Adaptation in On-line Recognition of Handwriting

Vuokko Vuori

18th January 1999

HELSINKI UNIVERSITY OF TECHNOLOGY **ABSTRACT OF MASTER'S THESIS**

Author and name of the thesis:

Vuokko Vuori
Adaptation in On-line Recognition of Handwriting

Date: 18th January 1999 **Number of pages:** 102

Department: Department of Engineering Physics and Mathematics
Professorship: Tik-61 Information Sciences

Supervisor: Professor Erkki Oja
Instructors: Docent Jari Kangas and Dr. Jorma Laaksonen

This work is related to on-line recognition of handwritten characters. The theoretical part of this work consists of a literature survey in which the essential background knowledge in the research field of handwriting recognition is reviewed and various recognition methods are described. The literature survey concentrates on the on-line recognition methods of isolated alphanumeric characters. Special attention is paid to those methods which are suitable for learning and adaptive systems.

As the practical part of the work, a learning on-line recognition system has been created and various adaptation strategies for it have been analyzed. The system is based on the k Nearest Neighbor rule. The classification of an input character is carried out by comparing it to stored prototype characters whose classes are known. The characters are compared by using various dissimilarity measures based on dynamic time warping (DTW). The system learns as the prototype set is modified during the run: prototypes are added, removed, and reshaped according to decision rules. The reshaping method is a modified version of Learning Vector Quantization (LVQ).

The handwritten characters used in the experiments were collected with a special tablet and pen. The characters were written one by one and recognized on-line. These recognition results were also used for analyzing the user's adaptation to the performance of the system. The experiments with preprocessing methods, dissimilarity measures, and adaptation strategies were carried out by re-recognizing the characters off-line.

Keywords:
pattern recognition, isolated handwritten characters, on-line recognition, intelligent user interface, adaptation, learning systems, user adaptation;
k Nearest Neighbor rule, Learning Vector Quantization LVQ, dynamic time warping DTW, elastic matching;

TEKNILLINEN KORKEAKOULU **DIPLOMITYÖN TIIVISTELMÄ**

Tekijä ja työn nimi:

Vuokko Vuori
Adaptaatio käsinkirjoitettujen merkkien reaaliaikaisessa tunnistuksessa

Päivämäärä: 18. tammikuuta 1999 **Sivumäärä:** 102

Osasto: Teknillisen fysiikan ja matematiikan osasto
Professuuri: Tik-61 Informaatiotekniikka

Valvoja: Professori Erkki Oja
Ohjaajat: Dosentti Jari Kangas ja TkT Jorma Laaksonen

Tässä työssä on tutkittu käsinkirjoitettujen merkkien reaaliaikaista tunnistusta. Työn alkuosa koostuu kirjallisuustutkimuksesta, jossa on esitelty käsinkirjoitetun tekstin tunnistukseen liittyviä ongelmia ja erilaisia, kirjallisuudesta löytyneitä tunnistusmenetelmiä. Kirjallisuustutkimuksessa on keskitytty menetelmiin, jotka soveltuvat yksittäisten, alfanumeeristen merkkien reaaliaikaiseen tunnistukseen. Erityistä huomiota on kiinnitetty menetelmiin, jotka pystyvät mukautumaan nopeasti käyttäjän käsialaan.

Työn keskeisin aihe oli kehittää käyttäjän käsialan oppiva tunnistusjärjestelmä ja tutkia erilaisia oppimismenetelmiä. Toteutettu tunnistusjärjestelmä perustuu k:n lähimmän naapurin menetelmään eli merkit tunnistetaan vertailemalla niitä tunnettuihin mallimerkkeihin eli prototyyppeihin. Merkkien vertailussa on käytetty etäisyysmäärittelyjä, jotka perustuvat *dynamic time warping* (DTW) -menetelmään. Järjestelmän oppiminen tapahtuu prototyyppijoukkoa muuntelemalla. Prototyyppejä joko poistetaan tai lisätään erilaisten tunnistustuloksiin perustuvien sääntöjen mukaisesti. Lisäksi prototyyppien muotoa muunnellaan oppivaan vektorikvantisaatioon (LVQ) perustuvalla menetelmällä.

Työssä tarvitut merkkinäytteet kerättiin erityisen kirjoitusalustan ja kynän avulla. Merkit tunnistettiin sitä mukaa kuin niitä kirjoitettiin. Näitä keräyksen aikana saatuja tunnistustuloksia käytettiin kayttäjän käsialan muutosten arvioimiseen. Erilaisia esikäsittelymenetelmiä, etäisyysmäärittelyjä ja oppimismenetelmiä tutkittiin tunnistamalla kerätyt merkit jälkikäteen uudelleen.

