ON-LINE RECOGNITION OF ENGLISH AND NUMERICAL CHARACTERS

BY

CHEUNG WAI-HUNG, WELLIS

A research thesis submitted in partial fulfilment

of the requirements for the degree of

Master of Science

in

The Department of Information Engineering

The Chinese University of Hong Kong

Hong Kong

June 1992

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Dr. Michael M. Y. Chang, for his invaluable guidance throughout the whole project. His stimulating suggestions have always made my work more fruitful. Being a part-time student, I am also grateful to my family for their patience with my studies in many late evenings. Their encouragement is the vital factor of my endurance during the period.

ABSTRACT

This report introduces an on-line cursive English character and Arabic numeral recognition system realized in an IBM compatible PC. Handwritings are entered via a graphical tablet. After handling the inputs by some standard pre-processing steps (stroke connection, rotation, scaling and de-skewing), the system breaks the strokes into small line or arc segments at all local maxima and minima and where strokes change direction abruptly. Delayed stroke segment is tagged to the segment in the primary stroke to facilitate subsequent recognition work. Learning is achieved by explicit user character segmentation and the prototypes are added into the dictionary by correlation technique. Each stroke segment is coded with its successive neighbour to form a bi-gram. By identifying the bi-gram patterns, the recognizer can extract the letter out by a sliding matching window. The candidate word is compiled by recursively fetching the letter from a double-track of candidate letters. Because of the feature used is the segment bi-gram codes and the recognition is on a template matching basis, the system is easily extendable to cater for various writing styles and different character shapes.

CONTENTS

1
INTRODUCTION

1.1 CLASSIFICATION OF CHARACTER RECOGNITION

Character recognition is a promising means to improve the user-friendliness of computers to human. [Govindan & Shivaprasad 90] and [El-Sheikh & El-Taweel 90] listed a good deal of applications of character recognition in computer operation. [Doster & Oed 84] also gave an example of how to integrate character recognition with some typical PC applications like word-processing. Depending on different modes of operation, character recognition can be classified as:-

(a) On-Line Character Recognition
The machine recognizes the writing while the user writes on some special devices (tablet or digitizer). Therefore it is also known as real-time recognition. However, because of recognition algorithm and hardware limitation, the recognition usually lags behind the writing to some extent.

(b) Off-Line Character Recognition
The recognition is performed after the writing is completed. The image is usually captured by an optical scanner.

There are many consequent differences due to the DIFFERENCE in capturing user input. The most important of all is the inputted data in on-line recognition is basically a temporal series of two-dimensional primitive patterns. Because of the availability of information like order of strokes, number of strokes, direction of strokes and even the speed, the recognition work is rendered much easier. On the other hand, the off-line recognizer requires time-consuming but imperfect pre-processing to trace the contours and to thin the traces. This results that with the same set of input data, on-line recognition can offer better performance than off-line recognition [Mandler *et al* 1985].

The other advantage of on-line recognition is that it is interactive - the user can correct the machine mistake immediately. This implies on-line recognizer can be tolerated with a higher recognition error rate. However, most on-line

recognizer can adapt to user's specific writing pattern or style and improves its recognition rate subsequently.

Naturally, on-line character recognition has its drawbacks. The most significant one is that user is required to use special writing instruments, which are not as comfortable and natural to use as pen and paper. Nevertheless, recently, some pressure sensitive LCD display is available. It can be regarded as *electronic ink* because this device can instantaneously display the trace of motion of the pen, a kind of immediate feedback to the writer as the pen and paper.

Another perspective to classify character recognition is by the style of writing. Broadly speaking, there are cursive writing and hand-print. However, there is not a clear distinction between these two extremes. Indeed a spectrum of writing styles can be observed. [Tappert 84] classified there are 5 stages:-

(a) boxed discrete characters
(b) spaced discrete characters
(c) run-on discrete written characters
(d) pure cursive script writing
(e) mixed cursive and discrete characters

Of course, the most difficult to recognize is the cursive script. The reason lies on the ambiguity of character boundary. This causes great difficulty in character segmentation. In non-cursive writing recognition, this segmentation can be performed on spatial or temporal delimitation. For instance, [Mandler 89] described a good character segmentation algorithm based on spatial relationship. The temporal information usually used by researchers is to assume a character is completed when the time difference between the end of a stroke and the beginning of the next exceeds a threshold. For example, Casio once marketed a calculator watch on which the user can write a character with his finger and a time out signifies the end of a character.

For cursive letters, the character segmentation usually adopts one of the two schools:-

(a) External segmentation

The isolation of characters is performed prior to the character recognition process. The segmentation points are usually guessed with some heuristics - for instance expected character width. However, this method can cause the segmentation error to affect the subsequent recognition result. The errors are usually corrected by string-correction algorithm later in the post-processing part.

(b) Internal segmentation

The segmentation requires recognition knowledge. This provides savings in computation and simplifies the job of recognition. Usually some loose segmentation is done at first (e.g. searching all local minima) to identify all potential character segmentation points. An example is in [Bozinvoic & Srihari 89].

It should be pointed out that some researchers simply abandoned the character segmentation stage and recognizes the word as a single entity [Frag 79]. Although this saves the character segmentation effort, a large word dictionary is required in the recognition process.

In fact there is much correlation between reading cursive script and understanding speech. Both need to process noisy inputs with indefinite delimitation and considerable variation in inputted data. However, much more research effort has been contributed to speech processing than cursive script. One possible reason is speech is the more natural way of human communication. The little research in recognition of cursive script has been restricted to lower case English only and there is still no commercial cursive character recognizer available on the market.

1.2 HISTORICAL DEVELOPMENT

Character recognition has long been studied to simplify the man-machine interface. From a chronological perspective, [Srihari & Bozinovic 87] divided the historical research into 3 phases of periods. The 1960's was typically represented by on-line approaches, among which the work of [Mermelstein & Eden 64] was regarded as the most notable. It made use of the velocity

measurement of pen strokes and composed letter syntactically from a standard set of strokes taken from a formal handwriting model. The second period, 1970's, started to investigate off-line solutions. There was also wider and better use of higher level knowledge (e.g. real/binary letter n-grams). In the third stage, 1980's, with the popularity of personal computer, greater user-friendliness when using these machines was expected. Furthermore the cost of digitizer is declining. As a result, on-line script recognition began to regain more research effort. During this period, many researchers utilized dynamic programming technique of elastic matching, a tool successfully applied to speech recognition problem, to the character recognition problem.

For a general review on character recognition development, [Govindan & Shivaprasad 90] can be referred. That paper emphasizes more on off-line character recognition. On the other hand [Tappert *et al* 90] gave a more comprehensive summary on the state of the art of on-line recognition.

1.3 RECOGNITION METHODOLOGY

Like the general pattern recognition problem, the most important issue is the data representation of the input data. Employing different features calls for different recognition methodologies. This leads to, of course, different system performance. It is noteworthy that most often the merit of a good data representation can be greater than a sophisticated recognition algorithm on bad data representation.

We can treat the problem as follows: The task of recognition algorithms is to identify the variation between different characters. This is based on the assumption that the variation between different characters should be greater than the variation between specimens of the same character. If the data representation can reveal this situation, then less burden rests on the recognition algorithm. [Wang 82] gave good definitions and explanation on the relevant concepts: *recognizability, learnability and ambiguity*. It was shown that the one with lower degree of ambiguity possesses an inherent advantage of recognition.

However, it must be pointed out that non-ambiguous features need not be good representation. If the classification is very fine and many templates of same character are resulted, this also causes inefficiency in subsequent recognition process. To conclude, the best feature is there is maximum difference between different characters, but minimum distance among different samples of same character.

Broadly speaking, the following lists some of the different features and different recognition methodologies adopted by researchers so far.

(a) Syntactic approach

Some of the commonly used features to be captured are:

- curvature of stroke
- direction of stroke
- length of stroke
- the number of loops, cusps, etc.

A tree representation of the captured image is created and then syntactic parsing analysis of the sequence is used to recognize the characters. This method makes use of the formal language-theoretic models. However, it was argued by [Govindan & Shivaprasad 90] whether this assumption is realistic.

> "Patterns are natural entities which cannot strictly obey the mathematical constraints set by the formal language theory. Imposing a strict rule on the pattern structure is not particularly applicable to character recognition, where the intra-class variations are infinite."

(b) Binary decision tree

The features are binary - the existence of ascender, descender, dot, cross, loop, cusp. [Frishkopf & Harmon 61] demonstrated an early example. [Tappert *et al* 90] considered that disadvantage of this method is that it usually does not give any alternative character choices, which are usually useful in subsequent recognition processing. Anyhow, this method can be a good choice to reduce the set of candidate characters to be analyzed by any other recognition algorithm.

(c) Feature Analysis

The classical pattern recognition techniques to separate the feature space into various decision recognition regions by hyper-surfaces are used in this case. The features are real-valued vectors. Examples can be Fourier coefficients, length of projection of strokes on various segmented axes, first, second or higher order of moments. A recent research utilizing Fourier coefficients can be found in [El-Sheikh & Guindi 88].

(d) Template Matching

This directly compares an input character with a standard set of stored prototypes. There are many useable features: x and y coordinates, slopes of stroke segments, chain codes. Because of the presence of variations among different samples of the same character due to minor perturbations of input character shapes, recently many researchers resorted to Dynamic Programming technique of Elastic Matching. It can normalize the expected difference in writing rate, which causes repeated writing of the same character to vary in the quantity of sample points and also causes non-linear time variation in the forming of a character. As estimated by [Nouboud & Plamondon 90], dynamic programming is common to 40% of the recent handwritten recognition system. However powerful this scheme is, it is very computationally intensive and still considered slow in the current PC technology environment. A solution to this problem is the use of a process dedicated to dynamic programming [Lu & Brodersen 84] or special VLSI architecture [Cheng & Fu 86]. Some famous examples utilizing dynamic programming are in [Tappert 82] and [Burr 83].

One of the difference between (a), (b) and (c), (d) recognition methodologies is that the former two require system designer to explicitly present the decision rules whereas the latter two can acquire the knowledge from learning.

