate strokes are part shapes of the same character because one stroke actually intersects the other,

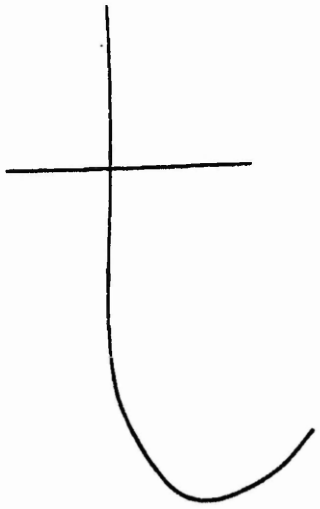


Figure 7.1 - Character with intersecting part-strokes (t)

However, for other characters this is not the case,

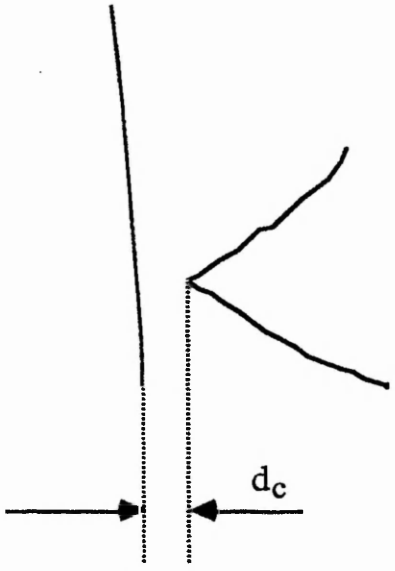


Figure 7.2 - Character with two non - intersecting pen strokes (k)

Hence, some threshold value needed to be determined in order to decide whether the space between the two separate letters 'l' and 'c' was actually an inter-character space or an intra-character space. For example, consider the following sequence of pen strokes,

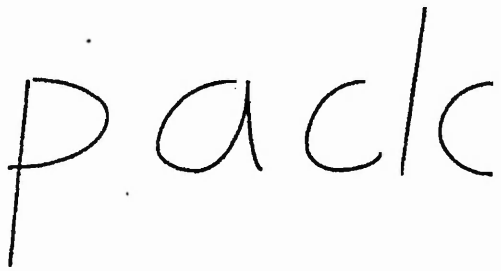


Figure 7.3 - Word 'pack'

By simply determining the horizontal spaces between these letters it would be difficult to decide that the last two pen strokes should actually be combined into a composite character, forming the letter 'k'. This problem will be discussed in more detail in the last chapter (Chapter 9).

By analysing the multi-stroke characters produced by our 112 user set it was possible to determine the criteria for the attempted matching of two separate pen strokes. These are as follows,

(i) If the next stroke crosses the path of any previously written stroke, attempt a match.

(ii) If the next stroke is a diacritical mark (i.e. a cross or a dot) and it is not a continuation of the present line (i.e. a dash or full stop respectively), search back along the sequence of stroke positional information in order to determine the previously written stroke which is closest to the mark, but exists in whole or in part below the diacritical mark. (This will cater for dots and crosses which are placed within close vicinity of their partner stroke as produced by some writers, while ensuring that the diacritical mark is not attempted to be matched with a letter from a previously written line).

(iii) If the newly written stroke is within a certain threshold distance of the last pen stroke written on the current line, it is deemed to be a part of that character. This threshold value was calculated by comparing the intra-character distances to the relative sizes of the composite pen strokes.

At this point it was decided that the techniques being developed for stroke matching and space detection could not be displayed to the user in this basic format, since not enough information is present while the first few pen strokes are being written to be able to display 'words' and 'characters' with any level of confidence. It was found, from analysis of pen stroke spacings, that the deviation between the first 12 spaces in some instances, was so great that it was not possible to decide correctly whether a particular space was,

(i) an intra-character space (S_p)

(ii) an inter-character space (S_c)

(iii) an inter-word space (S_w)

After only writing two pen strokes on a line, if a space is detected between them it is impossible to determine whether this space is an S_p, S_c or S_w space. Therefore the pen stroke sequence "lc" could be displayed as

(i) k (ii) lc (iii) l c

The decision procedure is not feasible until there are enough characters written along the line to be able to predict the spacing type with more certainty.

One way of reducing this uncertainty is to use some form of n-gram analysis. For example, in the case of our written word in Figure 7.3, possible alternatives *paclc* and *padc* will be discarded in favour of *pack* by virtue of analysing valid letter paths. This is discussed further in this chapter and also in some more detail in Chapter 9.

A similar problem arose with the display of a part character, before the writer has completed the second stroke. In some cases it is possible to display the recognised first stroke without confusing the writer (eg. display an 'l' before the writer crosses the 't'), however do we display an 'l' before a writer has completed an 'i'? A trial system was developed as shown below,

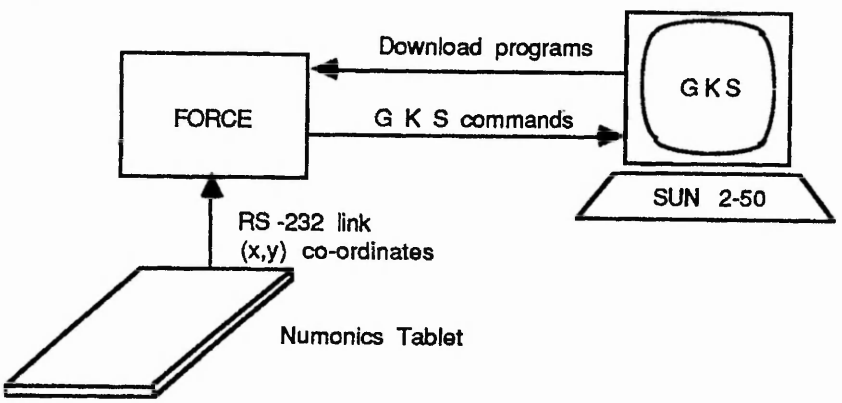


Figure 7.4 - Initial Script Input System

In order to be able to capture the tablet co-ordinate data and perform the recognition and display in real-time, it was necessary to transfer the data capture programs and recognition algorithms onto a machine having a real-

time operating system. A FORCE development system, running the PDOS operating system was chosen to perform the bulk of the processing, but it was necessary to retain the SUN to display the recognition results using the Graphical Kernel System to display the A4 page and show the text manipulation being performed. At this stage the display and manipulation of the recogniser output is the limiting factor in this demonstration system, since GKS running under the UNIX operating system on the SUN is very slow and memory intensive. Typically, incorporating the GKS into the run file increases the size from around 120Kbytes to around 1Mbyte.

In order to avoid user confusion, while a line of script is being written onto the tablet, the raw co-ordinates received from the data tablet are faithfully reproduced onto the SUN screen. Therefore the format on the screen of the SUN mimics the pen on paper actions of the writer while performing the stroke recognition, matching and space detection. Only when a new line is detected, will the final space detection processing be performed. Once this has been done the 'word' strings will be displayed in place of the raw data.

7.2.1. The Matching Procedure

When it has been determined that two or more pen strokes could be elements of the same character, it is necessary to decide on the identity of the composite character. The matching procedure uses the following information,

(i) the identity of the first stroke.

(ii) the identity of the second stroke.

(iii) the orientation of the two strokes with respect to one another.

A detailed analysis was undertaken of the characters written by our 112 user set which were made up of more than one single pen stroke. Of the total of 10352 characters written, the breakdown of the user set is as shown in the table below;

Character	Number Multi-stroke	Number Written	% Multi-stroke
a	7	221	3.2
b	30	206	14.6
c	-	209	0.0
d	30	405	7.4
e	14	815	1.7
f	145	202	71.8
g	11	310	3.5
h	6	300	2.0
i	346	619	55.9
j	76	207	36.7
k	119	208	57.2
l	-	215	0.0
m	2	205	1.0
n	2	409	0.49
o	-	411	0.0
p	64	207	30.9
q	12	215	5.6
r	-	407	0.0
s	-	313	0.0
t	297	306	97.1
u	7	514	1.4
v	2	213	1.0
w	1	206	0.5
x	202	209	96.7
y	16	207	7.7
z	16	208	7.7

TABLE 7.1 - Multi-stroke Character Breakdown

The results for the characters 'i' and 'j' were quite interesting in that they show that for our user base, around 45% of all i's written are not dotted, while almost 63% of all j's written were not dotted. Therefore we cannot rely on the writer dotting these characters in order to identify the character. Even in some instances where the 'i' or 'j' is dotted, the dot is not centred over the 'i' or 'j', but over it's preceding or succeeding neighbour.

We can also see that only 5 characters of the alphabet have not been formed from more than one stroke, these being the 'c','l','o','r' and 's'. Overall, 14.85% of all characters written were formed from more than one stroke. The most common multi-stroke characters were the 'f', 't' and 'x'.

Analysis of the multi-stroke characters has shown that they can be formed in a variety of different ways, and from a variety of composite shapes. The character 'k' is a particularly good example for indicating the different means of construction. It can be constructed from one, two or three single pen strokes. Figure 7.5 below shows the various styles of formation and the breakdown of the percentage occurrences for each particular style.

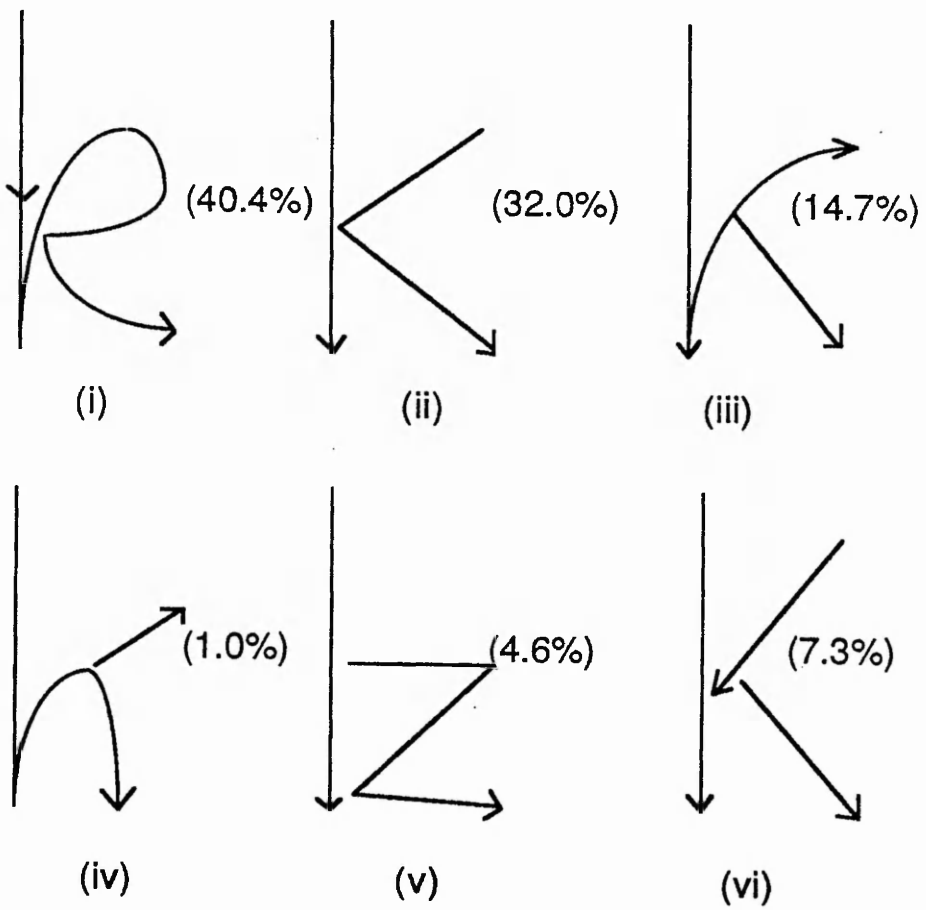


Figure 7.5 - Various formation styles for character 'k'

Hence the task of matching pen stroke pairings is not an insignificant one. We must not only decide whether the part strokes are valid elements of a letter, we must also examine the relative positioning, as in the following example where we have two opposite diagonals,

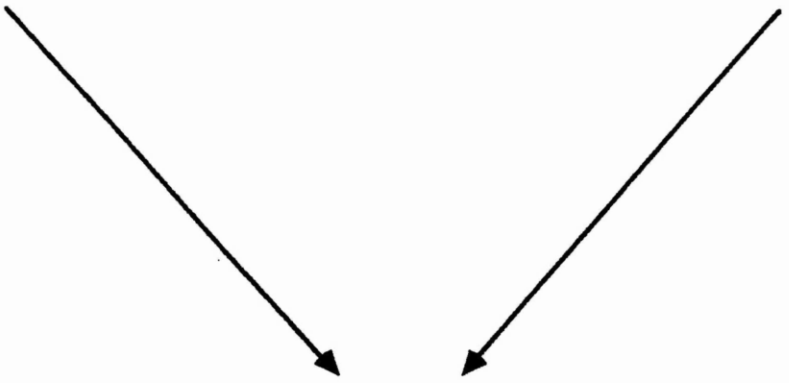
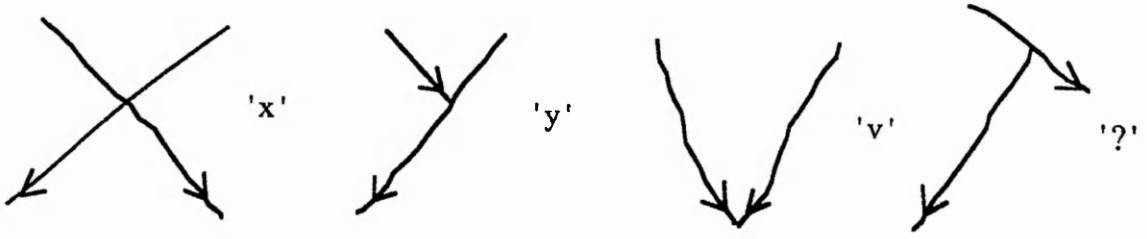


Figure 7.6 - Diagonal Stroke Pairings

Two such strokes in close proximity may produce a number of different shape results,



(i) (ii) (iii) (iv)

Figure 7.7 - Some Cross-diagonal Shape Permutations

(i) The two diagonals are of similar length and intersect roughly at their mid points. Character = 'x'.

(ii) Backward diagonal roughly twice the length of the other diagonal. First diagonal meets the backward diagonal roughly at its mid point from above. Character = 'y'

(iii) Diagonals roughly of similar length and come together at their lowest point. Character = 'v'.

(iv) The diagonals meet at their highest point. Not an alphabetic character.

7.2.1.1. The Matching Array

The technique used for pen stroke matching has been centred around the construction of a matching array. The design of such an array has been developed for the following reasons,

(i) optimisation of the matching algorithm.

(ii) ease of inclusion of new part stroke character constructions.

Each row of the matching array corresponds to a particular valid first stroke (first in time), and each column in the matching array corresponds to a particular valid second stroke. From the 112 user base, only a certain number of valid first and second strokes were found from all the multi-stroke characters written. Therefore, if we decide that a particular stroke S_{T_2} has been written within the minimum threshold of some previously written stroke S_{T_1}, we can determine the result of an attempted match by finding the appropriate element of the matching array. The procedure is as follows;

(i) A one dimensional array is searched in order to see if any element in the array matches the identity of our first temporal stroke, S_{T_1}. The array has previously been constructed from the identity of first stroke as written by our user base. This array is as shown below,

FIRST STROKE ARRAY = { [, ,], /, c, l, r, s, v, e, t, o, z, -, . }

If the stroke S_{T_1} does not match with any element of this array, we can exit from the matching procedure, as we have no previous evidence that any other stroke could be written after this one in such a way so as to produce a valid composite character. However, if the stroke does exist in the array, we note its position in the array (i.e. the n_{th} element) to be used as an entry to the two-dimensional matching array.

(ii) We now compare the identity of the second pen stroke against a second array which holds all the valid second strokes as written by the user base. This array is as shown below,

SECOND STROKE ARRAY = { [, ,], -, ., /, c, l, z, o, e }

If the stroke S_{T_2} does not match with any element of this array, we can exit the matching procedure as in the first case. However, if the stroke does exist, we note it's position in the array (i.e. the m_{th} element) to be used as the second entry into the two-dimensional matching array.

(iii) Having determined that the two part stroke identities are valid, we use their positional information in their respective arrays as means of entry into the matching array M. In this instance we interrogate element M[m,n]. The current matching array can be seen in the table below,

First Stroke	Second Stroke										
	[	\	]	-	·	/	c	l	z	o	e
[			x	f	<	f		`	k		
\	x		*	t	<	!	^	!		*	
]	x	x		f	j	x	x	x			
/	k	!	*	t	<		^	x		*	
c		a	g	t	<	t	^	~			
l	^	t	*	t	<	@	^		^	*	^
r		^		f							
s				f	<						
v		^		t							
e				t	<	t		d			
b		k		b							
t		k				k					
o		a						~			
f				f							
z				z							
-						x					
·			j								

TABLE 7.2 - Character Matching Array

AMBIGUOUS MATHCHES:

* : y, b, p

^ : k, NULL (no match if second stroke taller than first).

< : i, NULL (no match if second stroke below first).

! : y, v, x

@ : y, t

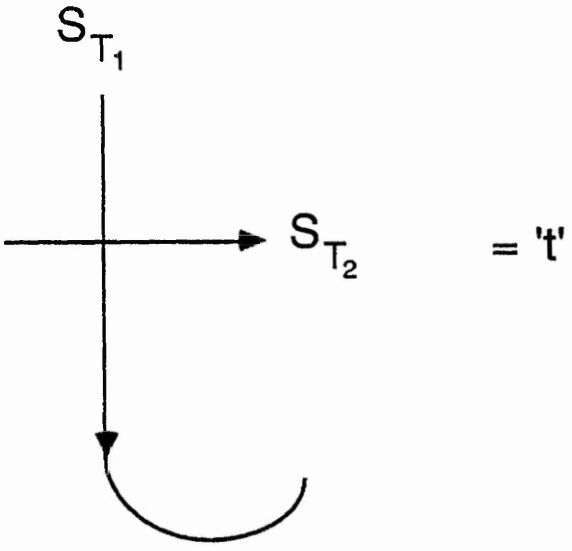
~ : a, q, d

The outcome of an interrogation of the matching array M may take one of the following forms;

1. The element in the matching array may contain a letter of the alphabet (a-z). This is the identity of the composite character. For example,

   M['l','-'] = M[5,3] -> 't'

   i.e.



2. A NULL character is found. In this instance we have valid first and second pen strokes but no previous knowledge that they may be combined to give a valid character. For example,

   M['/','/'] = M[1,1] -> NULL

3. A special character is found (eg. '+ ','^','*'). In this case we have found two valid first and second strokes, but we must do further analysis to determine the relative orientation of the two strokes before we can determine the identity of the composite character (as in the 'x','v','y' example previously).

This technique is equally applicable to characters of more than two strokes by simply making sure that the intermediate pair shape identifier is a column number in the matching array, i.e. a valid first stroke. Therefore, for the case of the three stroke 'k',

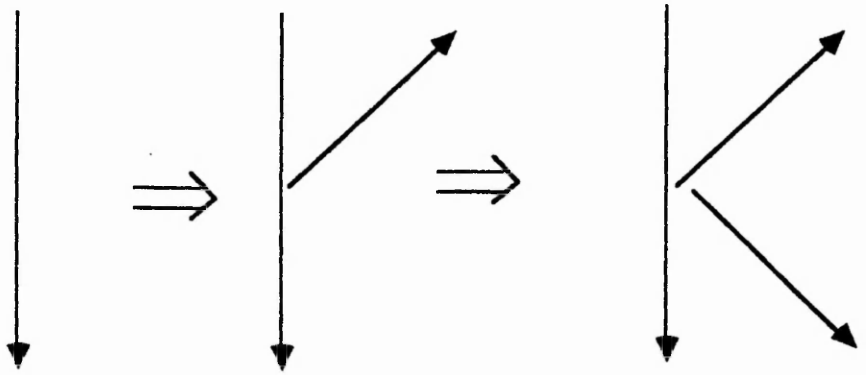


Figure 7.8 - Three stroke 'k'

When the first two strokes are detected, they will be input as a first stroke in the matching array. In this example, matching the first two strokes will produce the letter 't'. The letter 't' is now used as a valid first stroke, indicating another column in the matching array. Matching this with the second diagonal stroke will reveal the final character identity, 'k'.

7.2.1.2. Modified Matching Criteria

By studying the result of the recogniser output with the matching algorithm incorporated on the script from the user base, it was found that the matching algorithm was attempting to match pen strokes which should not be matched. In most instances this does not create a problem where either one or other of the strokes is invalid or the matching array indicates a NULL match. Typical characters which caused erroneous attempted matches were 'y' or 'g' as the second stroke. For example,

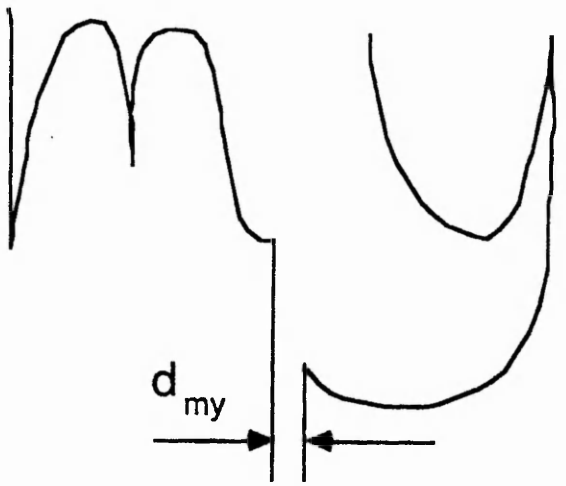


Figure 7.9 - Erroneous Pen Stroke Matching

The horizontal gap between the 'm' and 'y' is within the threshold for a possible match. This is due to the tail on the 'y' drawing the two characters together, and so trying for a match. However, neither the 'm' nor the 'y' are valid first or second strokes in our matching arrays, and so the matching algorithm will go no further than checking for the 'm' as a valid first stroke. However, if we do happen to have a stroke written before the 'y' or 'g' that is a valid first stroke, we can see from table 7.2 that neither the 'y' nor the 'g' are valid second strokes.

However, another instance does not produce such a fortunate result. Consider the case of the 'l' and the 'c' below. Again, the horizontal spacing between the two strokes indicates that a match should be attempted.

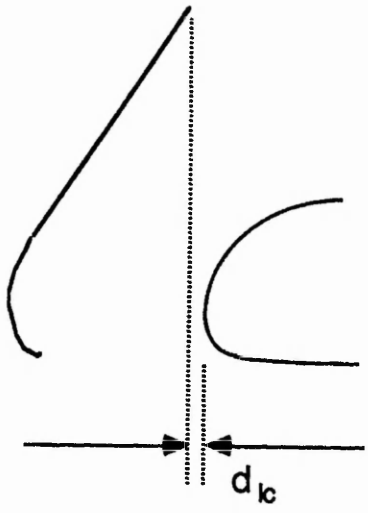
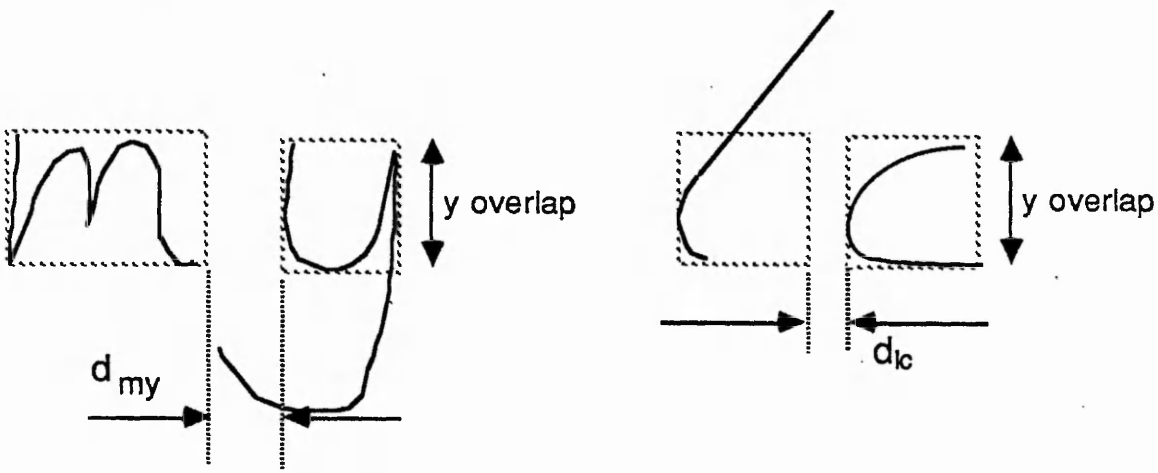


Figure 7.10 - Erroneous Pen Stroke Matching with Damaging Results

A match will be found, resulting in the character 'k' replacing the correct letter sequence "lc". In this instance, the reason for deciding to attempt a match is that the 'l' is slanted, therefore skewing its x boundary within the match threshold distance.

