# AILP (2010) Assignment 1 Report

| | |
|---|---|
| **Key1** | *ute cYe 26F* |
| **Key2** | *MhlCPrpChsl* |

## Abstract

*On-line character recognition is an important technological problem in modern life. With its increasing usefulness and popularity, it presents many new problems for programmers to investigate. In this report, an on-line recognition system based on dynamic time warping is examined, and key techniques in normalisation and multi-template selection are identified. These techniques are evaluated for performance based on accuracy and computational complexity, and the best are combined into a final system in order to develop an optimal level of accuracy.*

## 1. Introduction

With the increasingly widespread availability of touch devices, real time handwriting recognition is found in many different areas using many different technologies - whether it is simply signing for a package on a small handheld device, or taking complicated notes on a touch-sensitive laptop.

One of the key challenges that recognition systems face is the poor quality of the input data. The combination of the wide variation in human writing styles, and the imperfect hardware devices used to capture said writing means that the input data for the same character can be very different from person to person - and that is assuming they are using the same input device. In order to deal with this problem, input data is often pre-processed before it is passed to a recognition algorithm. In this report, I aim to examine various techniques from the area of normalisation pre-processing, and also to examine the use of multiple reference templates for recognition - a task that falls between the pre-processing and recognition algorithm steps.

I will consider these techniques based on the two main requirements of a character recognition system - accuracy and processing time. For a character recognition system to seem 'natural' to a user, the system must achieve above 95% accuracy[1, 2]. However, the system must also process the characters within a short time frame of the writer finishing them - short enough that it does not begin to 'fall behind' if a user writes many characters at once. Therefore, all of the methods examined in this report will be tested for speed, based on the pure CPU time required to compute them. No work will be done here in determining how to define the required speed - all comparisons and discussions are done relative to the speed of base system provided.

Four techniques from the area of normalisation will be considered, all of which are common to most on-line character recognition systems[3-11]. These are mean normalisation, scale normalisation, slant normalisation and smoothing. (Fig.1.) It is important to note that segmentation of letters, another common pre-processing technique[3-5,9-11], is not required here as the characters are being delivered in an isolated fashion.

As mentioned above, I will also consider the technique of multi-templating; specifically the task of determining the number of reference templates to use and how best to select these reference templates. While it is expected that the use of multiple reference templates will increase the accuracy of the system, they also introduce a significant processing overhead. This means the selection of reference templates must be examined closely in order to keep the ratio of accuracy to computation time at a reasonable level.

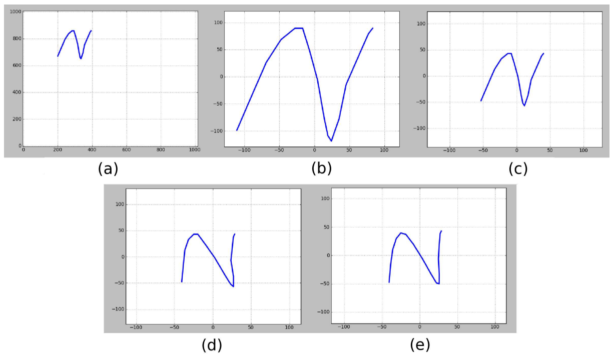

Figure 1: *Example character after: (a) No pre-processing; (b) Mean pre-processing; (c) Scale pre-processing; (d) Slant pre-processing; (e) Smoothing. (Note the change of axis scale between (a) and (b).)*

## 2. Experimentation

### 2.1. Overview

The base system used for this project was a Dynamic Time Warping (DTW) based classifier. The input data was a version of the MIT UNIPEN handwriting database, converted into the IIPL file format for simplicity. The system is a single reference template system, with the reference writers pre-chosen manually (s013 - s017).

All experiments were carried out using 5-fold cross validation. The set of writers were split into 5 sets, based on their remainder when divided by 5 - 0, 1, 2, 3 and 4. Experiments were carried out separately on the digit, lower case and upper case sets of characters, and independent results are given for each experiment. In each experiment, accuracy and CPU time were monitored - an average and standard deviation for both is given in the results table for the experiment.

