From Pixels to Features III: Frontiers in Handwriting Recognition
S. Impedovo and J.C. Simon (eds.)
© 1992 Elsevier Science Publishers B.V. All rights reserved.

Online recognition of unconstrained handprinting: a stroke-based system and its evaluation

T. Fujisaki, H.S.M. Beigi, C.C. Tappert, M. Ukelson and C.G. Wolf

IBM Watson Research Center, P.O. Box 704, Yorktown Heights, N.Y. 10598

Abstract

A recognize-then-segment recognizer of run-on (relatively unconstrained) handprinting uses a unified tablet-display to provide a paper-like computer interface. Whereas most handwriting recognition systems recognize characters, this one recognizes strokes then finds the best segmentation of strokes into characters. It classifies strokes, generates character hypotheses, and verifies hypotheses to estimate the optimal character sequence for each word. The system is implemented on an IBM workstation, accepts run-on characters written on a tablet, and performs recognition in real time. The system is evaluated from several aspects. A detailed analysis of recognition errors is given.

1. INTRODUCTION

The idea of using handwriting to interface with computers has attracted many people for many years. With recent advancements in device and packaging technology, a low-priced tablet-LCD, roughly the size and the weight of a book, has become a reality. On-line handwriting recognition is one of the key software technologies required to realize a very intelligent "paper-and-pencil-like" computer interface. An electronic tablet accurately captures x-y coordinate data of pen-tip movements, "electronic ink" gives the trace of the pen-tip on the screen surface of a unified LCD display, and recognition algorithms instantly convert the captured x-y coordinates into coded characters or symbols.

The "Paper-Like-Interface" project at the IBM Thomas J. Watson Research Center has aimed to realize such a sophisticated and natural application-generic interface. The current prototype, working on unified LCD-tablet devices and IBM workstations, supports the use of this interface for such applications as spread-sheet, host-terminal emulation, and music note editing [9], on AIX, OS/2, and Penpoint [1].

One of the essential features of a recognition system to be used in such an environment is that the system accurately recognizes relatively unconstrained handwriting. Figure 1 illustrates the range of types of handwriting for English. A combination of a "stroke segmentation process", grouping of strokes into character units, and a "shape identification process", shape identification of a character unit among candidate alphabets, are the two essential steps to perform recognition task for such writing. In Figure 1, handwriting types are listed in order of increasing difficulty of stroke segmentation required.

In the case of discrete letters written in boxes, the segmentation is trivial. This is because strokes are already segmented by writers when each letter is written in a box. In the case of spaced discrete writing, however, a machine procedure is required. Projection of X-Y coordinates of strokes onto the X-axis is one of the most often used techniques for grouping strokes into units corresponding to characters [5]. These units will then be sent to a character-shape identification process. Since the segmentation of strokes into character units is done prior to and independently of the character shape identification process, we call this a "segmentation-then-recognition strategy".

In the third type of writing, characters are run together, that is, strokes can touch or overlap one another. Since characters in run-on handwriting can not be easily or accurately separated one from another, the two processes of segmentation and shape identification of character units cannot be separated but should be heavily interrelated. In earlier methods of solving this problem of run-on characters [6, 7, 10], the process of checking many stroke combinations takes time and can be a bottleneck in real-time applications.

The approach presented here [3, 4] separates the recognition process into stroke recognition, stroke grouping (character recognition), and character shape validation. The improvement is achieved by swapping the order of stroke bundling (character segmentation) and shape recognition of strokes of the earlier approach. Stroke shape recognition is computation-intensive and proceeds as the strokes are entered. Compared to the earlier methods, early stroke classification provides information sufficient to greatly reduce the number of candidates required to be validated.

This paper briefly summarizes the search strategy of this stroke-based recognition system and then discusses the analysis of recognition errors and human writing errors in *run-on* handwriting.