Avainsanat:
hahmontunnistus, yksittäiset käsinkirjoitetut merkit, reaaliaikainen tunnistus, älykäs käyttöliittymä, adaptaatio, oppivat järjestelmät, käyttäjän oppiminen;
k:n lähimmän naapurin menetelmä, oppiva vektorikvantisaatio LVQ, dynamic time warping DTW, elastinen sovitus;

Acknowledgments

The research work presented in this thesis has been carried out in the Laboratory of Computer and Information Science (CIS) in the Department of Computer Science and Engineering at Helsinki University of Technology. The CIS-laboratory has been a most enjoyable and inspiring working environment, for which I wish to thank all my co-workers.

I would like to thank Professor Erkki Oja for supervising my work and introducing me to the fascinating world of artificial intelligence. I also express all my gratitude to Professor Teuvo Kohonen who has enabled this work by providing the excellent resources for research. This work has been funded by the Technology Development Centre Finland (Tekes) as part of technology programme Adaptive and Intelligent Systems Applications.

I have been guided by Dr. Jorma Laaksonen from the CIS-laboratory and Docent Jari Kangas from Nokia Research Center (NRC). I wish to thank both of them for their insights and suggestions. I am truly grateful to Jorma for all the time he has spent in helping to solve my various technical problems and improving my English. The two other members of our research group, Mr. Matti Aksela from the CIS-laboratory and Mr. Jukka Yrjänäinen from NRC, have also been most supportive.

I also owe my thanks to all my friends, relatives, co-workers, and the other volunteers who have provided the handwritten characters used in the experiments, and especially to Mr. Jarmo Hurri, M.Sc., who organized the first data collection.

I thank my family for all the financial and immaterial support they have provided during my studies.

Otaniemi 17th December 1998

Vuokko Vuori

Contents

Chapter 1

Introduction

On-line automatic recognition of handwritten text has been an on-going research problem for four decades. It has been gaining more interest lately due to the increasing popularity of hand-held computers, digital notebooks and advanced cellular phones. Traditionally, man-machine communication has been based on keyboard and pointing devices. These methods can be very inconvenient when the machine is only slightly bigger or the same size as the human palm. A keyboard is very difficult to integrate in small devices and it usually determines the size of the whole apparatus. This is especially true when the number of the characters is very large as in Chinese and Japanese. A pointing device, for example a track ball or a pen, is insufficient or very slow when used alone in applications in which textual input is also desired.

Because of these problems, new methods for input have been developed, for example systems that recognize speech and handwriting. Because both are very natural ways to communicate, people can easily learn to use them. Unfortunately, these recognition tasks are not that easy for computers whose artificial intelligence is different from that of humans. Neither one of the recognition problems have been completely solved yet. Very high accuracy is needed for such system before they can be commonly accepted. Handwriting recognition is the more attractive input method, especially in noisy environment and when privacy is needed.

At first sight, handwriting recognition does not appear to be a difficult problem. A recognition system should just choose the correct character, usually the one that most resembles the written one, from a limited set of characters. Unfortunately, this approach faces a number of difficulties. The most prominent problem in handwriting recognition is the vast variation in personal writing styles. There are also differences in one person's writing style depending on the context. The writing style may also evolve with time or practice. In addition, the mood of the writer and the writing situation can have an effect on style.

A recognition system should be insensitive to minor variations and still be able to distinguish different but sometimes very similar looking characters. Recognition systems should, at least in the beginning, be able to recognize many writing styles. Such multi-user systems usually have problems with recognition accuracy. One way to increase performance is adaptation, which means that the system learns its user's personal writing style.

1.1 Benefits of pen-based input

The most important benefit of pen-based input is combined pointing and text input. An application with graphical user interface requires the user to switch between a keyboard and pointing device if the user wants to manipulate icons, menus, windows, etc., or to enter text in different locations. Switching between a keyboard and pointing device means in practice that the user has to move his hand between the input devices and re-establish visual frame of reference among the display, keyboard, and pointing device. When both the pointing information and text input are used simultaneously, nonlinear text input is possible. Nonlinear text input means a writing process in which characters, words and sequences are not written in the same order as they are read (Ward and Blesser 1985).