An analysis of the accuracy and computation time of the base system can be found in Table 1.

|  | Avg Acc | Std Dev of Acc | Avg Time | Std Dev of Time |
|---|---|---|---|---|
| Digits | 50.58% | 5.213 | 0.505s | 0.021 |
| Lowercase | 40.41% | 4.832 | 1.748s | 0.030 |
| Uppercase | 48.11% | 4.343 | 1.777s | 0.039 |

Table 1: *Base system results.*

## 2.2. Normalisation

### *2.2.1. Mean Normalisation*

Mean normalisation solves the problem of variance in the position of the input characters. This is done in order to remove the necessity for the user to enter all of the input in a small set box. The normalisation is achieved by translating each character to be centred at the point (0,0), using the following formula (where N is the number of coordinates in the character data):

$$(x'_d, y'_d) = (x_d, y_d) - \frac{1}{N}\sum_{i=0}^{N}(x_i, y_i) \tag{1}$$

In order to gauge the effectiveness of mean normalisation, pre-analysis was run on the input data. The mean for each character was calculated and inserted into two histograms of x and y values for the entire data set. Characters were analysed in their separate classes - digits, lowercase and uppercase. The resultant graphs can be found in Fig 2. As would be expected, the results are Gaussian in nature, and fairly wide spread. This indicates that applying mean normalisation will be reasonably effective in increasing the accuracy of the system, as it would bring all of the means to a single point rather than the spread out nature that can be observed in the graphs. In terms of the computation time, the calculations involved in mean normalisation are fairly simple (one pass through the data to find the mean, and another to correct the points), and so it is expected that the CPU time will not rise by any significant percentage.

|  | Avg Acc | Std Dev of Acc | Avg Time | Std Dev of Time |
|---|---|---|---|---|
| Digits | 74.76% | 6.995 | 0.477s | 0.011 |
| Lowercase | 66.11% | 4.977 | 1.704s | 0.014 |
| Uppercase | 65.34% | 4.655 | 1.660s | 0.086 |

Table 2: *Mean normalised results.*

### *2.2.2. Scale Normalisation*

Scale normalisation attempts to correct the different sizes of characters that may be present in the input, by resizing all of the characters to fit within a pre-defined square bounding box, of length 150 pixels. The coordinate points were re-sized using the following algorithm (where H and W are the height and width of the character):

$$(x'_d, y'_d) = (x_d, y_d) * \frac{150}{\max{(H, W)}} \tag{2}$$

As with mean normalisation, pre-analysis was carried out on the input data - analysing the bounding box size for each character and storing it in a histogram. The results are presented in Fig 3. As before, the graphs are mostly Gaussian in nature, although less so than with the mean analysis. The scale normalisation Gaussians are also thinner, with a standard deviation of less than 25. This would imply that there would be reasonable increase in accuracy when applying scale normalisation (and again for a reasonably small CPU cost due to the simplicity of the calculations), although probably not as high as might be expected from mean normalisation. However, it is important to note that since the inputs are not mean normalised, the scaling may take them out of proportion to each other and thus lower this expected increase in accuracy.

(Note that the size of 150 for the bounding box was chosen based on the pre-analysis, although some experimentation showed that the size of the box was largely unimportant as long as it was not small enough to crowd the points - usually around 25 to 50 pixels long.)

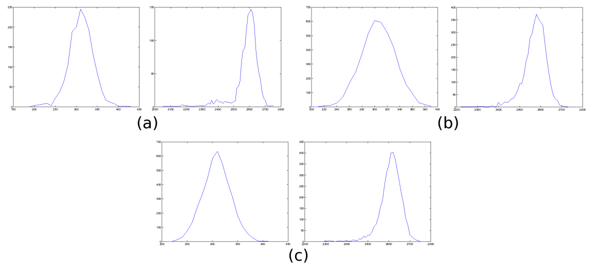

Figure 2: *Mean analysis graphs (both x and y axes) for: (a) digits; (b) lowercase; (c) uppercase characters.*

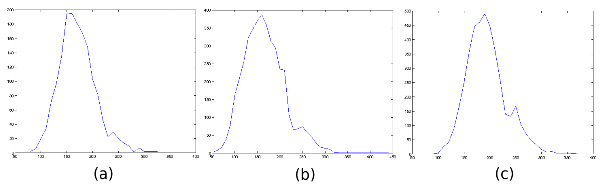

Figure 3: *Scale analysis graphs (both x and y axes) for: (a) digits; (b) lowercase; (c) uppercase characters.*

|  | Avg Acc | Std Dev of Acc | Avg Time | Std Dev of Time |
|---|---|---|---|---|
| Digits | 14.96% | 3.016 | 0.485s | 0.023 |
| Lowercase | 10.16% | 2.960 | 1.696s | 0.011 |
| Uppercase | 8.84% | 2.792 | 1.652s | 0.039 |

Table 3: *Scale normalised results.*

### *2.2.3. Slant Normalisation*

Slant normalisation (or 'deskewing') attempts to correct the natural slant that most humans write with. This is a far more complex operation than mean or scale normalisation, due to the difficulty of estimating the degree of slant that exists in the character. Normally deskewing is performed on an entire word - with individual characters there is very little data available to estimate the slant angle.