In order to overcome above problems, it was decided that we should only analyse the the horizontal distance between two pen strokes only over their vertical region of overlap. This will eliminate such occurrences from the matching algorithm. i.e.,



(i) TAILS (ii) SLANT

Figure 7.11 - Matching Distance Re-Calculation

This new region measurement was found to

- speed up the matching by not attempting to match due to 'g' or 'y' tails.
- increase the overall recognition by not attempting to match separate slanted pen strokes.

In a number of instances it was found that the matching algorithm did not result in a correct word simply due to the way that the strokes were placed. In the case below we will never decide that the most likely word is 'pack'.

Figure 7.12 - Indeterminate Matching Problem

The matching algorithm will always decide on the letter sequence 'padc' as the most likely word. Such problems will be discussed in the concluding chapter.

7.2.2. Space Detection

Closely allied to the character matching algorithm is the algorithm for determining the spacing between words along a line. We assume that along a line of text, the size of the characters will not vary significantly. If they were to do so, the space detection would not be able to sort out the word spaces from the character spaces. However, it is not assumed that every line of text be written at a similar size. The space detection is done on a line by line basis.

As a line of text is being written the absolute position of the pen strokes is monitored until it is detected that a new line has been started. This is quite a simple task and will be described later. Once the new line is commenced it is possible to analyse the horizontal pen spacings that have previously been recorded.

Initially, it was planned to determine whether a horizontal space between two characters as some function of the relative dimensions of the characters. This idea soon proved to be impractical as it was found that there was a

large degree of variability of spacing, independent of the relative character dimensions. It was also found to be impractical to attempt to determine whether a space was between characters in a word or between words by working out an average spacing and using some thresholds above or below the average to differentiate between the two. This is because no two writers produce word and character spaces which produce a typical standard ratio. In fact, in many instances, there is a large variation in the sizes of the horizontal spacing between characters, and this was found to be severely detrimental to any type the original ideas for space detection, which involved some form of averaging mechanism.

After studying the types of space formation by the user set, a method was devised which was independent of both the character size and the space size. Of all the sentences in the user set, no space between characters in a word was found that was over 5mm, in fact this was roughly the size of a space between two words. Therefore, a one dimensional array of 20 elements was set up which will represent a count of the space sizes detected along a line. When a line is detected as being completed, the largest space between successive strokes along the line is used as a maximum bound for the array. Hence if the maximum space detected is 10mm, the array will be divided into twenty equal divisions, representing space divisions 0-0.5mm, 0.5-1.0mm and so on. Similarly, a maximum space detected of 8.0mm will cause the array to be divided into increments of 0.4mm, going 0-0.4mm, 0.4-0.8mm and so on. Therefore, a space measuring 2.3mm will increment the 6th element in the array divided for a maximum space of 8.0mm.

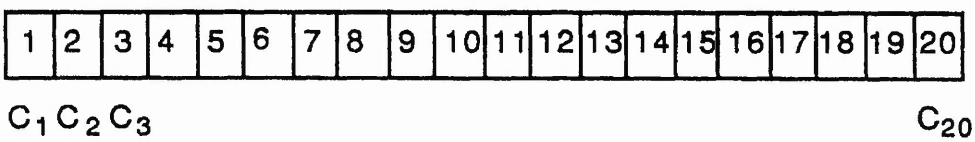


Figure 7.13 - Space count array

Therefore, the first element of the array will contain a count of the number of spaces in the line whose length was between 0 and 0.4mm, and so on. Consider a typical written sentence,

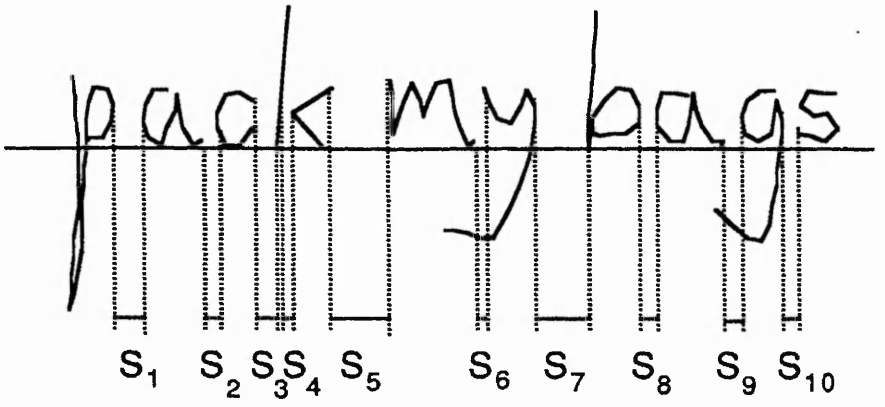


Figure 7.14 - Spacing Analysis

With space distances:-

S_1 = 0.41mm	S_6 = 0.61mm
S_2 = 0.56mm	S_7 = 5.65mm
S_3 = 0.32mm	S_8 = 0.13mm
S_4 = 0.14mm	S_9 = 0.18mm
S_5 = 3.92mm	S_{10} = 0.29mm

Maximum space detected is 5.65mm, giving a span of 5.65/20mm per element in the spacing array (0.28mm), producing a distribution,

3 4 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1

Now, a simple examination of the array enables us to determine the inter-word space from the inter-letter spaces. A sequence of three consecutive empty elements after the initial group of space sizes detected is used as a delimiter between inter-letter and inter-word spacings. Therefore, in this example, the inter-letter spaces occupy the first three bands of the array, and we have a gap of 10 empty bands until the next space size is noted. Therefore, all spaces greater than or equal that particular lower band limit are identified as inter-word spaces. The decision for opting for three empty bands was chosen as a result of studying the space arrays as produced by the user sentence sets. This was found to be a very robust method for word boundary detection, only failing to detect a word boundary when the user writes the sentence in such a way that it is impossible to find the boundary

simply from the space information.

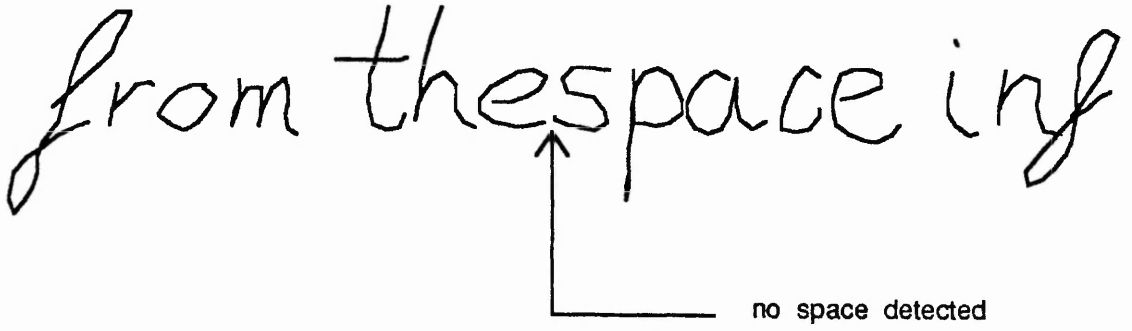


Figure 7.15 - Undetected Word Boundary

Again, application of n-grams to this problem is one possible method of resolution. Since *thespace* is not a valid word, a technique of word-splitting followed by n-gram analysis could be used to identify possible word boundaries.

th espace -no valid words
the space -both words valid
thes pace -only one word valid

This problem will be discussed in the Chapter 9.

7.2.3. Line Detection

Before the spaces can be determined as word or character spaces, we must be able to detect that a new line has been started before processing the spaces on the previous line prior to displaying the recognised output. In a very simplified decision process we could say that a writer producing a sheet of text would write sentences from left to right across the page, gradually working their way down from top to bottom. In such an instance, it is fairly straightforward to detect a new line by simply comparing the absolute pen position of the new pen stroke to the absolute pen position of the last pen stroke.

Figure 7.16 - New line Detection

The test for new line is :-

$$\text{if } [(S_{n+1}).y_{min} < (S_n).y_{min}] \&\& [(S_{n+1}).x_{min} < (S_n).x_{max}]$$
$$NEWLINE$$

This takes into consideration that a writer may go back along a line performing diacritical mark operations, crossing and dotting incomplete characters. From the sentences written by the user base, it was found that some people dot and cross characters immediately while others will only complete them after writing the other characters in the word.

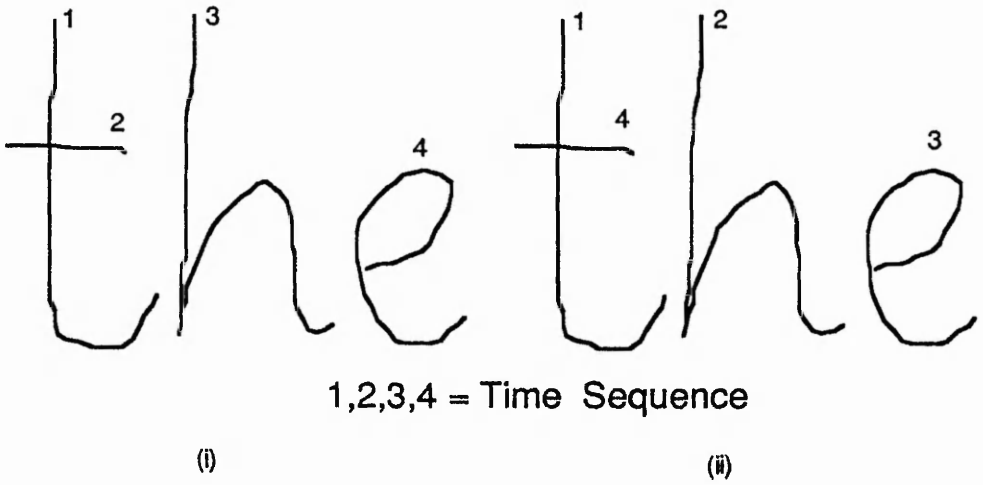


Figure 7.17 - Diacritical Mark Time Sequencing

From the user base it was found that some 88% of people would cross f's and t's immediately while the other 12% would wait until completion of the word. However, only 43% of people who dotted i's or j's would do so immediately while 57% would wait until the completion of the word. In some instances it was also observed that some writers would read the line and then cross and dot incomplete characters, or correct mis-spellings. Hence it would follow, that on a page of text, that the writer would want to go back to any previously written line and make some changes to that line. Therefore, not only must we detect that a writer has commenced a new line, we must also detect any new actions on any previously written lines. This facility is discussed here as part of the functionality of the script input system, but the human interaction allowable as a result of this is only discussed in the concluding remarks. Detection of the line identifier is quite simple as long as the lines written are constrained so as not to overlap in the y plane. A measure is kept of the maximum and minimum y extremities of each line. Therefore, when a new pen stroke is read in and does not represent a new line starting, we can determine whether it is written on the present line or some previous line.

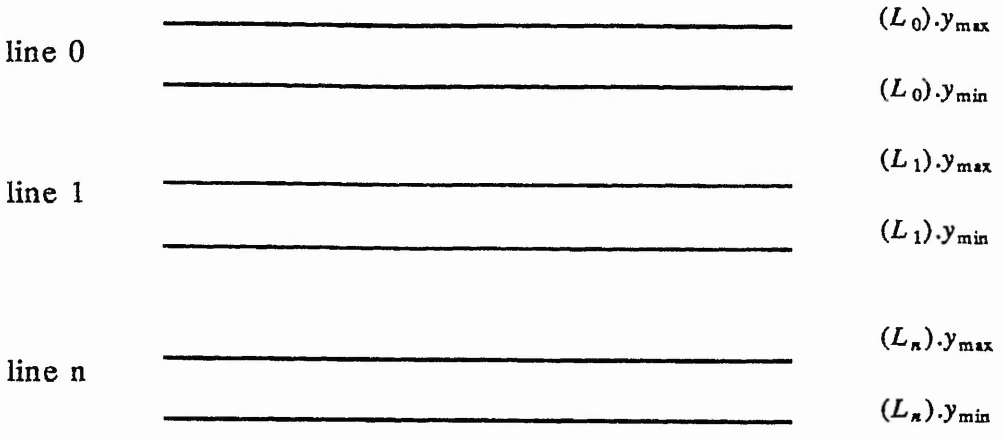


Figure 7.18 - Line Detection

If the mid-point of the present stroke lies within the bounds of a particular it is said to exist on that line, unless the stroke is a diacritical mark. This is because a dot or a cross could be placed above the maximum y bounds of the line they are on. Therefore, for line n,

```
if(pen_stroke = diacritical )
    if(mid_point_pen_stroke > (L_n).y_min) && (mid_point_pen_stroke < (L_n-1).y_min)
        line_number = n
else
    if(mid_point_pen_stroke > (L_n).y_min) && (mid_point_pen_stroke < (L_n-1).y_max)
        line_number = n
```

If the pen stroke is not a diacritical mark and falls outside the bounds of the lines previously written, we assume no particular significance, and simply ignore the pen stroke.

8. RESULTS

8.1. Introduction

The recognition results for the algorithms have been broken down in order to assess the performance at the various stages of the recognition procedure. The results specifically highlight the following areas:-

(i) The performance of the Freeman algorithm on the strokes in isolation.

(ii) The performance of the XY trend algorithm on the strokes in isolation.

(iii) The combined or correlated results of the above algorithms.

(iv) The result of the matching algorithm, combining part characters to produce a recognised character string.

(v) The space detection algorithm, detecting line and word boundaries in the recognised text string to produce the recognised sentences.

The initial results that are given were performed on the data collected from the 112 user test writer set, in order to assess the effectiveness of the algorithms on the user dependent set of writers. However, further results are also given for 10 completely untrained writers, whose character styles are not incorporated in the XY and Freeman databases. This will help assimilate how well the algorithms cope in a user independent environment. The level of degradation of these results from those of the user dependent set will give an indication as to how well the present user styles represent a wider user range, and give some indications as to how much work is still required to complement the databases and matching array, possibly with the inclusion of the data sets collected by the students of Trent Polytechnic (another 300 or so authors).

Various papers written on the subject of script recognition have considered the case of a person, given isolated hand-printed characters to identify. On average, it was found that only 96% of all characters shown to people could be recognised correctly without the aid of context. A large amount of information is elicited by a reader from their understanding of the context and semantics of the sentence and/or paragraph. Both Suen et al[42] and Harmon[54] quote 96% as the absolute maximum recognition by a human without context, therefore this must be taken into consideration when analysing the results of the recognition algorithm. Suen et al[42] also consider the sizes of databases required for the production of systems that could handle 'user independence'. More importantly, they consider the circumstances under which they have been produced. Early databases were very limited, both in number of samples and style and size of the letter set. Three databases were quoted which consist of more than 100,000 alphanumeric characters printed by multi-authors, however much research work requires that the writer print characters according to specified rules and models. For example, 0 with a slash, I with top and bottom horizontal bars.

At present, our lower case database contains the writing style of 112 authors, having produced over 13,000 lower case letters. Since the construction of the initial database, another 326 authors have been sampled, collecting another 40,000 lower case characters, 50,000 upper case letters, and over 10,000 numeric and special characters. This new data is presently being processed in order to produce a more representative user independent database.

Before realising the results of the recognition algorithms, it must be stressed that any figures given for recognition rate of a particular system have no meaning unless some clear indication of environment, allowed user writing style and alphabet set are given. In some papers surveyed, no indication of conditions or circumstances were given. In other papers, for example Caskey[28], the character set is limited to allowable FORTRAN characters only. However, they must be written so as to conform to the national ANSI standard for hand-printed character style.

The two test sentences performed by the authors for the collection of the lower case alphabetic data can be seen in Appendix B. The only constraint on the writer is that they form the characters from left to right along the page, in a reasonably straight line, performing their natural writing style. There is no constraint on style, speed or size of writing whatsoever. It was decided at a very early stage that only by attempting to aim for as user-friendly a system from the outset would it be possible to progress the early results to the end goal of a completely natural real-time document creation system.

8.2. The Recognition Procedure

Due to the large amount of data to be analysed, a number of programs and utilities have been developed to enable the rapid evaluation of the recognition algorithms for the aspects specified in points (i) to (v) in the introduction. In order to reliably test the results of recognising the test sentences (and to produce valid error free databases as described in Chapter 6) it was necessary to confirm the validity of each character or part-character written by an author with respect to the data sent by the tablet. This is not always completely reliable, mainly due to the problems of data capture outlined in Chapter 3. All the raw (x,y) co-ordinate data is held in ASCII files. The format of these files is as follows,

```
{*          START OF HEADER
= packmybagswithfivedozenextra.....
$ packmybagswl.l-h[-l.vedozene/\l-ra.....
! name_id
% tablet_type
& date_of_creation
@ user_specific_features
*}          END OF HEADER
D xxxxx yyyyy
```

```
D xxxxx yyyyy
D xxxxx yyyyy  FIRST  STROKE
.
.
.
D xxxxx yyyyy
U    0    0
D xxxxx yyyyy
D xxxxx yyyyy  SECOND  STROKE
.
.
.
D xxxxx yyyyy
U    0    0
etc
```

8.2.1. Header Description

The header is delimited by the sequence {*.....*}. Elements inside the header are as follows,

8.2.1.1. Text String Sequence

This string is a textual representation of the written letters as produced by the author, and is inserted manually into the header after studying the written words. This must be done manually, as a significant number of writing mistakes were found to have been made. Namely, mis-spelling or transposing letters in words, and in some instances, omitting a word altogether.

8.2.1.2. Stroke String Sequence

This string is a textual representation of the actual breakdown of each separately produced pen-down action performed by the writer during the process of creating the two test sentences. By identifying the sequence of stroke creation and classifying each shape, it is possible to perform a measure of isolated shape analysis as a first level of the recognition algorithms.

8.2.1.3. Name Identifier

The authors name.

8.2.1.4. Date of creation

The date of creation of the file, format DDMMYY (day-month-year).

8.2.1.5. Tablet Type and Parameters

This gives some detail of the tablet and how it was set up during creation of the test sentences. The Numonics 2200 tablet was selected as the most suitable from an evaluation of current technology (Chapter 2). However, two other input devices were purchased,

(i) a Penpad upper case script recognition system. The tablet which is part of this product has a stylus with a pen-down switch especially designed so as to detect the presence of the pen on the paper even for the lightest writer. However, it was not possible to divorce the data capture part of the product from the recognition software in order to be able to evaluate its performance.

(ii) the Elographics Touch screen. This provided the best user interface, where the author can use their own personal writing device, but is does not offer the same resolution and data transfer rate parameters as available from the Numonics tablet. Nevertheless, the Elographics device is seen as one possible progression to an early 'electronic paper' prototype, especially as there is a transparent version of the tablet already available.

8.2.1.6. User Parameters

These are not used at the present time, but they have been recorded in the event that some useful information may be extracted at some future date. Parameters consist of

(i) Left of right handedness

(ii) Age

(iii) Gender

(iv) Occupation

It is considered that other factors such as state of health, haste and mood can also affect the style and shape of the characters formation, Kutlinski[80].

8.2.2. Stroke Representation

The rest of the ASCII file contains sequences of pen co-ordinate information, the 'D' status before the (x,y) co-ordinate denotes the pen on the paper. Sequences of pen down co-ordinates are delimited by an up-stroke marker, namely (U 0 0).

8.2.3. Stroke Analysis

A batch process has been written which produces a stroke recognition breakdown for each of the 112 reference files containing the raw co-ordinate data. The data in each file in turn is encoded into both the Freeman and XY representations. These encodings are compared against the appropriate data-base in order to determine the recognised identity of the stroke. Therefore, a sequence of such recognition results is produced for each data file for

(i) the Freeman algorithm alone.

(ii) the XY trend algorithm alone.

(iii) a correlation of the Freeman & XY results.

Each sequence is then compared independently to the stroke sequence in the header of each file. The header is produced by analysing the contents of each file graphically on the SUN workstation. A program reads each stroke sequentially from the file, and, using a GKS (Graphical Kernel System) utility, plots out the graphical representation from the points onto the screen. If the stroke is a complete characters pen-stroke, there is no problem in assigning an identity to the pen-stroke, having also the hard-copy of the written text for comparison and validation. However, if some characters are written by producing more than one single down-stroke, it is the decision of the editor to enter the identity of the composite strokes. For most composite characters there is no problem in this respect. For example,

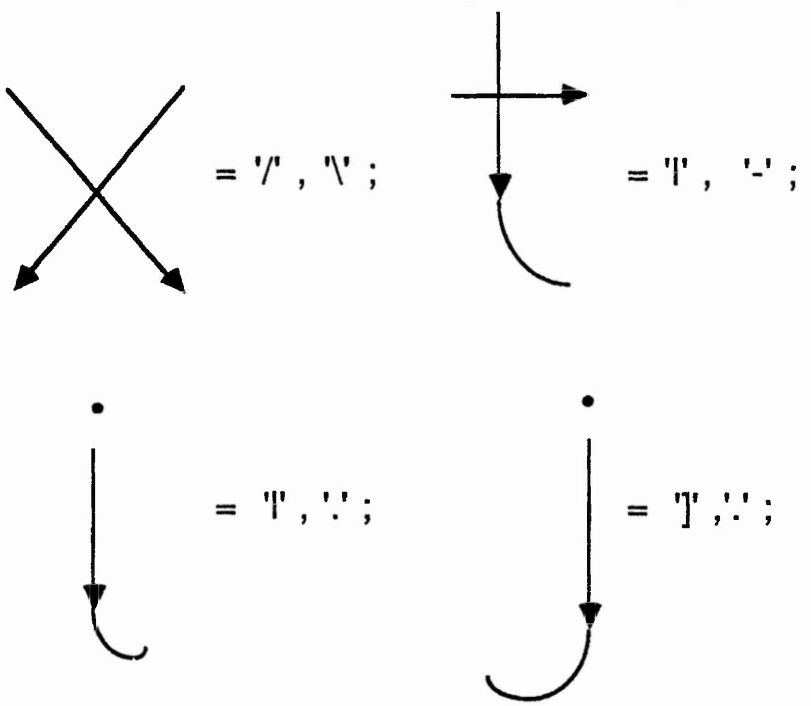


Figure 8.1 - Composite Stroke Classification

However, in other cases it is not so simple to classify the composite strokes.

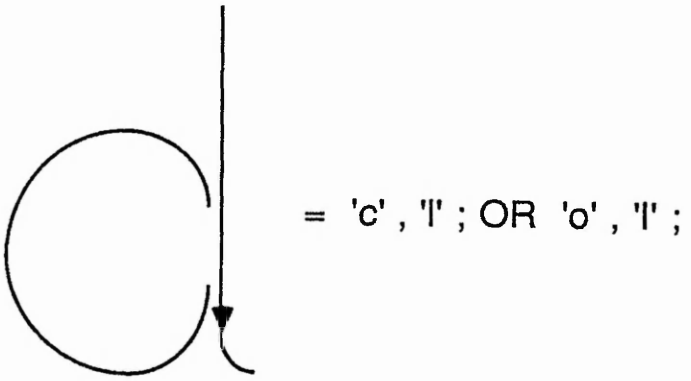


Figure 8.2 - Uncertainty in Composite Stroke Classification

In a number of instances there is more than one likely identity of such a part stroke. Therefore, whereas the complete character may be recognised with no ambiguity by the editor, there is sometimes a degree of uncertainty in assigning identities to some part strokes. The results of the stroke analysis are broken down in the following sections.

	RECOGNISED AS :-					
CHARACTER	-	·	/	\	|	%age CORRECT
-	366	23	7			92.42
·	2	507	1		1	99.22
/	2	2	161		8	93.06
[			2		9	
\	3	10	1	171	6	89.53
]					5	
c	1	1				
d					1	
e			1			
i					1	
l		29	28	20	755	90.75
o			1		1	
r	1		2		1	
v					1	
w					1	
y					1	

TABLE 8.1 - Simple Stroke Breakdown

Table 8.1 gives the result of the pre-processor to the recognition algorithms, which filters out straight line strokes before these are passed to the Freeman and XY algorithms for processing. It was found that 20.60% of all pen down strokes written by the author set could be picked out very easily and quickly as being simple straight line strokes or dots, these being |,\,/,- or the dot mark (.). The detection of such strokes was performed by analysing the angular variation between successive incremental line elements over the entire travel of the stroke. If these increments do not exceed some threshold value, the stroke is a straight line. Its orientation is easily deduced from its end points, and this enables its classification. The dot mark is simply identified by interrogating the size of the stroke in both the x and y directions. If both dimensions are below some threshold for a dot, the dot is identified. Only in instances where a pen slip has occurred during the creation of the dot will the dot not be detected. However, in such cases the stroke will not resemble a dot to a human reader.