Compared with a small keyboard, writing with a pen is easier and not slower than typing. With a miniature keyboard integrated into a small device, typing speed is 10-21 words per minute (wpm), and with a pen 15-18 wpm. Both speeds are considerably lower than typing speeds with touch screen and ordinary keyboards, 20-32 wpm and about 60 wpm, respectively (Guyon and Warwick 1996), (Ward and Blesser 1985).

Compared to text input based on speech recognition, handwriting offers better privacy. Handwriting is not sensitive to noise and it does not cause disturbance in environments where silence is required, for example in a library. In addition, the user does not need to spell difficult words, such as foreign words and names, letter by letter or remember long and complicated sentences while dictating text to the system. Also, editing of the text afterwards is easier (Mel et al. 1988).

1.2 Requirements for recognizer performance

Recognition accuracy is the key factor in determining the acceptability of a handwriting recognition system and the whole application in which it is implemented. According to experiments performed by Frankish et al. (1995), the relationship between the user acceptance of pen interface and the recognition accuracy is highly task-dependent.

Guyon and Warwick (1996) have organized an experiment in which good typists wrote with a special keyboard that made random errors with a predefined rate. These tests showed that writers tolerate errors up to 1% while 0.5% is unnoticeable and 2% is intolerable. These error rates are lower than error rates for human readers. A study organized by Neisser and Weene (1960), which involved 200 writers and 9 readers, showed that average human recognition error rate for isolated, handwritten, and context free characters is 4.1%. Almost the same result (4.0%) was obtained in a more recent experiment in which context free characters from 10 writers were recognized by 10 readers. The best writer-reader couples had zero error percentages but the error rate for the worst couple was as high as 21.7% (Parizeau and Plamondon 1994). However, the human ability to recognize isolated alphanumeric characters is considerably lower according to the authentication test described by Kassel (1995). In that test, three readers labeled over 29 000 characters written by 150 subjects. Only 87.6% of the characters were correctly recognized. The case substitution error rate was 6.6%. A case substitution error means that the lower and upper case versions of a letter are confused.

These experiments clearly show that the required recognition rate for a handwriting recognizer is very high – even higher than humans can perform. Another requirement for the recognition systems, besides the recognition rate, is reliability. High reliability can be achieved if the system rather rejects characters than falsely recognizes them.

1.3 Aims and overview of this thesis

The work described in this thesis has been carried out in cooperation with Nokia Research Center. It has been funded by the Technology Development Centre Finland as part of the technology programme Adaptive and intelligent Systems Applications. The aims of this thesis are to discuss the essential background knowledge of the research field of handwriting recognition, make a literature survey of the most important and recent recognition methods, and implement one of the state-of-art methods in practice.

The theoretical part of this work, presented in Chapter 2, mainly consists of the literature survey. It concentrates on the on-line recognition methods for isolated handwritten alphanumeric characters. Special attention is paid to those methods which are suitable for learning systems. Several recognition algorithms, mainly found from articles in the various journals and conference proceedings dealing with pattern recognition, artificial intelligence, and novel man-machine interface problems, are illustrated with examples.

The practical part of this thesis consists of creating a learning on-line recognition system by using a well-known recognition method, namely elastic matching. This algorithm has been proved to be effective and accurate by the earlier research works described in the literature survey. In addition, it can easily be used in an adaptive system: it is based on model matching and a new model can be formed from a single training sample. The description of the recognition method applied in this work is presented in Chapter 3. The first aim of the practical part is to select the preprocessing and normalization methods and dissimilarity measure which are most suitable for a multi-writer system. The second – more important – aim is to turn the initial writer-independent system into a writer-dependent system. Several on-line learning strategies are suggested and tested both on-line and via simulations. The learning strategies are described in Chapter 4. In addition, the user adaptation is discussed.

The initial recognition system is formed by using data collected earlier but all the tests are performed with new data collected especially for this research work. Data used in the experiments and the preprocessing and normalization methods are described in Chapter 5. All the experiment are described and their results are presented and discussed in Chapter 6. The major conclusion drawn from the results are collected in Chapter 7.

Chapter 2

Recognition of handwriting

Handwriting recognition systems can be subdivided into many different categories. Most important discriminating feature of the systems is the time when the text is recognized. Recognition can be done either off-line or on-line. In addition, the different sequences of text which form the unit to be recognized as a single entity discriminate problem types: the text can be considered as a sequence of characters, words or whole sentences. The set of characters, writing style, and constraints on the writing also have a great influence on recognition methods. These distinctions are described in more detail in the following sections.