2.
ORGANIZATION OF THIS REPORT

The project described here is an on-line character recognition system for cursive English letters and Arabic numerals. Like other recognition systems, there are the following phases:-

(a) Data Sampling
(b) Pre-processing
(c) Stroke Segmentation (or Feature Extraction)
(d) Learning
(e) Prototype Management
(f) Recognizing
(g) Post-processing

Part (a) to part (f) are implemented in the project. In subsequent chapters they are described sequentially. Although part (g) is not implemented, this project also reviewed some state of the art achievement in the post-processing part of character recognition.

The system developed in this project is employing bi-gram template matching technique to recognize the inputted script from the standard ones. The features used is the different small segments (straight line or arc) connected some interesting points of the script. Owing to this segmentation process, much pre-processing and recognizing work is facilitated or eliminated.

3.
DATA SAMPLING

3.1 GENERAL CONSIDERATION

The user input of on-line character recognition is usually by digitizer or writing tablet. Currently there are mainly two types of digitizers widely available on the market.

(a) Electromagnetic / electrostatic tablets
Perpendicular grids of conductors are equally spaced (0.1 to 0.5 inch apart) in the tablet. There is a loop of wire in the stylus tip. An electromagnetic pulse is applied to either the grid or the loop. Then the other can detect the induced voltage. By scanning the tablet conductors to locate the pair closest to the loop and interpolating, the position of the stylus can be located.

(b) Pressure sensitive tablets
There are layers of conductive and resistive material in the tablet. Between the layers is mechanical spacing. An electrical potential is applied across one of the resistive layers to set up a voltage gradient. Pressure from the stylus tip at a point causes the conductive layer to pick up the voltage from the resistive layer. Thus the position of stylus can be determined.

Typically the tablets can have positional resolution up to 200 points per inch and temporal resolution 100 points per second. Depending on the recognition algorithm, there is effect of resolution and sampling rate on the handwriting recognition accuracy. [Kim & Tappert 84] gave a study on this issue.

The data can also be sampled spatially or temporally. If temporal information (e.g. velocity of stroke) is required in subsequent stage, usually the coordinates of the input are sampled at constant time interval. On the other hand, if the positional information is required, it is preferable to have the sampled point nearly evenly distributed. (If some parts are clustered too close together, there is great instability in slope measurement.)

Another important information to be captured is the pen lift. In on-line character recognition, a pen lift simply signifies a new stroke.

3.2 IMPLEMENTATION

The digitizer adopted is Summagraphics Model MM1201. A resolution of 100 dots per inch is used with a sampling rate of 110 samples per second. An IBM PC/AT compatible computer (Intel 80386 CPU at 33 MHz clock) is used to poll the serial port periodically to fetch the data.

The module to capture tablet data supports the following functions:-

(a) capturing the tablet data and displaying them on the PC monitor

(b) outputting the coordinates to subsequent modules. The pen lift is represented by a physically non-existent coordinate (-1,-1)

(c) Sampling the point either
- at a constant polling cycle; or
- with the distance between consecutive sampled point at least a threshold. (A threshold of 1 means no two consecutive points coincide together.)

(d) displaying the segmented script (outputted by the latter Stroke Segmentation module) on the PC monitor.

4.
PREPROCESSING

4.1 GENERAL CONSIDERATION

The main purpose of pre-processing is to minimize the discrepancies among different samples of the same character, so that the subsequent processing is facilitated. The distortion encountered are from two sources:-

(a) different writing styles (character size, slant and orientation)
(b) noisy input due to incorrect positioning of pen lift, zigzag of the pen movement or occurrence of occasional spurious point usually due to hardware problem.

The significance of pre-processing is particularly important in cursive script recognition because usually much individuality is manifested between different writers' writings and even from the same writer.

There are many good literatures to cover this aspect, for instance, [Burr 82], [Brown & Ganapathy 83] and [Bozinovic & Srihari 89].

The pre-processing steps relevant to on-line character recognition can be of the following types:-

(a) Smoothing
It is to correct the zigzag pen movement. Depending on the features to be used in recognition, this process is very important. (Remark: The feature used in this project is immune to minor pen movement fluctuating because the smoothing process is already intrinsically incorporated in the feature analysis process.) The simplest way to perform smoothing is by averaging the coordinates with its neighbours. For curvilinear writing, especially cursive English, a harmonic oscillator has long been regarded to represent a good writing model. Therefore [Burr 82] also adopted a Fourier series representation to smooth the sampled data.

(b) Reference line finding

There are four important lines in cursive script, namely, lower line, lower baseline, upper baseline, and upper line (Fig.1). As quoted form [Bozinovic & Srihari 89],

> "There is psychophysical evidence of early use of such information by humans in lower-case print reading - ascender and descenders are among the most prominent features used for defining shape for letter."

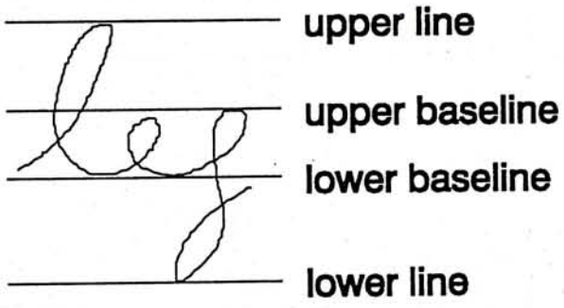


Fig.1
Reference lines in cursive script

(c) Rotation (Baseline drift correction)

It is to re-orient the script so that the baseline becomes horizontal.

(e) De-skewing (Slant adjustment)

This process corrects the different slants of different writers.

(f) Hook elimination

Hooks are small segments having abrupt direction change at the beginning and the end of strokes. The reasons of presence of hook can be:

- the writer cannot manage to operate the styles, especially at high writing speed, causing inaccurate pen-lift operation
- The tablet has some inertia to detect pen lift position, this reporting the wrong pen lift coordinates.

The effect of hook on cursive writing is less than that on hand-printed writing. In the former case, the characters are usually already connected by non-essential ligatures.

(g) Over-stroke and Double-stroke elimination

Over-stroke is a special writing style of retracing a previously written strokes. Double stroke is a special writing styles of retracing the same stroke immediately. [Mandler 89] gave a comprehensive study on the elimination of these redundant strokes.

(h) Stroke connection

It is to eliminate extraneous pen lifts which are usually separated by a very small distance compared to the nominal character size.

(i) Dot reduction

This step simply reduces strokes on small length to single points

4.2 IMPLEMENTATION

It is obvious that different pre-processing schemes have different significance to various recognition algorithms. Therefore not all the above-mentioned pre-processing schemes are implemented in this project. The following describes the pre-processing used.

4.2.1 Stroke connection

It is a simple task. The module just detects whether two strokes has distance less than a threshold. If so, the pen-lift mark is eliminated.

4.2.2 Rotation

This involves lower baseline finding. The pseudo code is as follows:-

```
locate all local minimum points
perform linear regression on these points
do {
  outside-flag = false
  for all local minimum points
    if its distance from regression line > threshold
      eliminate this points from regression sample space
      re-calculate the linear regression coefficients
      outside-flag = true
      }
  } while outside-flag &&
          (number of points in regression sample space > 5)
```

Obviously the lower-base line contains local minimum points. But not all local minima constitutes the baseline. Therefore the linear regression method is used to eliminate these points. The current scheme to drop out those points separating from the best-fitting straight line by a certain threshold is proved sufficiently robust. To be more sophisticated, the point which has the maximum separation distance can be eliminated one by one. Note that the criteria to exit the loop by checking the remaining points in the regression sample space is to avoid dropping too many points even though they are correct. Anyway, this is a pure mathematical approach and does not call for other character shape information. [Brown 83] adopted a different way to eliminate the wrong local minimum points by checking whether they occur on retrograde strokes, dots, crossbars above the script or at cusp locations.

The rotation algorithm is simply a coordinate rotation.

```
find the inclination of the baseline Θ
find the centre point of all samples (mx, my)
for all points in the sample {
  new x =  (old x - mx) cos Θ + (old y - my) sin Θ + mx
  new y = -(old x - mx) sin Θ + (old y - my) sin Θ + my
  }
```

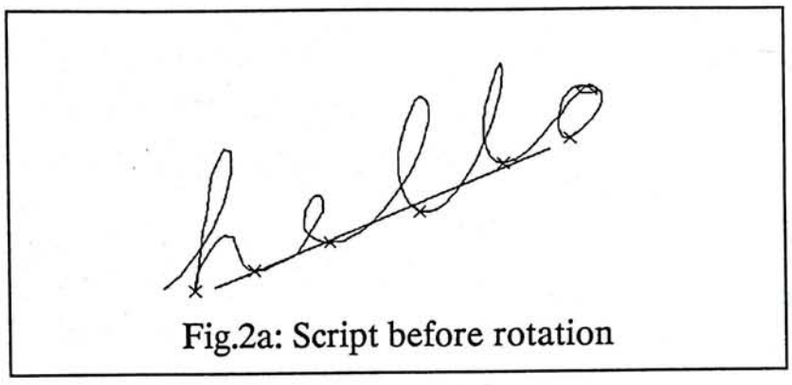


Fig.2a: Script before rotation

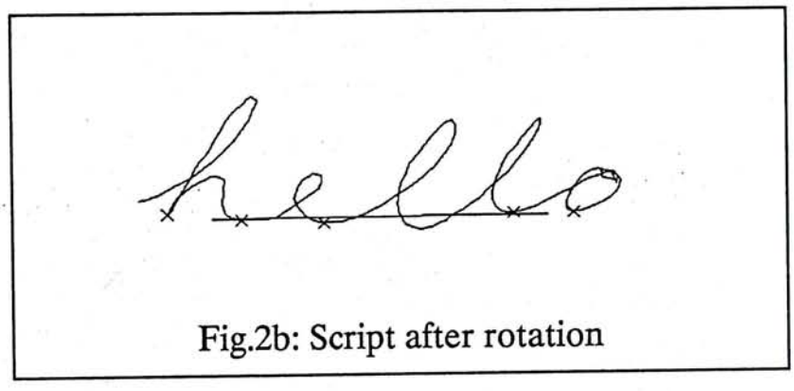


Fig.2b: Script after rotation

4.2.3 Scaling

Since the cursive script has great variation in character widths among different characters, there is no point to perform character width normalization. In fact, most of the researches only scale the script vertically.

This calls for finding lower and upper baselines. Since the script after rotation is already horizontal, another approach based on y-projection intensity analysis can be adopted to find the lower and upper baseline together. This sophisticated method adopting histogram was first described by [Hayes 79]. The captured data are projected onto the vertical axis. An array is used to store the number of projected points at each vertical location. It was found that the resultant histogram is usually bell-shaped and the 75 percentile points are good estimates of the locations the upper and lower baselines.