2. SEGMENTATION AND RECOGNITION STRATEGY

2.1 Search Space and Hypothesis Generation

The segmentation of run-on writing is a difficult task. To handle this, the system adopted a *Generation and Test* paradigm which is known to be a very powerful problem solving technique in Artificial Intelligence. Arrival of *a stroke* from some external source triggers the *generator* to generate the most-likely hypothetical solutions. We will simply call this a *hypothesis*. The *tester* receives each hy-

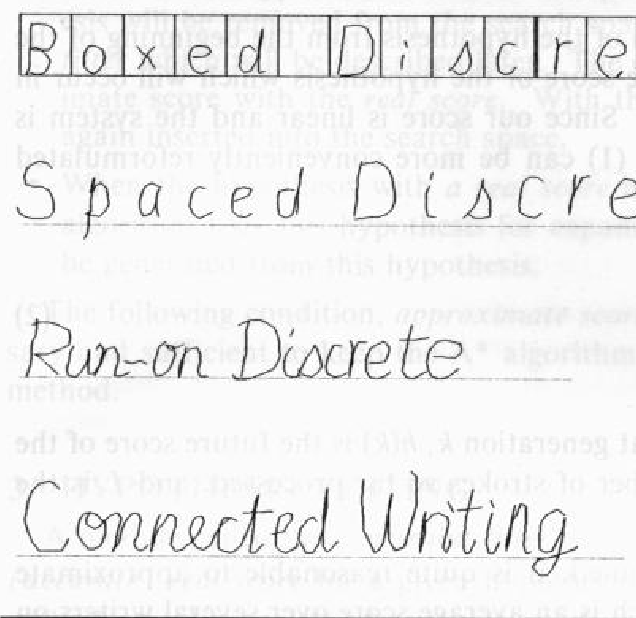


Figure 1. Types of English Handprinting

pothesis and tries to verify it. If it rejects the hypothesis, it will request the next hypothesis from the *generator*. This iteration continues until a hypothesis satisfies the *tester*. When a solution is found, the system is ready to accept the next stroke.

The first component of the system is the stroke-shape recognizer, which starts processing on receiving a stroke from the tablet. A stroke is matched to stroke prototypes that correspond to strokes of characters, such as *1/2 of A* (1st stroke of a 2-stroke *A*). An input stroke produces a list of stroke labels (each of which corresponds to a stroke prototype), and corresponding distance scores that indicate the degree of dissimilarity between the input stroke and the prototype. For example, a vertical-bar stroke could result in a list of labels such as *1/2 of B*, *1/2 of D*, *1/4 of E*, *1/3 of F*, *1/3 of H*, and *3/3 of H*. A sequence of input strokes produces a sequence of lists of labels. The system then generates hypothetical character paths, that is, sequences of stroke labels that are potentially meaningful sequences of character fragments. For example, a sequence of *1/2 of A*, *2/2 of A*, *1/2 of B*, *2/2 of B*, and *1/1 of C* is a valid hypothetical path corresponding to *ABC*.

2.2 Ordering of hypotheses

Since searching for the best solution by the generation-and-test method is a sequential process, it is essential that the hypotheses be generated and sent to the tester in order of plausibility.

The standard A* algorithm [8] ensures finding the most plausible hypothesis from a search space as described above. Associated with each hypothesis *h* is a cost function

$$F(h) = G(h) + H(h), \tag{1}$$

where $G(h)$ is the current score (cost) of the hypothesis from the beginning of the task until now and $H(h)$ is the future score of the hypothesis which will occur in the future until the end of the task. Since our score is linear and the system is synchronized with strokes, equation (1) can be more conveniently reformulated as:

$$F(h) = \sum_{j=i}^{n} g(j) + \sum_{j=n+1}^{L} h(j), \tag{2}$$

where $g(k)$ is the score of the stroke at generation k, $h(k)$ is the future score of the stroke at generation k, n is the number of strokes so far processed, and L is the total number of strokes in the input.

Since $h(k)$ is a score not yet obtained, it is quite reasonable to approximate $h(k)$ by a constant bias value B, which is an average score over several writers on various writings. Because L ***is unknown*** in an on-the-fly recognition environment, the cost function $\hat{F}(n)$ shown below can be used conveniently for our purpose. Note that this cost function ***does not rely on L.***

$$\hat{F}(n) = F(n) - \sum_{j=1}^{L} B = \sum_{j=1}^{n} g(j) + \sum_{j=n+1}^{L} h(j) - \sum_{j=1}^{L} B = \sum_{j=1}^{n} (g(j) - B) \tag{3}$$

2.3 Real and approximate scores

In order to make the A* algorithm work effectively, all hypotheses should have scores that can be compared uniformly regardless of the status of the hypotheses. Accurate scores from character matching, however, can be obtained only when all of the constituent strokes of the character are available. Thus, a character matching score is not available for those hypotheses that correspond to an intermediate state of constructing a character from multiple strokes. In order to handle this situation, we introduce approximate scores when real scores are not available.