2.1 Off-line recognition

The handwriting recognition task heavily depends on the application. First applications were systems that recognized text that had originally been written on paper. In such systems, paper sheets are digitized into two-dimensional images. Features for recognition are first enhanced and then extracted from the bitmap images by means of digital image processing. This type of recognition is called off-line recognition as the text is not recognized at the same time as it is produced but after the writing task is completed. Off-line handwriting recognition is a subset of optical character recognition (OCR) (Tappert et al. 1990).

Off-line methods are not suitable for man-machine communication because they do not facilitate real-time interactivity. They are suitable for automatic conversion of paper documents to electronic documents which then may be interpreted or post-processed by computers. Typical applications for optical character recognition are systems which need to automatically handle a huge amount of information in paper form. Such recognition systems have been developed for numerous application areas but are mainly used for machine-printed text. Applications for handwritten text include, to name a few: automatic sorting of mail, handling of financial documents such as checks, writer recognition, signature verification, and shorthand transcription (Govindan 1990).

2.2 On-line recognition

In the case of on-line recognition, text is input with special equipment and it is recognized in real time. The input medium is usually a tablet or a flat display that can capture information on the location and motion of a pen or some other pointing device moving on its surface. Writing can be performed with any ordinary pen or a pen-like device especially designed for this purpose. Location and possibly also pressure data are frequently sent to the computer which performs the recognition task.

On-line recognition is considerably different from off-line recognition because of the dynamic information on writing: the writing speed, acceleration, angle of the pen and its pressure against the writing surface are available for the system (Nouboud and Plamondon 1990). Additional information that is absent in OCR, can be derived from the number, order and drawing direction of the strokes. On the other hand, the dynamic information can be abandoned and methods developed for off-line recognition used instead. Just the preprocessing phase needs to be modified. The dynamic information improves the recognition accuracy (Tappert et al. 1990).

The main applications for on-line character recognition allow only a limited interaction as they are typically just for filling out forms. In more advanced applications, a pointing pen and a handwriting recognition system can replace the mouse and keyboard by emulation. Such systems usually have a special tablet for writing and the recognition is carried out by the tablet's own microprocessor as the algorithms are very dependent on such things as the sampling rate and resolution. The recognition result is sent to the main application whose software needs only minor modifications if any (Tappert et al. 1990).

A good example of a commercial product is Smart Paper developed at the IBM's Thomas J. Watson Research Center. It is a tablet that can be used with an ordinary PC. The text to be recognized is written on ordinary paper on top of the tablet which has a little display for feedback to the user and shortcut buttons for commands. Writing is done with a special pen. The tablet's recognition algorithm is adaptive

Figure 2.1: *A typical example of a PDA without a keyboard, 3Com's Palm Pilot.*

so it can learn the user's writing style (Gardner 1997).

The writing surface can also be integrated into the device itself as is the case in novel hand-held devices called personal digital assistants (PDAs), pen-based notebook computers, or organizers. As stated by Schomaker (1998), telecommunication capability is required to make such a mobile device truly useful and to have some added value and not just being another isolated information processing unit of convenient size. The first hand-held device with both a pen-based user interface and telecommunications, namely Personal Communicator, was introduced by the EO company in 1993 Schomaker (1998).

Apple Newton, which was released in 1993, was the first PDA without a keyboard. It has a touch-sensitive screen and writing is carried out in natural handwriting style with a special stylus. The system has problems with recognition accuracy mainly because of new users not familiar with handwriting recognition expected the system to recognize also peculiar styles. This problem has been avoided in 3Com's Palm Pilot in which the style of writing is constrained and the system recognizes only special characters called Graffiti. Despite the fact that the user has to learn a new character set, Palm Pilot has dominated the PDA markets for the last two years (Comerford 1998). Palm Pilot is illustrated in Figure 2.1.

2.3 Character sets

The character set which forms the text to be recognized affects crucially the recognition algorithms used. The recognition task is naturally easier when the number of characters is small and they are dissimilar. Recognition systems have been developed for numerous languages and character sets: Roman alphabets, Arabic numerals, Japanese, Chinese, Indian, Arabic, Korean, Cyrillic, Hebrew, Thai, Greek, Berber, and shorthand writing (Govindan 1990). The most difficult characters are

Chinese due to the enormous number of them – over 50 000. Chinese characters can also be written in two different styles, block or cursive.