[Brown & Ganapathy 83] refined this algorithm by placing horizontal thresholds over the script at pre-determined intervals and counting the number of times the script crosses each threshold. The resultant histogram thus is not biased by the actual density of the sample points, which is highly depending on the writing speed.

In this project, the scheme is further enhanced. The straight line segment between each sampled point is wholly projected onto the vertical axis. Using the line segment between sampled points can eliminate the writing speed problem and can achieve true intensity at the same time. Since the 't' cross is a misleading stroke in this algorithm, it should be eliminated in the histogram formation. A simple check on the local slope and ignoring those points with slopes smaller than a threshold can effectively drop out these points.

After finding the lower and upper baselines, the character height is normalized so that the distance between these two baselines is 30 pixels.

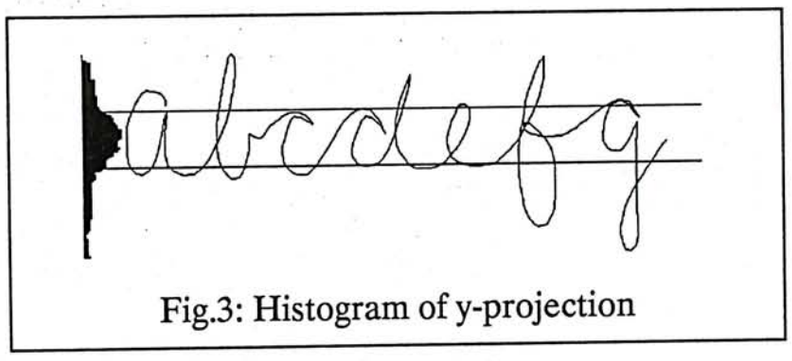


Fig.3: Histogram of y-projection

4.2.4 De-skewing

The core is to find the slant of the word. Three different algorithms are tried in the project. The first is due to [Burr 82] to find the most prominent angle θ of the downward portion of the trajectory. The slope of the slant is a weighted harmonic mean of the ratio of vertical component of pen velocity (v_y) to the horizontal component (v_x). The weighing factor was suggested to be the third power of v_y.

$$\cot \theta = \frac{\Sigma\ v_y^3\ v_x}{\Sigma\ v_y^4}$$

Burr suggested it could give better performance if higher power of v_y is used. cot θ instead of tan θ is used lest the infinity value of vertical slope causes calculation inconvenience.

The second method to find the slant is simpler. It is just to find the harmonic mean of all slopes of the small segments of the script if they have slopes greater than a threshold.

The third method due to [Burr 83] is even less complicated. All the captured points are divided into 2 parts. The first lies above the x-axis (which is in-between the upper and lower baselines) and the second below. The respective centres of gravity are found and the slope between these two points represents the slant of the script.

It is interesting to find that the performances of the three above-mentioned algorithms have no great difference.

After finding the slant, the captured data are tilted horizontally so that the resultant slant is vertical.

$$
\begin{aligned}
&\text{new } x = \text{old } x - (\text{old } y - y_0) \cot \Theta \\
&\text{new } y = \text{old } y \\
&\text{where } y_0 \text{ is the y coordinate of the lower baseline} \\
&\qquad \tan \Theta \text{ is the original slant of the script}
\end{aligned}
$$

5
STROKE SEGMENTATION

5.1 CONSIDERATION

As mentioned before, the features used by the recognition play a key role in the system performance. A good data representation can save much effort in the pre-processing stage and minimize the complexity of the recognition step. In this project, segmentation of strokes into small arcs and line segments is used as the feature of the strokes.

There were many similar approaches adopted by other researchers previously. The simplest methods to do segmentation are by sampling the script at constant time interval or constant geometrical separation ([Tappert 82] and [Burr 83]). This results in many short segments per character (typically 10). Due to writing variation, there is difficulty to perform correlation between inputted data and the standard prototypes. Therefore researchers usually adopted elastic matching to overcome this problem. However, the tough computational demand renders a great system burden.

There were other enhanced segmentation schemes. [Berthod & Maroy 79] described a (local) Angular Segmentation method. Simply put, it is to re-sample the data point so that the angle difference between successive segments is at least a certain angular threshold. This scheme has the advantage that region of greater curvature has more sample points.

Another similar scheme based on angular information was developed by M. Berthod in 1975 and was described in [Belaid & Haton 84]. In this approach the first 3 points (p_1, p_2, p_3) are considered. If the angle between $(p_1, p_2)^T$ and $(p_2, p_3)^T$ and the length of $(p_1, p_2)^T$ are both greater than some thresholds, the point p_2 is considered as a segmentation point. Otherwise p_2 and p_3 are progressively moved forward. Let us call this method the Progressive angular segmentation.

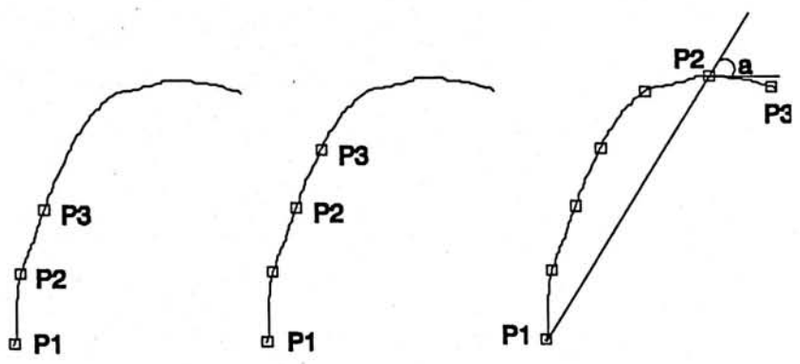


Fig.4: Progressive angular segmentation

In the project both segmentation methods by angular information are tried. It was found that the latter method does not perform as desirable as the former in the case of cursive English writing. For instance, consider the points X in the cursive letters 'q' and 'z' in Fig. 5. If the (local) Angular Segmentation is used, the abrupt angular change at points X is very obvious. However, in the case Progressive Angular Segmentation, supposed P is the previous segmentation points, the angle change is 'diluted' by the straight line approximation of the arc. Of course, a smaller angular threshold can remedy the case. However, too small this value will cause too many segmentation points. Therefore the (local) angular segmentation scheme was adopted in the project.

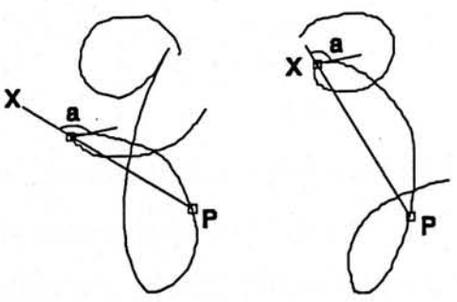


Fig.5: Angle dilution

The problem on irregular segmentation on different samples of same character still needs to be addressed. Considering the character segmentation problem, it is well known that the local minimum points are good candidates for the character delimitation points. If they are also the stroke segmentation points, more consistency between different samples can be achieved.

In the project, both the local minima and the local maxima are always stroke segmentation points. Special consideration is made in case where plateau or basin occur instead of local maximum or minimum. In these situations, only one end of the plateau or basin is the segmentation point; otherwise the spirit to minimize writing variation by using less segmentation point is deviated.

Another kind of definite segmentation points usually occurring in Arabic numerals and printed characters is the point of inflexion. Consider the 'S' and '8' in Fig. 6. The cross marks are already segmentation points. But there is ambiguity in segmenting the central region. It can be observed that the middle segment has a point of inflexion (the derivative of slope is zero). However, practical experiments show that there is great difficulty in detecting this point as the rate of change of slope is so small that it is not easy to be discriminated against noise.

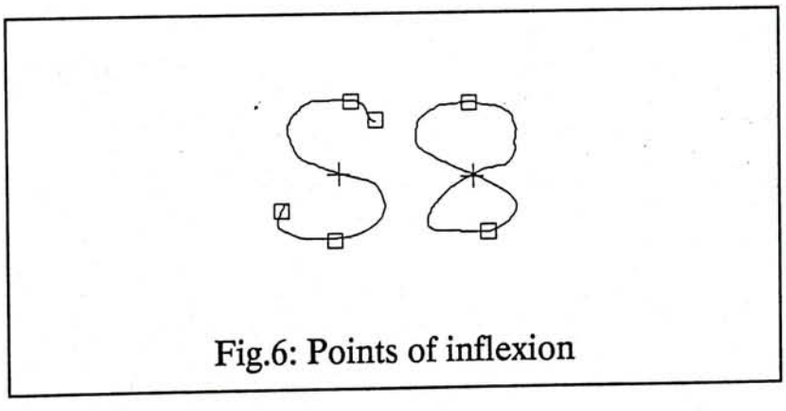


Fig.6: Points of inflexion

Therefore an alternative approach using left-right extreme is used. In this algorithm, all left and right extreme points are located. If within a segment both left and right extreme points are encountered, the last one is considered a new segmentation point. This method proved to be robust enough.

It is noteworthy this segmentation procedure also performs smoothing intrinsically. Another pre-processing step, hook elimination, can also be conveniently performed afterward. Simply check the first and the last segments. If they have length less than a threshold, then they can be discarded.

Dot reduction can also be performed by examining the isolated segments. If they have very short length, then it can be regarded as a dot.

5.2 IMPLEMENTATION

Many previous system treats segmentation as a kind of data purification or filtering. However, because the segmentation algorithm uses simple segment matching technique, the segmentation stage basically becomes the feature analysis as in other recognition system. Therefore consistency is very important.

First the script is delimited by the following points

(a) local maximum / minimum
(b) the second left / right extreme point
(c) where there is a local abrupt angular change

The first 2 conditions are definite. The last is not, depending on angle threshold. It was found that a value of 80° can give the system sufficiently stable segmentation points.

Fig.7a: Script before segmentation

Fig.7b: Script after segmentation

Then the segments are labelled. Since considerable curvature still exists in most segments in cursive script, it is necessary to differentiate them from straight line segments by labelling them with different codes.