Real score is a score provided from character matching. *Approximate score* is an approximation to the real stroke matching score and is given to every hypothesis. The computation of the *approximate score* is fast and is complete in the sense that every hypothesis regardless of its status can be given an *approximate score.* Because invocation of the character matcher is expensive, the *real score* is computed only for hypotheses which are chosen by the A* algorithm based on their *approximate score.*

The A* search algorithm handles hypotheses with *approximate score* and *real score* as follows:

- When a hypothesis with an *approximate score* is chosen as the best one, the score of this hypothesis will be re-computed. In order to do this, the hypothesis will be removed from the search space and will be sent to the *constraint filter* which will be described later. The constraint filter replaces the approximate score with the *real score*. With this new score, the hypothesis will be again inserted into the search space.
- When the hypothesis with *a real score* is chosen as the best hypothesis, A* algorithm uses this hypothesis for expanding the stack. Possibly, a child will be generated from this hypothesis.

The following condition, *approximate score is greater than real score*, is necessary and sufficient to keep the A* algorithm optimal with this two-stage scoring method.

3. HYPOTHESIS TESTING

A module called the *constraint filter* is used for two purposes. It computes the *(accurate) real score* for a given hypothesis and replaces the approximate score with the real score. It also serves as the *test*-component in *generate-and-test* paradigm. Different independent filters are cascaded to form this constraint filter. Some filters perform verification while others compute the real score.

When a hypothesis is selected as the best one by the A* search, the hypothesis is passed through the filters one at a time. If a hypothesis satisfies the constraint, the hypothesis will be passed to the next filter (*verified*). But if the hypothesis conflicts with the constraint, the hypothesis will be eliminated (*rejected*) from the search space. A filter for real score computation works similarly to re-compute the score and replace the score of the hypothesis.

3.1 Character shape verifier

Character shape verifier is one of the filters in the constraint filters. It *tests* a hypothesis and computes its *real score*. When the last stroke of the hypothesis ends a character, this shape verifier invokes the *elastic shape matcher* and obtains the character shape matching score for the last character of the hypothesis. For instance, if "E" is the label of the last four strokes of a hypothesis, these strokes are passed to the character shape matcher with the label "E". The matcher verifies the topological relationship of the four strokes by comparing to the information in the "E" template. The *approximate score of the last four strokes* will be replaced with the one obtained from the character shape matching.

A hypothesis will be rejected if the new character score is worse than a threshold which is a function of the approximate score, or if the new character score is worse than a *constant* times *the best score ever obtained* with the same strokes.

3.2 Language Models

The system uses four different language models in different stages of the recognition process. These models are Predictive Language Model (PLM), Template Language Constraint (TLC), Case Transition Model (CTM), and Dictionary. A brief discussion of each of these language models is given.

3.2.1 Predictive Language Model (PLM)

The Predictive Language Model is based on predictions of a character, given a context of N previous characters. This model uses a context of N=3 (up to three characters). This model uses a data base created by extracting the most frequent tetra-grams (sequences of four characters) from a 270,000-word dictionary. The large number of dictionary words generated a rather robust set of tetra-grams. The PLM data base includes the probability of occurrence of each of these tetra-grams.

The PLM is used in the search process to modify the search penalties and to reorder the search hypotheses. Therefore, the PLM is being used as a control variable, similar to other control variables like the stroke matching scores. The introduction of additional control variables necessitates balancing the contribution from each of the variables in the search. Use of the PLM allows the most probable hypotheses to be generated first, giving them a higher probability of pursuance in the search process.

3.2.2 Template Language Constraint (TLC)

The Template Language Constraint works as the search is in progress to restrict the generated hypotheses to those allowed by predefined templates. These templates have several forms: Picture Templates, Character List Templates, and Word List Templates.

A Picture Template fixes the format of a writing field. For example, if the writer is asked to enter a date, the recognition is forced to accept only the following format: dd/dd/dd where d stands for a digit. To do this, the TLM will eliminate any hypothesis in the search that does not fit the specified format.