The processing of these strokes was performed much more quickly than encoding the stroke by the Freeman and XY algorithms and searching the databases.

8.2.3.1. Freeman Stroke Results

Our test database was constructed from a total of 112 writers. Each person wrote two test sentences, comprising 9 and 8 words respectively, a total of 80 letters. (See Appendix B). The overall recognition rate for the Freeman algorithm was calculated at 90.53%. Characters 'f' and 'x' gave 100% recognition. This was due in the main to the fact that there were very few single stroke 'f's or 'x's written and these gave very distinct Freeman encodings, which were not easily confused with any other character encodings.

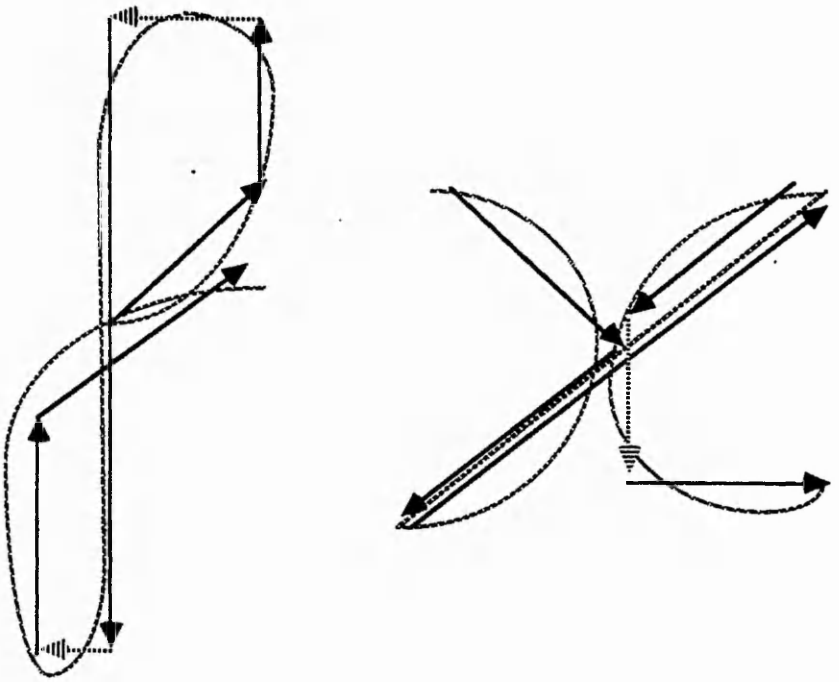


Figure 8.3 - 100% Freeman Recognition

Analysing Table 8.2 shows a recognition range as low as 79.66% for the character 'v', and up to 98.45% for the stroke ']' (i.e. the undotted 'j'). If we look more closely at the character 'v' we can see that the main reason for its poor performance is that, out of all the 'v's misrecognised, (48 in all), 32 or 67% of the mis-recognition is as the character 'u', and 9 or 19% of the mis-recognition is as a character 'r'.

CHARACTER	RECOGNISED AS : -	·	/	[	\	]	a	b	c	d	e	f	g	h	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	?	%age CORRECT
-	83		12											1					1													85.57
.	2							1								1							1									
/	3		42						1		1					3						1				1							80.77
[			4	113		1					1					4																91.87
\	3				72				2							4																88.89
]						382							1						2										1		2	98.45
a						2	382		9	8	3			2				2	3		3	1	1		12			1			2	88.63
b								166		1	1				1				2	5	1				2						5	90.22
c	2			2	4				370		5				1	10			10					1								91.36
d				1			13			356						1		1	4		4				5			1			2	91.05
e							1	1	22	1	80					7		3	1						1					1	7	94.59
f												57																				100.00
g						1	1	2		2	1		264						1		18		6					1	10		10	83.28
h			1				1	4						295		3	1	7							3		2				1	92.77
k								1						3	78			2							3						1	88.64
l	3		7	2	27			1	45		1		1			508	1								6	1				1	2	83.83
m						1	1	1		1	1			1	1		89	6			1				2		8				5	86.70
n						2			1	2				2	1	1		413		3	1	2	1		6						1	94.72
o						3	1		3	1	1								443				1		9	2					16	92.29
p								1				1								139						1			1		1	96.53
q							4						18			1					176				1	1					6	85.02
r	1		3	1		2			1			3		1		2		2	1	4		403	1	3	2	20						89.56
s						1				1			1	1					1				425			1				1	5	97.25
t												1												14								93.33
u	1						2		1	5			1			5	1	13	5			1			480	25	3				3	87.91
v									1			1				2						9	1	1	32	188					1	79.66
w								1			1						2		12						7	1	183				11	83.94
x																												7				100.00
y				1		1		1					7			1					1				1				86		2	92.54
z												3	1																	222	3	96.94

TABLE 8.2 - Freeman Error Matrix

Further investigation into the formation style for the character 'u' showed that the reason for the large number of mis-recognised 'v's was due to the fact that a number of writers produced 'u's with no downstroke at the end of the character, and that this particular shape was identical to the second most popular formation style for the creation of a 'v'.

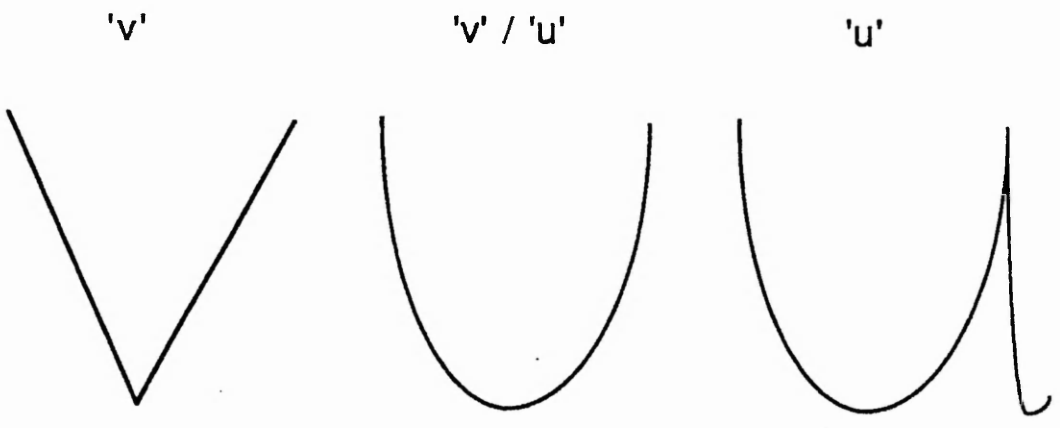


Figure 8.4 - 'v' and 'u' Confusion

Therefore it is impossible to distinguish the second 'v' type, which has a gradual reversal in the y travel, from the less common 'u' which has no down stroke.

Similarly, investigation on the shapes of the letters 'r' and 'v' show an area where the slope of the down stroke is at such an angle, slightly off the vertical, where it is impossible to say whether a 'v' or an 'r' was meant to be written.

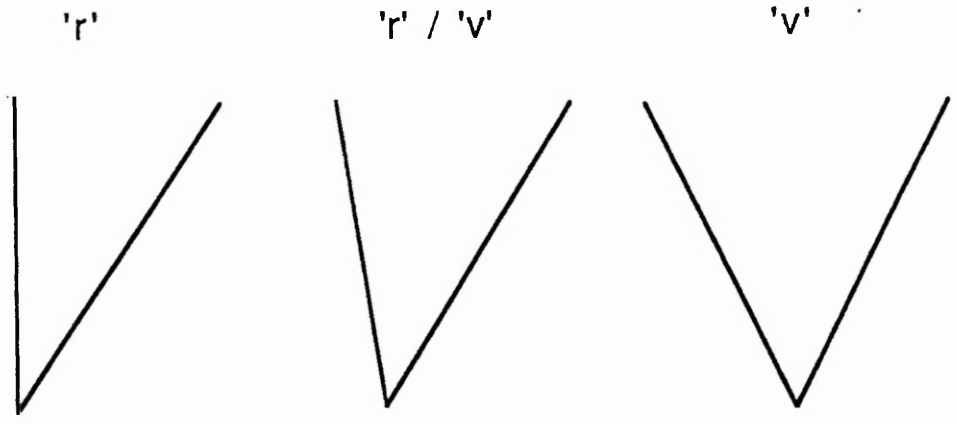


Figure 8.5 - 'v' and 'r' Confusion

Analysis of the letter confusion matrix for the Freeman algorithm (Table 8.2) reveals that the vast majority of incorrect recognition decisions is due to the shapes of some characters or strokes resembling another character or stroke in its shape. Table 8.3 shows, for the cases where a particular stroke has been mis-recognised, which strokes make up the incorrectly deduced identities. In almost every case, the 'best' misrecognition is of a stroke which is of a very similar shape to the misrecognised stroke or character. Some are more obvious than others, e.g $r \rightarrow v$, $u \rightarrow v$, $n \rightarrow u$, $d \rightarrow a$, $m \rightarrow w$, $y \rightarrow g$.

Number of incorrect occurences		Algorithm Character Identifier in order descending order					
Character	Misrecog Count	1st	%age	2nd	%age	3rd	%age
-	14	/	85.7	h	8.3	o	8.3
/	10	l	30.0	-	30.0	———	
[	10	/	40.0	l	40.0	———	
\	18	l	44.4	-	33.3	c	22.2
]	9	o	50.0	y	25.0	g	25.0
a	49	u	24.5	c	18.4	d	16.3
b	18	p	27.8	o	11.1	u	11.1
c	35	l	28.6	o	28.6	e	14.3
d	35	a	40.6	u	15.6	q	12.5
e	46	c	48.9	l	15.6	n	6.7
f	0	———		———		———	
g	53	p	35.3	y	19.6	s	11.8
h	23	n	28.0	b	16.0	k	12.0
k	10	h	30.0	u	30.0	n	20.0
l	98	c	45.9	\	27.6	u	6.1
m	29	w	27.6	n	20.7	u	6.8
n	23	u	26.1	p	13.0	q	8.7
o	37	u	24.3	c	8.1	\	8.1
p	5	b	25.0	y	25.0	e	25.0
q	31	g	58.1	a	12.9	u	3.2
r	47	v	42.6	p	8.5	t	6.4
s	12	]	8.3	o	8.3	d	8.3
t	1	f	100.0	———		———	
u	66	v	37.9	n	19.7	o	7.6
v	48	u	66.7	r	18.8	l	4.2
w	35	o	34.3	u	20.0	m	5.7
x	0	———		———		———	
y	15	g	46.7	q	6.6	]	6.6
z	7	e	42.8	f	14.3	———	

Table 8.3 Breakdown of incorrect character Identifiers for the Freeman algorithm

In the case of the very simple strokes, misrecognitions occur where strokes fall outside the threshold region.

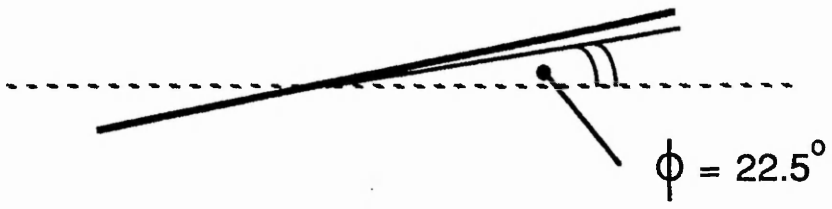


Figure 8.6 - Simple Stroke Mis-Recognition

For example, a cross-stroke, the orientation of which is greater than 22.5° to the horizontal, will be decoded as a diagonal. In some instances misrecognitions do occur in the cases where the Freeman algorithm might produce similar vector strings for two strokes/characters whose shapes look quite different.

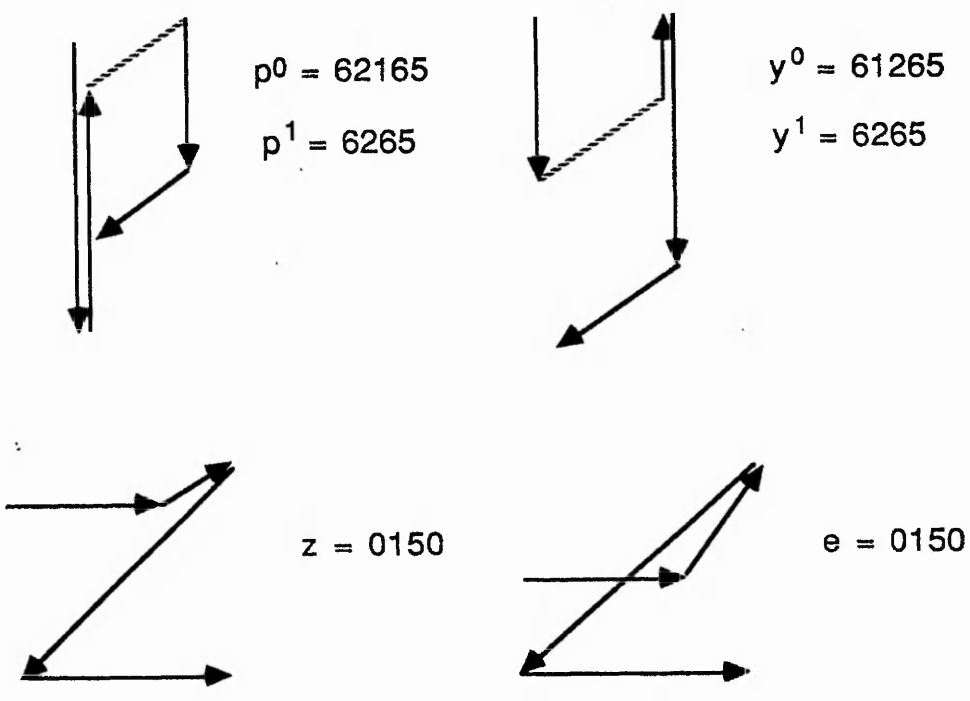


Figure 8.7 - Freeman Encoding Confusions

The Freeman algorithm passes on the five best alternatives (if indeed there are five alternatives) in descending order. Therefore, in the cases where a wrong decision was made as the best choice, it was found to be common to find the real character identity in one of the four other choices as shown in Table 8.4,

	1st	2nd	3rd	4th	5th
Comparisons	8219	615	137	55	25
Correct	7439	426	64	11	2
Percentage	90.53	69.27	46.72	20.00	8.00
Cumulative	90.53	96.70	97.49	97.63	97.65

TABLE 8.4 - Freeman Recognition Results

In the cases where the stroke or character has been misrecognised, if the second choice is analysed, in 426 out of 615 cases (or 69%), the actual identity is found in the second most popular alternative. Hence, by assuming that the identity of the stroke or character resides in one of the first two alternatives the recognition rate rises from 90.53% to 96.70%. However, only a further 0.95% increase is obtained by assuming the top 5 alternatives. Therefore, the majority of the confusion for the Freeman algorithm is mainly between one of two possible stroke or character shapes.

8.2.3.2. XY Trend Stroke Analysis

The XY trend algorithm gave an overall recognition rate of 78.10%. Individual results are broken down as shown in Table 8.5. Results are not as good as the Freeman algorithm. In particular, it can be seen that recognition of the simple straight line strokes that are not filtered out by the preprocessor (mainly due to having leading or trailing ticks) is particularly poor. The cross-stroke is only recognised 14.43% of the time. More significantly, the majority of cross-strokes (53 out of 97) are not recognised at all. Similarly, poor results can be seen for '\', '/', and 'l'. For a more complex character, the leading or trailing tick will not be such a large percentage of the total character travel and so will be eliminated at an early stage in the encoding reduction. However, as the straight line stroke consists of two or, at the most, three distinct trends in each direction (x & y), the tick will not be removed.

	RECOGNISED AS :-																													
CHARACTER	-	·	/	[	\	]	a	b	c	d	e	f	g	h	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	?	%age CORRECT
-	14		4	1	2			3	1	1						4		2	1			7			2	2					53	14.43
·					1															1			1								2	
/	1		14	2		1										4						2									28	26.92
[	1		5	91					7							4			11				3								1	73.98
\	2			2	36											5		2					1			1					32	44.44
]			2	2	1	270		1					10			7		1	28	1		1	37		5				6	16		69.59
a						1	391	2		18		2		2	1			1	3		2	4			4							90.72
b								18						2										1								98.37
c	11		4	3	6		5	16	256		12			1		65		2	18		4				1	1						63.21
d				4			7			870	1		1						5			1			2							94.63
e				4			3	33	8	27	683	15		1		48		8	12		1				4					4		80.26
f			1	1		1					1	52												1								91.23
g						1				1	1		295								8		2						9			93.06
h			1			2	11	8	1					257	7	1		8					6	7	5		1			2		80.82
k														4	82									1	1							93.18
l	10		17	13	81	7	1	26	55	3	10	2		5		229	1	6	4	1	1	1		1	10	7			1	26	88	37.79
m											1						207	4			1				1	1	3					94.95
n					2		3			3				5		2	1	385			1	3	5		24		1		1			88.30
o						2	2	10	3	20	1	1				2			407	2	1	12	4		12	1						84.79
p								1										1		40					1	1						97.22
q							4			1			12		1						178	1						10				85.99
r	3		6	17		1	1	3				1	2	7		3		9	3	17	1	331		10	7	25					3	73.56
s			1	7		6		3		23			21	11		3		2	1	1			347							11		79.41
t																						2		13								86.67
u	2			1	1	2	1	6		3			1	5		3		25	13			4	4	1	419	53	1				1	76.74
v	8						1	1	2							11						28		4	33	148						62.71
w															1			1				1			2	1	212					92.75
x																												7				100.00
y						2							10			1					2	1	1		2				182			90.55
z											1				1	3				2										222		96.94

TABLE 8.5 - XY Trend Error Matrix

The reason for the poor performance of the XY algorithm on the simpler strokes is highlighted in the figure below,

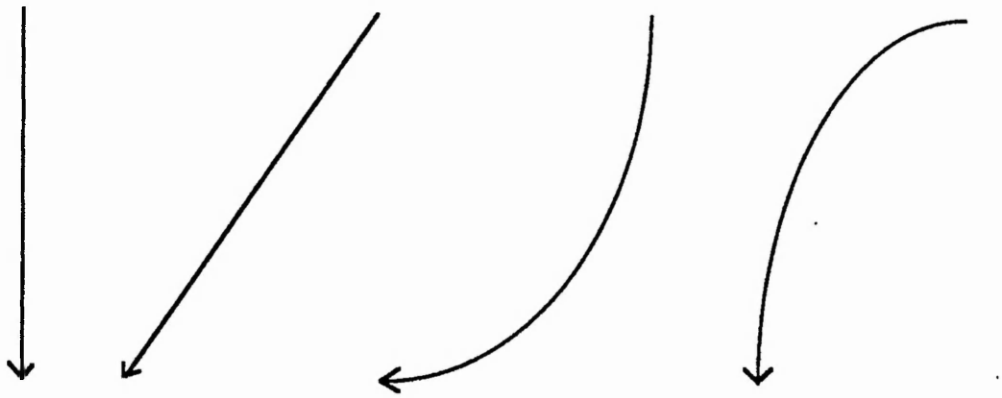


Figure 8.8 - Simple XY Stroke Confusions

In other words, the 'l' cannot be differentiated from the '/', '[', or ']' strokes. In order to avoid such confusions, the XY algorithm does not attempt to process encodings with a trend count of one in each direction.

Investigation of confusions arising for the more complex XY encodings can be highlighted in the following example. Consider the XY encoding for the character 'l',

$$l = 0.15\ -0.23\ 0.27\ 0.34\ /\ 0.18\ -0.51\ -0.19\ 0.09$$

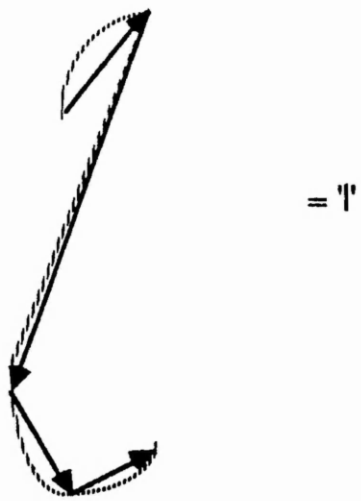


Figure 8.9 - XY Encoded Character 'l'

However, there are also a number of other characters and strokes in the XY database which have a similar trend encoding,

$$XY = + - + + / + - - + = \{ l, v, u, o, f, a, z, c, q, e, d, b, n, h \}$$

Recreating these encodings produces the following shapes,

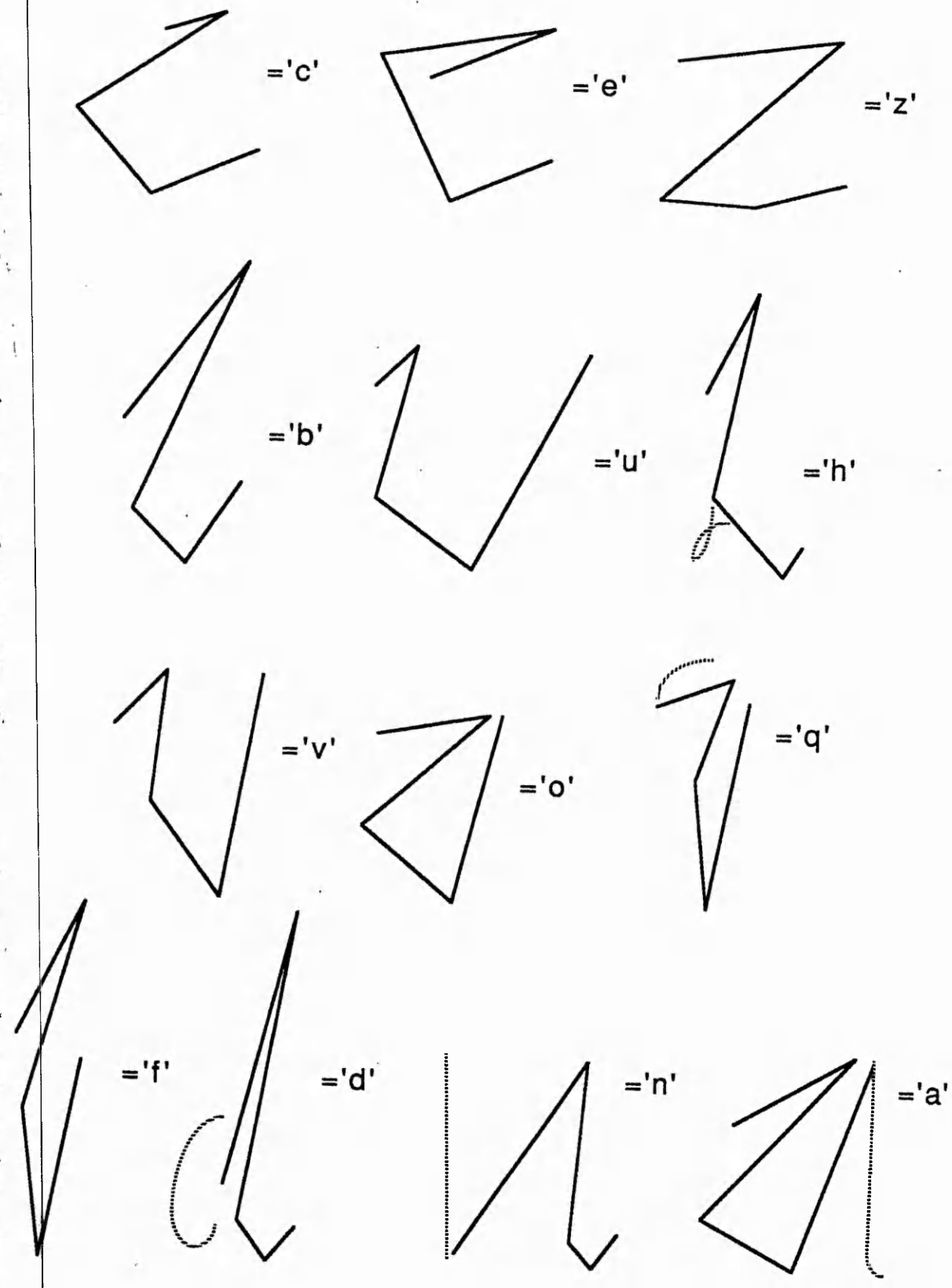


Figure 8.10 - Similar Trend encodings

A number of encodings are reasonably good representations of the character shape that they are portraying, namely {e, z, b, f}. However, a number of the encodings represent the character shape with a leading tick, as produced by some authors, namely {l, c, u, v, o}. However, the remainder of the encodings from the database appear to be caused by reducing the XY trend until the character shape is lost, namely {h, q, d, n, a}.