Therefore to calculate the slant angle I made the generalisation that if you deskew a character, its width will decrease (see Fig 1 for a visual example of this reduction in width.) I then used the following formula to slant each input character from -30 to +30 degrees, in 5 degree steps. The resultant data with the smallest width was then taken as the correctly deskewed character. (Note that N is the number of coordinates in the character data.)

$$x'_d = x_d - (y_d - \frac{1}{N}\sum_{i=0}^{N} y_i) * tan(\theta) \tag{3}$$

Due to the difficulty with calculating the slant angle, it is problematic to analyse the input data with regards to it. However, random sampling of 20 characters from each of the classes showed little visible slanting within the input data. Therefore, it is probable that the slant correction algorithm will not improve the accuracy of the system by a significant percentage, if at all. It should be slightly more computationally expensive than mean or scale normalisation, since deskewing each character requires 12 applications of the slanting algorithm.

| | Avg Acc | Std Dev of Acc | Avg Time | Std Dev of Time |
|---|---|---|---|---|
| Digits | 51.35% | 7.741 | 0.502s | 0.024 |
| Lowercase | 40.24% | 4.950 | 1.583s | 0.062 |
| Uppercase | 46.80% | 3.526 | 1.723s | 0.049 |

Table 4: *Slant normalised results.*

### 2.2.4. Smoothing

The combination of natural hand movements and the flaws in the hardware devices used to capture the characters can lead to imperfect input data[4, 8, 9, 12]. There will often be small 'kinks' in the data during straight line sections, and curved sections will often be quite angular in nature due to the low sampling rate of the input device.

To attempt to correct these problems, the data can be smoothed. There are many ways to smooth data - for example using a Gaussian or non-Gaussian low-pass filter[7] (or 'mask'), using piecewise-linear curve-fitting[6], or using a backwards averaging filter[8, 10] (which only relies on already smoothed points in order to smooth a new point). Experimentation with the Gaussian, non-Gaussian and backwards averaging filters showed that they all performed at roughly the same level for the input data, so the backwards averaging filter was chosen, with a small modification of the formula presented in the report upon which it is based. I found by experimentation that the 3:1 ratio in favour of the previously smoothed point was far too strong for the sparsely sampled data set, and that it was actually better to have the opposite ratio to the one Groner presents - a 3:1 ratio in favour of the about to be smoothed point.

$$(x'_d, y'_d) = \frac{1}{4} * (x_{d-1}, y_{d-1}) + \frac{3}{4} * (x_d, y_d) \tag{4}$$

As with slant normalisation, it is difficult to pre-analyse the input data to determine how smooth it is. It is also hard to use a random sampling to analyse the data as smoothing is a far more subtle feature. However, due to the sparsely sampled nature of the input characters, I expect that smoothing will not improve the accuracy of the system by a significant percentage. Smoothing using the chosen algorithm is also computationally inexpensive, so I do not expect a large increase in the time taken either.

| | Avg Acc | Std Dev of Acc | Avg Time | Std Dev of Time |
|---|---|---|---|---|
| Digits | 48.95% | 5.981 | 0.486s | 0.042 |
| Lowercase | 39.35% | 5.387 | 1.717s | 0.046 |
| Uppercase | 47.16% | 4.327 | 1.693s | 0.049 |

Table 5: *Smooth normalised results.*

## 2.3. Multi-templating

In a single template character recognition system, such as our base system, the choice of reference template is directly linked to how accurate the system is. This is problematic for two main reasons. Firstly, if the chosen template is a poor quality sample, the system will have poor accuracy on the inputs. Secondly, even if the reference template is a good sample, the system may still have poor accuracy. This is due to the differences in writing style discussed in the introduction - when based on a single reference, the system will perform well for those characters by writers with a similar style to the reference, and poorly for characters by writers with a different style. As an example, Fig 4 shows the letter 'f' written by four different writers. Only the characters written by writers (a) and (b) look remotely alike and so if a system is designed using one of these writers as the sole reference, it might struggle to recognise the other 'f's.