A Character List Template allows only user-provided characters to be recognized. Therefore, a hypothesis is eliminated if it does not belong to the provided character list.

A Word List Template allows only user-provided words to be recognized. In this case, at any point in the search, hypothesized paths that are not present in any of the listed words are discarded.

3.2.3 Case Transition Model (CTM)

The Case Transition Model is similar to the TLC. For the CTM, the templates used are those dictated by the grammar of the language. For example, in English, with the exception of some special cases, there is no word that begins with a lowercase letter and has some or all of the following letters in uppercase. The CTM includes a data base of all possible transitions between adjacent characters, based on the previous context. Any hypothesized path that violates these grammatical rules is eliminated.

3.2.4 Dictionary and Case Post-processing

While the previous procedures concerned control variables in the search, this section describes postprocessing that is performed on the recognized sequence of characters.

A 100,000-word dictionary, compressed into a data base of about 100 kBytes, corrects words that are not in the dictionary by picking the dictionary word closest to the recognized character string. The measure of closeness is based on several heuristics, such as the percentage of incorrect characters in the recognized word (relative to the dictionary word) and the length difference between the recognized and dictionary words.

Once a dictionary word is picked to replace the recognized sequence of characters, several case-transition grammatical rules are used to estimate the most probable case (Upper or Lower) for each letter.

4. EXPERIMENTS

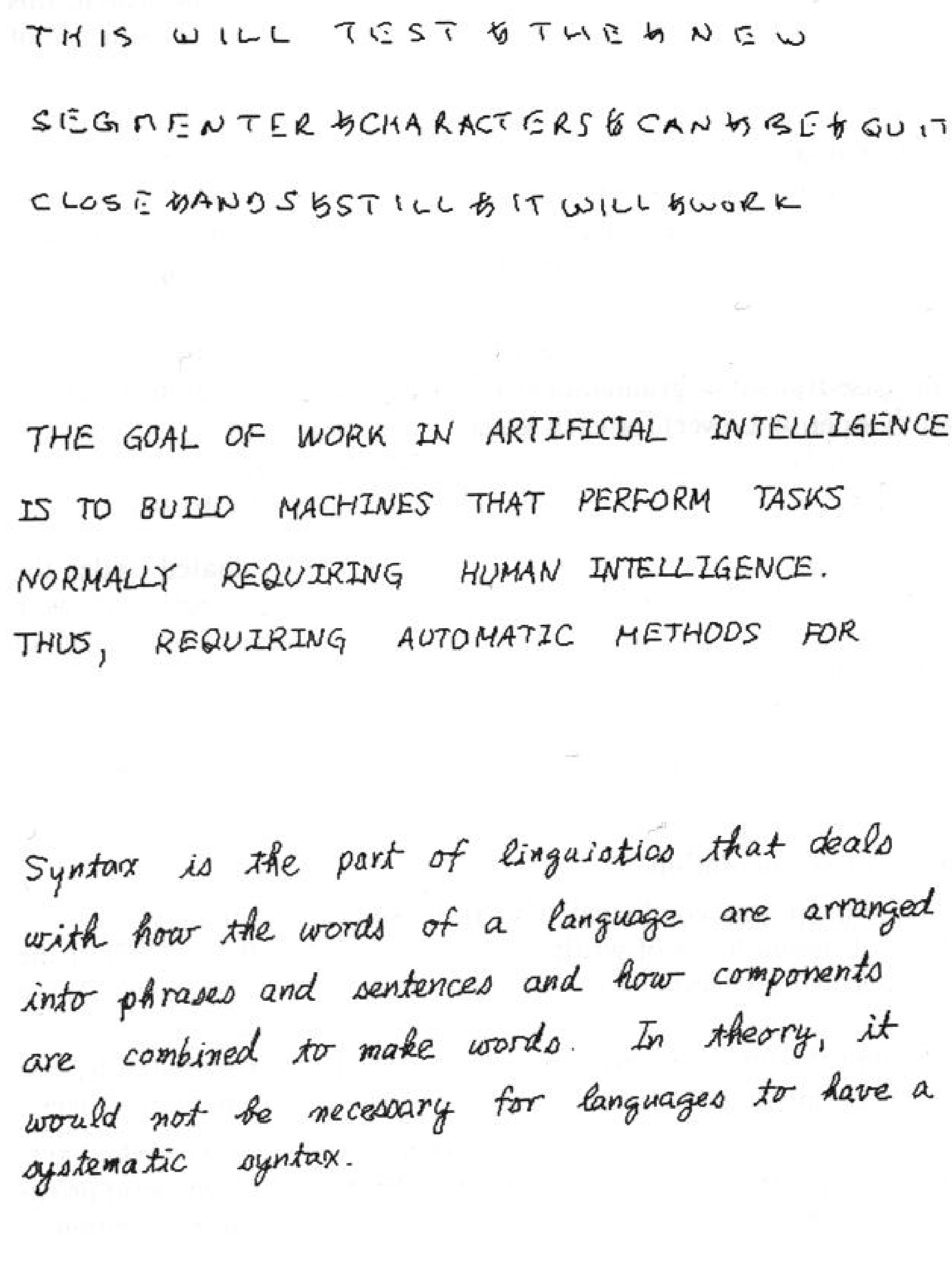
The recognition software system was written in C and evaluated under the IBM OS/2 and the IBM AIX operating systems. Data for these experiments were collected using a Photron FIOS-6440 LCD-tablet device, which consists of an electromagnetic tablet superimposed with a VGA-compatible LCD. Resolution of the tablet was 50 points/inch and the sampling rate was 70 points/second.