Therefore, it appears that the XY algorithm is not as robust as the Freeman algorithm for reduction of the encoding, especially for characters which have been initially poorly written. For no characters did it appear to be better in terms of recognition than the Freeman algorithm. For the more complex character shapes, 'w', 'g' and 'm' it performed as well as the Freeman algorithm. Reduction tends to produce a very large subset of character id's with similar trend patterns which might cause a processing problem in real-time. The above example has 14 different possibilities, and this is not uncommon. A breakdown of character identifiers in such cases where the algorithm chooses the incorrect character or stroke identity is given in Table 8.6. From it we can see that, as for the Freeman algorithm, most instances are due to shape similarity.

Number of incorrect occurences		Algorithm Character Identifier in order descending order					
Character	Misrecog Count	1st	%age	2nd	%age	3rd	%age
-	84	r	8.3	l	4.8	/	4.8
/	38	l	10.5	r	5.3	[	5.3
[	32	o	34.4	c	21.9	/	15.6
\	45	l	11.1	n	4.4	[	4.4
]	118	s	31.3	o	23.7	z	13.6
a	40	d	45.0	u	10.0	r	10.0
b	3	h	66.7	t	33.3	——	——
c	149	l	43.6	o	15.2	b	13.6
d	21	a	33.3	o	23.8	[	19.0
e	168	l	28.6	b	19.6	d	16.1
f	5	/	20.0	[	20.0	t	20.0
g	22	y	40.9	q	36.4	s	9.1
h	61	]	18.0	n	13.1	b	11.5
k	6	h	66.7	u	16.7	t	16.7
l	377	\	21.5	c	14.6	b	7.1
m	11	n	36.4	w	27.3	u	9.1
n	51	u	47.1	s	9.8	g	9.8
o	73	d	27.4	r	16.4	b	16.4
p	4	b	25.0	n	25.0	u	25.0
q	29	g	41.4	y	34.5	]	13.8
r	119	v	21.0	[	14.3	p	14.3
s	90	d	25.6	g	23.3	h	12.2
t	4	r	100.0	——	——	——	——
u	127	v	41.7	n	19.7	o	10.2
v	88	u	37.5	r	31.8	l	12.5
w	6	u	33.3	v	16.7	n	16.7
x	0	——	——	——	——	——	——
y	19	g	52.6	u	10.5	]	10.5
z	7	k	42.8	p	28.6	e	28.6

Table 8.6 Breakdown of incorrect character Identifiers for the XY Trend algorithm

It is interesting to note that if we consider the first five alternatives for character identity, in the case of the XY algorithm (Table 8.7), a recognition rate of 98.14% is observed, higher than that found for the Freeman algorithm. This is due in main to the imprecision of the XY algorithm.

	1st	2nd	3rd	4th	5th
Comparisons	8219	1578	617	325	168
Correct	6416	954	291	147	53
Percentage	78.06	60.46	47.16	45.23	31.55
Cumulative	78.06	92.01	95.64	97.48	98.14

TABLE 8.7 - XY Trend Recognition Results

8.2.3.3. Combined Algorithm Stroke Analysis

The results of the correlated stroke output are given in Tables 8.8 and 8.9. The correlated recognition rate is 94.04%, a 3.5% increase over the better technique, the Freeman algorithm. Although a more mathematically precise method of combining the results from the Freeman and XY algorithms might have been found, this method does prove to be particularly effective and has the advantage that the correlation processing is very slight compared to the other processes, e.g. encoding, reducing, searching, matching.

We can see from Table 8.8 that the correlated output has no non-recognised characters (column '?' in the table). The Freeman algorithm had 86 non-recognised characters (1.05%) and the XY algorithm had 208 non-recognised characters (2.53%). Again, most mis-recognitions are due to strokes of similar shapes. The correlated results give the best individual recognition rate for each character except the 'c' and '[', where a higher recognition by the Freeman algorithm alone. The correlated results are seriously degraded by the output of the XY trend algorithm.

Due to the variability in the performance of the XY algorithm between the simple strokes and the more complicated characters, the possibility of applying some weighting mechanism to the output of the XY algorithm might well improve its performance. Hence, a low weighting would be applied to the simpler strokes, e.g. 'l', 'c', '[', '/' and so on, and a higher weighting to the more complex characters, e.g. 'm', 'w', 'g' and so on.

The correlated results show little improvement over the separate recognition algorithms over the first five recognition choices. Indeed, it is not as effective as the XY algorithm alone, (97.94% to 98.14% respectively).

	RECOGNISED AS :-																														
CHARACTER	-	·	/	[	\	]	a	b	c	d	e	f	g	h	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	?	%age CORRECT
-	445	23	20		1				1					1					1			1										90.26
·	4		1					1								2							1									
/	2	2	206	1	1				1							11						1										91.56
[			5	118		1										10																88.06
\	6	10	1		244				1							9										1						89.71
]						383							1			5			2				1						1			97.46
a							412		3	7				1							2	1			5							95.59
b								181											1	2												98.37
c	3	1			3				370		5					18			6						1							90.91
d				1			6			378	1		1			1			2		1				1							96.43
e			1				1		17	1	325	1				3		2							1							96.83
f												57																				100.00
g										1	1		296						1		6		4						8			93.38
h			1				2	2						300	1	1		7							3		1					94.34
k														1	86										1							97.73
l	2	29	41	3	56				27	1	2					1261	1								8				1	2		87.97
m											1						213	3									1					97.71
n									1					1				431			1		1		1							98.85
o			1			1			2		1					1			464				1		9	2						96.27
p						1	1													141						1						97.92
q																					205				1				1			99.03
r	2		4											1		1				1		425	1	7	1	11						93.61
s														1				1		1			433							1		99.08
t																								14		1						93.33
u	1					1	2		1	2						1	1	4	3			2			304	23	1					92.31
v					1				2							3						15			26	190						80.17
w																1			1						3	1	213					97.26
x																												7				100.00
y				1									2			2					1				2				194			96.04
z											1																			228		99.56

TABLE 8.8 - Combined Algorithm Error Matrix

	1st	2nd	3rd	4th	5th
Comparisons	10352	443	102	56	41
Correct	9735	336	46	12	10
Percentage	94.04	75.85	45.10	21.43	24.39
Cumulative	94.04	97.29	97.73	97.85	97.94

TABLE 8.9 - Correlated Recognition Results

8.2.4. Character Analysis

These results incorporate the matching algorithm, which is fed the stroke information output from the correlator. If the matching algorithm were 100% efficient we would expect the stroke recognition rate of 94.04% to be converted into a character recognition rate of 94.04%. The results were calculated by comparing the character string as written by the author with the character string as output from the matching algorithm. However, in this instance, there is not necessarily a one to one correspondence between characters the same distance along each string. For example,

"packmybags....." = *REFERENCE STRING*

"paclcmybags....." = *RECOGNISED STRING*

A utility was written which does an automatic comparison of the two strings in order to determine the recognition rate. However, in our example above, a match for a 'k' has not been made. This leads to two characters appearing in the recognised string compared to only a single one in the reference string. Therefore, the utility needs to perform some forward searching in order to get back into step for the next comparison. In order to be able to do this, we must assume that over the next five characters in each string, there should be sufficient correspondence to be able to to determine the next comparison points in each string.

If we consider a particular author from our 112 writer set, the analysis of their two test sentences is laid out below as a typical example;

Filename : 05081005

Strokes : *l]ackmybagswll−h]−lvee]cl−rallquor]ugsl]ol−hwlzenedmenqulclzly]uged]−oursharpvl]cens*

Reference : *packmybagswithfiveextraliquorjugsbothwizenedmenquicklyjugedfoursharpvixens*

Recognised : *puckmqbagswithfireextraciquorjvgsbothwizenedmenquicklyjugedgoursharpuixens*

Ref	*Recognised as*
a	*u*:91 *a*:82 *q*:58 *d*:56 *n*:53
y	*q*:88 *y*:87 *w*:46 *g*:46 *u*:33
v	*r*:89 *v*:73 −:24 *i*:16 *i*:11
l	*c*:84 *l*:78 *o*:54 *u*:42 *b*:26

u	*o*:71 *u*:69 *c*:47 *l*:34 *v*:34
u	*v*:68 *u*:68 *l*:34 *r*:22 *b*:20
f	*g*:50 *j*:39 *l*:37 *f*:21
v	*u*:73 *v*:71 *r*:39 *l*:36 *b*:27

	Correct	*Non−rec*	*Rec−err*	*Seg−err*
Counts	65	0	8	0
%age	89.04	0	10.96	0

Figure 8.11 - Character Analysis by Author File

In this particular example, all the matches have been identified. Therefore there is no problem in determining the correspondence between elements of the recognised and reference strings.

The matches found were,

'l' + ']' = 'p'
'l' + '-' = 't'
']' + '-' = 'f'
']' + 'c' = 'x'
'l' + ']' = 'b'
'l' + 'z' = 'k'

In this instance, therefore, the recognition performance is not degraded by the matching procedure, and all the errors detected are due to mis-recognitions at the stroke level. Of the 8 recognition errors found, in seven cases the actual identity can be found in the second alternative, and in the other instance, the identity is found in the fourth and last alternative. The types of errors are also indicated above. These are broken down as,

(i) recognition errors, occurring at the stroke level

(ii) non-recognition errors, again at the stroke level where no entry in either database can be found

(iii) segmentation errors, occurring where the matching algorithm finds an incorrect match for two separate characters.

The confusion matrix for the character recognition level is given in Table 8.10.

	RECOGNISED AS :-																									
CHARACTER	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	%age CORRECT
a	389		3	6				1							1		2		1		5						94.42
b		201					1									2								1			97.57
c			190	2	2				5		1	1			5					1				1			90.91
d	6		1	391		1						1		1	2		1										96.54
e	1		22		775	1			1			2		1	1					9							95.09
f						175	1			2										22							86.63
g					1		287								1		9		3						8		92.58
h	1	2						280			1	1		7						4	3		1				93.33
i			6			1			552		2	22				1	7			6	7	6		4		4	89.18
j									2	198		3								1							95.65
k			2					1	4		165	4								4	2						79.33
l			3	1	3				31			171					1			1						1	79.53
m													201	2									1				98.05
n								1	3					402						1	1						98.29
o				1	1					1					395			1	1		8	2					96.11
p		1								1						202											97.58
q			1												1		207				2				1		92.28
r	1								4	1						1		386	1	4	2	6					94.84
s		1							1	1				1		1			307							1	98.08
t			2			1			7		1	15								277	1				1		90.52
u	2			1			1		2				1	4	3			1	1	3	464	24	1				90.27
v				1					3									9		1	28	169					79.34
w															1						3	1	201				97.57
x			1						7											4		1		189			90.43
y						1	2		1	1		1					1				2	1		2	188		90.82
z					1																					207	99.52

TABLE 8.10 - Character Level Error Matrix

The 112 data files were processed in batch order, the histogram below showing a breakdown of the recognition rate on an author level.

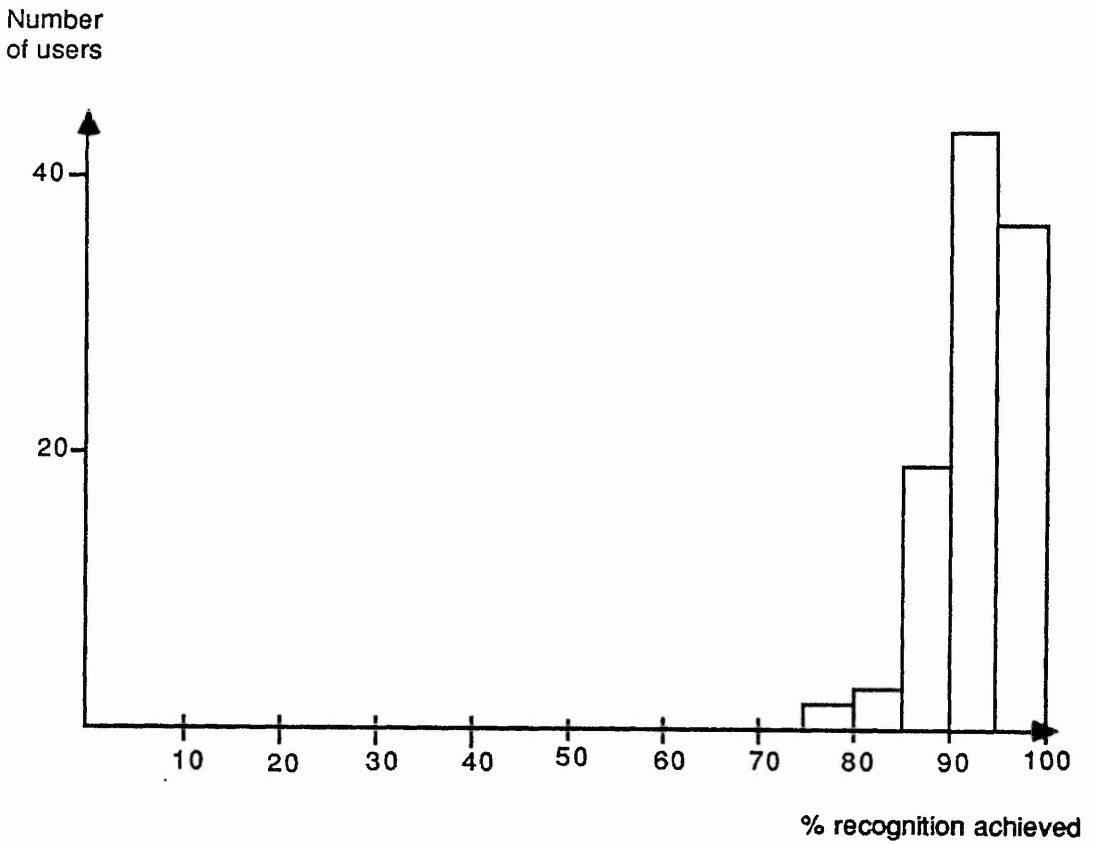


Figure 8.12 - Recognition Rate by Author Script

Sorting through the results, it was possible to determine the effectiveness of the matching algorithm. Errors in matching are broken down in the following table,

Composite Character	Number Found	Number Recognised	%age correct
a	7	2	28.6
b	30	30	100.0
c	0	0	-
d	30	30	100.0
e	14	5	35.7
f	145	123	84.8
g	11	8	72.7
h	6	3	50.0
i	346	343	99.1
j	143	143	100.0
k	119	97	81.5
l	0	0	-
m	2	0	0.0
n	2	0	0.0
o	0	0	-
p	64	62	96.9
q	12	6	50.0
r	0	0	-
s	0	0	-
t	297	283	95.3
u	7	7	100.0
v	2	1	50.0
w	0	0	-
x	202	189	93.6
y	16	8	50.0
z	16	16	100.0
TOTAL	1471	1356	92.18

TABLE 8.11 - MATCHING PERFORMANCE

The errors encountered in the matching procedure can be broken down as follows:-

(i) The two part strokes are not sufficiently close for a match to be attempted,

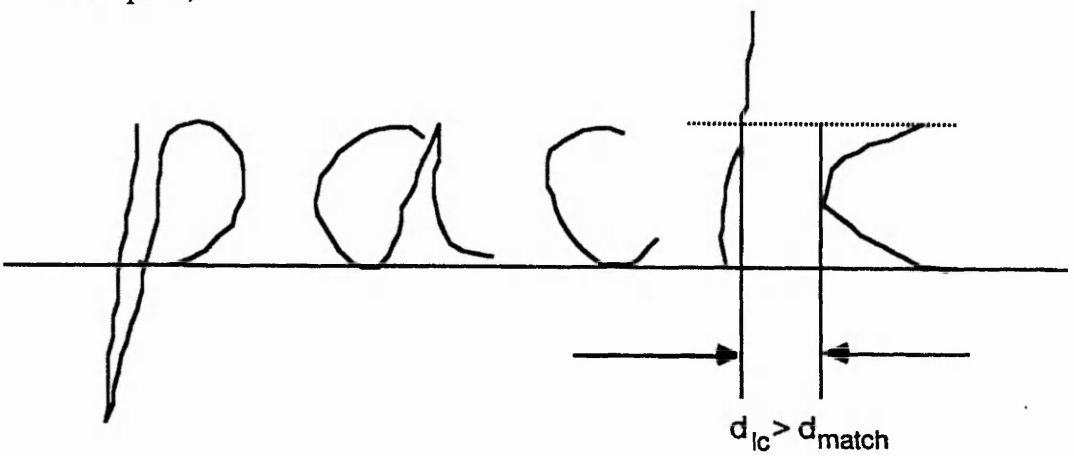


Figure 8.13 - Character Part Strokes Exceed Threshold

(ii) The two part characters are within the threshold, but are not in the matching array,

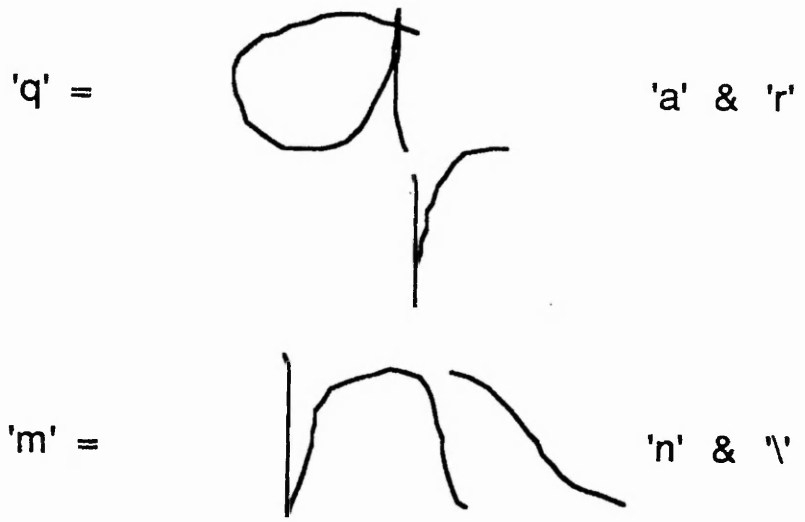


Figure 8.14 - New Combination of Character Part Strokes

(iii) A match is not found because the first stroke of the composite character has already been matched to a previous character,

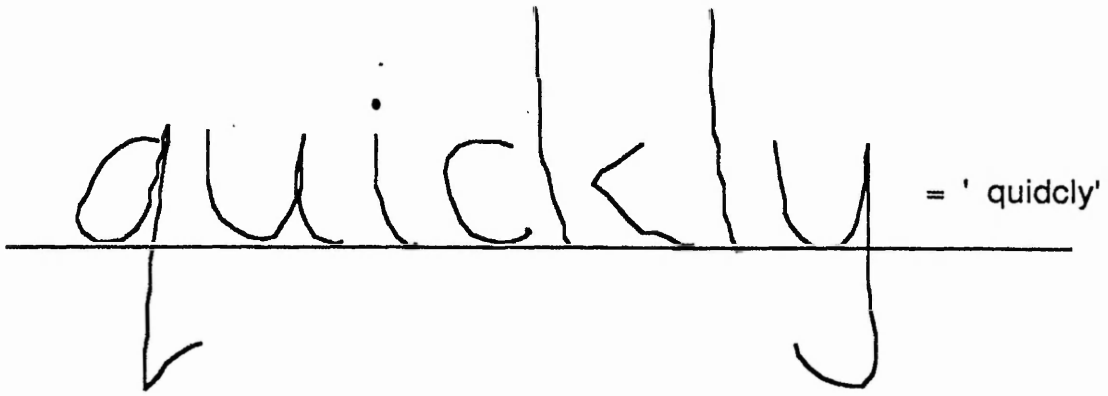


Figure 8.15 - Incorrect Part Stroke Combining

Of the most common composite stroke characters, 'i', 't', 'f', 'k', 'j' and 'x', the 'k' shows the worst matching performance with an 81.5% success rate. This problem was found to arise because an 'l' is not matched to a '< ' if they are of a similar y dimension. This is needed in order that we do not erroneously match undotted 'i's to 'c's.

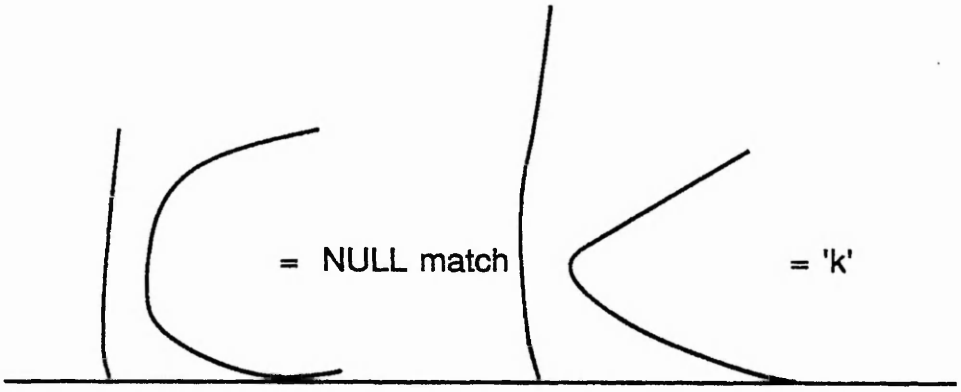


Figure 8.16 - Differentiating between 'k' and 'ic'

Although most writers do produce 'k's as in case (i) above, a number of such k's (around 15-20%) were like case (ii) above and so did not get recognised. This problem could be resolved by analysing the stroke dimensions with respect to the other strokes in the word in order to decide on a valid match, as below in Figure 8.17,

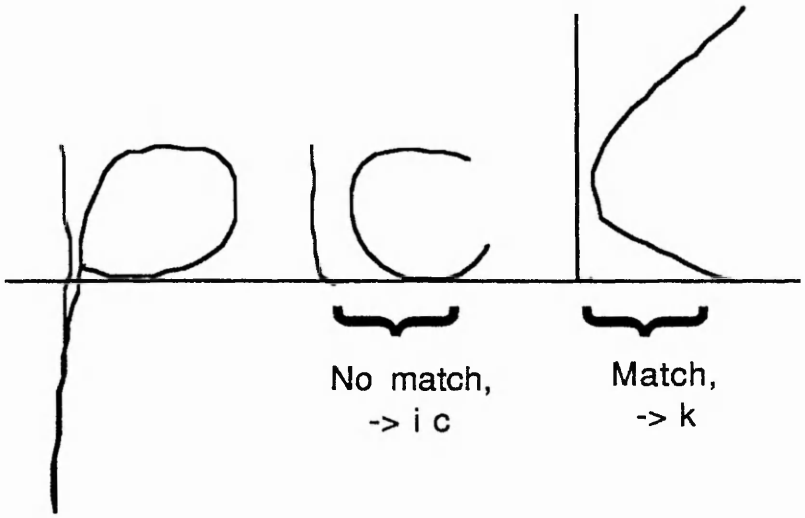


Figure 8.17 - Matching by means of relative word dimension analysis

This possibility is discussed in the concluding chapter.

8.2.5. Space Algorithm Performance Results

The space detection algorithm results are given below for the data presented to it from the 112 writer files. A human reader can quite easily define the correct word separation by recognising the characters. However, this algorithm has no prior knowledge of the character identities, only their distance apart. Hence, a human reader can delimit words that would not otherwise be separated, having only the spacing information. Errors in the algorithm only arise due to sloppy writing by the user. The results are,

Number of word spaces	1615
Number of spaces not detected	107
Number of extra spaces detected	14

This allows some measure of the performance of the space detection algorithm,

$$\frac{(1615 - 107 - 14)}{1615} * 100\% = 92.5\%$$

The problem of space detection would be greatly simplified in the analysis of more natural hand-writing where the relative sizes of the spaces between words are usually significantly greater than the spaces encountered between characters within a word. The main reason for the poor delimitation between words for those writers that the space detection algorithm did make errors on could be due to the fact that producing such unconnected script is not natural for most writers, and concentrating on not connecting characters tends to make the writer leave a larger gap between characters than they would otherwise tend to do.

8.3. Untrained Writer Results

In order to assess the robustness of the recognition algorithms and gain some idea as to how representative the databases are to the styles of any user, we decided to test 10 completely untrained writers. They were given two new test sentences to write and the data files were passed through the recogniser. The hard-copy of their attempts can be seen in Appendix C, along with a more detailed breakdown as output from the assessing program. The overall recognition rate determined was 88.14%, some 5% lower than that for those data samples in the 112 user set. A breakdown of results is given in Table 8.12,

	1st	2nd	3rd	4th	5th
Comparisons	843	92	41	36	34
Correct	743	51	5	2	2
Percentage	88.14	93.51	12.19	5.55	5.88
Cumulative	88.14	94.19	94.78	95.02	95.25

Table 8.12 - New Writer Results

Over the first and second alternatives, a recognition rate of 94.19% is achieved. The non-recognition was found to be evenly broken down between,

(i) bad stroke matching

(ii) bad tablet data (only partly captured co-ordinates)

(iii) no Freeman or XY encoding for a particular stroke shape.