Other important characteristics of the character sets are the typical number of strokes in one character and the variety of the stroke forms. The Chinese characters have an average of 8-10 strokes in block style. It is considerably more than the average of about 2 strokes in capital Roman alphabets (Tappert et al. 1990). In Japanese hiragana, almost all strokes are curved whereas the strokes in the Chinese characters mainly consist of straight lines and only a few corners (Wakahara and Odaka 1997).

2.4 Writing style variations

Handwritten characters have enormous variety in shape compared to machine-printed characters. Variations occur mostly between writers but also with one writer. Handwritten characters may be regarded as distorted versions of idealized character models and the distortion can be interpreted to be caused by several factors. According to Kuklinski (1984), sources of variations can be classified as personal background factors, situational factors, and material factors. These factors can affect both the generation and recognition of the characters.

2.4.1 Variations of characters

A writing style is based on the alignment and variable forms of characters. Handwritten characters can vary in both their static and dynamic properties (Tappert et al. 1990). Static properties are the underlying, ideal model of the character and the geometrical properties such as relative position and size of the strokes, corners and retraces, ornamentals, aspect ratio, size, and slant. Dynamic properties are more involved with the generative aspects of the characters. Characters can look similar although their number of strokes, stroke order and direction may vary considerably (Kuklinski 1984). Because of these numerous variations, the number of styles to write a certain character can be considered as infinite.

The number of strokes varies as sequential strokes have a tendency to be connected within a character. This is caused by pen lift failures. Stroke connecting is more likely to occur between sequential vertical or horizontal strokes whose ends are close to each other than between strokes of which one is horizontal and the other is vertical. Retraces of strokes may be made for same reasons. Very short retraces at the

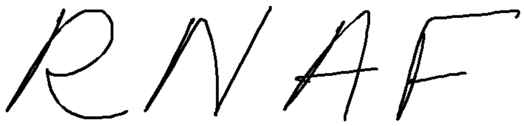


Figure 2.2: *Characters with retraces.*

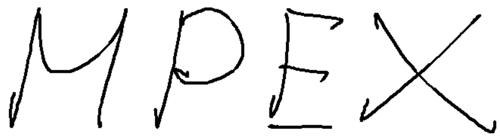


Figure 2.3: *Characters with hooks and continuations.*

beginning and end of the strokes are called hooks and continuations, respectively. Retraces, hooks, and continuations are illustrated in Figures 2.2 and 2.3. The number of strokes increases when a stroke is broken by a false pen lift. The direction of a stroke varies if the writer tries to minimize the time the pen is lifted up or by some personal preferences. Usually, vertical strokes are drawn before horizontal strokes and strokes on left before strokes on the right (Ward and Kuklinski 1988).

In addition, the form of the strokes varies – nominally straight strokes can be curved as bows or s-curves. The forms of loops and cusps change as loops have a tendency to collapse into cusps and, on the contrary, cusps can change into loops. Distinguishing the cusps and loops is a difficult problem because they produce similar features (Ward and Kuklinski 1988).

2.4.2 Alignment of characters

Handwriting styles can be classified into four basic classes: boxed discrete, spaced discrete, run-on discrete, and connected or pure cursive handwriting. Boxed discrete style means that all characters are written in boxes. Spaced discrete style means that the writing is less constricted and the characters are separated spatially by a significant space or temporally by a predefined time difference between the characters. In the run-on discrete style, characters can touch each other but a stroke cannot be included in more than one character. This means that pen must be lifted at least between characters. In the cursive style, all the letters in one word are connected. Hence, the form of a letter is greatly affected by the surrounding letters. Usually, natural handwriting style is a mixture of discrete and cursive styles (Bellegarda et al. 1994), (Fujisaki et al. 1991). Some examples of text samples with

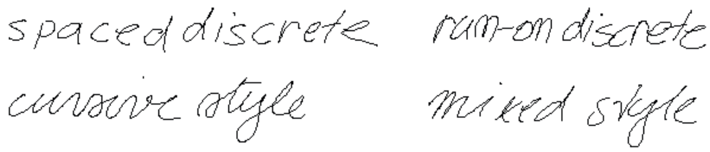


Figure 2.4: *Different aligments of characters.*

differently aligned characters are shown in Figure 2.4. All the samples are written by the same writer.