In these experiments, an 82-character alphabet was used: A-Z, a-z, 0-9, ,.?!:;'"()+-*/=><$%# .

4.1 Evaluation of segmentation

The first experiment was conducted with three subjects familiar with tablet digitizers. Three different levels of writings, as shown in (A), (B), and (C) of the figure 2, were collected and tested with different segmentation strategies. These three cases differ in the degree of character separation. Case (A) is spaced-discrete writing and the other two are run-on discrete writing. Compared to case (B), case (C) uses script-shaped characters and has more overlapping of strokes.

Training consisted of printing each character in a writing area which had vertical guide marks separating adjacent characters. Each subject wrote approximately 240 training characters to generate the stroke templates for recognition.


THIS WILL TEST THE NEW
SEGMENTER CHARACTERS CAN BE QUITE
CLOSE AND STILL IT WILL WORK

THE GOAL OF WORK IN ARTIFICIAL INTELLIGENCE
IS TO BUILD MACHINES THAT PERFORM TASKS
NORMALLY REQUIRING HUMAN INTELLIGENCE.
THUS, REQUIRING AUTOMATIC METHODS FOR

Syntax is the part of linguistics that deals
with how the words of a language are arranged
into phrases and sentences and how components
are combined to make words. In theory, it
would not be necessary for languages to have a
systematic syntax.

Figure 2. Handwriting Samples

Character errors in run-on writing can be categorized as one-to-one substitutions, in which one character is substituted for another, and multi-character substitutions, in which one character is substituted for *n* characters, *n* characters are substituted for one character, or *n* characters are substituted for *m* characters. These multi-character substitution errors will be called *segmentation errors* since these errors occur when strokes are incorrectly grouped into characters. In one-to-one substitution errors, there is nothing wrong with the segmentation, and we simply call these *substitution errors*.

In order to measure the capability of segmentation, the three handwriting samples (A, B, and C) were recognized by three different segmentation methods. Table 1 lists the segmentation error rates (in percent), that is, the number of segmentation errors over the total number of characters.

In this experiment, method-1 corresponds to the segmentation strategy that uses x-axis projected gap between strokes [5]. Characters are segmented where the gap between shadows of two strokes exceeds a predefined writer-independent threshold.

Method-2 corresponds to the segmentation strategy that uses actual two-dimensional distances between strokes. The distance between two strokes is defined as the minimum of the distances between constituent points of each stroke. Characters are segmented when the distance between two strokes exceeds a predefined writer-independent threshold [2].

Method-3 corresponds to the proposed "recognition-then-segmentation" strategy based on generation-and-test. In this method, the segmentation decision involves the analysis of stroke shapes and verification of the character shape.