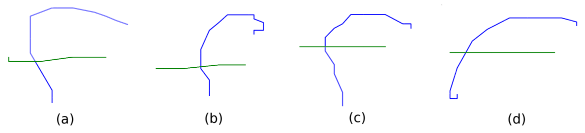

Figure 4: *The character 'f' written by four different writers.*

One way to overcome this dependency is to use multiple templates as references. Input points are classified against each reference pattern, then a k-nearest neighbours algorithm is applied to choose the closest match - usually with k being 1 (i.e. just a nearest neighbour search). The question then becomes how to select the reference templates that will be used, and how many such templates to select. In this report I will examine three methods - an 'everything-else' selection method, a 'best-single' selection method and a random-choice selection method.

Note that for every method within this section, the time taken to run the knn algorithm was included in the recognition CPU time, as it is an essential part of the multi-template system.

### 2.3.1. 'Everything-else' Selection

The simplest of the multi-template methods, 'everything else' selection works by taking the data that is not being used as the input, and applying it as the reference patterns. This can either be done at a high level stage - splitting the data into two groups of "input" and "reference" - or at a lower level, where each input writer is compared with all other writers in the data set. In this report I will only examine the higher level method.

For each set of input data (0 to 4), the reference writers were taken as being the set of writers that were not being used by the input data. It is expected that the accuracy of the system will increase by a high percentage as it is unlikely that any one test set would have a writer with a style that is not similar to any of the other writers in the training data, and using a large number of writers will also avoid the problem of poor quality references However, it is also expected that this will be very computationally expensive in comparison to the base system, as each writer will be compared to around 130 writers instead of just 1.

| | Avg Acc | Std Dev of Acc | Avg Time | Std Dev of Time |
|---|---|---|---|---|
| Digits | 93.84% | 1.993 | 9.121s | 0.546 |
| Lowercase | 86.70% | 2.043 | 50.804s | 1.658 |
| Uppercase | 90.93% | 1.945 | 52.311s | 3.277 |

Table 6: *'Everything else' multi-templating results.*

*2.3.2. 'Best-single' Selection*

This method assumes that the templates that work best for single reference template recognition will work even better if combined. Therefore, the top eight reference templates for each input set are calculated and used for the multi-template recognition. Note that when determining the top reference patterns, the average accuracy of a writer on *all* of the classes (digits/lowercase/ uppercase) is taken, since it is assumed that any system would eventually need to analyse all of the character sets. Higher accuracy could probably be achieved by splitting the analysis into the three classes. The value of 8 for the number of writers was determined using experimentation - values lower than this lost accuracy without being much faster computationally, whilst higher values began to have a larger computational cost increase than gain in accuracy.

One potential flaw in this method is the assumption that the chosen templates will 'cover' for each other - that is, the reference writers chosen will not be similar in style. If they are then there would not be as high an accuracy increase as might be expected, as all of the reference templates would simply make the same mistakes - for example, perhaps they all mistake 'e's for 'c's. Despite this, I still expect that this method will show a substantial increase in accuracy over the base system, although less than the 'everything-else' method. I also expect the CPU time to be largely reduced from the 'everything-else' method, as this only requires 8 templates as opposed to around 130.

| | Avg Acc | Std Dev of Acc | Avg Time | Std Dev of Time |
|---|---|---|---|---|
| Digits | 84.18% | 3.814 | 1.175s | 0.101 |
| Lowercase | 73.72% | 2.209 | 5.436s | 0.330 |
| Uppercase | 78.85% | 3.265 | 5.651s | 0.223 |

Table 7: *Pre-selection multi-templating results.*

*2.3.3. Random Selection*

Another potential method is simply to randomly choose 8 reference templates each time the recognition is run. This method relies on the assumption that multi-templating is overall a good step, and that any 'bad' choices made by the random chooser are likely to be mitigated by the other choices. As in the pre-selection case, eight writers are chosen to be the reference templates.