It was particularly encouraging that the non-recognition due to not finding encodings in the databases was particularly low (around 1-2% of strokes). This bodes well for the construction of new Freeman and XY databases from 500 sample sets, and it is envisaged that the size of such databases will not be more than double the size of the present databases.

9. CURSIVE SCRIPT

9.1. Introduction

Although research into the field of script recognition has been conducted since 1960, as yet, little of this effort has resulted in a successfully marketed product. In order to achieve a greater degree of acceptability by potential users, certain requirements must be given particular emphasis.

It is important that the system be capable of recognising a writers natural writing style, although, obviously a writer is expected to observe some degree of neatness and consistency in their writing. Even so, the user should not be made to feel severely bound when using the system. Hence, the system should be able to recognise the full range of character sets that a writer might use when writing on a piece of paper. The obvious character sets would be:-

['a','b','c',.....'z']
['A','B','C',.....'Z']
['0','1','2','3','4','5','6','7','8','9']
['?','$','%','+',.....]

To date, script recognition products have limited the user to one or two of these character sets (e.g., upper case letters and numerals). Naturally, this limits potential applications. However, a far greater limitation is the style and placement of the writing. Systems hitherto have been confined to the recognition of unconnected letters only. The most successful product to date, the Penpad [90], requires the user to write either upper case, numerals and some punctuation characters within separate boxes on a piece of special graph paper. No training is required explicitly, although examples of the shape and styles of characters that can be recognised is given. Under such constraints, the system will produce very good recognition results (95+ %). Such a product is ideal for form filling applications, where a writer is required to construct his letters neatly and precisely. Recently, another US company has brought out a script recognition product, the Linus Write-Top [92]. This product includes an extremely extensive training and tutorial package. Once the system has learned a users writing style, it can subsequently recognise around 96% of written characters. Although not as tightly constrained as the Penpad product, the user must write on preset lines and ensure that small and large letters are written below and above a dotted guideline respectively. The constraint on size and placement of characters is a particular problem caused by the limitations of the recognition algorithms.

A more severe constraint on the user, however is the necessity that the user form their words using unconnected characters. When we were sampling our test data set of writers, it was observed that a number of people could not write a complete sentence of text using totally unconnected letters unless they had several attempts and gave the task their complete concentration.

An observation of peoples natural writing styles from memos or written notes shows that, in general, when a person writes a word, the letters within the word may be:-

- entirely unconnected
- some mixture of connected and unconnected letters
- entirely connected

The degree of connectivity between letters within a word depends on a number of factors, including:-

(i) word length.

(ii) the letters themselves (for example 'i's and 't's often cause pen breaks in a word).

(iii) the writers confidence in being able to spell the word.

This chapter discusses the preliminary work into the recognition of natural handwriting. Development of the work has placed particular emphasis on ease of usage. The requirement of a natural environment is of particular importance. Increasingly, a good deal more effort has been directed towards the requirements of the user interface by researchers over the last few years. The Linus product incorporates the 'electronic paper' concept which many people see as the corner stone of future developments. This hardware configuration can emulate the situation of a person writing on a pad or piece of paper with a pen or pencil.

9.2. Cursive Script Recognition - A Resume

From the state of the art review, a small number of researchers were found to have considered the problem of recognising connected handwriting dynamically. The obvious approach being to build on the techniques already developed for the analysis of isolated characters by identifying the bounds of letters within words. Alternatively, some researchers adopted a completely new approach. This being to consider each written word as a single unit to be recognised. One reason that this method found favour was the fact that humans, when reading a piece of text, are considered to identify words by considering the shape as a whole, rather than breaking the word down into its constituent letters. Hence, broadly speaking, two distinct approaches have evolved:-

- character level analysis
- word level analysis

9.2.1. Character Level Analysis

In this instance the word is broken down into its character components followed by separate letter identification. Characteristics within the word are identified and used as the basis for splitting the word into possible letter segments. Providing each letter position is successfully found, the problem of subsequent recognition is reduced to one similar to unconnected letter

recognition. It is of particular importance that the character recognition technique gives a very good recognition rate. For example, the incorrect identification of only one letter per word results in a word recognition rate of 0%.

A number of techniques have been used for word segmentation, including the detection of all y minima with a word by Mermelstein and Eden [93]. Harmon [94] performs segmentation by estimation of letter widths, and extracts the features from the resulting segments. However, the correct identification of the letter bounds within a word is particularly difficult, especially when attempting to apply it as a general method for any writer. The detection of one letter bound too many or one letter bound too few will make the subsequent task of identifying the word impossible, since the algorithm will now be operating under the wrong assumption in trying to process the wrong number of letters. To give an instance of the problems facing the technique of segmentation, let us consider the cursive word *'mummy'*. This word actually contains 5 letters. However, a technique of minima detection would find up to 13 segments, suggesting that the word might contain as many as 13 letters.

9.2.2. Word Level Analysis

As a result of the difficulties encountered in word segmentation, it became increasingly popular to perform recognition on the word as a whole. Features such as down-strokes, arcs, loops, and cusps are identified. This feature sequence is compared against a database of pre-written word features. To date, this method has proven to give better recognition results than the former. However, it does have some serious limitations:-

(i) the vocabulary size is very restrictive. One technique by Farag [11] used a dictionary of only 10 words. Wong and Fallside [66] apply a dynamic programming technique based on a technique for the recognition of continuous speech. However the results are only given for a small word sample (less than 10). In order for such a system to recognise a particular word, such systems need to be trained with at least one prior example of that word provided by the potential user. Hence a training phase would be necessary to allow the system to build up a database of vocabulary features, in which the potential user must write at least one example of every word that they might subsequently want the system to recognise. Even for a limited vocabulary set of 10-15000 words, such a task would prove too daunting for most people.

(ii) closely related to vocabulary size is the processing time necessary per written word. In general, processing time will be directly proportional to vocabulary size. This is not the case for the segmentation method where processing time is proportional to the length of the word. Hence the feasibility of such a system performing in real time on a large vocabulary is doubtful as a marketable product.

9.3. Word Segmentation

We decided that the technique of word segmentation would be attempted. If a reliable segmentation method could be found, it would then be a case of operating on the letter segments as in the case of unconnected script. Six writers were asked to write the two test sentences as before in Appendix B. The Freeman coding technique was used to encode the raw data. Analysis of the vector string showed that writers who wrote with little or no slant formed a ligature, which, when vectorised, produced a consistent, repeatable vector sub-string within the word. Depending on the complexity of the ligature, this would comprise some sequence of the vectors '0', '1' and/or '2'. This is best illustrated by an example. Figure 9.1 shows how the cursive word *and* is encoded:-

Figure 9.1 - Freeman Encoding of the word 'and'

The Freeman string produced by this encoding is:

and = '345670126701610767105670126 7'

This encoding highlights 5 ligature elements separating 6 segments (indicating that the word could contain a maximum of 6 letters).

seg_1 = 'c' = 34567, lig_1 = 012,

seg_2 = 'i' = 67, lig_2 = 01,

seg_3 = 'i' = 6,

$lig_3 = 10,$

$seg_4 = 'r' = 767,$

$lig_4 = 10,$

$seg_5 = 'c' = 567,$

$lig_5 = 012,$

$seg_6 = 'r' = 67.$

At this stage, if we did not know that the word *and* had been written, it is not possible to determine which of these ligatures are valid connections between letters (inter-letter shapes) and which are actually a part of a letter (intra-letter shapes). For our word *and* we can see that ligatures 1,3 and 5 are intra-letter shapes and ligatures 2 and 4 are valid inter-letter elements and not part of a letter. In order to reconstruct the letter shapes for this word, we simply recombine the appropriate neighbouring segments via their joining ligature shape, ie,

$$seg_{12} = 'a' \equiv 3456701267,$$

$$seg_{34} = 'n' \equiv 610767,$$

$$seg_{56} = 'd' \equiv 56701267.$$

In this case we know which segments to join together to obtain the correct letter shapes. However, had we not had this prior knowledge, we could equally have joined the segments either side of the valid ligature shapes. Quite often this will also lead to the formation of a valid Freeman letter vector string. In this instance we get,

$$seg_{23} = 'u' \equiv 67016,$$

$$seg_{45} = '?' = 76710567.$$

The first combination gives another valid ligature shape, but the second does not. Therefore, it is necessary to investigate each possible route through the word in order to determine which route or routes results in a valid word. Figure 9.2 shows a letter net constructed for the word *and*. Each node in the net being a letter possibility resulting by decoding and identifying the Freeman string as per the unconnected character analysis.

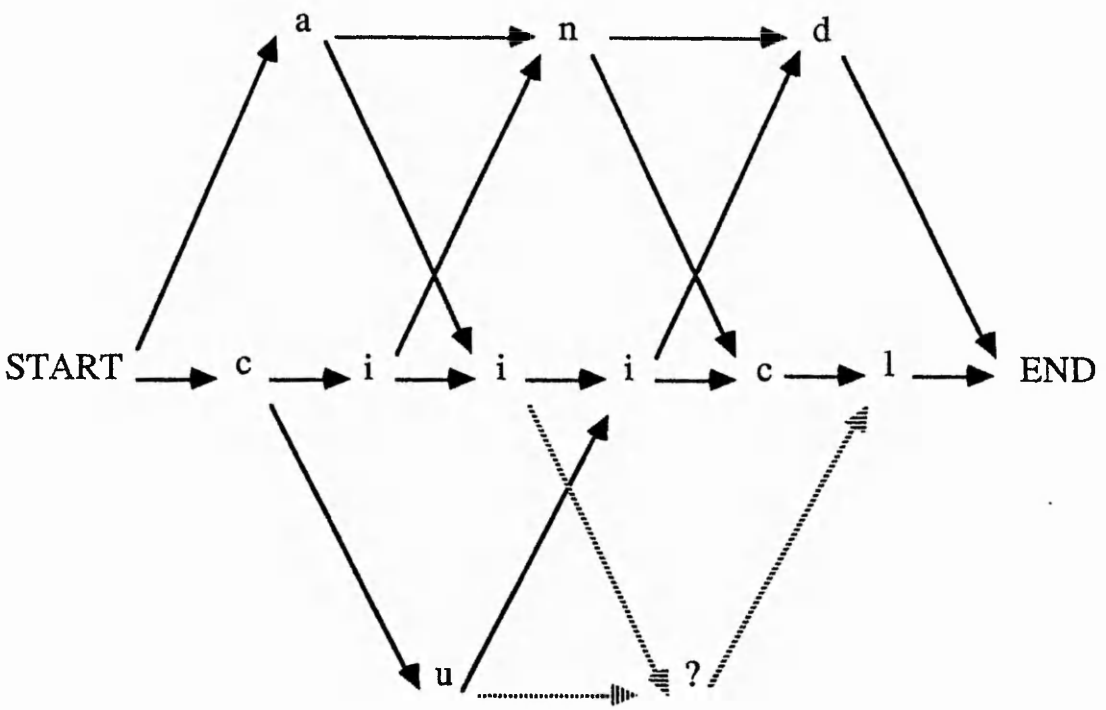


Figure 9.2 - Letter Net Constructed for the word 'and'

The number of routes through the letter net follows the Fibonacci number series;

$$F(0) = 0,\ F(1) = 1,\ \ldots\ldots\ldots\ldots F(n+1) = F(n) + F(n-1),\ n \geq 0$$

i.e. 0, 1, 2, 3, 5, 8, 13, 21, 34, 45,..... (9.1)

Therefore, the six segments detected in the word *and* indicate that a total of 13 routes exist through the letter net. These produce the following string alternatives,

ciiicl - no 2 segment combinations

aiicl - one 2 segment combination
cuicl
cincl
cii?l
ciiid

ancl - two 2 segment combinations
ai?l
aiid
cu?l
cuid
cind

and - three 2 segment combinations

At this stage we discovered a flaw in the method. The technique did not allow for an 'm' within a word to be recognised. This is because the letter 'm' comprises not two but three successive segments. Therefore, once all possible one and two segment letters had been processed it was necessary to identify any possible 'm' occurrences within the word. This is performed by the identification of three successive small, straight single segments. These can be identified after Freeman analysis as having the identity 'i'. Our word *and* in fact has three such elements. Therefore we should also consider the possibility of an 'm' existing within this word.

seg_{234} = *'m'* = 6701610767

This adds a further level of complexity to the letter net, as in Figure 9.3.

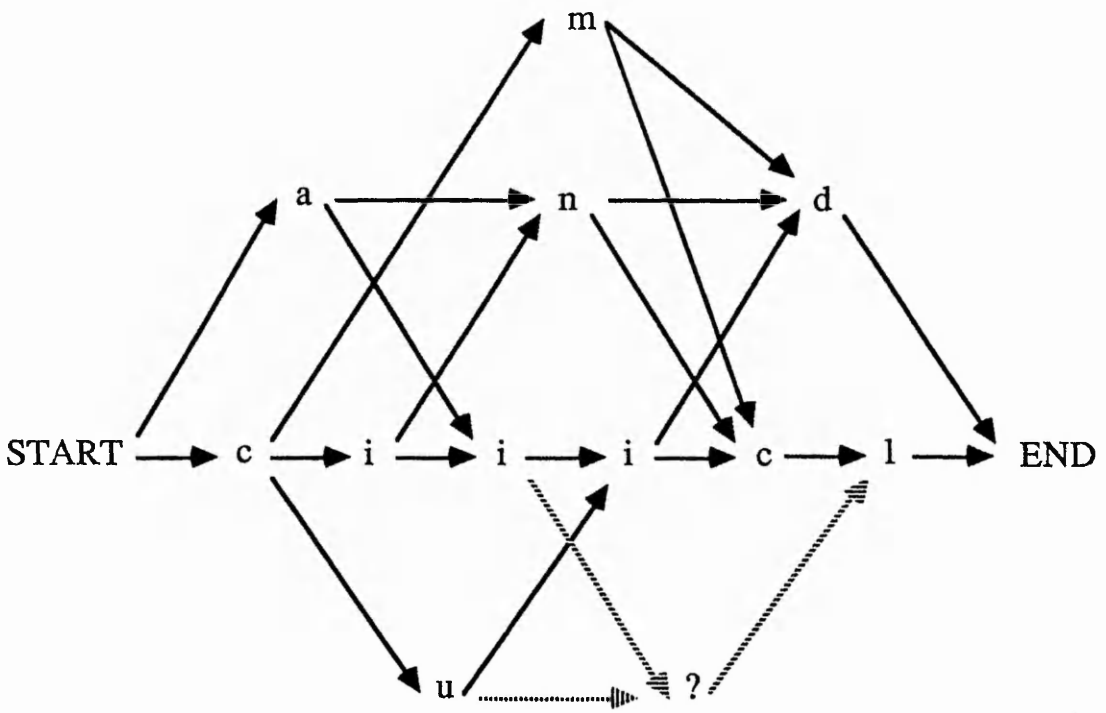


Figure 9.3 - Inclusion of 'm' in the letter net

This produces two further letter sequences,

cmd - one three segment combination
cmcl

In the case of a small number of letters it is necessary to analyse the shape of the ligature between the segments, since in some instances it is actually part of the letter itself. This is best illustrated diagrammatically in Figure 9.4.

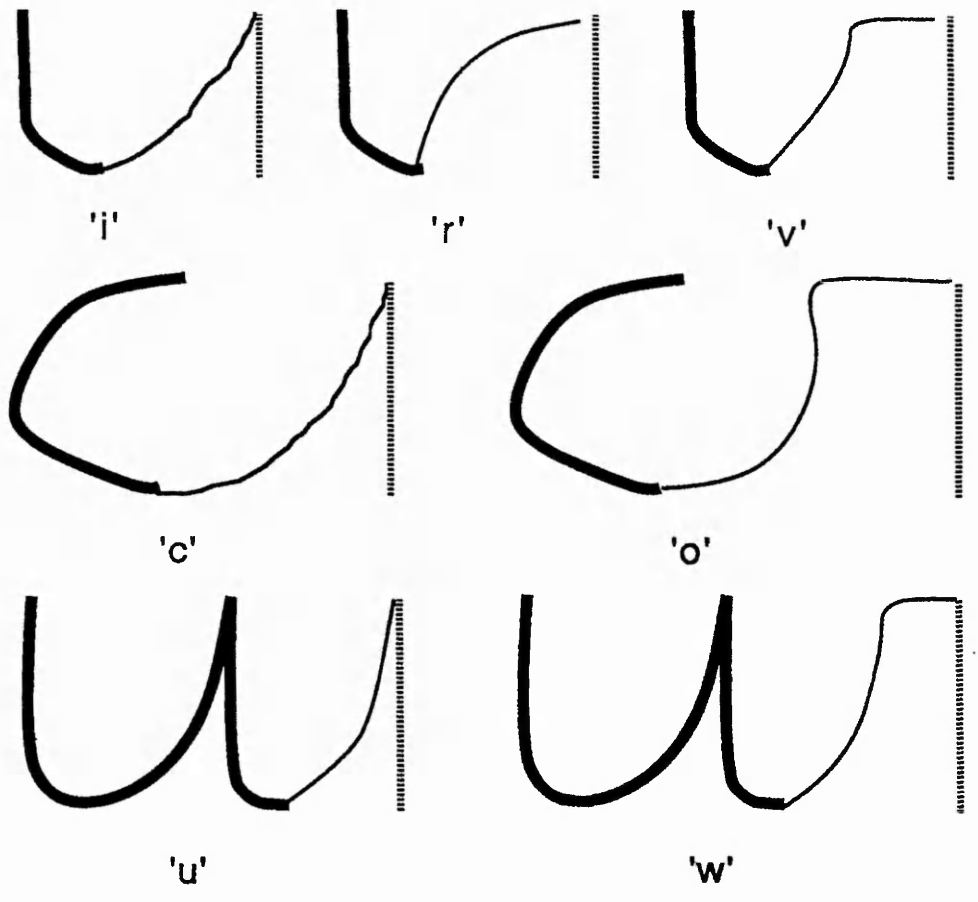


Figure 9.4 - Letter identification by ligature shape analysis

In these instances the ligature shapes are used to order the possible character identities rather than eliminate some possibilities. For example, although a ligature shape might indicate a character 'r' had been written, characters 'i' and 'v' are not excluded at that particular node, they are simply to be found further down the list with lower confidences.

Assuming that the Freeman algorithm has identified the character shape successfully at each node, it is necessary to determine which of these letter sequences, if any, gives a valid letter sequence, and thus is a valid word possibility. However, we have shown in the results in Chapter 8, that the recogniser sometimes identifies the character as only its 2nd, 3rd, 4th or 5th choice. After word segmentation, the techniques for identifying the possible letter regions, based on the unconnected letter identification algorithms, produced very good recognition rates at the character level. The two main reasons for this are:-

(i) there is less confusion at the individual character level due to the ticks produced by the pen-up and pen-down action. This is because there is now less letters delimited by the movement of the pen onto or off of

the paper.

(ii) because the initial work is being performed on a writer dependent basis, the Freeman database is more specific to a single users style, greatly reducing character confusions.

Therefore, the individual character recognition rate can be 95+ %. Unfortunately this figure quoted is not particularly meaningful when considering cursive script. Consider 10 ten letter words written cursively. In each case 9 out of the 10 letters in each word has been recognised correctly as first choice. This would correspond to a recognition rate of 90%. However, it also corresponds to a word recognition rate of 0%. The problem that we have is that it is not possible to tell (with a high degree of certainty) exactly how many letters there are in a cursive word. In our example *and* this number was anywhere between 3 and 6. For a particular individual, if it were possible to achieve 100% recognition at the character level over say the first 5 or 6 alternatives, this could be the basis for some means of higher level analysis that would identify routes through the letter net by some means of comparison with a dictionary of allowable words. Some initial analysis using N-gram techniques has been undertaken in section 9.6 to determine the initial performance of the cursive script recognition program. However, more formal and advanced techniques already exist. L. Evett et al [95] describe the work being undertaken at Trent Polytechnic into the analysis of letter sequences and methods for dictionary look-up procedures.

9.4. Segment - Ligature Correlation

Having segmented a word into a number of segment and ligature substrings, it became apparent that most writers would form certain characters within a cursive word by some combination of successive segments and ligatures. As a rule, we can classify the lower case alphabet in terms of such combinations:-

Single segment

c, e, i, j, l, s, z

Segment ligature combination

o, r, v

Segment ligature segment combination

$a, b, d, g, h, k, n, o, p, q, u, x, y$

Segment ligature segment ligature combination

w

Segment ligature segment ligature segment combination

m

Segment and *cross stroke*

f , *t*

This is not meant as an exhaustive list. Even from our small sample set, it was observed that the same writer would form a specific character in a different way depending on its position within a word (especially for characters commencing a word). Characters most often formed in different ways were,

b, *f*, *k*, *p*, *s*, *t*, *u*, *x*, *z*

It was apparent, from studying our very small initial sample set, that it was not possible to rely on dotting information in order to identify the position of the letters 'i' or 'j' within a word. In many instances a writer would omit the dot altogether, and in many instances when the writer would dot the word, it would not be over the top of the 'i' or 'j' that it was meant for, but over some other letter. Therefore it was decided that the letters 'i' and 'j' would have to be identified without the help of the dotting information.

In some instances, it was also necessary to interrogate the shape of the ligature leading up to the character shape. Figure 9.5 shows how the characters 'c', 'e' and 'z' are put into order of confidence by analysing the leading ligature.

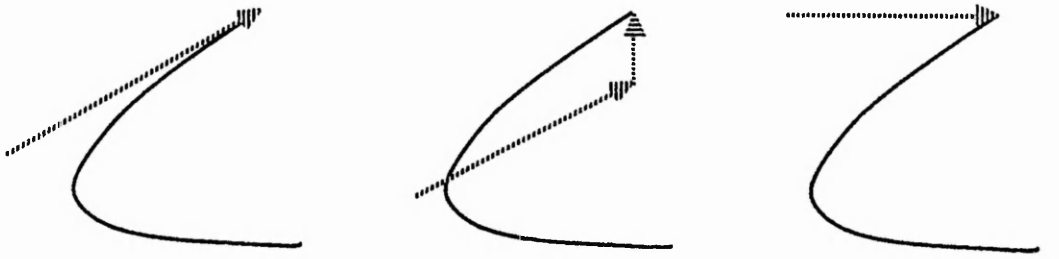


Figure 9.5 - Analysis of Ligature Shape

9.5. Natural Handwriting

As was pointed out in the introduction, people tend to use some combination of connected and unconnected letters within words when writing a piece of text. In some cases, it may be a users natural style to connect every letter within every word that they write. On the other hand, some writers are most comfortable forming every letter disconnected from the last. In general however, people use some mixture of unconnected and cursive text. We want a system that can cope with any mixture of writing style. Initially it was thought that it would be of great help to the recogniser if people did generally make pen breaks during the process of writing a word. Such pen breaks would mean the elimination of an inter-letter ligature. This would mean an easier task for the recogniser. For example, take the case where a user writes our word *and* but lifts the pen after forming the *a*. Instead of a six segment Fibonacci search we would have a two plus four segment Fibonacci search. From 9.1 this would produce:-

and	6 *segments*	13 *letter strings*
a nd	2 + 4 *segments*	2 *x* 5 = 10 *letter strings*

It was soon realised that this would not be a viable assumption. From table 8.11 in the results (Chapter 8) it can be seen that, for our sample set of unconnected text a total of 1471 character were formed from more than one single pen stroke. Subtracting the dottings of 'i's and 'j's this left a total of

982 characters. From a total of 8960 written letters, this represents around 11% of all written letters. A person is equally likely to form some characters with two pen strokes when writing in their natural style. The obvious candidate is the letter 'k'. If this is the case then our previous thinking would be invalid. In the case of the two-stroke 'k' we would assume a letter sequence 'lc' and never consider the possibility of a 'k' being written. Figure 9.6 shows the extremes that could possibly occur in different users writing styles.