Method	Sample A	Sample B	Sample C
(1) X-projection segmentation	6.2	75.4	87.6
(2) Stroke-distance segmentation	2.5	55.8	77.2
(3) Recognition then segmentation	2.5	0.0	0.0

Table 1. Error Rates (percent) for Segmentation Strategies

This result shows that the recognition-then-segmentation strategy (method-3) performs well in various ranges of writing, whereas the other segmentation methods (method-1 and method-2) do not work for writing without sufficient separation between characters.

4.2 Evaluation of recognition

Another experiment was conducted with 12 subjects not familiar with tablets or handwriting recognition systems. At the start of the first session, subjects were given suggestions for obtaining successful recognition; they were told to try to make similar characters, such as O and 0, differently, and not to connect adjacent characters as they would in cursive writing.

Each subject wrote approximately 1341 characters to generate the stroke template for the recognition. Training consisted of printing each character in a writing area which had vertical guide marks separating adjacent characters. The test was done on 1544 characters of run-on writing. For the test, characters were written on base lines without guide marks. More detail of the test procedure is given in [11].

Average character-recognition accuracies on twelve writers for run-on writing are shown in Tables 2 and 3. (These results were obtained during the revision of the paper by re-running the data through an improved version of the system.) Table 2 shows the percentage of correctly recognized characters for the alphabet consisting of all 82 characters and for the restricted alphabets of uppercase only and digit only (both input and recognized output character is restricted). Table 3 shows the percentage of correctly recognized characters of different types in the full 82-character alphabet (input restricted, recognized output unrestricted). It should be noted that these numbers do reflect character segmentation errors but not word segmentation errors. For example, if the phrase "We went out" was recognized as "We wen to ut," all characters would be scored as correct.

82-character	Uppercase	Digits
90.7	96.3	97.2

Table 2. Restricted-alphabet Recognition (percent correct)

Uppercase	Lowercase	Digits	Special
97.2	89.0	89.6	82.1

Table 3. Full alphabet: accuracy within subset (percent correct)

The following observations were made from this character recognition evaluation.

1. Recognition accuracy tends to be a function of alphabet size. Here, recognition accuracy of the restricted alphabets (digits and uppercase) is clearly better than that of the full 82-character alphabet (Table 2).
2. Templates generated from writing made between guide marks can effectively be used for recognition of run-on handwriting written without guide marks.
3. Special characters are the most difficult to recognize (Table 3). Highly confusable were: two single quotes and a double quote, comma and single quote, semicolon and single quote followed by comma, + and t, + and f, +

and X, colon and exclamation point, * and x, * and ′=t, 1 and /, l and /, < and L, $ and S.

4. Lower case letters appear to be difficult to recognize because they have more variations than upper case letters. Also, it was observed that some writers have difficulty writing lower-case characters discretely, especially in run-on writing.

For word segmentation, the system uses a simple segmentation algorithm previously called method-1. It uses shadows of consecutive strokes projected on to the x-axis. Words are segmented when the gap between two shadows is greater than a predefined writer-independent threshold. Because of this simple segmentation strategy, accuracy of word segmentation was only 88.3%.

In order to understand the reason for recognition errors in detail, approximately one eighth of 2,200 recognition errors were chosen randomly and analyzed by hand. In order to obtain objective categorization, each error was inspected by two individuals in parallel. All different assignments of error categories were resolved in face-to-face meetings. Seven categories of error causes were introduced.

- Tablet problem - misrecognitions caused by tablet or pen malfunctions: penskip, excessive stroke hooks, extra points, and stroke tapping.
- Writer's error - misrecognitions caused by the writer; for example, the writer left the writing area blank or wrote symbols other than those requested.
- Run-on writing habits - misrecognitions caused by writing habits which this system could not handle. "Connected strokes" (adjacent characters are connected) and "excessive delayed strokes" (strokes written later to complete "t", "i". etc.) are the causes of misrecognition in this category. Connected strokes were observed more frequently (60%) than excessive delayed strokes.
- New variation of shape - misrecognition due to the introduction of new shape at recognition time. This only causes errors when the new shape significantly differs from the ones given during training.
- Segmentation - misrecognition due to the invalid grouping of strokes into characters, causing multi-character substitution errors.
- Case confusion - one-to-one character confusion errors between uppercase and lowercase characters such as "s" and "S"; "x" and "X"; and "c" and "C".
- Shape confusion - one-to-one character confusion errors from other than the above causes; for example, "2" and "Z", "t" and "+", "V" and "U".