One of the major problems in recognition of cursive style handwriting is the segmentation of the words into characters. A well known paradox states that in order to attain a good segmentation, the word should be recognized first. But, before the recognition can be performed, the word must be segmented. External segmentation methods, which are performed prior to the recognition, cannot sufficiently isolate a character in the case of non-constrained cursive handwriting in which several letters can be drawn with a single stroke (Tappert et al. 1990). These methods can be used to locate words. Several different segmentations into characters can then be tried for each word and the segmentations yielding the best recognition results in the sense of the decision criteria are chosen. A lexicon and other linguistic knowledge can be used to prune the alternatives. Segmentation can also be done iteratively on the basis of the recognition results. The segmentation methods based on the recognition are called internal segmentation (Tappert et al. 1990). Sometimes segmentation and recognition are carried out simultaneously in an integrated process. Such a method is described by Hu et al. (1998).

2.4.3 Personal background factors

One of the most important personal background factors of writing style is handedness as left-handed and right-handed writers use muscle and tendon groups involved in writing differently. For example, left-handers prefer to draw horizontal lines from right to left and right-handers prefer to do it in the opposite direction. This can be explained by the fact that writers usually prefer dragging the pen behind the hand to pushing it ahead (Ward and Kuklinski 1988). Age and health affect motor control of writing and thereby the resulting writing style. The education and origin of the writer have important roles in the writing style because different type characters are taught in different schools. In addition, some professions affect style, especially when a neat writing style is needed (Kuklinski 1984).

2.4.4 Situational factors

Situational factors are haste, motivation, and the method of presentation. Haste can be caused by several reasons and it has major effects on style as many people have different styles for different writing speeds. In addition, fatigue and mood affect the writing speed. The motivation of the writer also determines how carefully characters are written, for example, the address of an important letter is very likely to be written carefully while an ordinary shopping list can be almost unreadable even to the writer himself. Text can be written differently depending on whether it is self generated or copied (Kuklinski 1984).

2.4.5 Material factors

The writing instrument, surface, and form constitute the material factors of the writing style. The writing instrument has an effect on the writing style depending on its size and overall comfort. Writing surface's friction and position also affect the style. The form factors, such as the size of the blank writing area, the length of the writing line, or the size of the writing boxes for characters, can have a dramatic effect on the style (Kuklinski 1984).

2.4.6 Constraints on writing

As the variations are the key problem of automated handwriting recognition, writing style is usually more or less constrained in such systems. Usually, characters are written in boxes to ease their separation, or the system provides guiding lines to help the users write more consistently. These constraints are designed to improve the performance of the recognizer and they can be seen as mediate *a priori* decision rules. If the constraints are too strict, they may cause the system to be rejected by the users. According to experiments carried out by Suen (1979), constraints do affect the style and speed of writing and the recognition results.

In the most constrained systems, users are expected to use specific characters. These systems perform well assuming that the user writes in the required style. The drawback of such a system is that the users tend to write in their own style and, as the styles vary a lot, the system is awkward for most of the users (Ward and Blesser 1985). In Figure 2.5, the character set used in 3Com's Palm Pilot is shown as an example of a constrained writing style.

Figure 2.5: *Graffiti characters.*

2.5 Context

Text is a string of characters which can be subdivided into basic units such as syllables, words, phrases, sentences, lines, paragraphs, and so on. Context of the text consists of the surrounding units which give the meaning to the text. The use of the context facilitates the recognition of the otherwise very confusing parts of the text and thus improves the recognition performance of the system. According to Srihari (1985), contextual knowledge includes the following: the probabilities of letters, letter pairs, letter triplets, etc., the probabilities of words; the legal combinations of letter pairs, triplets, etc., a lexicon or list of words acceptable in global or local context, a grammar describing the syntax of the language, and a semantic network representation of the concepts embodied.

The context is very important in recognition of cursive script, especially during the character segmentation phase. Contextual information can also be used when recognizing discrete characters and it usually improves the system's performance (Tappert et al. 1990). In some problems, no contextual information exists. The language used can be unknown and the input text totally unconstrained so that the text may not have any apparent meaning. A situation like this is very likely to occur in a PDA application in which the user typically writes down names and nicknames, addresses, telephone numbers, codes and other short notes. It is not possible to know what language is used, when it is changed, or make a dictionary with all existing words in it.

2.6 Recognition methods

The character recognition problem has been approached in many ways and various recognition methods have been suggested. Some of the methods have been especially developed for character recognition but most of them are borrowed from other fields of pattern recognition, signal processing, and image analysis. Recognition methods can be roughly classified into four major groups: statistical, structural and syntactical, model matching, and neural network methods. Sometimes, different methods are combined, for example, simple methods are used for preclassification and final decision is made with more sophisticated methods.