Codes	Classification	Criteria		
0	dot	$l < 50$		
1	straight line segment	$\|a\| < 45°$		
2		$\|a-90°\| < 45°$		
3		$\|a-180°\| < 45°$		
4		$\|a-270°\| < 45°$		
5	arc segment	$\|c\|>135°$	starting point higher than ending point	$c > 0$
6				$c < 0$
7			starting point lower than ending point	
8				$c > 0$
9		$\|c\|<135°$	starting point higher than ending point	
10				$c < 0$
11			starting point lower than ending point	
12				$c > 0$

Legend : l - the length of segment
a - the angle of the inclination of segment
c - the curvature of the (arc) segment

In fact smaller number of classes has been tried (only 4 types of arc) to increase the immunity of writing variation. However, the result is that many different letters have the same sequence of codes, leading to ambiguity of recognition.

This touches a philosophical issue. If the classification of samples is too fine, there is a proliferation of prototypes and it is easy to do recognition by separation. But the computation load is heavy. On the other hand, if the division is too coarse, the number of clusters goes fewer but the clusters so divided will overlap to a great extent and burden will again rest on the recognition task to do some ad hoc tests for discrimination.

It is worth mentioning the method to find the curvature of an arc. Here, curvature means the angle subtended by the two ends of the arc at the centre of the circle. Anti-clockwise turn is defined to be positive. If the arc is truly circular, from Fig.8, it is obvious the curvature can also be found by angle difference between the tangents at the starting point and the ending point. Since it is inconvenient to find the centre of circle, this method was first tried to find the curvature. Unfortunately the fluctuation of slope, especially in the starting and ending position of the arc often made the result unrepresentative. Therefore another method is devised.

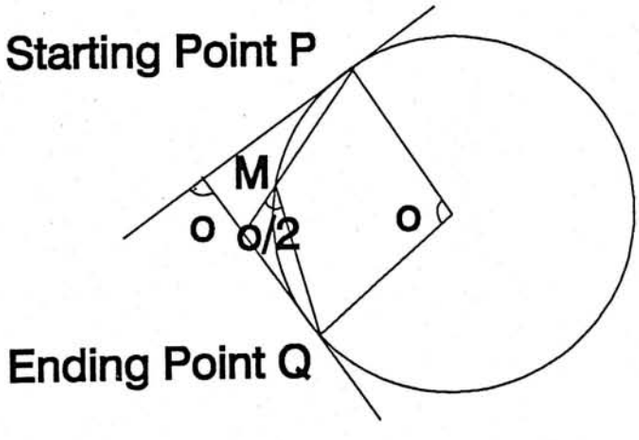


Fig.8
Method to find curvature of an arc

Again from Fig.8, if point M lies on anywhere on the arc, simple geometry tells us that PM and MQ has an angle change of ½o. This method proved to be robust if we take the point M to be nearly in the middle part of the arc.

After doing segmentation, there is one more step peculiar in cursive script recognition, namely handling delayed stroke. It is the stroke which is used to complete a character but does not immediately follow the first portion of that character. Usually the dot in 'i', 'j', the crosses in 't' and 'x' are delayed strokes in cursive writing.

It is most convenient for subsequent processing that the delayed stroke is linked to the segment of the primary stroke as a tag. If the delayed stroke is a dot, then it is linked to the primary segment which has the starting position closest to it. If it is a straight line segment, it is linked to the segment(s) with which it crosses. Since one delayed stroke can cross two 't' in one stroke if they are nearby, this arrangement can make sure the linkage does not miss any primary strokes.

The flow chart to handle delayed strokes is as follows.

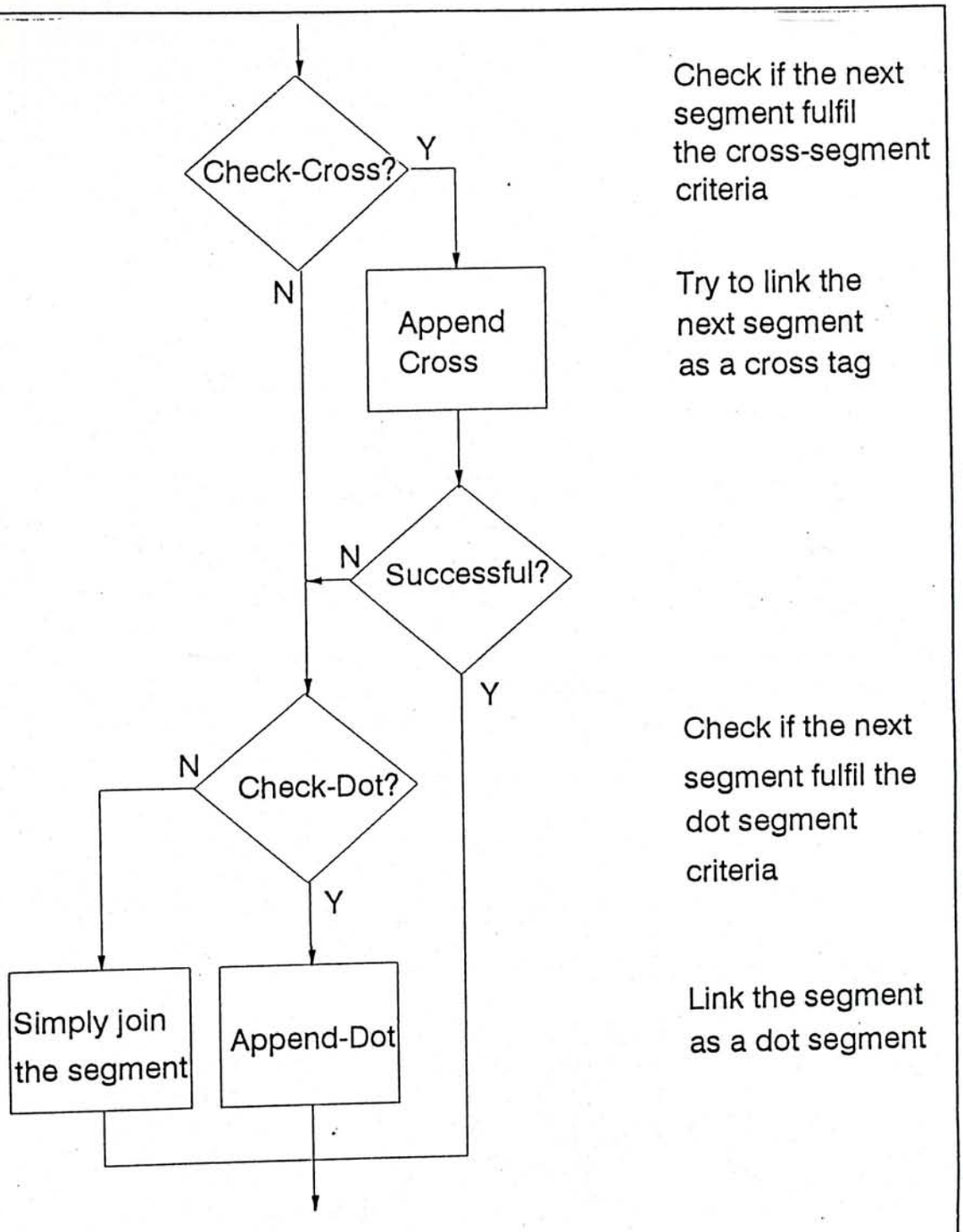


Fig.9:
Flowchart to handle Delayed Stroke

```
CHECK-CROSS :=
  second stroke has only a straight line segment &&
  (its starting x position is within the previous
     segment's x-range ||
   its ending x position is within the previous
     segment's x-range)
```

```
APPEND-CROSS
  for all line segments in the previous stroke {
    if it is a straight line {
      check whether it crosses the proposed delayed
        stroke
    else { /* it is an arc */
      for all small constituting segments
        in the arc
        check whether it crosses the proposed
          delayed stroke
      }
    if crossing occurs
      append the delayed stroke to this segment
    }
  if at least one cross occurs
    return TRUE
  else
    return FALSE
```

```
CHECK-DOT :=
  second stroke has only one segment &&
  it is a dot &&
  its starting x position is within the previous
    segment's x-range &&
  its starting y position is upper than the
    previous segment's baseline
```

```
APPEND-DOT
  for all segments in the previous stroke {
    find one which is going downward &&
      length > threshold &&
      has a minimum distance between starting
        position and the proposed dot
    }
  append the dot to this segment
```

6
LEARNING

In order to build and append prototype library for the recognition process, the user has to supply the explicitly character-segmented prototypes to the system. Since the stroke segmentation already considered the prospective character delimitation points as the stroke segmentation points, the user has to simply inform the system which of these points is the character break-points and spell the word explicitly. In this project, a user-friendly module was created to facilitate this process.

7
PROTOTYPE MANAGEMENT

Since the recognition process uses template matching, it is important that the system manages the set of prototypes efficiently. This module is called *character librarian* in the system. It can support the following operations:-

(a) adding the output information from the learning module to the character dictionary (the set of character prototypes)

(b) sorting the dictionary according to
- alphabetical order; or
- by the occurrence of same prototypes in the dictionary

(c) purging the seldom hit prototypes from the dictionary

The functions (b) and (c) are solely for house-keeping. The major role (a) deserves more description. Since there are variations in different prototypes of the same character, it is difficult to determine if the new entry is already in the dictionary. If every different template is regarded as new item, the number of prototypes will grow to so large that the dictionary is difficult to manage. Therefore a scheme based on correlation is adopted in this system.

The stroke segmented script can already be described by a chain of codes; each code specifying a kind of dot, line or arc segment. The correlation is based on string matching technique between the new item and the existing prototypes. If the length of common part of the *string* is great compared with the original prototypes, this common part is regarded as the *essential* part of the character. The non-essential part will be dropped out from the dictionary. It is not difficult to imagine that these non-essential segments are usually the linking segments between characters in cursive writing and will vary in direction and length depending on which two characters are linked together. It should be cautious to keep the length of common part to at least 3 segments long, or else many successive correlation steps will drop out the basic part of the character later.