ERROR TYPE	PERCENT
Tablet problem	2.6
Writer's error	3.9
Run-on writing habits	6.4
New variations of shape	16.2
Segmentation	16.6
Case confusion	21.1
Shape confusion	33.2
TOTAL	100.0

Table 4. Types of Run-on Errors

Shape confusion was further classified into two subcategories: confusable even for human testers, 11.4%, and human can discriminate correctly but machine could not, 21.8%.

Confusions among *O*, *0*, and *o* accounted for 8% of the total error; among *5*, *S*, *s*, and *$(dollar)* for 6%; and among *double quote, single quote, comma, period,* and *i* for 5% of the total error. Confusion is also high within the characters *t*, *T*, + and *1*, *l*, *I*, */*. Eliminating confusions among the ten confusable characters (*0, t, i, S, O, double quote, 1, 5, p, quote*) would remove 32% of total recognition errors.

Use of linguistic constraints is essential to reduce errors for some categories. Approximately two third of case confusion errors were removed by using the character type transition model described previously. A 4.6% accuracy improvement was obtained over the number in Table 4 by using character tri-gram linguistic constraint, as described previously.

Our run-on recognition system assumes that users spend a certain amount of time for enrollment (training). It is assumed that all variations of writing characters can be captured during this enrollment. Errors in category of *new variations of shape*, however, indicated that new variations of writing characters can occur during recognition time.

The following is a breakdown of the error category "new variations of shape" (16.2%).

1. shape is not new, differs in stroke order - 0.37%
2. shape is not new, differs in stroke direction - 0.87%
3. differs in stroke number, shape may be new. - 5.67%
4. same stroke number, but shape is new - 9.29%

This break-down, together with the shape confusion errors (Table 4), indicates the necessity of training from run-on writing instead from boxed-discrete writing. Training in the context of use after a recognition error may also be helpful.

It is also observed from the break-down of new variation errors that stroke orders and stroke directions of characters are quite stable. Efforts to support flexible stroke order and direction in characters were not successful.

The category of *run-on writing habits* reveals that 3.9% of errors can be reduced by relaxing the constraint on connected strokes and 2.5% of errors can be reduced by relaxing the constraint on delayed stroke. We do not support connected strokes. We also have a limit for delayed strokes (e.g. crossing of *t*) to complete a character; strokes cannot back up more than 4 characters for finishing previously written characters. The initial instructions were not sufficient to eliminate such habits in run-on writing.

4.3 Effect of training

An adaptive capability is at the heart of the design of the system. This stems from our belief that the system should adapt to the user rather than force users to adapt to the system. This adaptive capability yields a system with recognition accuracy that improves over time. We believe that this capability is key for user acceptance and satisfaction. Measurement was conducted with the same 12 subjects to observe the accuracy evolution over time. The accuracy of run-on recognition was measured after initial training as well as after training with additional characters. Figure 3 shows how accuracy improved through training. The X-axis shows the minimum number of training samples for a character at the time of measurement. No linguistic constraints are used in this measurement. The figure indicates that accuracy increased between a minimum of 2 and 3 samples per character and then levelled off.