The recognition procedure is basically very simple: after preprocessing, some features are extracted from the unknown character which is then classified to the class whose members have the most similar features. The problems involved in this process are: 1) finding both descriptive and discriminating features, 2) selecting a way to compare them, and 3) creating the classification rules. Of course, these problems are highly dependent on each other.

In the next sections, some character recognition methods are briefly described. In the recent years, the number of commercial applications for handwriting recognition has increased. Due to the hard competition for market shares, research is not so open as it used to be and it is difficult to find information on the most recent methods.

2.6.1 Statistical methods

In statistical methods, the recognition of a character is carried out by choosing the character class which is most probable or has the minimum measure of expected classification error or some other risk. Features are assumed to be variables with statistical distributions and different classes to have different values for the distribution parameters. Values for the parameters can be estimated from measurements. If the parameters are updated on-line, the system is adaptive and thus able to adjust to different writing styles. In addition, *a priori* class probabilities can be used in the decision rules.

Next, some methods are described in more detail as examples of statistical methods. The recognition rates of different methods are not as such comparable as the classifiers have been created and tested with different data.

Example 1: A simple statistical method for on-line character recognition is described by Odaka et al. (1982). Isolated characters are preclassified according to the number of strokes. Each stroke is then approximated with a small number, N, of points which divide the stroke into straight segments of equal lenght. The reference patterns consist of the mean vectors and covariance matrices of the strokes. Final classification is carried out by minimizing the distance between the feature vector of the unknown character and the reference patterns. The feature vector of a character consisting of S strokes is:

$$L = (x_{11}, y_{11}, x_{12}, y_{12}, \ldots, x_{SN}, y_{SN}). \tag{2.1}$$

The distance between a feature vector L and a reference pattern of class θ is defined as follows:

$$D(L, \theta) = (L - M_\theta) \sum\nolimits_\theta^{-1} (L - M_\theta)^T, \tag{2.2}$$

where M_θ is the mean vector and $\sum_\theta$ is the covariance matrix of the reference patterns. The mean vectors and covariance matrices are estimated for each class using the corresponding training samples.

It was found that the suitable number of points per stroke is five for alphanumeric characters and over 100 training samples per class are needed. The system was trained with data from several writers. The recognition rates achieved for alphanumerics were surprisingly high: 100% for the learning set and 99.9% for the test set.

Example 2: The method described in the previous example is more suitable for characters that consist of straight strokes as curly strokes cannot be estimated properly with few points. Another on-line method for curly characters is proposed by Arakawa (1983). In that method, the feature vector of a stroke consists of P Fourier coeffients for the pen-point movement loci. The feature vector of a character of S strokes is:

$$L_s = (a_{11}, \ldots, a_{1P}, b_{11}, \ldots, b_{1P}, \ldots, a_{S1}, \ldots, a_{SP}, b_{S1}, \ldots, b_{SP}), \tag{2.3}$$

where

$$a_{ij} = \frac{2}{N_i} \sum_{j=1}^{N_i} x_{ij} \cos(\pi ij / N_i) \tag{2.4}$$

and

$$b_{ij} = \frac{2}{N_i} \sum_{j=1}^{N_i} y_{ij} \cos(\pi ij / N_i). \tag{2.5}$$

(x_{ij}, y_{ij}) is the jth point in the ith stroke. N_i is the total number of points in the ith stroke.

The method is otherwise very similar to the method described in the previous example. Classification is based on the Bayesian decision rule: the distributions of the pattern feature vectors are assumed to be multivariate normal and the most probable class is chosen. The probability of a feature vector L belonging to class θ is:

$$P(L,\theta) = \frac{1}{(2\pi)^{n/2}|\sum_\theta|^{1/2}} \exp\left[-\frac{1}{2}(L-M_\theta)\sum_\theta^{-1}(L-M_\theta)^T\right], \tag{2.6}$$

where M_θ is the mean vector and $\sum_\theta$ is the covariance matrix of the reference pattern in the class θ. $n = PS$ is the length of the feature vector.

With this method, high recognition rate for alphanumerics and multi-writer data was obtained. The training data was collected from 25 writers and it consisted of five characters per class written by each subject. The method was tested with writer-independent data. The test data was collected from ten writers and it also had five samples per class written by each writer. The recognition rate was 99.2%.