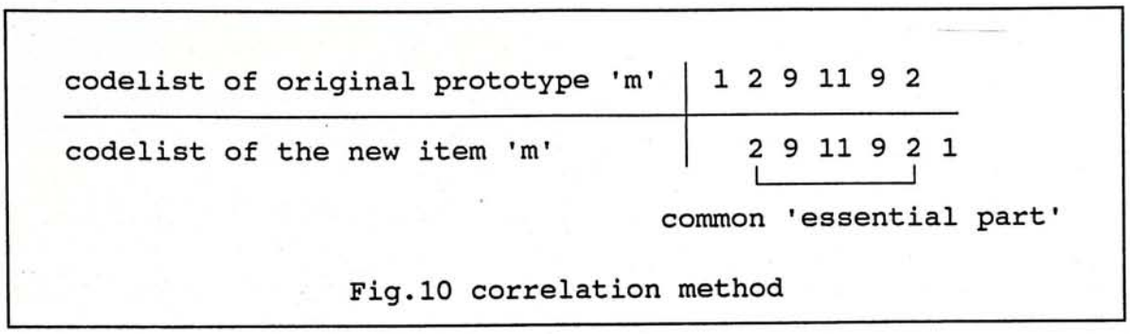


Fig.10 correlation method

This correlation serves the same purpose as the elastic matching technique but is simpler. Furthermore the computational burden now goes to the off-line librarian step and the on-line recognition can be freed to do simple matching.

The library maintains the following information of the segments:-

(a) coordinates of the starting position
(b) length (geographical distance between starting point and the ending point) of the segment
(c) angle (the slope of the imaginary line joining the starting point and the ending point)
(d) curvature of the segment

Each of the above has 3 parameters:-

(a) maximum value
(b) minimum value
(c) mean value

In case the old prototype already exists in the dictionary, the librarian simply updates these 3 parameters according to the new entry. The 3 parameters are used to cater for the variation in writing. Some characters may have greater deviation in some segments that the others. Maintaining this information can facilitate the later recognition process.

8
RECOGNITION

8.1 CONSIDERATION

Recognition module is the core part of the system. Since simple matching technique is used, some special mechanism is used to ensure maximum efficiency during the recognition process. The following are the special consideration points.

8.1.1 Delayed Stroke Tagging

In the stroke segmentation stage, all the delayed strokes are re-arranged so that the primary stroke to which it belongs is attached together. Besides this linking process, the code of the primary segment is also changed as follows

```
new code = old code * 2 + tag-exist-flag
```

Therefore a segment with delayed-stroke linked can easily discriminated from those without delayed stroke.

8.1.2 Bi-gram

This refers to the case that two successive segments are encoded together as a single entity.

original codelist	1	2	3	4	5	6
bi-gram codelist	1+2N	2+3N	3+4N	4+5N	5+6N	6

where N = maximum number of code enumeration

Fig.11: Example of bi-gram coding

Using bi-gram coding in matching process can facilitate the comparison process as two segments are compared simultaneously. Furthermore the bi-gram notation also serves as a kind of higher level feature extraction. For instance, in the cursive letter 'p' in Fig.12, the second and the third segments forms a sudden 180° angle change. Treating these two segments together by a bi-gram code can represent this feature easily. Of course tri-gram or higher order n-gram

may be better for this purpose. But this will be more subject to writing variation.

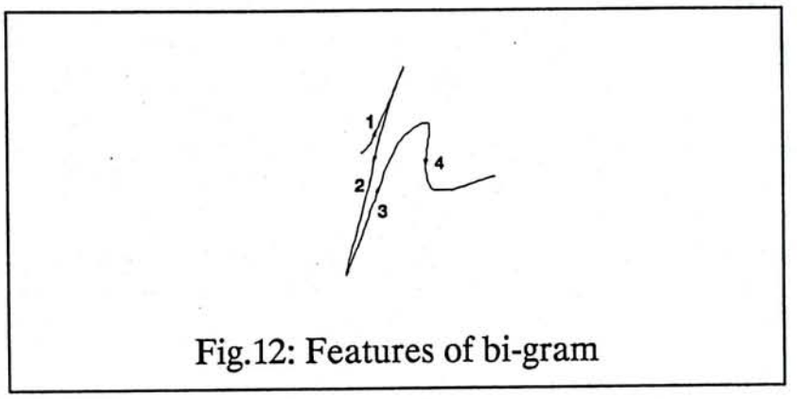


Fig.12: Features of bi-gram

8.1.3 Character Scoring

In order to have the best candidate character to be recognized out, it is very important that the scoring scheme is representative. A good scheme can make the best alternative score much higher than the others. An analogy is the Energy function in general pattern recognition problem.

Two different cursive character, for example, 'h' and 'p' can have the same bi-gram codelist. What they differ is the parameters of each segment. Therefore the elementary scoring problem is how to define the similarity between two segments. In the project, it is made up of the following multiplication factors:

(a) similarity in length of the segments
(b) similarity in inclination of the segments
(c) similarity in curvature of the segments (only applicable if the segment is an arc)

The *similarity* here means:

$$\frac{1}{1 + \dfrac{\text{difference with the mean}}{\dfrac{\text{range in prototypes}}{2}}}$$

The maximum value is unity if there is no discrepancy and the score goes asymptotically to zero if the deviation is great. The deviation is normalized by

the difference between the maximum and minimum values of prototypes in order to take care of the different properties in different characters

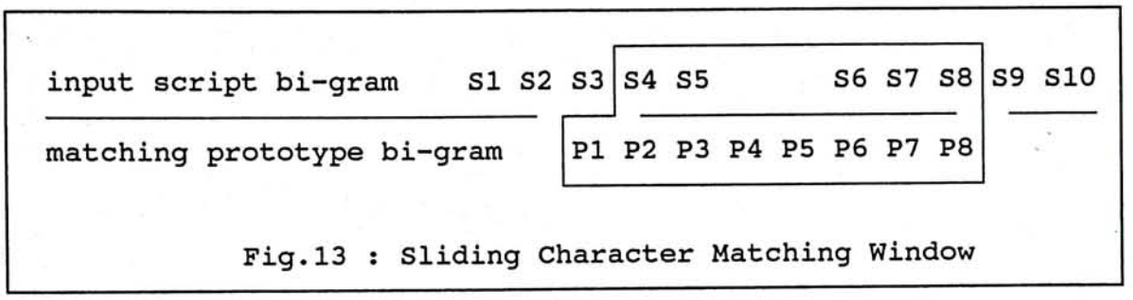


Fig.13 : Sliding Character Matching Window

Since the inputted script has no character boundary, the character score is calculated based on a sliding character matching window, in which the bi-grams of the input match sequentially with those of the proposed prototypes. Fig.13 shows the sliding window. The character scoring pseudo-codes are as follows:

```
total bi-gram score = 0
P = P1 /* starting bi-gram of the prototype */
for all bi-gram S in the input script
  /* here starting at S4 */
  if S is found from bi-gram list starting from P {
    record the position in P /* say P2 */
    calculate the segment similarity
    if delayed segment exists
      calculate the delayed segment similarity
    add the score to total bi-gram score
    }
  else {
    character score = 0
    return
    }
character score = total bi-gram score /
                  no. of segments processed
if no. of segments processed deviates from
  the length of prototypes with a large number
  character score = 0
```

This algorithm supports some intermediate stroke mismatch so long that the number is small compared with the total number in the prototypes. This running window scanning scheme effectively performs the character delimitation at the same time.

8.1.4 Ligature Handling

Because cursive writing contains many linking segments between characters. These are usually unable to be recognized out, so are some roughly written characters. Unrecognized segments are skipped in the recognition algorithm. In order to represent these skipped segments, their count is inserted between recognized letters as negative number. This can help the subsequent post-processing to identify the best answer.

byte number	0	1	2	3	4	5
recognized character	b	-1	o	-2	o	k

8.1.5 Word Scoring

Because of lack of explicit character segmentation, the recognition scheme can recognize some consecutive simple characters as a single complex character or vice versa. To exaggerate the problem, Fig.14 shows some sketches. The first part can be recognized as 'mm, 'nuv' etc. In the second part, the 'o' can be treated as a hand-printed 'o' and a dangling 'c' - the joining ligatures can also be recognized as some rough writing sometimes.

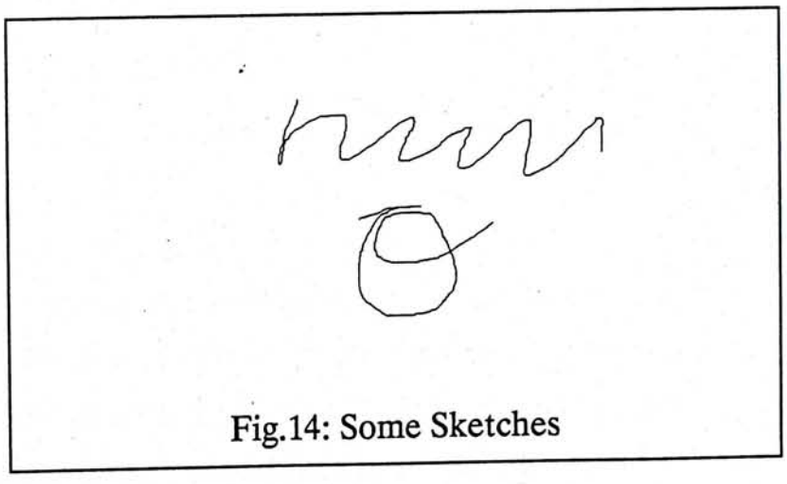


Fig.14: Some Sketches

Naturally skipping the rough writing (usually characterized by lower character score) can make the resultant word score higher. However, this would make the program easily drop out some roughly written characters. Therefore to make a balance on this situation, the word score is defined as:

$$\frac{\text{total character score}}{\text{number of character}} \times \left(1 - \frac{\text{no. of skipped segment}}{\text{total no. in the stroke}}\right)$$

Of course, some later processing like word dictionary can also solve the problem. But it is best that maximal information is utilized by the recognizer first before relying the subsequent steps. Furthermore using such word scoring scheme can sort the word candidate in a more realistic order so that the chance of early hit in word dictionary is greater.

8.2 IMPLEMENTATION

Based on the sliding window schemes, three recognition algorithms have been devised. The details, merits and demerits are described below.

8.2.1 Simple Matching

It is the faster method out of the three. But the problem is that it only gives the best answer and misses the other alternatives which can be the actual correct result. The following lists the pseudo codes.

```
for all segments in the input script {
  find the maximum character score in the sliding window
    of all prototypes in the dictionary
  if found
    skip the segments until the end of the window
  else
    skip one segment
  }
```

8.2.2 Best First Search Matching

This scheme is an enhancement of 8.2.1, in the sense that it allows character alternatives. For instance in Fig.15, the first part of script can be recognized as 'cl' or 'd' or even a distorted 'u'. The algorithm searches the tree from the most promising partially expanded alternative by Best-first search. The partial score is the average character scores. Again, if no alternative is found at a certain leaf of the tree, a segment is skipped and searching is restarted.