Due to this being a random procedure, the experiment was repeated ten times and the outcomes averaged to give a reasonable overall average of the method. I expect that the overall outcome should again be more accurate than the base system, but potentially slightly less accurate than the pre-selection method due to the probability of choosing 'bad' writers. The computation time should be similar to the pre-selection method, as the random choosing of numbers is not computationally significant.

| | Avg Acc | Std Dev of Acc | Avg Time | Std Dev of Time |
|---|---|---|---|---|
| Digits | 79.48% | 4.732 | 1.169s | 0.126 |
| Lowercase | 67.29% | 3.202 | 5.555s | 0.344 |
| Uppercase | 74.80% | 3.054 | 5.615s | 0.352 |

Table 8: *Random choosing multi-templating results.*

### 2.4. Combined

In order to build the 'best' system (in terms of accuracy with regards to CPU time), I combined the mean and scale normalisation techniques with the pre-selection multi-templating method. Although scale normalisation performed poorly when used as the sole normalisation technique, when combined with the mean normalisation it gave an accuracy boost of around 2-3%. I expect that the resultant combined system will show high accuracy, with a reasonable increase in CPU time over the base system.

| | Avg Acc | Std Dev of Acc | Avg Time | Std Dev of Time |
|---|---|---|---|---|
| Digits | 91.96% | 2.109 | 1.033s | 0.028 |
| Lowercase | 84.43% | 2.488 | 4.533s | 0.110 |
| Uppercase | 89.14% | 2.461 | 4.635s | 0.063 |

Table 9: *Results for the 'best' system - mean and scale normalisation with pre-selection multi-templating.*

## 3. Discussion

### 3.1. Normalisation

As expected, the mean normalisation showed a great increase in accuracy over the base system - an average of a 22.3% increase in accuracy over the three classes (Table 2). It also was computationally insignificant - the CPU time actually decreased by an average of 60 milliseconds, most likely due to simple fluctuations in processing speed.

The scale normalisation did not perform as expected, showing a large *decrease* in the accuracy of the system of around 30% per class (Table 3). I theorise that this was caused by the predicted issue of characters being taken out of proportion to each other by the scaling, with a far greater negative effect than predicted. More research would have to be done to confirm this. Note that if both mean and scale normalisation are applied, the accuracy increases over just mean normalisation, by roughly 2.5-3% in each class - lending weight to the prediction that scale normalisation is a positive step as long as the input data is mean normalised first.

As predicted, the slant normalisation did not affect the system significantly. In terms of accuracy there was about a 1% increase for digits, no significant change for lowercase letters, and a 2% decrease for uppercase letters (Table 4). As discussed above, this was expected due to the fairly straight data set that was involved in this project. This decrease in accuracy was most likely due to the problem of 'guessing' the slant angle correctly.

Finally, smoothing also produced results that were close to what was expected. There was on average a 1% decrease in accuracy when using smoothing (Table 5). This was likely due to the sparseness of the input set, which makes smoothing correctly difficult. One solution that could be researched in the future would be to re-sample the points to increase the density of the character data[7], and to potentially combine this with a wider ranged filter than the backwards averaging filter used in this report, which only examines the previous point when smoothing a point.

Overall, the normalisation results were as expected, apart from the poor performance of the scaling normalisation. However, applying the scaling normalisation after mean normalisation showed a small increase over just using mean normalisation, therefore confirming that, once the inputs have been normalised in regards to their mean, scaling can be an effective normalisation technique.

### 3.2. Multi-templating

The 'everything-else' selection method for multi-template recognition performed as expected. It showed a large percentage increase on the base system - on average 44.12% - but was also far more computationally expensive, taking between 18 and 30 times longer (for Digits and Uppercase characters respectively) to run. This makes it an excellent step for accurate recognition, but shows that it has a very high computational price for this accuracy. (Table 6.)

'Best-single' selection also performed as expected, showing a 32.47 percentage increase over the base system on average (Table 7). It also took only around 2-3 times as long in terms of CPU time, for all three classes. This means that it gave almost 75% of the increase in accuracy that the 'everything-else' selection gave, in only 8-13% of the CPU time that the previous method took. It is likely that in a real world system, this method would be far preferable to the 'everything-else' method.

The random selection method performed almost as well as the 'best-single' method, with a 27.5% increase in accuracy over the base system (Table 8). Like the 'best-single' method it also took around 2-3 times the CPU time required by the base system, which was expected since they both calculate the recognition accuracy using 8 reference templates. Therefore, the random selection method could also perform reasonably well in a real world situation - however the random nature of the method might cause problems and should therefore be avoided. Since the previous method is deterministic, has about the same computational complexity and has a slightly higher accuracy, it would be a far better choice for a real world system.