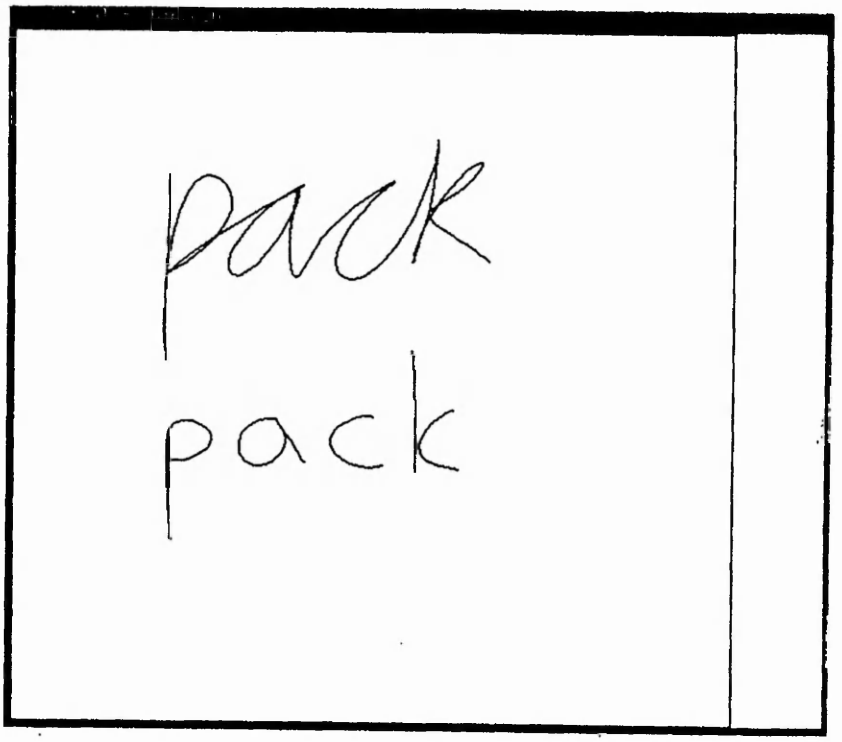


Figure 9.6 - Extremes in Writers' Styles

Therefore, no prior assumptions were made as to the construction of a word until a number of pen strokes were identified as being a complete word. Since we are still working with a disjoint tablet and screen, we have a similar problem as to when to display the recognised text. It was appropriate at this stage to display each word after it was recognised, since, presently, no post-processing is performed after the word analysis. This will lead to initial confusion for a writer, since the system will not display a recognised word until the next word has been started. If a writer is not fully conversant with a particular word, he may stop for a number of seconds after writing part of the word in order to decide how the rest of the word is spelt. Therefore, he would become very confused if, at that point, the system decided the writer had finished the word and processed and displayed a half complete word. Therefore, the only way of being certain that a writer has finished a word is to detect that he has started a new word. Figure 9.7 shows a breakdown of the natural handwriting recogniser. The general construction is very similar to the unconnected script recogniser shown in Figure 10.2.

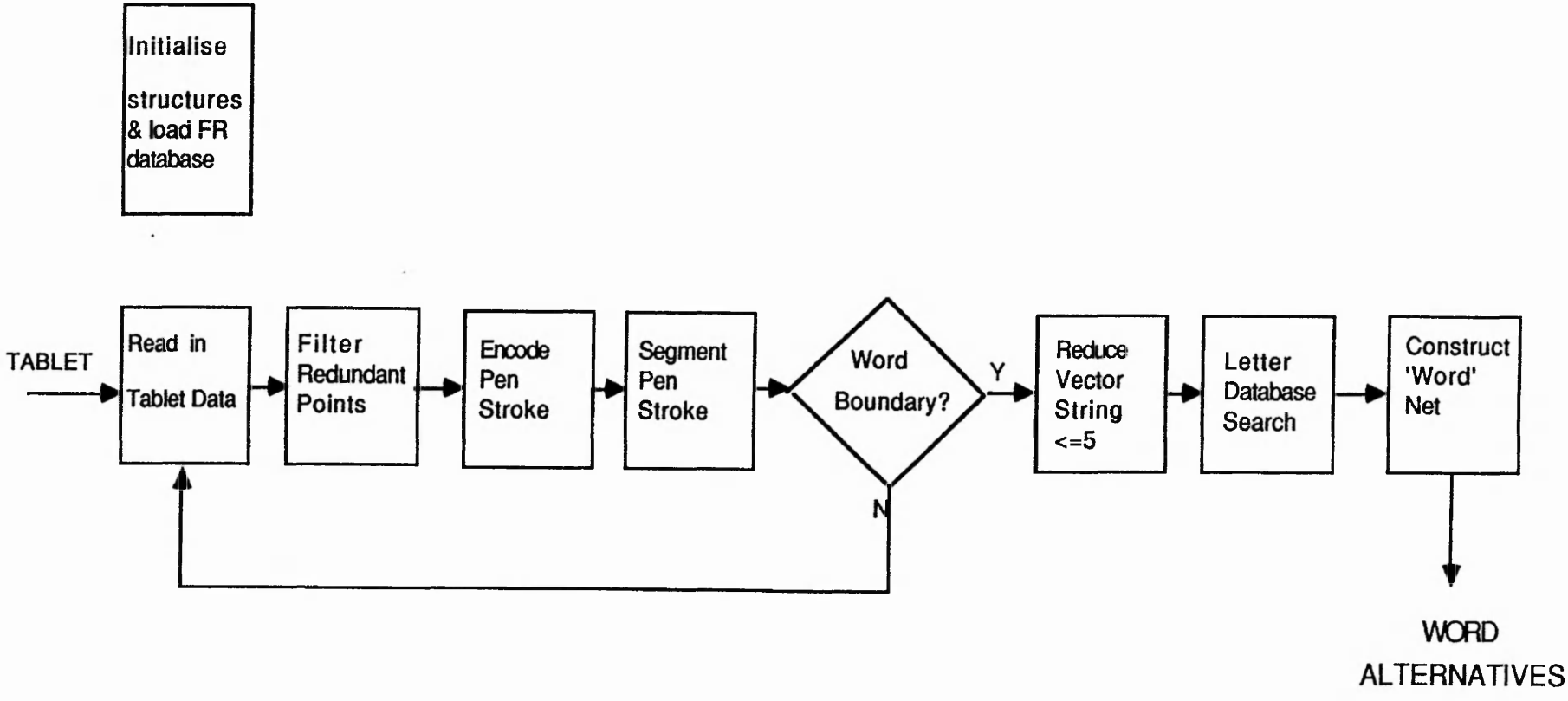


Figure 9.7 - Cursive Script Recogniser

In order to cater for any multiple pen stroke characters, pen breaks detected within a word (apart from cross-strokes for 't's and 'f's which are treated separately) are removed by the insertion of a ligature from the end of the last pen stroke to the beginning of the new pen stroke. This effectively converts every unconnected letter string into a cursive word. Obviously, in the cases where the letters are a single pen stroke we seem to be complicating the problem. However, we have now developed a method of treating every written word in the same way. Figure 9.8 shows the technique of ligature joining,

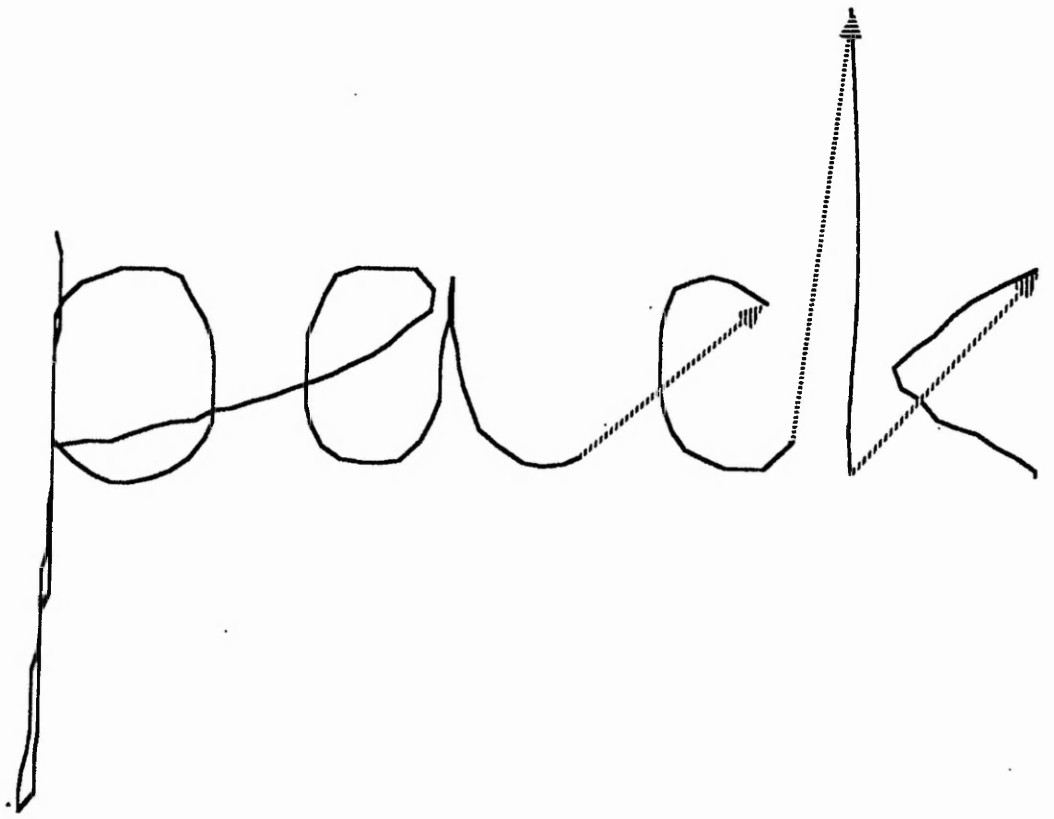


Figure 9.8 - Ligature Joining

9.6. Initial Results

Initial results of the natural handwriting recogniser have proved to be very promising. A user, having trained the system, can achieve 95+ % at the word level corresponding to a character level recognition rate of 99+ % (taking the first six alternatives). Depending on the number of words in the dictionary search tree, some of the recognised words may not come out as first choice. A user is able to look through the option list by simply dotting the

stylus onto the appropriate word. This will reveal the 2nd choice, 3rd choice and so on to a maximum of 10 alternatives. Figure 9.9 shows the raw pen motion produced from the Numonics 2200 tablet when a user writes some text on the tablet.

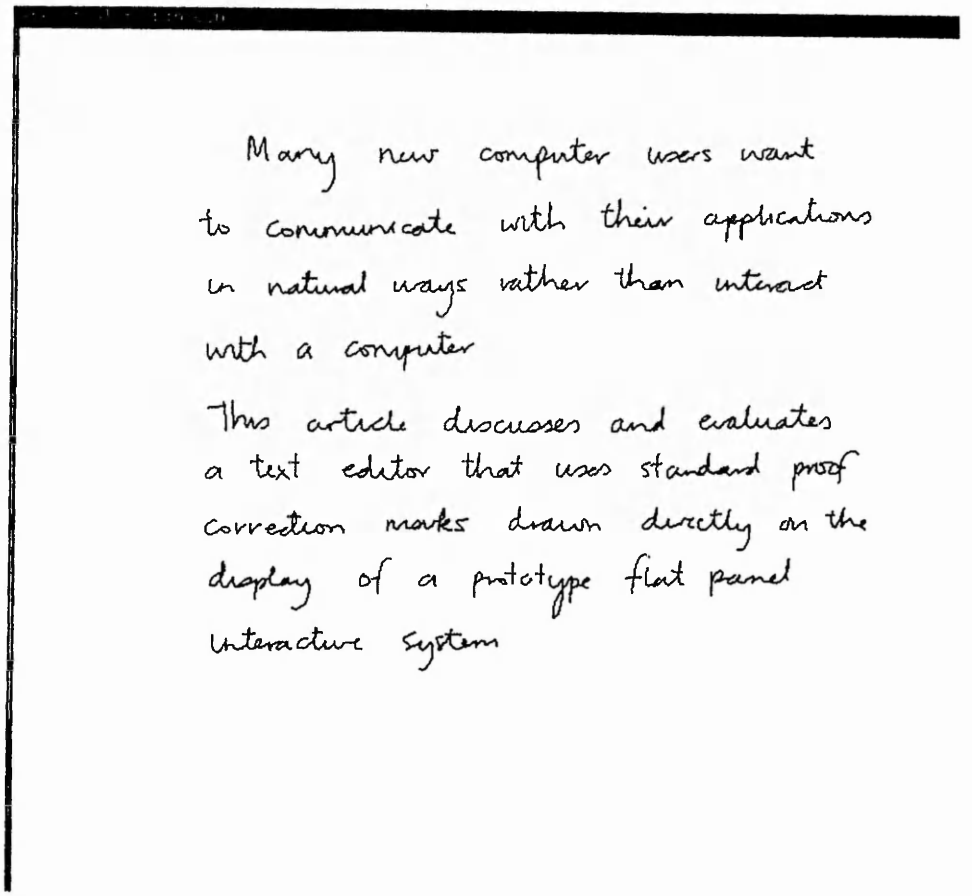


Figure 9.9 - An example of natural handwriting

Letter nets were produced resulting from the recognition of the raw text (shown above in Figure 9.9). By identifying each possible letter sequence from the start to the end of each letter net, it was possible to compare the sequence against some lexicon in order to gain some insight into the performance of the script recogniser. The lexicon used was the UNIX dictionary. At this stage one point became apparent. The lexicon must contain all the words required to be identified. Hence the UNIX dictionary was supplemented with those words in the test script that it did not already contain. The result of comparing each possible letter sequence for a word match in the lexicon produces the word alternatives shown over the page. As can be seen, some words have a number of alternatives. These alternatives have

been ordered using the individual letter confidences. However, this is not seen as a definitive method of finding the most likely word. For example, consider the word *many* producing the word alternatives *many*, *maim* and *norm* :-

1. *m*:75 *a*:89 *n*:69 *y*:79 *Confidence* = (75+89+69+79)/4 = 78

2. *m*:75 *a*:89 *i*:75 *m*:45 *Confidence* = (75+89+75+45)/4 = 71

3. *n*:79 *o*:49 *r*:67 *m*:45 *confidence* = (79+49+67+45)/4 = 60

Many	new	computer	dais	want	to	communicate	with
Maim	mu		users	mint	it		
Norm	rim						
	now						
	inn						

their	applications	m	natural	ways	hither	them	interact
then		on		loam	rather	than	
fum		in				thorn	
friar						thou	

with	a	computer.	
		couturier.	

this	article	discusses	and	evaluates	a	text	editor
flip	follicle		ana			auto	
tub			dud				
trip			curd				

that	uses	standard	pilot	correction	marks	drawl	directly
float			pivot	coalition	marry	drawn	quietus
flout			proof		mealy	drank	
trot							

on	the	display	of	a	prototype	flat	panel
en	floe			d		foot	band
oil	floc					tart	bard

interactive system.

This resulting word alternative sequence shows that 39 out of the 45 words written are found as the most likely word alternative, a recognition rate of 86.7%. However, if we encompass the first three alternatives for each recognised word we achieve a recognition rate of 100%.

The UNIX dictionary, consisting of only 24473 words, cannot be considered as an adequate lexicon for general usage, but it does give some indication as to the word alternatives that might arise. For example, *new* and *now* would seem to be reasonable options. However, it is not so easy to explain *ways* and *loam*. Below is given a breakdown of the word distribution by character size for the UNIX dictionary:-

Letter count	*Number*	*Percentage*
1	26	0.01
2	91	0.37
3	759	3.10
4	2142	8.75
5	3097	12.65
6	3795	15.50
7	4045	16.53
8	3578	14.62
9	2970	12.14
10	1890	7.72
11	1072	4.38
> 11	1008	4.12

It is reasonable to assume that a larger dictionary would have a similar spread of word sizes. Therefore, one would expect that letter sequences 7 characters long would produce the largest number of word alternatives.

9.7. Future Work

Areas for future work which have arisen as a result of developing the cursive script recognition system include:-

- Word splitting and word joining. In some instances it is not possible for the cursive script recogniser to detect a word boundary when a writer places two words very close together. Likewise, when a writer breaks a word up, he often makes such a large gap that the recogniser interprets it as meaning two separate words. In such cases, analysis of the letter nets by the post-processor usually results in no valid word being found. In such cases splitting or joining of letter nets can often result in the correct word boundaries being detected.
- The investigation of a dictionary look-up technique that can be applied to the output of the cursive script recogniser. The code for the recogniser has been downloaded onto a FORCE micro-system. This has a 68020 processor and runs at 12.5 MHz, operating under the real-time operating system, PDOS (Programmable Disk Operating System). Reading of the tablet information, vector encoding, segmentation and generation of the letter nets has proved to run in a real-time environment with no observable delay for a user writing on the tablet. The important two factors for the dictionary look-up will be the size of the dictionary. What can be considered a reasonable lexicon? 15,000 words, 30,000 words, 60,000 words? The size of the lexicon will affect the amount of comparison that must be performed for each letter net. Hence the mechanism for searching the letter nets and comparing against the database must be very efficient, both in terms of memory and time if real time operation is to be achieved.

- One area that will be of particular importance is the investigation of syntactic analysis as an aid to the word recognition. A syntax analyser can check whether a sentence of words are grammatically correct. For example, if a word within a sentence is recognised with equal weighting as *dog* or *clog*, sentence level analysis could resolve the choice by comparing the two words within the context of the sentence as a whole.

There is also still a good deal of work to be done on the recognition algorithms. A further investigation into natural writing styles has shown that generally, the segmentation techniques holds good for a large number of writing styles. However, for people who write with particularly severe slant to the left, analysis of the ligature sections for these writers indicated that apart from the '0','1' and '2' vectors, the ligature was often found to contain the '3' vector. The possibility of including the '3' vector into the segmentation algorithm is being investigated.

Another area which is under investigation is the design of a training program, whereby a user writes some test sentences and the program extracts the characters from the text and constructs a user specific database from the input for subsequent use when the writer wants to use the system. At present, a program has been written which allows the user to enter examples of lower case, upper case, numerals and special characters and constructs the appropriate databases. However, we could not use the program as it is to build a database for use when a person uses the cursive script recogniser. Most people form a good proportion of connected characters completely differently from the way they write unconnected characters (most obviously 'f', 's', 'x', 'z'). Hence a system needs to be devised that can accept cursive words and extract the character shapes from them, using the prior knowledge of the word identity.

Apart from allowing a user to dot previously written words in order to check alternatives where the word is not recognised as the best choice, a number of other simple editing functions have been implemented on the demonstration system on the FORCE computer. These are,

(i) Word over-write. In some instances the recogniser will not recognise a word at all. This will be the case where a character shape in the word is not known to the recogniser. The user can enter the 'EDIT MODE' and try again by writing over the top of the word and the recogniser will process the new attempt and display the result in place of the old word. Other instances where the word is not recognised correctly can be due to the writer mis-spelling the word or the word being written not existing in the post-processors dictionary.

(ii) Word delete. Again, by entering the 'EDIT MODE', the user can delete any number of previously written words by simply striking a horizontal stroke through the words he wishes to delete.

(iii) Word insert. The user enters the 'EDIT MODE' and identifies the two words between which he wants the text inserting by marking the gap between them with an inverted 'v' symbol. He then proceeds to write

the text to be inserted. The reformatted line, incorporating the inserted text will be displayed after the user has exited 'EDIT MODE'.

These editing functions have really shown the requirement for electronic paper. The complexity in manipulation of recognised text display increases dramatically with successive editing permutations.

Thought must also be given to the construction of an upper case, numeral, and special character recogniser before the consideration of a recogniser with the capability to cope with the full range of characters and styles a writer might want to use. The ability of the cursive script recogniser to cater for unconnected script has resulted in a rethink in the design of the upper case, numeral and special character recogniser. This is discussed in the concluding chapter.

10. CONCLUSIONS AND FURTHER WORK

The results of the work to date have shown promise in terms of both recognition rate achieved and capability of the algorithms to run in a real-time environment. While the overall recognition rate for the trained sample set (Table 8.9) gives a performance of 98% for the first five alternatives, it is particularly encouraging to achieve a 95% recognition rate (Table 8.12) for a completely untrained writer set. This gives credence to the assumption that the size of the database required for a more representative user independent database should follow the extrapolation indicated in Figures 6.10 and 6.11.

10.1. A Real-Time Environment

The development environment did allow for a degree of real-time operation. Figure 10.1 shows the development environment.

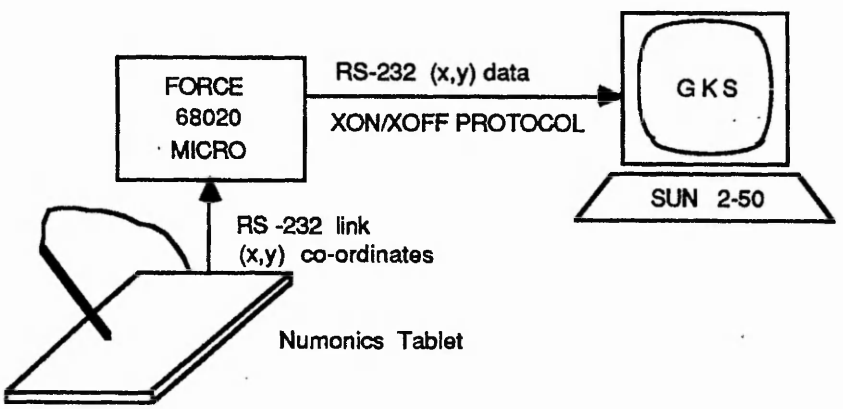


Figure 10.1 - Initial Development Environment

The FORCE micro-computer acted simply as a buffer to ensure that no pen motion data was lost as the user wrote on the tablet. This is necessary since the UNIX on the SUN, not being a real-time operating system, could not schedule the reading of data from a serial port in order that no data was lost. However, it was still found to be necessary for the writer to periodically pause when using the system to allow for the SUN to 'catch up' by clearing the script buffer on the FORCE system. The serial link to the SUN operated on a simple XON-XOFF protocol. The processed textual information was displayed on the SUN using the Graphical Kernel System (GKS) graphics utility. GKS provided the flexibility necessary to rapidly manipulate areas of text on the screen, so replicating the actions of the writer creating the text being written over the Numonics tablet. GKS has also proved to be an invaluable tool in other aspects of the research work. It has been incorporated in the design of a number of development tools. Validating input data was of particular importance. Tablet data often became useless during prolonged operation. This was due to the electromagnetic field being adjusted to the surface of the tablet in order to detect pen up states without needing to use the pen-switch which proved useless for analysing handwriting. As the tablet was switched on for long periods, the field would waver around the surface of the tablet, so that either pen motion was detected before the pen tip reached the tablet surface, or (the other extreme) pen motion was missed even though the tip was on the tablet surface.

The other major task was checking the shapes in the Freeman and XY databases. It was often possible to enter a character shape into the database with the wrong character identity. In order to avoid mis-recognitions due to this error it was necessary to periodically check the integrity of the databases after a major addition to the number of codings contained therein. GKS allowed easy development of a shape regenerator. Hence erroneous entries could be easily detected and removed.

However, the use of GKS in this form beyond the development stage was not seen as particularly viable. The executable run file size on the SUN without GKS was 0.25Mbyte, but this increased to around 1Mbyte by linking in the GKS run-time code. Since we are using only a small fraction of the full GKS capability, a graphics interface with more specific functionality would be better suited for a standalone demonstration of the recognition algorithms.

It has been possible to achieve a better degree of real-time capability of the algorithms by dowloading the C code onto the FORCE and re-linking to run on the FORCE. This has been achieved both for the unconnected recognition program and the cursive recognition algorithms described in Chapter 9. In fact, the cursive recognition program, building up the letter nets and incorporating the simple editing functions described in chapter 9 resides in less than 60Kbytes of code on the FORCE. Presently, we are still using the serial link to the SUN. In this instance it is used to send the GKS calls to the SUN in order that the GKS can still be used for display purposes.

10.1.1. Possible Speed/ Memory Improvements

It was decided to investigate speed improvements that might be made in the unconnected script recognition code once they had reached a relatively bug free, stable state. Figure 10.2 shows a breakdown of the separate tasks within the program, indicating the relative time spent by the processor in each function during the course of a text creation session.