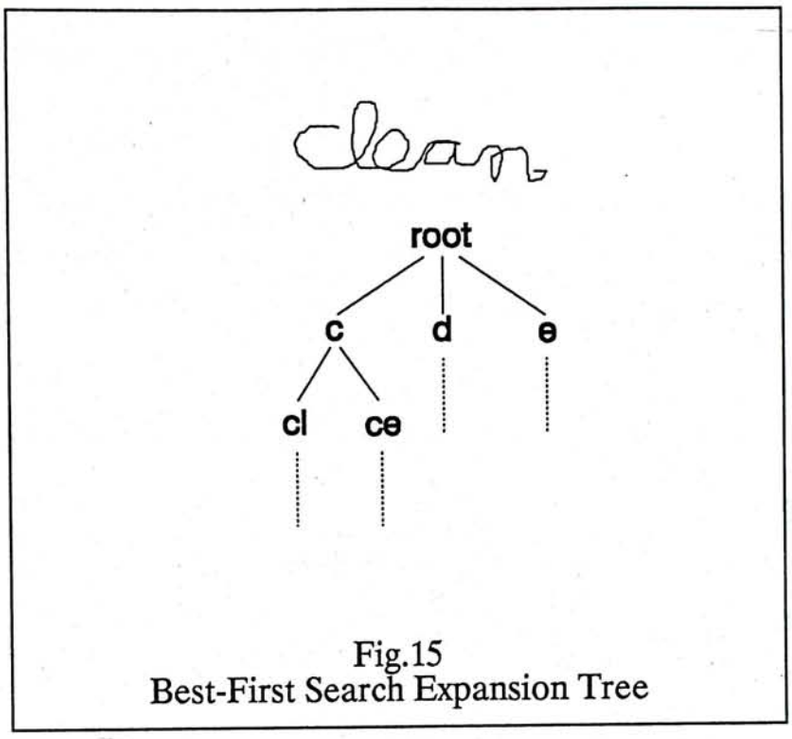


Fig.15
Best-First Search Expansion Tree

One problem of this scheme is that supposed the first portion ('d' part) of the word is roughly written but the subsequent part is very tidy. Unfortunately character 'c gets a marginally higher score than 'd' and therefore the 'c' sub-tree is expanded first. Because of the tidy subsequent part, the search cannot resume to the 'd' subtree until the 'c' subtree is exhausted.

To remedy the case, two solutions are devised.

(a) amend the scoring scheme so that the partial solution with the smallest number of character is expanded first (effectively the search becomes a breadth-first search)

(b) increase the length of the partial solution queue

Study showed that the first solution gave a very slow performance because even the unlikely candidate subtree is also expanded. Therefore the second solution is used. In fact it was found that the queue length seldom went beyond 50.

In fact, the greatest problem of the Best-first search method is that it does not skip segment if it can find at least one candidate letter. For example, in Fig.16, the letter 'a' is to be recognized. If the sliding window start at point A, then the letter 'e' will be recognized out and the recognition process continues on point

'c'. If the sliding window starts at point B, the character 'a' can be recognized out.

This problem can be solved by leaving a partial solution with one segment skipped in expanding all sub-tree. But this will cause proliferation of the number of leaves and degrade the recognition performance tremendously.

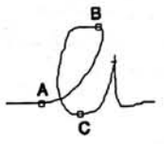


Fig.16
Different Sliding windows results in different characters

8.2.3 Multiple Track Method

A track means one set of alternative character. There can be more tracks but it was found that usually two are enough (Fig.17).

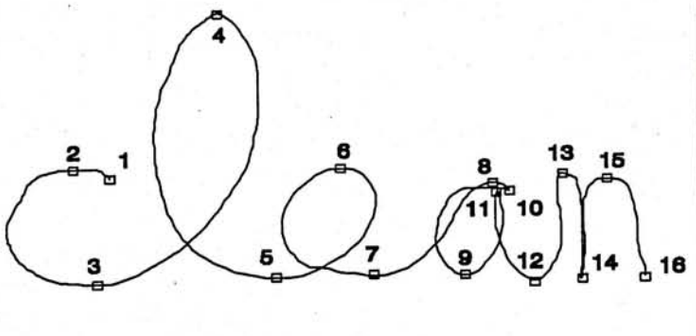


Fig.17 Multiple-Track Method

```
Segmentation    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
Alternative 1   c  c  l     e        a  e  v     n  n
Alternative 2   d     e     l
                recognized character at the sliding window
                starting at the specified segmentation points
```

The following is the pseudo code:

```
for all segments in the input script
  for all prototypes in the dictionary
    if it is the same as either one of the last
      alternatives in the track
      skip this prototype
    else {
      if character score at the sliding window >
         character-score-threshold
         mark it in track 1 or track 2
      }
```

There may be cases when same character is recognized in sliding window starting at two successive points (e.g. point 1, 2 in Fig.17 for letter 'c'). Therefore the latter is skipped to avoid duplication.

After finding all character alternatives, the candidate word is composed by a self-recursive procedure *word-expand*.

```
word-expand:
  if end of stroke is reached {
    if word-score > word-score-threshold
      store the answer in the word alternative queue
    }
  else {
    while track 1 and track 2 have no character alternatives
      skip one segment

    skip one more segment
    word-expand
    restore the one segment

    take out the character alternative in track 1
    skip that number of segments
    word-expand

    take out the character alternative in track 2
    skip that number of segments
    word-expand
    }
```

8.3 SYSTEM PERFORMANCE TUNING

The effect of character-score-threshold and the word-score-threshold is very important in system performance. Varying these two parameters can effectively tune the system.

(a) Character score threshold

Controlling this parameter can make the system adapt to the tidiness of the input script. If the input is well-written the average character score is high and the character score threshold can be set higher so that the system can consider fewer alternatives.

One kind of usually mis-recognized segments is the ligature strokes. They can be recognized as 'e' or 'c' characters with low score. Setting suitable character score threshold can eliminate these alternatives.

There are two types of errors in character recognition, namely:

- rejection error : no candidate character is proposed
- incorrect recognition : the wrong candidate character is presented.

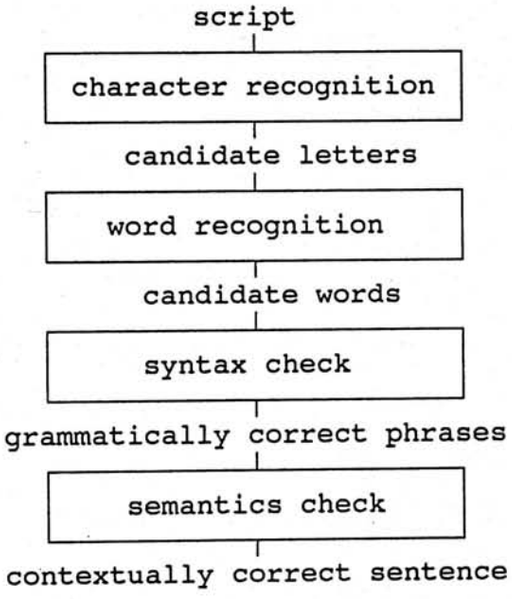
In fact, by adjusting the character score threshold, the two kinds of error rate can be changed complementarily depending on requirement.

(b) Word Score Threshold

After processing all character alternatives, the prospective word candidates are built. Some has greater number of skipped segments but may have higher net score; some may have reverse situation. A suitable word score threshold can effectively eliminate the word candidate with too many skipped segments buy keep correct answers properly.

9
POST-PROCESSING

Although this part is not implemented in the project, it is worthwhile mentioning some of state of the art post-processing techniques that can improve the performance of a recognition system. A complete character recognition process can be viewed as a successive recognition at different level of knowledge.



According to [Suen *et al* 77], even human beings can make about 4 percent mistakes when reading in the absence of context. Therefore to make a robust recognizer, post-processing is very important. Currently most of the researchers in character recognition only worked on the word level to give the best word candidates. For syntax and semantics checks, which are mainly of Artificial Intelligence concern, are separately studied by other relevant researchers.

There are basically two streams of methodologies to check word. The first is based on probability approach and the second on word dictionary.

9.1 PROBABILITY MODEL

It is usually based on the probability of the existence of n-gram to determine whether to reject a word proposal. For instance, no word has the triple 'bzq' and therefore if a proposed word has this pattern, it should be wrong. The great advantage of this approach is the small requirement on storage space and the

ease of table look-up. In fact it is readily incorporated in the character recognition stage. One recent example is in [Kundu *et al* 89] and [Bozinovic & Srihari 89].

The major drawback of probabilistic approach is that the probability is only good on *a prior* knowledge on majority and can cause errors in some exceptional case. A word with n-gram of small occurrence likelihood does not mean that it is wrong. Another problem is the difficulty to update the probability values based on different applications. For instance a medical writer may frequently specify some character patterns which, however, is seldom used in normal English. Finally, to be a serious Markov model, bi-gram may not be sufficient accurate. However, a tri-gram notation needs 26^3 probability entries, which means already a comparable storage demand as a small word dictionary.

9.2 WORD DICTIONARY APPROACH

This is based on looking up the word dictionary to determine whether a proposed entry has correct spelling or not. [Burr 83] already implemented a nice dictionary of 160 kilo-bytes storage. Because of the large memory requirement, Burr put most of the dictionary into the disk.

To use the dictionary efficiently it is not enough to check whether a word is correct or not; it is desirable the dictionary can give a list of similar but correct entries even though the input is wrong in spelling. [Kohonen & Reuhkala 78] gave an associate loop-up method based on redundant hash addressing. [Doster 77] accessed the dictionary by a special hash function using the first or the second half of a word.