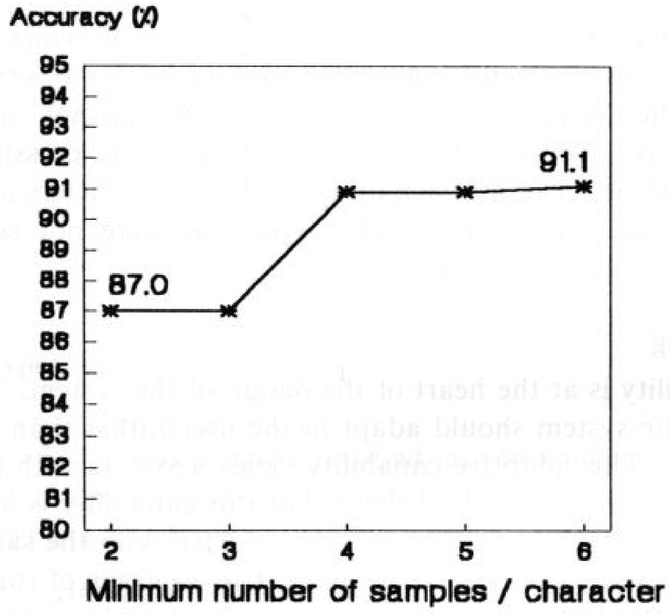


Figure 3. Effect of training

4.4 Effect of Language Models

The Predictive Language Model (PLM) controls the search to increase the chances of hypotheses most frequent in the language to be expanded. This is particularly important for writer-independent recognition, where the shape matching information for the character is not always reliable. For The speed of writer-independent recognition is also increased by the PLM because the PLM expands only hypotheses that are allowed by the language and are of high probability. The PLM increases the accuracy of writer-independent recognition by about 1.5% and trained recognition by about 0.8% on the test data used here.

The Template Language Constraint (TLC) and the Case Transition Model (CTM) also highly reduce the number of hypotheses generated in the search, thus increasing the accuracy and speed of the system by lowering the rate of confusion. The effects of the TLC on recognition are similar to those seen in Table 2 by restricting the alphabet. No results are given for general TLC's since their contribution is highly context dependent and specific to the test conducted. However, Table 2 could be seen as generated through character list templates.

The dictionary increases both word and character accuracies, and is most suitable for usage with everyday text creation. The dictionary has been designed to be robust, so that careful writing can be recognized correctly even for words not in the dictionary. Word accuracy is typically increased from 5 to 20 percent and character accuracy from 0.5% to 2.0%.

As a whole, the above language models have been designed to ensure robustness and to increase the accuracy in all cases. Most require less time for their private over-head than the time saved in the recognition process by reducing confusion, etc.

5. DISCUSSION

From the experiments conducted, the recognize-then-segment strategy for the recognition of run-on handprinting appears promising. This unique strategy successfully sped up the recognition of run-on writing without degrading recognition accuracy. The improvement is achieved by separating and swapping the order of stroke bundling (character segmentation) and shape recognition (of strokes or characters) of the earlier approach. With the information obtained from the stroke recognition process, the number of hypothesized characters generated during the segmentation process was significantly reduced. We have also demonstrated that prototypes obtained from the discrete mode can be effectively used for runon recognition.

Acknowledgements

We thank Tom Chefalas and Joonki Kim for their work on the design and the implementation of the system.

6. REFERENCES

1 R. Carr and D. Shafer, *The Power of PENPOINT*, Addison Wesley, 1991.

2 A.S. Fox and C.C. Tappert, "On-line external word segmentation for handwriting recognition," *Proc. 3rd Int. Symposium on Handwriting and Computer Appl.*, July 1987.

3 T. Fujisaki, T.E. Chefalas, J. Kim, and C.C. Tappert, "Online recognizer for runon handprinted characters," *Proc. IEEE 10th Int. Conf. Pattern Recognition*, June 1990.

4 T. Fujisaki, T.E. Chefalas, J. Kim, C.C. Tappert, and C.G. Wolf, "Online Run-on Character Recognizer: Design and Performance," *J. of Pattern Recognition and Artificial Intelligence*, vol. 1, May 1991.

5 G.F. Groner, "Real-time recognition of handprinted text," *Proc. FJCC*, pp. 591-601, 1966.

6 H. Murase, "Online recognition of free-format Japanese handwritings," *Proc. 9th Int. Conf. Pattern Recognition*, pp. 1143-1147, November 1988.

7 H. Murase, T. Wakahara, and M. Umeda, "Online writing-box free character string recognition by candidate character lattice method," *Trans. Inst. Electron. Commun. Eng. Japan J68-D*, pp. 765-772, April 1985. (Japanese)

8 N.J. Nilsson, *Problem solving methods in artificial intelligence,*, McGraw-Hill, 1971.

9 J.R. Rhyne and C.G. Wolf, "Gestural interfaces for information processing applications," *IBM Research Report RC12179*, September 1986.

10 C.C. Tappert, Recognition system for run-on handwritten characters, United States Patent, 4,731,857, March 1988.

11 C.G. Wolf, A.R. Glasser, and T. Fujisaki, "An Evaluation of Recognition Accuracy for Discrete and Run-on Writing," *Proceedings of the Human Factors Society 35th Annual Meeting*, pp. 359-363, 1991. also IBM Research Report RC16946