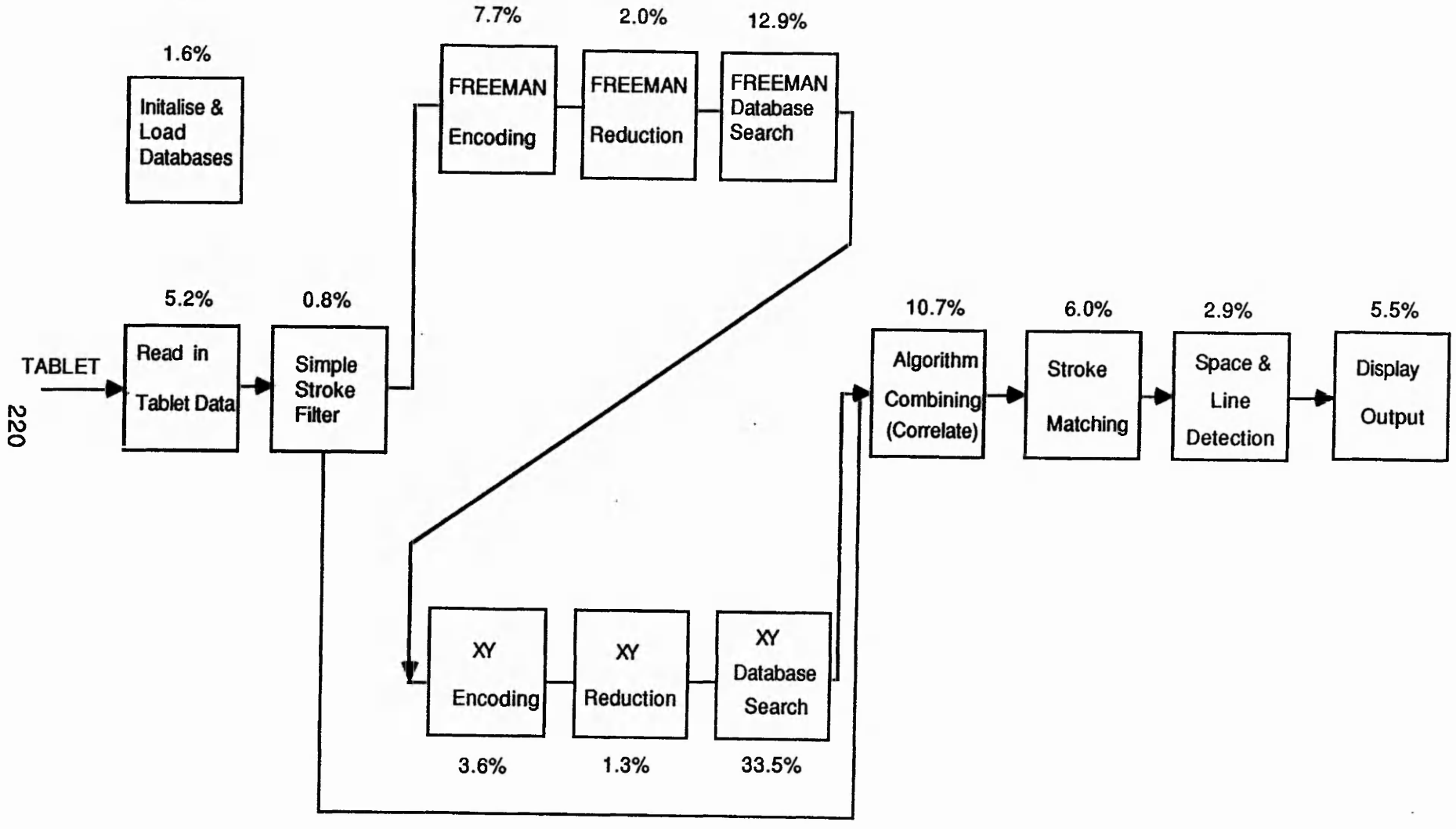


Figure 10.2 - Process Breakdown By Time

This analysis was obtained by running the recogniser on the SUN using the UNIX utility, *gprof*. Dummy C routines were linked in, in place of the GKS code for reasons of clarity in analysing the *gprof* output. The code was then recompiled and run to produce an execution profile. The UNIX command is simply:-

gprof executable_file

Most of the time is spent doing the actual character recognition tasks. This is a total of 60% of the processing for the XY and Freeman tasks combined. One development route might be the design of a dual-processor architecture, paralleling the XY and Freeman tasks. This alone would result in a reduction of around 20% in overall processing time.

Within the algorithms themselves, it was noted that the largest sub-tasks were database searching and string manipulation. The XY database in particular, with its more complicated encoding format, would benefit significantly in terms of processing requirement by simplifying the search. This could be achieved by reformatting the XY encoding so that storage requirements and processing time are both reduced. At present we have a typical encoding format:-

'a' −0.10 −0.25 0.20 0.25 0.20 / 0.10 −0.20 −0.10 0.30 −0.30

Each trend is assigned a signed floating point number for storage. This encoding format could have been represented thus:-

'a' −10 −25 20 25 20 / 10 −20 −10 30 −30

These XY parameter distance could have been stored in a *char* variable in C, making the string manipulation considerably faster. Storage size would also be greatly reduced. The floating point number requires 32 bits for storage, while the character byte only requires 8 bits. Hence a space saving of around three quaters for the XY database.

It has become increasingly apparent, from the results obtained for the unconnected script recogniser, and from the development of the cursive script recognition system, that the Freeman algorithm has proved significantly more robust and versatile than has the XY algorithm. However, investigation is necessary to determine what role, if any, the XY algorithm has to play in the further development of the cursive script algorithms. Although the XY algorithm has shown that it is not particularly good at recognising characters as best choice, it is marginally better than the Freeman algorithm when incorporating the best five alternatives. (Compare Tables 8.4 & 8.7), and so may yet play some part in the cursive recognition algorithms. As an initial idea, the raw data could be re-processed with the XY algorithm if no valid word match is found after processing by the Freeman algorithm.

10.2. Extension of the Character Base

Some initial work has been done with regards to the recognition of upper case characters, numerals and some of the more commonly encountered special characters. (See Appendix B). Around two-thirds of the 112 writers used to create the lower case databases also gave examples of the above mentioned character sets. It would be interesting to see how the unconnected script algorithms would cope with a different character base.

The major difference in the design of the recogniser was the construction of the matching array required to identify the various part character shapes that may be produced. The task of database construction was particularly easy in comparison. Although these new character sets contain more complex character shapes, it was found that such characters were created by forming a number of strokes, often just a number of straight line elements. For example, 'A','E', considerably fewer shapes to encode (supported by the fact that the Freeman and XY databases were smaller), even though the number of characters supported in the databases was about twice the number for the lower case alphabet.

Consider the four-stroke 'E':-

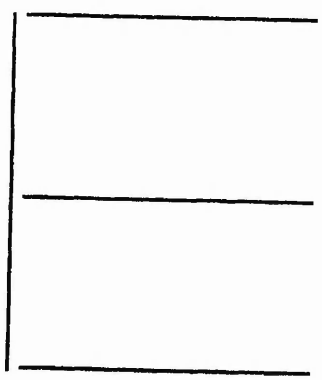
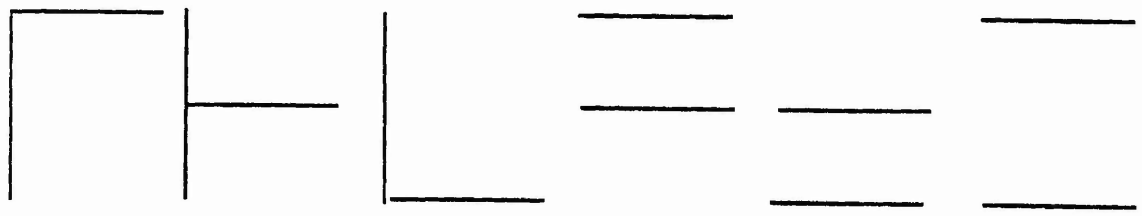


Figure 10.3 - Four-Stroke 'E'

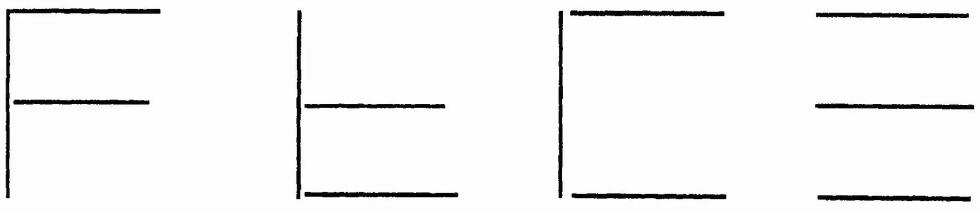
The number of alternative formation routes is determined as $n!$, n being the number of strokes in the character. Therefore, the four-stroke 'E' can be formed by 24 different stroke sequences.

The variety of partly formed character shapes that needs to be identified for this example alone was found to be:-



(i) (ii) (iii) (iv) (v) (vi)

Intermediate two-stroke shapes

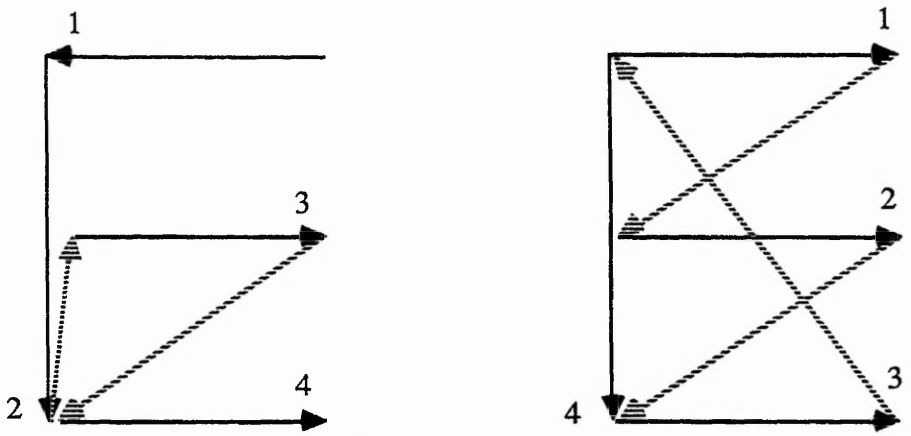


(i) (ii) (iii) (iv))

Intermediate three-stroke shapes

Figure 10.4 - Character 'E' Intermediate Shapes

At this stage it became apparent that the basic approach to the matching would not be suitable for the upper case aphabet. Instead, the concept of character stroke combination, adopted in the cursive recogniser to cater for multi-stroke characters, was seen as a far simpler and more effective approach. Figure 10.5 shows how this combination technique would operate on two differently formed character 'E's:-



E_1 = '462050'

E_2 = '0505036'

Figure 10.5 - Character Joining

This approach means that the upper case, numerals and special character sets can be added into the recogniser by incorporating the new character shapes, most probably by the construction of a complementary character shape database to that for the lower case alphabet.

Obviously, such a combination of letter sets would require an extra level of post-processing beyond that currently implemented. The increased amount of character confusions that might ensue might involve characters from all four letter sets. For example:-

'l', '1', 'I', 'i', '!', '(', ')'

One technique which could provide valuable information would be to analyse the size and vertical position of characters along a particular written line. However, it is particularly important that the zonal boundaries of each word or character grouping can be accurately distinguished. Zonal position of characters would be of great benefit in ambiguity reduction.

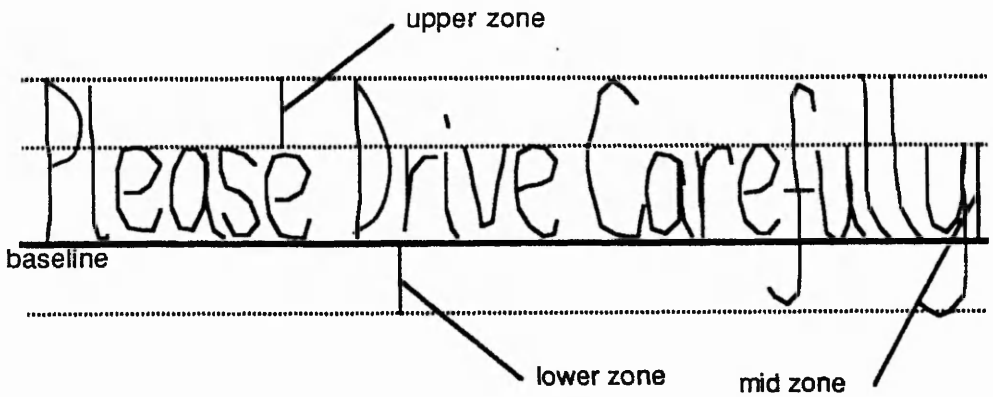


Figure 10.6 - Zonal Boundary Detection

A technique for zone detection must be able to cater for problems such as:-

- slanted lines
- inconsistent character sizes along a line
- wavy baslines ('hill and dale' writing)

10.3. Market Opportunities

Although the cursive script recogniser shows promise as a tool that might be incorporated into some future market product, it must be said that, to date, script recognition related products have not found widespread acceptance. In the UK, Ferranti and Quest Microsystems have marketed products, but with little success. In the USA, Pencept [91] have achieved some degree of market success. Linus Technologies [92] are pointing the way towards the vehicle for script recognition acceptance. We are finally moving towards products which have advantages over using a keyboard. One area which would be of particular importance would be the integration of a natural editing function with the recognition software. Many computer and word-processor users find it particularly annoying to have to learn a new editor each time they use a new machine. This is regarded as a particularly important feature that the electronic paper enviroment could provide.

Optical character recognition (OCR) devices have had much greater market penetration due to the amount of unconnected script documents in the office produced in the form of typed pages. In many instances these exist only in hard-copy form. Scanning and recognition allows the most economical form of entry to a computer, for updating and correcting, or simply for easier archival. OCR has begun to diversify to the recognition of hand-

printed documents also. This might have some uses, however, documents produced by hand are not usually written entirely in block printing. It is usually some combination of upper and lower case script, and, what is more, connected letters.

The ideal environment for dynamic script recognition (DCR) would be a situation as close as possible to the natural environment of working with a pen and paper as being used by Linus Technologies in their new product. Technology has now advanced to the stage where this scenario can be re-created electronically by putting a transparent tablet over the top of a flat screen. The cost and resolution of flat screen technology has made sufficient advances in recent years so as to make such a device economically viable. A flat screen, either LCD or gas plasma, having a resolution of 640 x 400 pixels and measuring 12" along its' diagonal length currently costs around 700 US dollars (not including display driver software). A transparent tablet which could be placed over the top of the display would cost a further 1300-1900 US dollars with its power supply and controller. Therefore, a prototype 'electronic paper' system could be purchased for 2000-2600 US dollars. Tappert et al [86] have already built such a device and investigated its' suitability as a medium for a handwriting recognition system. Figure 10.7 shows an example of script recognition being operated in an electronic paper environment.

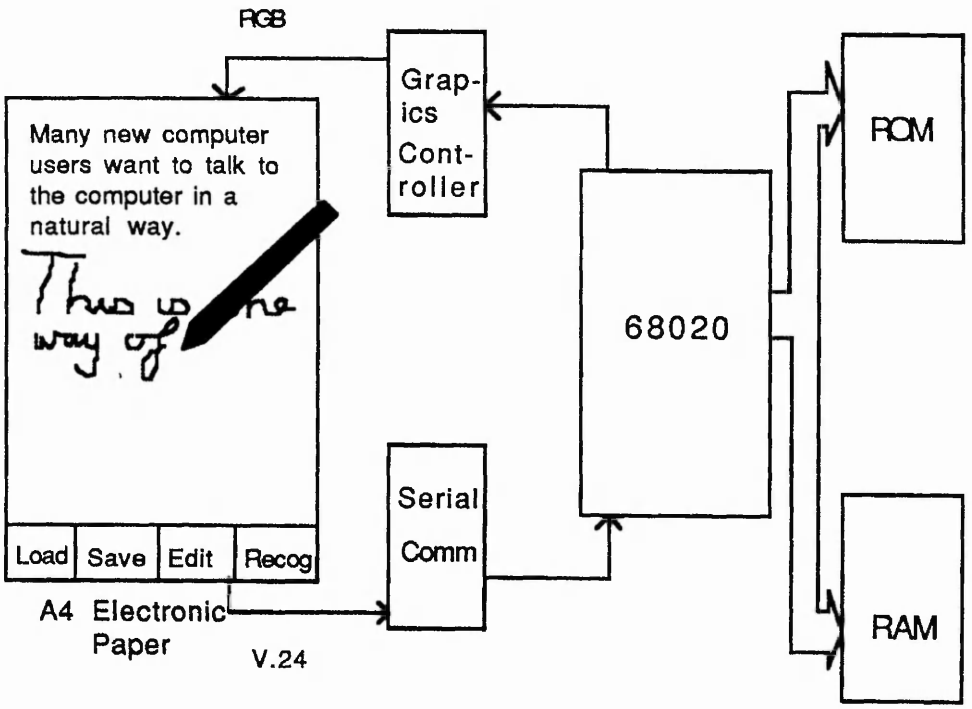


Figure 10.7 - An Electronic Paper Hardware Environment

Problems that must be overcome are largely,

(i) the real-time display of the pen movements ('electronic ink').

(ii) ease of use by the writer. They found that this was in direct relation to the problem of parallax.

The initial electronic paper set-up had a distance measure of 0.45" from the surface of the tablet to the display plane. A thinner transparent tablet having a gap of only 0.17" separation was also tried, and it was found that people could write much faster, due to the reduced parallax problem. Most flat-screen vendors also sell associated graphics drivers, these are a pre-requisite for displaying the raw textual information, or 'electronic ink'.

The advent of the 'electronic paper' system will herald the development of a more natural means of file/document manipulation. This can be performed by the user as they would if they were editing a document with a pen on paper. By defining a set of natural editing symbols, it is possible to make the electronic paper file editing a particularly simple task, even for the untrained user.

The combination of the realisation of electronic paper with a natural handwriting and editing environment has enormous scope for market development. It gives the machine the means of adapting to the most fluent method of communication for the human user.

The design and integration of a natural sketch recognition and editing task into such a product would result in a totally self-contained mixed mode (text & graphics) document creation system, having a natural means of human interface. Such a concept could form an integral role in the functionality of the office workstation of the future. However, a variety of lesser, but more immediate market opportunities can be defined which are more immediately attainable,

(i) Portable memo-pad, an electronic paper device that can be used in the field to record dynamic manuscript and sketch graphics, that can be transcribed by recognition software back at a base unit.

(ii) Signature recognition/verification devices. There is enormous potential for such products. Banking and point-of-sale terminals are prime areas for such an application.

(iii) Handwriting recognition as an identification mechanism in a secure environment.

(iv) Remote recognition via a telephone link. An example might be remote form-filling where the form is sent down the telephone line to be displayed on the users screen, say, from an insurance company. The client can the fill in the form on an electronic paper device. Their raw data is transmitted back to the insurance company offices where it is recognised and the recognised information transmitted back to the client for viewing and/or alteration.

Appendix A: BIBLIOGRAPHY

[1]
Acoustic Radar Graphic Input Device.
P. de Bruyne, FIT, Zurich.
AC M0-89791-021-4/80/0700-0025 1980.

[2]
On-line Computer Recognition of Handprinted Characters.
R.M. Brown.
IEEE Trans. Electronic Computers vol. EC-13 pp750-752 Dec.1964.

[3]
Recognition of Handwritten Characters By Topological Feature Extraction.
J.T. Tou, R.C. Gonzalez.
IEEE Transactions on Computers, July 1972 pp.776-784.

[4]
Learning In Syntactic Recognition of Symbols Drawn on a Graphic Tablet.
M.Berthod, J.P. Maroy
Computer Graphics and Image Processing, 1979, Academic Press.

[5]
Morphological Features and Sequential Information in Real-time Hand-printing Recognition.
M.Berthod, J.P. Maroy
Proc. Second Int. Joint Conf. on Pattern Recognition, Aug 1974.

[6]
A Description of Handwriting Dynamics
E.H. Dooijes
Simulation of Systems 1979, North-Holland Pub. Co. 1980.

[7]
On Line Cursive Script Recognition: A Structural Approach With Learning
M. Berthod, S. Ahyan
Proc. 5th Int. Conf. On Pattern Recognition 1980,Vol. 2 pp.723-5,IEEE

[8]
On-line Handwritten Character Recognizer
K. Odaka, T. Wakahara, S. Hashimoto
Trans. Inst. Electronic and Communication Eng. Japan,
section E vol. E65 no.8, Aug 1982

[9]
On The Automatic Reading of Cursive Script
Y.S. Eisa

Colloquium on 'Coding of Documentary Information' 1-3,
March 1982 London, IEE.

[10]
Cursive Script Recognition System By Elastic Matching
C.C. Tappert
IBM Journal of Researchand Development (USA), vol. 26 no.6 Nov 1982.

[11]
Word-level Recognition of Cursive Script
R.F.H. Farag
IEEE Transactions on Computers, vol. C-28 no.2 Feb 1979

[12]
A Numeric Script Recognition Processor For Postal ZIP Code Application
L.R. Focht, A. Burger
Proc. IEEE Int. Conf. on Cybernetics and Society 1976

[13]
Elastic Matching In Automatic Pattern Recognition
A.J. Szanser
Proc. Conf. on Machine Perception of Patterns and Pictures,
Teddington Middlesex Apr 1972.

[14]
The Use of Context In Pattern Recognition
G. T. Toussaint
Pattern Recognition(GB) vol. 10 no.3 1978, IEEE Conf. Pattern Rec. and Image Processing, 1977.

[15]
On-line Recognition of Hand-Printed Korean Characters
Y.H. Huh, H.L. Beus
Pattern Recognition Vol 15 No6 (pp445-453) 1982.

[16]
An On-line Character Recogniser
R.M. Simmons
Interface Age,pp 110-114, 1971.

[17]
Experiments in Text Recognition with the Modified Viterbi Algorithm.
G.T. Toussaint
IEEE Trans. Pattern Analysis Machine Intelligence (USA) Vol. PAMI-2 pp 184-93
April 1979.

[18]
Machine Recognition of Roman Cursive Scripts
K. Badie, M. Shimura
Proc. 6th Int. Conf. Pattern Recognition pp 28-30 vol.1 1982 IEEE

[19]
Cursive Script Recognition
M. K. Brown, S. Ganapathy
Proc. Int. Conf.on Cybernetics and Society pp47-51 1980 IEEE

[20]
Hand-Written Numeral Recognition - 'The Organisation Degree Measurment'
S. Impedovo, N. Abbattista
Proc. 6th Int. Conf. Pattern Recognition pp40-3 vol.1 1982 IEEE

[21]
Expansion of Pen Movement Stroke Extraction Method to Hiragana Character Recognition
T. Yamamoto, W.S. Hsu, S. Ozawa
Trans. Inst. Electronic and Comms. Eng.of Japan, Sect E. Vol E65 No. 11 Nov 1982.

[22]
On-line Recognition of Hand-written Characters Utilising Stroke Vector Sequences
K. Ikeda, T. Yamamura, Y. Mitamura, S. Fujiwara, Y. Tominaga, T. Kiyono
Pattern Recognition (GB) vol.13 no.3 pp 191-2, 1981.

[23]
Online Handwritten Character Recognition Fora Personal Computer System
K. Yoshidaand H. Sakoe
IEEE Trans. Consumer Electronics (USA) vol. CE-28 no.3 1982

[24]
An On-line Data Entry System For Hand-printed Characters
H.D. Crane, R.E. Savoie
Computer (USA) vol. 10 no.3 pp43-50 1977.

[25]
Algorithm For the Recognition of Handwritten Text
S.A. Guberman, V.V. Rozentsveig
Automation and Remote Control (USA) vol.37 no.5 pt.2 1976.

[26]
Sequence Detection Using All-magnetic Circuits
H. D. Crane
IEEE Trans. Electronic Computers pp 155-160 vol. EC-9 no. 2, 1960.

[27]
Devices for Reading Handwritten Characters
T.L. Dimond
Proceedings of the Eastern Computer Conference, pp 232-237, 1961.

[28]
Machine Recognition of Handprinted Characters
D.L. Caskey
Sandia Labs, Albuquerque, New Mexico

[29]
Real Time Recognition of Hand-drawn Characters
W. Teitelman
Proceedings- Fall Joint Computer Conference, 1964, pp559-575

[30]
An On-line Character Recogniser with Learning Capabilities
G. Gaillat
Central Research Labs, Thomson-CSF

[31]
A Feature Extraction Method for the Recognition of Handprinted Characters
D.J. Hunt
Research and Advanced Developement Centre, ICL

[32]
Algorithm for a Low Cost Hand Print Reader
A.W. Holt
Computer Design, Feb 1974, pp85-89

[33]
Use of Handwriting in Construction of Models
M. Hosaka, F. Kimura
Scientific Information Systems in Japan, 1981, pp83-90

[34]
Pen Direction Sequences in Character Recognition
V.M. Powers
Pattern Recognition, 1973, Vol 5, pp 291-302

[35]
Designing a Handwriting Reader
D.J. Burr
IEEE Trans PAMI, Vol PAM-5, No.5, Sept 1983, pp554-559

[36]
Some Simple Contextual Decoding Algorithms Applied to Recognition of

Hand-printed Text
G.T. Toussaint, R.W. Donaldson
Proc. Annu. Canadian Computing Conf. 1972, pp422 101-116

[37]
Analysis and Synthesis of Handwriting
J. Vredenbregt, W.G. Koster
Philips Technical Review, Rev32, No3/4, 1971, pp73-78

[38]
A Tree Classification Algorithm For Handwritten Character Recognition
M. Shridhar, A. Badreldin
Proc.7th Int Conf. on Pattern Recognition, pp.615-8 1984 IEEE

[39]
Segmentation and Recognition of Symbols For Handwritten Piping and Instrument Diagram
M. Futura, N. Kase, S. Emori
Proc.7th Int Conf. on Pattern Recognition, pp.626-9 1984 IEEE

[40]
Experiments in Contextual Recognition of Cursive Script
R.W. Ehrich, K.J. Koehler
IEEE Transactions on Computers, Vol C-24, No 2, Feb 1975, pp 182-194

[41]
Automatic Recognition of Handprinted Characters- The State of The Art
C.Y. Suen, M. Berthod, S. Mori
Proc. IEEE vol.68 no.4 April 1980

[42]
Advances in Recognition of Handprinted Characters
C.Y. Suen, M. Berthod, S. Mori
Proc. 4th Int. Joint Conf. Pattern Recognition, pp30-44 Nov 1978, Kyoto Univ.