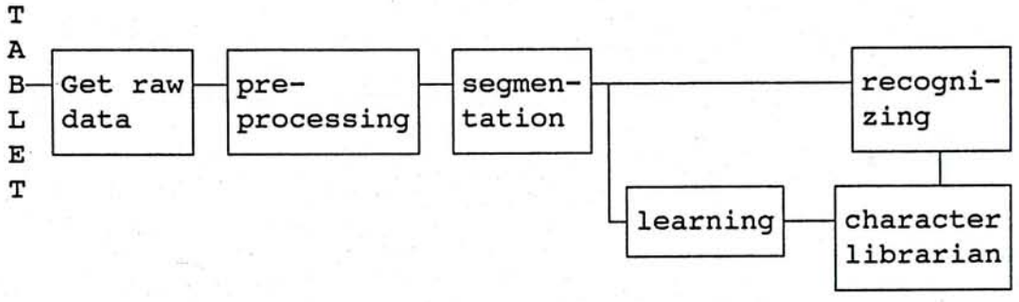
In cursive English character recognition, some classes of character can be easily mixed up. For instance 'n' and 'u'; 'l' and 'e'; 'p' and h' can easily be inter-changed. Therefore the concept of *Equivalence class* in [Aldefeld *et al* 80] is very useful to correct these errors.

With the availability of more and cheaper solid state memory on computer, storage requirement is less of a problem than the requirement of a fast associate searching algorithm. Recently [Takahashi *et al* 90] described a sophisticated spelling correction scheme based on best-match method. [Wells *et*

al 90] presented an improved 26-way tree lookup procedure to achieve real-time searching.

10
SYSTEM IMPLEMENTATION AND PERFORMANCE

Each recognition step is coded as a separate module to ease maintainability. All the modules are implemented in C language (Turbo C 2.0 and Microsoft C 5.1) on an IBM PC/AT compatible computer with Intel 80386 CPU at 33 MHz. The modules can be linked up by piping in MSDOS as follows:



Currently there are about 200 prototypes storing 26 lower case cursive English characters. SMALL memory model is sufficient to handle the data requirement. Depending on which recognition schemes (simple, best first search or multiple track) and the score thresholds, the recognition time is different and is not linear with respect to the number of characters. Just for illustration purpose, the time to recognize the word 'cheung' in Fig.18 is shown in the following table. The time does not include pre-processing step and the time to load the template dictionary from disk to RAM.

character score threshold	word score threshold	recognition time
35	80	4 seconds
10	10	7 seconds

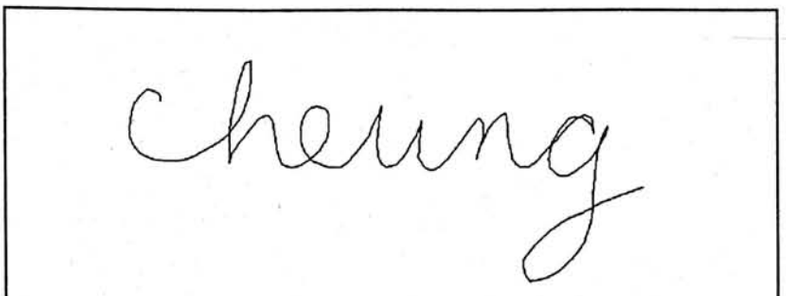


Fig.18
Sample pattern to be recognized

There is no recognition error rate available because this value can only be realized by extensive objective testing. [Nouboud & Plamondon 90] described a detailed test procedure for character recognition system.

11
DISCUSSION

The current method gains its merit by simple data representation. Although the number of templates per character is already around 10, it does not cause a great memory burden in current PC technology. Another advantage of this system is that it can easily adapt to different user's writing. If we accept that a recognizer can be personalized, the system implemented in this project can fulfil the requirement of an on-line system. It can easily capture more prototypes into the system.

One great difference between this system and the others is its little reliance on explicit knowledge on features of cursive English writing. Currently the only rules adopted is the handling delayed strokes and segmenting strokes in all local minimum points for prospective character boundaries. Other than these, all the knowledge is captured from user inputted script. This aspect of concept is similar to neural network approach.

Therefore the system can recognize very cursive writing, so long that it has learnt that style before. On the other hand, even though some scripts may look well-written, it is possible that this system fails to do correct recognition. The following figures depict some of the correctly recognized writings:-

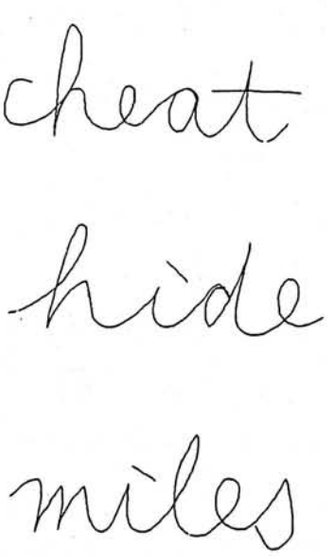


Fig.19: Correctly Recognized Scripts

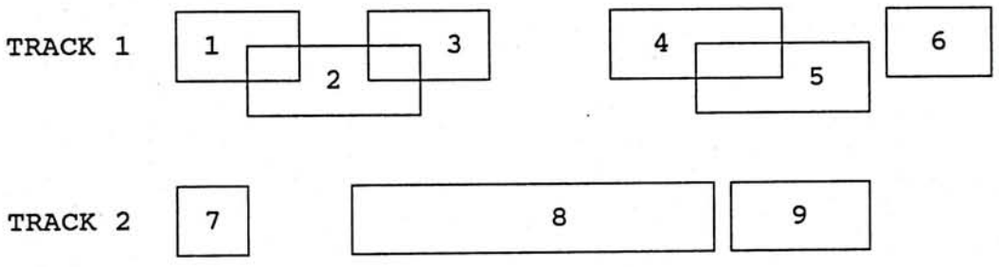
Because of the incremental nature of the system, in the production phase, the system should be equipped with a character template dictionary of a wide variety of different writers' scripts. The personalization stage can be achieved by allowing each new user to input some new writings to supplement the master dictionary. By this way, the system can be built robust enough.

From Chapter 10, it can be found that the system can usually recognize a word in several seconds using double-track method. In fact, most of the time is spent on building the word candidates from character alternatives. Because of the bi-gram coding, all the character candidates can easily be selected in around one second. Actually currently the system simply uses a simple *for* loop to search all dictionary entries for the existence of a certain bi-gram. If the dictionary is indexed or hashed by the bi-gram codes, the time to give character candidate can even by shortened drastically, especially when the character dictionary grows to the size of several thousand templates.

The longer time consumed by word building from recognized characters is due to the tree parsing (as in Fig.15) by recursion. This makes the time not linear but exponential to the length of the word. Actually the tree is already pruned when the partial word score is below a threshold. But there is a side-effect to stop expanding some initially unpromising branches which, however, can give a correct ending finally. This phenomenon corresponds to a word badly written on the left but the script looks acceptable near the end, because the word is built from left to right during the tree parsing. Incidentally, in cursive writing, it can be found that the extremes of a word are usually less tidy than the middle portion, because of stroke linking.

One simple method to cut down system search time is to make the system adjust the character and word score thresholds dynamically to the previous writing tidiness. If the writing is good, higher thresholds can be set so that less character alternatives (less branching points at each node) or less interim word candidates (pruning the unpromising tree branches) can be realized automatically.

On the other hand, if the parsing is to be abandoned, is there any better alternative algorithm? Currently the writer is thinking of the following fragmentary ideas. Using the double track recognizing algorithm, the character alternatives can be represented as follows (each numbered box for a character alternative):-



If tree parsing is used, the following ten word candidates are examined:-

1-3-4-6	7-2-4-6
1-3-5-6	7-2-5-6
1-3-9-6	7-2-9-6
1-8-5-6	7-8-5-6
1-8-9-6	7-8-9-6

However, if we inspect the figure macroscopically, it can be seen that the sequence 1-8-5-6 looks the best. The word 'macroscopically' here has linked to the concept of global information. Tree parsing gathers global information by fetching local information sequentially (assumed there is no tree pruning). However, it is possible that the expansion does not start from the left extreme but from some middle good character alternative. Here if we can notice that box 8 is a good point to start, many subsequent effort is saved. This means that we need not build the word from left to right, but can start from some promising middle island. But this still leaves the problem of how to connect these character alternatives. This question can be modelled as optimal path

connection between these 'islands'. But the algorithm should not be exponential to the number of connecting points.

Another innovative approach is similar to the neural network approach. Each promising node (character alternative) can give reinforcement marks to the adjacently connected nodes, but give negative scores to the nodes occupying the same region. This Kohonen method has the advantage that all nodes are being equal in the context of expansion priority. So long that one node has good score, it can easily suppress other conflicting character alternatives and spread its field strength to other linking nodes, effectively building a bigger and bigger node (grouping of consecutive character alternatives). Finally a whole word can be built.

12
EPILOG

The main contributions of this project to build an on-line character recognition system are the following:

(a) segmenting the stroke into a list of small straight line segments or arcs at some well-behaved positions, with delayed stroke handling;
(b) coding the list of segments as a list of bi-gram codes;
(c) building the template library by correlation technique;
(d) recognizing by successive bi-gram matching with a multiple-track scheme.

In fact [Ward 90] pointed out that handwriting character recognition is an engineering problem rather than a scientific problem, in the sense that there is not a single fundamental theoretical approach. After implementing this system, the writer begins to appreciate more this view. In order to make the system robust, the designer has to take care all the unusual but possible response from user. Just looking at character recognition, the system should cater for retrace like double-stroke or over-stroke. In document level, the system should expect user to edit the inputted script by deleting, inserting or modifying past inputs. Therefore besides character recognition alone, the user writing behaviour and expectation need to be studied thoroughly. This may call for many man-years of study.

APPENDIX I
PROBLEMS ENCOUNTERED AND SUGGESTED ENHANCEMENTS ON THE SYSTEM

They are categorized by the different modules.

1. Data Sampling

(a) Currently the coordinates of the stylus of the digitizer are fetched from the asychronous port by polling. This causes intermittent data loss especially during a disk data transfer. Interrupt-driven data fetching should be used to overcome this problem.

(b) The input from digitizer is only displayed on the computer screen; there is no trace on the digitizer of previous pen location. This causes great difficulty for user to write delayed stroke - the cross of 't' or the dot in 'i' may be wrongly placed on the top of other characters. Therefore the pressure sensitive LCD panel is suggested to use. It can provide the electronic ink feature by displaying what the user writes on the surface.