The multi-templating methods worked almost exactly as theorised, showing large increases in accuracy at a varying cost of CPU time. One topic that is not addressed in this report is how the multi-templating systems translate to a real world system, where there is no test data set to use as the 'everything-else' reference or to calculate the 'best single' reference for - input characters arrive sequentially with an unknown gap of time between them. In this real-world situation the 'everything-else' method would be modelled using a large and diverse set of characters as the pattern references for all new input characters, the 'best-single' templates would be calculated based on a similar test set and chosen from that, and the random writers would be chosen similarly to the 'best-single'.

### 3.3. Conclusion

In this report I have presented two main areas important to on-line character recognition (normalisation and multi-template reference selection) and examined key techniques within these areas and how they performed on the MIT data set of characters. Of the normalisation techniques, mean and scale normalisation showed the largest improvement over the base system. When combined with the best multi-template method for accuracy with regards to CPU time (the 'best-single' selection method), I was able to reach an average accuracy of 88.5% at a reasonable CPU time - twice that of the base system, on average. (Table 9.)

Although the figures given by the systems created in this report are very close to the level of accuracy that human users demand (above 95%), it must be noted that the systems presented here all treat the three classes (digits, lowercase and uppercase characters) as discrete classes. No consideration was given here to the 'real-world' situation where all three sets (and potentially many more characters!) could be in the input data. Further research into this area would be necessary to fully examine the techniques presented here. In terms of future improvements to the accuracy of the system, there are both many more techniques in the areas of normalisation and reference selection that could be examined (for example, equi-distant re-sampling[7] or serif removal[4]), and also refinements that could be made to the techniques examined here - especially the slant correction and smoothing algorithms. Finally, a further study could be made into either the encoding of the input data or improvements to the Dynamic Time Warping algorithm, both of which are likely to yield potential accuracy improvements.

## 4. References

[1] LaLomia, M. J., "User Acceptance of Handwritten Recognition Accuracy", 1994

[2] CharacTell Advanced Character Recognition Technology, "Advanced Character Recognition, Technology and Performance Analysis White Paper", September 2003

[3] Paszkowski, B., Bieniecki, W., and Grabowski, S., "Preprocessing for Real-Time Handwritten Character Recognition"

[4] Loy, W. W., and Landau, I. D., "An On-Line Procedure for Recognition of Handprinted Alphanumeric Characters", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.PAMI-4, No. 4, July 1982

[5] McInerney, S., and Reilly, R. B., "Hybrid Multiplier/CORDIC Unit for Online Handwriting Recognition", 1999

[6] Ito, M. R., and Chui, T. L., " On-Line Computer Recognition of Proposed Standard ANSI(USASI) Handprinted Characters", Pattern Recognition, Vol. 10, 1978

[7] Namboodiri, A. M., and Jain, A. K., "Online Handwritten Script Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 26, No. 1, January 2004

[8] Arakawa, H., "On-Line Recognition of Handwritten Characters - Alphanumberics, Hiragana, Katakana, Kanji", Pattern Recognition, Vol. 16, 1983

[9] Brown, M. K., and Ganapathy, S., "Preprocessing Techniques for Cursive Script Word Recognition", Pattern Recognition, Vol. 16, 1983

[10] Groner, G. F., "Real-Time Recognition of Handprinted Text", 1966

[11] Tappert, C. C., Suen, C. Y, and Wakahara, T., "The State of the Art in On-Line Handwriting Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 8, August 1990

[12] Odaka, K., Arakawa, H., and Masuda, I., "On-Line Recognition of Handwritten Characters by Approximating Each Stroke with Several Points", IEE Transactions on Systems, Man and Cybernetics, Vol. SMC-12, No. 6, November/December 1982

[13] Burr, D. J., "Designing a Handwriting Reader" IEE Transactions on Pattern Analysis and Machine Intelligence, Vol.PAMI-5, No.5, September 1983

[14] Hanaki, S., Temma, T., and Yoshida, H., "An On-Line Character Recognition Aimed at a Substitution for a Billing Machine Keyboard", Pattern Recognition, Vol. 8, 1974

[15] Wilfong, G., Sinden, F., and Ruedisueli, L., "On-Line Recognition of Handwritten Symbols", IEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18, No. 9, September 1996