[43]
A Normalising Transformfor Cursive Script Recognition
D. J. Burr
Bell Laboratories, 1982 IEEE Transactions pp 1027-1030

[44]
Handwriting Recognition Accuracy Versus Tablet Resolution and Sampling Rate
J. Kim, C.C. Tappert
7th International Conference on Pattern Recognition, 1984, Vol II pp917-918

[45]
Real-Time On-Line Symbol Recognition Using a DTW Processor

P. Lu, R. Brodersen
7th International Conference on Pattern Recognition, 1984 Vol II pp 1281-1283

[46]
On-line Recognition of Shortforms in Pitmans Handwritten Short-hand
C.G. Leedham, A.C. Downton
7th International Conference on Pattern Recognition, 1984 Vol II, pp 1058-1060

[47]
A Microcomputer System to Recognise Handwritten Numerals Using Syntactics-Statistics
G.Y.Tang, P.S Tzeng, C.C. Hsu
7th International Conference on Pattern Recognition, 1984 VolI Ipp 1061-1064

[48]
Knowledge-based Cursive Script Interpretation
R. Bozinovic, S. Srihari
7th International Conference on Pattern Recognition, 1984 Vol I Ipp774-776

[49]
Signature Verification Based on Nonlinear Time Alignment: A Feasibility Study
M. Yasuhara, M. Oka
IEEE Transactions on Systems, Man,and Cybernetics, March 1977 pp 212-216

[50]
Reading Handwritten Words Using Hierarchical Relaxation
K. Hayes
Computer Graphicsand Image Processing No. 14, pp344-364 1980

[51]
Recognition of Handprinted Characters for Automated Cartography:
A progress report
M. Lybanon, R. Brown, L. Gronmeyer
Image Understanding Systems II, 1979, pp 165-173

[52]
Machine Recognition of Handprinted Words: A Progress Report
K. Sayre
Pattern Recognition, 1973 Vol5, pp 213-228

[53]
On-line Cursive Script Recognition
C. Higgins, R. Whitrow
Interactive Conf. 1984, Elsevier Science Publishers

[54]
Automatic Recognition of Print and Script

L. Harmon
Proceedings of the IEEE, Vol 60, No. 10, Oct 1972, pp 1165-1176

[55]
A New Character Recognition Scheme with Lower Ambiguity and Higher Recognisability
P.S. Wang
Proceedings of the IEEE, 1982, pp37-39

[56]
Preprocessing techniques for Cursive Script Recognition
M.K. Brown, S. Ganapathy
Computer Graphicsand Image Processing, 1983, pp447-458

[57]
Ambiguity Reduction in Writing with Ambiguous Segmenting and Uncertain Interpretation
S. Peleg
Computer Graphics and Image Processing 10, pp 235-245, 1979.

[58]
Experiments in On-line Script Recognition
E. Mandler, R. Oed, W. Doster
AEG-Telefunken Research Institute

[59]
Digitiser Technology: Performance Characteristics and the Effects on the User Interface
M. Phillips, J. Ward
IEEE Computer Graphics & Applications Feb 1987.

[60]
A Case for Digitiser Tablets
T.T. Kutlinski
Computer Graphics World, May 1985.

[61]
Components of Hand-Print Style Variablility.
T.T. Kutlinski
IEEE 7th International Conf. on Pattern Recognition.

[62]
A Comparative Study of Some Recognition algorithms in Character Recognition.
C.C. Kwan, L.Pang, C.Y. Suen
Proc International Conferenceon Cybernetics.

[63]

Approach to Smart Document Reader System
I. Masuda et al.
1985 IEEE CH2145-1/85/0000/0550.

[64]
Word Shape Analysis in a Knowledge-Based System for Reading Text
J.J. Hull
2nd International Conf On Artificial Intelligence Applications, Miami, Florida 1985.

[65]
Postional Representation of English Words
S. Mukherjee, M. Sloan
2nd International Conf On Artificial Intelligence Applications, Miami, Florida 1985.

[66]
Dynamic Programming in the Recognition of Connected Handwritten Script
K.H. Wong, F. Fallside
2nd International Conf On Artificial Intelligence Applications Miami, Florida 1985.

[67]
The Write Stuff
T. K. Worthington (U. S.patent 4513437)
IEEE Spectrum Oct. 1985

[68]
Method for Selecting Constrained Hand-Printed Character Shapes for Machine Recognition
R. Shinghal, C.Y. Suen
IEEE Trans. PAMI, Vol. PAMI-4 No.1. Jan. 1982.

[69]
Handwriter Identification From One-bit Quantized Pressure Patterns
K.P. Zimmermann, M.J. Varady
Pattern Recognition vol. 18 no 1. pp63-72

[70]
A High Accuracy Syntactic Recognition Algorithm for Handwritten Numerals
M. Shridhar, A. Badrelin
IEEE Transactions on Systems, Man, and Cybernetics, Feb 1985.

[71]
Character Recognition
Alan Howard
Systems International, Nov 1986.

[72]
Recognition of Handprinted Hebrew Characters Using Features in the Hough

Transform Space
M. Kushnir, K. Abe, K. Matsumoto
Pattern Recognition Vol 18, No. 2 1985

[73]
Computer Recognition of Handwritten Numerals by Polygon Approximation
T. Pavlidis, F. Ali
IEEE Transactions on Systems, Man and Cybernetics, Nov 1975.

[74]
Compact Large-area Graphic Digitizer for Personal Computers
P.de Bruyne, FIT Zurich.
IEEE Comp. Graph.and Appl. Dec 1986 pp49-53.

[75]
Recognition of Handprinted Characters by an Outermost Point Method
K. Yamamoto, S. Mori
Pattern Recognition Vol. 1 2pp 229-236 1980.

[76]
A Note on Human Recognition of Hand-Printed Characters
U. Neisser, P. Weene
Information and Control3, pp 191-196 1960.

[77]
On the Recognition of Printed Characters of Any Font and Size
S. Kahan, T. Pavlidis, H. Baird
IEEE Transactions, PAMI, Vol9 No.2, pp 274-287, March 1987

[78]
Handwritingand Pattern Recognition
M. Eden
IRE Transactions on Information Theory pp 160-166 Feb 1972.

[79]
Recognition of Isolated and Simply Connected Handwritten Numerals
M. Shridhar, A. Badreldin
Pattern Recognition Vol 19, No.1 pp 1-12 1986.

[80]
Psychophysical Techniques for Investigating the Distinctive Features of letters
R. Shillman, T. Kutlinski, B. Besser
Int J. Man-Machine Studies, 8, pp 195-205 1976.

[81]
On The Encoding of Arbitrary Geometric Configurations
H. Freeman

IRE Transcations on Electronic Computers, pp 260-268, June 1961.

[82]
A Sketchphone System
IEEE Transactions in Communications, Com 29, No. 12, 1981.

[83]
Handwriting Recognition on a Transparent Tablet over a Flat Display
C.C. Tappert et al
SID International Symposium, 6-8 May 1986, pp308-312

[84]
Script and Graphics Recognition- A State of The Art Study
P.T. Wright, N.G. Baker, P.D. Moulds
ESPRIT PROJECT 295, 18 Dec 1984

[85]
Elographics Touch Screen
Elographics Inc,
105 Randolph Rd,
Oak Ridge, Tennessee 37850, USA.

[86]
Handwriting Recognition on a Transparent Tablet Over Flat Display
C.C. Tappert, A.S. Fox, J. Kim, S.E. Levy, L.L. Zimmerman
SID86 Digest, pp308-312

[87]
On-line Handwriting Recognition- A Survey
C.C. Tappert, C. Y. Suen, T. Wakahara
1988

[88]
Opto-electronic Paper at the CCTA
P. Christmas
Information Media& Technology, Vol 20 No.3, 1988

[89]
Photron FIOS-6440
Photron Limited,
Jingumae6-12-15, Shibuya-Ku,
Tokyo 150, Japan

[90]
WH-515 LCD Integrated Tablet,
Wacom Co. Ltd, Tokyo, Japan

[91]
Penpad,
Pencept Inc.,
39 Green Street,
Waltham, Mass., USA.

[92]
Linus Write-Top
Linus Technologies,
1889 Preston White Drive,
Reston, VA 22091, USA.

[93]
Experiments on Computer Reciognition of Handprinted Words
P. Mermelstein, M. Eden
Information Control, pp255-270, 1964.

[94]
Methods and Apparatus for Reading Cursive Script
L.D. Harmon
US Patent 3111646, Nov 1963.

[95]
Post-Processing Techniques for Script Recognition
L.J. Evett, C.J. Wells, L.J. Dixon, F.G. Keenan, R.J. Whitrow
ESPRIT PROJECT 295 Research Report, August 1988.

Appendix B: SCRIPT TEST SHEETS

Each writer giving a sample of writing style was asked to produce the following as examples of their :-

(i) lower case unconnected

(ii) upper case, numeral & special character

(iii) cursive (or natural)
handwriting style.

ESPRIT PROJECT 295 - TEST SHEET 1

Please write the following sentences in lower case unconnected lettering:-

pack my bags with five dozen extra liquor jugs

both wizened men quickly judged four sharp vixens

ESPRIT PROJECT 295 - TEST SHEET 2

Please copy the following:-

PACK MY BAGS WITH FIVE DOZEN EXTRA LIQUOR JUGS

---

0 1 2 3 4 5 6 7 8 9 & % $ * ( ) { } [ ] @ ! ? + =

ESPRIT PROJECT 295 - TEST SHEET 3

Please write the following sentences in your natural handwriting style:-

pack my bags with five dozen extra liquor jugs

---

both wizened men quickly judged four sharp vixens

Appendix C: New Writer Hard-Copies & Results Breakdown

The following results are those obtained from the 10 data sets collected from new, untrained writers whose script styles have not been incorporated in the database. There are also hard-copies of the scripts that these writers produced.

1. Filename: amk.ref Dated: 18-3-88
   Tablet: Numonics

   Reference: the quick brown fox jumps over the lazy dog
   able grown men have quickly found their sexy jokes zapped

   Recognition: the quick brown fox jumps over the lazy dog
   able grown men hqve quickly fousd their sexy johis zapped

ERRORS:

Ref	Recognised as
a	q:77 a:73 d:69 u:67 g:47
n	s:44 p:38 z:33 o:31 j:27
k	h:86 n:57 r:49 p:47 s:26
e	i:99

	Correct	Non-rec	Rec-err	Seg-Err
Counts	81	0	4	0
Percentage	95.29	0.00	4.71	0.00

2. Filename: dc.ref Dated: 18-3-88
   Tablet: Numonics

   Reference: the quick brown fox jumps over the lazy dog
   able grown men have quickly found their sexy jokes zapped

   Recognition: the quoek brown fox jumps ouer tne lazy aog
   able grown men naue quuckly found tlieir sexy jokes zupped

ERRORS:

Ref	Recognised as
i	o:89 v:76 u:68 b:39 d:23
c	e:84 d:46 i:44 o:36 c:35

v	u:87 v:79 r:42 t:37 b:34
h	n:80 h:74 r:38 a:25 k:24
d	a:95 d:85 q:59 n:48 u:46
h	n:80 h:80 u:57 k:54 y:28
v	u:71 v:66 b:16 f:13 o: 9
i	u:99 v:84 i:78 r:51 b:33
h	l:99
	i:80 j:63 i:50 n:40 -:19
a	u:92 a:76 d:74 o:63 q:62

	Correct	Non-rec	Rec-err	Seg-Err
Counts	75	0	9	1
Percentage	88.24	0.00	10.59	1.18

3. Filename: esp.ref Dated: 18-3-88
Tablet: Numonics

Reference: the quick brown fox jumps over the lazy dog
able grown men have quickly found their sexy jokes zapped

Recognition: the quick brown foc jumps over the lazy dog
gble grown men have quickly found tueir sexy jekes zapped

ERRORS:

Ref	Recognised as
x	c:62 r:30 j:11
a	g:91 q:85 a:70 d:68 n:51
h	u:90 a:67 h:66 n:63 w:43
o	e:78 o:72 z:50 c:47 f:40

	Correct	Non-rec	Rec-err	Seg-Err
Counts	81	0	4	0
Percentage	95.29	0.00	4.71	0.00

4. Filename: ghm.ref Dated: 22-3-88
Tablet: Numonics

Reference: the quick brown fox jumps over the lazy dog
able men have quickly found their sexy jokes zapped

Recognition: the qvick brown tox jumbs orer the zazy dog
able men havc qviculy found rheir sexy jores zpjed

ERRORS:

Ref	Recognised as
u	v:77 u:71 r:41 i:31 b:17
f	t:98
p	b:61 h:60 n:49 g:37 p:37
v	r:63 v:57 i:17
l	z:55 c:43 l:37 a:30 o:30
y	g:92 y:71 b:23 z:18 s:12
e	c:85 e:64 o:48 i:24 1:12
u	v:84 u:46 c:46 r:46 i:30
k	u:69 v:64 l:43 r:32 b:31
t	r:64 t:35 d:15 e: 7 a: 7
k	r:91 v:84 l:27 i:21 c:19
a	p:77
e	j:93 z:48 g:25 o:23 y:23
d	e:97 c:37 o:33 i:27 f:26

	Correct	Non-rec	Rec-err	Seg-Err
Counts	65	0	14	0
Percentage	82.28	0.00	17.72	0.00

5. Filename: hwt.ref Dated: 18-3-88
Tablet: Numonics

Reference: the quick brown fox jumps over the lazy dog
able grown men have quickly found their sexy jokes zapped

Recognition: tbe quick brown fox jumps ooer the lazg dog
aple growa meh haue guicley found tha sexy jokes zabped

ERRORS:

Ref	Recognised as
h	b:81 h:75 n:52 p:30 s:24
v	o:83 v:43 r:33 u:31 p:25
y	g:81 y:71 s:36 p:25 b:19
b	p:99 b:96 n:47 s:46 g:37
n	a:66 m:65 h:63 q:50 u:43
n	h:84 m:73 u:73 a:67 q:63
v	u:74 v:59 r:41 b:26 t:25
q	g:82 q:81 d:52 w:42 a:41
k	l:70 f:39 e:37 b:36 z:33
	e:94 c:74 z:65 q:21 k:15

e	a:90 q:85 u:83 w:74 h:68
i	s:72 z:41 a:25 h:22 f:18
r	e:88 d:36 i:27 q:24 c:23
p	b:82 p:51 n:38 h:37 g:33

	Correct	Non-rec	Rec-err	Seg-Err
Counts	71	0	13	1
Percentage	83.53	0.00	15.29	1.18

6. Filename: mjs.ref Date: 18-3-88
Tablet: Numonics

Reference: the quick brown fox jumps over the lazy dog
able grown men have quickly found their sexy jokes zapped

Recognition: the duick brown fox jumps over the ldzg dog
dble yrown meh hdve guilkly found lhilr sexy jokes zapped

ERRORS:

Ref	Recognised as
q	d:92 a:87 g:46 q:44 u:42
a	d:86 a:77 u:55 b:41 g:35
y	g:90 y:74 b:61 p:31 q:24
a	d:88 a:82 q:56 u:55 g:43
g	y:87 g:73 b:39 s:37 p:36
n	h:61 n:33 p:28 b:27 r:23
a	d:86 a:79 u:65 v:27 o:27
q	g:86 q:85 d:61 a:59 n:59
c	l:74 c:68 u:57 v:25 i:23
t	l:99
e	i:71

	Correct	Non-rec	Rec-err	Seg-Err
Counts	73	0	11	1
Percentage	85.88	0.00	12.94	1.18

7. Filename: mpc.ref Date: 22-3-88
Tablet: Numonics

Reference: the quick brown fox jumps over the lazy dog
able grown men have quickly found their sexy jokes zapped

Recognition: the quick brown rox jumps over the iazy dog

able growr mep have quiekly fouud thir sexg jokes zappcd

ERRORS:

Ref	Recognised as
f	r:97 n:77 p:68 h:37 t:19
l	i:99
n	r:83 k:73 n:28 i:28 m:26
n	p:78 n:43 b:28 g:27 y:24
c	e:86 c:73 a:50 i:40 o:29
n	u:85 q:38 n:34 h:33 a:31
e	i:99
y	g:94 y:63 b:35 u:23 n:22
e	c:91 e:79 f:39 d:36 i:29

	Correct	Non-rec	Rec-err	Seg-Err
Counts	76	0	9	0
Percentage	89.41	0.00	10.59	0.00

8. Filename: rje.ref Date: 22-3-88
Tablet: Numonics

Reference: the quick brown fox jumps over the lazy dog
able grown men have quickly found their sexy jokes zapped

Recognition: the quick brown tox jumps orer thi iazg dog
able groon men have guictly foqnd their sexy jokes zapbed

ERRORS:

Ref	Recognied as
f	t:82
v	r:78 v:43 i:23 q:22 u:20
e	i:83
l	i:72 i:71 c:15 -:12 i: 0
y	g:94 y:70 s:45 z:19 b:17
w	o:79 u:35 v:32 b:29 i:25
q	g:83 q:76 a:69 w:31 y:29
k	t:79
u	q:68 u:48 d:43 g:38 a:38
p	b:83 p:72 s:30 h:18 g:17

	Correct	Non-rec	Rec-err	Seg-Err

Counts	75	0	10	0
Percentage	88.24	0.00	11.76	0.00

9. Filename: sds.ref Date: 18-3-88
Tablet: Numonics

Reference: the quick brown fox jumps over the lazy dog
able grown men have quickly found their sexy jokes zapped

Recognition: the quuek brown fox idnts ovar the lazg dog
dble grown men hqve qvicnly found tbeir sexy jokes sapped

ERRORS:

Ref	Recognised as
i	u:89 i:88 c:67 v:62 f:26
c	e:79 o:66 c:55 a:40 l:39
j	i:97
u	d:97 u:94 a:58 v:57 n:41
m	n:74 m:71 h:41 k:35 d:28
p	t:79 p:72 r:67 n:53 h:35
e	a:73 e:71 o:69 f:67 d:66
y	g:52 b:32 y:30 f:27 j:24
a	d:88 a:80 u:64 q:61 g:60
a	q:70 a:69 n:64 d:55 u:52
u	v:76 u:71 r:31 -:25 b:22
k	n:74 k:60 h:49 b:38 p:31
h	b:65 w:55 q:46 h:34 d:25
z	s:82 z:63 y:34 -:30 q:22

	Correct	Non-rec	Rec-err	Seg-Err
Counts	71	0	14	0
Percentage	83.53	0.00	16.47	0.00

10. Filename: sed.ref Date: 22-3-88
Tablet: Numonics

Reference: the quick brown fox jumps over the lazy dog
able grown men have quickly found their sexy jokes zapped

Recognition: thc quialh brown fox jumps ooer the ldzy dog
abie grown men huue duickiy found their sexy jokes zapped

ERRORS:

Ref	Recognised as
e	c:86 e:85 o:47 f:30 a:29
c	a:96 c:92 e:52 i:51 f:47
k	l:99
	h:85 c:74 t:53 b:36 a:30
v	o:54 u:33 v:30 d:27 f:27
a	d:95 a:70 q:62 g:59 u:58
l	i:99
a	u:82 a:76 q:72 d:72 b:51
v	u:69 v:46 o:44 d:25 f:20
q	d:85 a:67 g:39 u:38 o:23
l	i:99

	Correct	Non-rec	Rec-err	Seg-Err
Counts	75	0	9	1
Percentage	88.24	0.00	10.59	1.18

OVERALLRESULTS:

	Correct	Non-rec	Rec-err	Seg-Err
Counts	743	0	97	3
Percentage	88.14	0.00	11.51	0.36

ESPRIT PROJECT 295

Please write the following two test sentences in lower case unconnected letters.

DATE: 22/3/88 WRITER: A.M. King M/F: M

L/R handed: R TABLET: NUMONICS AGE: 23

COMMENTS: —

the quick brown fox jumps over the lazy dog

the quick brown fox jumps over the lazy dog

able grown men have quickly found their sexy jokes zapped

able grown men have quickly found their sexy jokes zapped

ESPRIT PROJECT 295

Please write the following two test sentences in lower case unconnected letters.

DATE: 18/3/88 WRITER: D. Charnley M/F: F

L/R handed: R TABLET: NUMONICS AGE:

COMMENTS: —

the quick brown fox jumps over the lazy dog

the quick brown fox jumps

over the lazy dog

able grown men have quickly found their sexy jokes zapped

able grown men have quickly found

their sexy jokes zapped

ESPRIT PROJECT 295

Please write the following two test sentences in lower case unconnected letters.

DATE: 18/3/88 WRITER: E.S. Powell M/F: F

L/R handed: R TABLET: Numonics AGE: 21

COMMENTS:

the quick brown fox jumps over the lazy dog

the quick brown fox jumps over

the lazy dog

able grown men have quickly found their sexy jokes zapped

able grown men have quickly

found their sexy jokes zapped

ESPRIT PROJECT 295

Please write the following two test sentences in lower case unconnected letters.

DATE: 22/3/88 WRITER: G.H.Maslin M/F: M

L/R handed: R TABLET: NUMONIC AGE: 40

COMMENTS: —

the quick brown fox jumps over the lazy dog

the quick brown fox jumps

over the lazy dog

able grown men have quickly found their sexy jokes zapped

able men have quickly found their

sexy jokes zapped

ESPRIT PROJECT 295

Please write the following two test sentences in lower case unconnected letters.

DATE: 18/3/88 WRITER: H.W.Thomas M/F: M

L/R handed: R TABLET: NUMONICS AGE: 22

COMMENTS:

the quick brown fox jumps over the lazy dog

the quick brown fox jumps over the lazy dog.

able grown men have quickly found their sexy jokes zapped

able grown men have quickly found their sexy jokes zapped.

ESPRIT PROJECT 295

Please write the following two test sentences in lower case unconnected letters.

DATE: 18/3/88 WRITER: M. Stephens M/F: M

L/R handed: L TABLET: Numonics AGE: 28

COMMENTS: —

the quick brown fox jumps over the lazy dog

the quick brown fox jumps over the lazy dog

able grown men have quickly found their sexy jokes zapped

able grown men have quickly found their sexy jokes zapped

ESPRIT PROJECT 295

Please write the following two test sentences in lower case unconnected letters.

DATE: 22/3/88 WRITER: M.P. Collar M/F: M

L/R handed: R TABLET: NUMONICS AGE: 25

COMMENTS: —

the quick brown fox jumps over the lazy dog

the quick brown fox jumps over the lazy dog

able grown men have quickly found their sexy jokes zapped

able grown men have quickly found their sexy jokes

zapped

ESPRIT PROJECT 295

Please write the following two test sentences in lower case unconnected letters.

DATE: 22/3/88 WRITER: R.J.Evans M/F: M

L/R handed: R TABLET: NUMONICS AGE: 36

COMMENTS:

the quick brown fox jumps over the lazy dog

~~t~~ the quick brown fox jumps over

the lazy dog

able grown men have quickly found their sexy jokes zapped

~~able grown m~~

able grown men have quickly found their

sexy jokes zapped

ESPRIT PROJECT 295

Please write the following two test sentences in lower case unconnected letters.

DATE: 16/3/88 WRITER: S.D Saunders M/F: M

L/R handed: L TABLET: NiMONICS AGE: 27

COMMENTS: —

the quick brown fox jumps over the lazy dog

The quick brown fox jumps over the lazy dog

able grown men have quickly found their sexy jokes zapped

able grown men have quickly found their sexy jokes zapped

ESPRIT PROJECT 295

Please write the following two test sentences in lower case unconnected letters.

DATE: 22/3/88 WRITER: S.E. Dickie M/F: F

L/R handed: L TABLET: NUMONICS AGE:

COMMENTS: —

the quick brown fox jumps over the lazy dog

The quick brown fox jumps over the lazy dog

able grown men have quickly found their sexy jokes zapped

able grown men have quickly found their sexy jokes zapped