2. PRE-PROCESSING

(a) Because many pre-processing steps involves trigonometric functions, which are usually implemented in double precision floating point in C language, they cause large computational demand, especially on a PC without numerical co-processor. In fact, the accuracy of these operations needs not be very high and can be replaced by integer operation. For instance,

To find	Method
angle between two lines	user vector dot product
square rooting	successive odd number subtraction $n^2 = 1 + 3 + 5 + .. + (2n-1)$
tan Θ	sqrt $(1 - \cos^2\Theta) / \cos \Theta$

3. SEGMENTATION

(a) To use integer operation instead of floating point, as in 2(a).

(b) More features can be used in coding the segment or as segment parameters:

- ascender or descender (to help discriminate 'p' and 'h')
- loop width (to help discriminate 'ee' or 'u' in Fig.20)
- loop closure distance (to help discriminate 'u' or 'o')

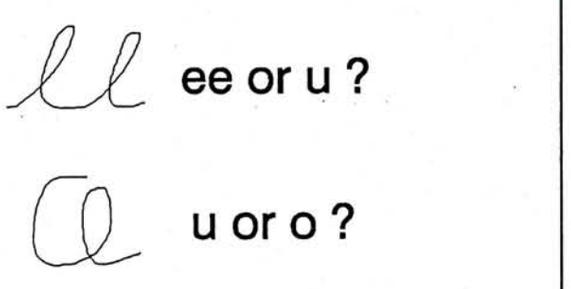


Fig.20: Ambiguity due to loop width

(c) Since the local maximum and local minimum points currently are definite segmentation location. If the user writes a horizontal line in a zigzag manner (Fig.21), then many small straight line will be resulted. To remedy this case, a segment parser can be built to check every consecutive segments. If they have angle difference less than some threshold, the segmentation point can be eliminated.

In fact, a separate segment parser can serve more functions. For instance point (b) can be realized in this parser because of the flexibility.

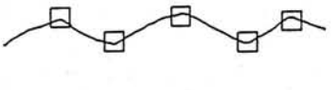


Fig.21: Zigzag line segment

4. RECOGNITION

(a) As the prototype dictionary grows larger, search by indexing should be used to shorten processing time. An easy implementation is to index the dictionary entries by the bi-gram. An occurrence of a certain bi-gram can point to a list of character templates.

(b) Currently the character score threshold and the word score threshold are explicitly defined. However, it is easy to modify the program so that these values can vary with the user input style. In this case, the system can adapt dynamically for different writers at different times.

5. APPLICATIONS

(a) Finally to make the system productive, it is necessary to hook it to PC applications. [Doster & Oed 84] has already suggested a way on Concurrent CP/M-86. Recently Microsoft Windows might be a good platform for character recognition system integration.

APPENDIX II
GLOSSARIES

In the field of character recognition, there are many common terms used with special meaning (as in other research areas). Therefore to avoid ambiguity, it is worthwhile giving exact definitions of some terms used in this report. Some of the definitions are quoted from the specified literatures.

ambiguity [Wang 82]
: The degree of ambiguity of a character α, in the domain Ω under the encoding scheme Γ, is defined as the maximum number of interpretations of α in Ω under Γ.

character segmentation
: separating a word into letters

curvature
: It is the signed angle subtended from the centre of the circle to the starting and ending points of an arc. An anti-clockwise turn is defined to be positive.

delayed stroke [Tappert 82]
: It is the one used to complete a character but which does not immediately follow the first portion of that character. For example, in 'city', the dot in 'i' and cross in 't' are delayed.

double stroke [Mandler 89]
: A special writing style of retracing the same stroke

over stroke [Mandler 89]
: A special writing style of retracing a previously written stroke

recognizability [Wang 82]
: The degree of recognizability of a character α is defined as the reciprocal of the degree of ambiguity if α is in the dictionary; other it is defined to be zero.

stroke [Tappert *et al* 90]
: The writing from pen down to pen up

stroke segmentation
: segmenting a stroke into a number of small line segments or arcs

REFERENCES

Aldefeld, B., S.E. Levinson and T.G. Szymanski 1980
A Minimum-Distance Search Technique and its Application to Automatic Directory Assistance, The Bell System Technical Journal, Vol.59 No.8 Oct, pp.1343-1356

Aoki, K. and K. Yoshino 1989.
Recognizer for Handwritten Script Words Using Syntactic Method, in R. Plamondon, C.Y. Suen and M.L. Simner (Eds.), Computer Recognition and Human Productions of Handwriting, Word Scientific Publ. Co. pp.5-18

Belaid, A. and J.P. Haton 1984
A Syntactic Approach for Handwritten Mathematical Formula Recognition, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.PAMI-6 No.1, Jan, pp.105-111

Berthod, M. and J.P. Maroy 1979
Learning in Syntactic Recognition of Symbols Drawn on a Graphic Tablet, Computer Graphics and Image Processing, Vol.9 pp.166-182

Bozinovic, R.M. and S.N. Srihari 1989
Off-Line Cursive Script Word Recognition, IEEE Trans in Pattern Analysis and Machine Intelligence, Vol.11 No.1, pp.68-83

Brown, M.K. and S. Ganapathy 1983
Preprocessing Techniques For Cursive Script Word Recognition, Pattern Recognition, Vol.16 No.5 pp.447-458

Burr, D.J. 1982
A Normalizing Transform for Cursive Script Recognition, in Proc 6th Int. Conf. Pattern Recognition, Munich, West Germany, Oct, pp.1027-1030

Burr, D.J. 1983
Designing A Handwriting Reader, IEEE Transactions on Pattern Analysis and Machine Intelligence Vol.PAMI-5 No.5 Sep pp.554-559

Cheng, H.D. and K.S. Fu 1986
VLSI Architecture for Dynamic Time Warp Recognition of Handwritten Symbols, IEEE Trans. Acoust, Speech, Signal Processing, Vol.ASSP-34 Jun, pp.603-613

Doster, W. 1977
Contextual Postprocessing System for Cooperation with a Multiple-Choice Character-Recognition System, IEEE Trans on Computers, Vol.C-26 No.11 Nov, pp.1090-1101

Doster, W. and R. Oed 1984
Word Processing with On-Line Script Recognition, IEEE Micro Oct pp.36-43

El-Sheikh, T.S. and R.M. Guindi 1988
Computer Recognition of Arabic Cursive Scripts, Pattern Recognition Vol.21 No.4 pp.293-302

El-Sheikh, T.S. and S.G. El-Taweel 1990
Real-Time Arabic Handwritten Character Recognition, Pattern Recognition Vol.23 No.12 pp.1323-1332

Farag, R. 1979
Word-Level Recognition of Cursive Script, IEEE Trans on Computers Vol.C-28 Feb pp.172-175

Frishkopf, L.S. and L.D. Harmon 1961
Machine Reading of Cursive Script, in C. Cherry (Ed.), Information Theory (4th London Symp.) London, England, Butterworths, pp.300-316

Govindan, V.K. and A.P. Shivaprasad 1990
Character Recognition - A Review, Pattern Recognition Vol.23 No.7 pp.671-683

Hayes, K.C. Jr 1979
Reading Handwritten Words Using Hierarchical Relaxation, Ph.D. dissertation, University of Maryland, College Park

Kim, J. and C.C. Tappert 1984
Handwriting Recognition Accuracy versus Tablet Resolution and Sampling Rate, in Proc. 7th Int. Conf. on Pattern Recognition, Montreal, Canada, Jul-Aug, pp.917-918

Kohonen, T. and E. Reuhkala 1978
A Very Fast Associative Method for the Recognition and Correction of Misspelt Words Based on Redundant Hash Addressing, in Proc 4th Int. Joint Conf. on Pattern Recognition, Japan pp.807-809

Kundu, A., Y. He and P. Bahl 1989
Recognition of Handwritten Word: First and Second Order Hidden Markov Model Based Approach, Pattern Recognition, Vol.22 No.3 pp.283-297

Lu, P.Y., and W. Brodersen 1984
Real Time On-Line Symbol Recognition Using a DTW Processor, 7th ICPR, Vol.2 pp.1281-1283

Mandler, E. 1989
Advanced Preprocessing Technique for On-Line Recognition of Handprinted Symbols, in Computer Recognition and Human Productions of Handwriting, R. Plamondon, C.Y. Suen and M.L. Simner (Eds.), Word Scientific Publ. Co. pp.19-36

Mandler, E., R. Oed and W. Doster 1985
Experiments in On-Line Script Recognition, in Proc 4th Scandinavian Conf. Image Anal, June, pp.75-86

Mermelstein, P., and M. Eden 1964
Experiments on Computer Recognition of Connected Handwritten Words, Inf. Control Vol.7 pp.255-270

Nouboud, F. and R. Plamondon 1990
On-Line Recognition of Handprinted Characters: Survey and Beta Tests, Pattern Recognition Vol.23 No.9 pp.1031-1044

Srihari, S.N. and R.M. Bozinovic 1987
A Multi-Level Perception Approach to Reading Cursive Script, Artificial Intelligence Vol.33 pp.217-255

Suen, C.Y., R. Shinghal and C.C. Kwan 1977
Dispersion Factor: A Quantitative Measurement of Quality of Handprinted Characters, in Proc Int Conf Cybernetics and Society pp.681-685

Takahashi, H., N. Itoh, T. Amano and A. Yamashita 1990
A Spelling Correction Method and its Application to an OCR System, Pattern Recognition Vol.23 No.3/4 pp.363-377

Tappert, C.C. 1982
Cursive Script Recognition by Elastic Matching, IBM J. Res. Develop. Vol.26 No.6 Nov, pp.765-771

Tappert, C.C. 1984
Adaptive On-Line Handwriting Recognition, in Proc. 7th Int. Conf. Pattern Recognition, Montreal, Canada, Jul-Aug, pp.1004-1007

Tappert C.C., C.Y. Suen and T. Wakahara 1990
The State of the Art in On-Line Handwriting Recognition, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.12 No.8 Aug, pp.787-808.

Ward, J.R. 1990
One View of On-Going Problems in Handwriting Character Recognition, In Proc Int. Workshop on Frontier in Handwriting Recognition, Montreal, Canada, Apr, pp.101-107

Wang, P.S.P. 1982
A New Character Recognition Scheme with Lower Ambiguity and Higher Recognizability, in Proc 6th Int. Conf. Pattern Recognition, Germany, Oct, pp.37-39

Wells, C.J., L.J. Evett, P.E. Whitby and R.J. Whitrow 1990
Fast Dictionary Look-Up for Contextual Word Recognition, Pattern Recognition, Vol.23 No.5 pp.501-508