Chain codes

Different kinds of chain codes are widely used in character recognition methods. Chain coding means that a character is divided into segments which are then coded. The segmentation can be based on the local extreme points (left, right, up, down, maximum or minimum velocity, etc.) or by evenly spaced points. The codes can present straight lines of equal length with discrete directions, lengths of straight lines and continuous angles between them, or different types of lines, curves, loops and corners.

Classification of chain coded characters can be carried out in several ways, for example by using statistical methods (Loy and Landay 1982), string comparison methods (Nouboud and Plamondon 1991) or dynamic programming (Yuen 1996).

Example 3: A statistical method for classifying chain coded characters is described by Loy and Landay (1982). In their method, segmentation is based on the maximum curvature points and the character is approximated with a polygon. The feature vectors consist of the distances of the line segments from the origin and the angles between the line segments and the horizontal axis. The feature vector classes are assumed to have similar distributions as in the previous example. In addition, classification rules similar to those in the two previous examples have been used.

The method was tested with data from a single writer. The recognition rates for both classification rules were over 98% and when the distribution parameters were updated on-line, the recognition rates improved up to 99%.

Hidden Markov Models

The speech recognition problem has many similarities with the handwriting recognition problem. Therefore, many methods and ideas have been adopted from the former problem to the latter. Hidden Markov models (HMMs) are currently the state-of-art method in speech recognition and they seem to be gaining popularity also in handwriting recognition. They are especially suitable for the recognition of cursive script.

HMMs were initially introduced and studied in the late 1960s and early 1970s. HMMs have also been called Markov sources and probabilistic functions of Markov chains. They are used for describing the statistical properties of a signal. The modeled signal is assumed to be a random process whose parameters can be estimated in precise, well-defined manner. HMMs are based on Markov processes whose states are not directly observable – therefore they are called hidden Markov models. The observations are probabilistic functions of the states.

A basic HMM is characterized by the following parameters: 1) the number of states in the model, N, number of distinct observations per state, M, 2) state transition probabilities, $A = \{a_{ij}\}$, where i and j are the indices of the corresponding states, 3) observation probabilities for each state, $B = \{b_j(k)\}$, where j is the index of the state and k is the index of the observation, and 4) initial state distribution ,$\pi = \{\pi_j\}$, where j is the index of the state (Rabiner 1989). HMMs used in handwriting recognition are typically referred to as left-right models (Hu et al. 1996). These models have the property that the state index increases or remains unchanged as the time elapses. A direct path (the state index increases by one in every state transition) through such a model corresponds to the average realization of the modeled process. Self loops (the state index remain unchanged) correspond to lengthened versions of the process and state skips correspond to shortened versions (Nathan et al. 1993).

A HMM can represent several things in a handwriting recognition system. Early HMM-based on-line recognition systems had separate models for each word. Because the number of models increases with the size of the vocabulary, systems using character models have become more popular (Nathan et al. 1993), (Veltman and Prasad 1994). As the characters usually contain patterns that are common to more than one character, stroke and other subcharacter models can also be used and concatenated to form letter models (Hu et al. 1996). Training of the system is easier when the number of models is small. In the case of a large vocabulary, it can be difficult to obtain an adequate training set as several samples for each word model are needed for proper estimation of the model parameters.

In speech recognition systems, the observation probabilities for each state are usually mixtures of continuous probability densities $b_j(O) = \sum_{m=1}^{M} c_{jm}\mathcal{N}(O, \mu_{jm}, U_{jm})$, where O is an observation vector, c_{jm} is the mixture coefficient for the mth distribution in the jth state, and $\mathcal{N}$ is any log-concave or elliptically symmetric probability density function. $\mathcal{N}$ is typically a Gaussian distribution. Discrete probability densities are more suitable for handwriting recognition according to a comparison carried out by Rigoll et al. (1996). Similar result was obtained by Bellegarda et al. (1995). In a writer-dependent task, the recognition results of the discrete parameter method were as good as the results of the continuous parameter method, and in a writer-independent task, the results of the discrete parameter method were only slightly worse. In both tasks, the discrete parameter method required only a few percents of the CPU time consumed by the continuous parameter method.

Three basic problems associated with HMMs are: 1) computing of the probability of the observation sequence for a given model, 2) choosing the corresponding state sequence for the given observation sequence and model, and, 3) adjusting the probability parameters A, B and π of the model according to the given observation. The first problem is the same as choosing the model which best matches the observation. Solving the second problem corresponds to uncovering the hidden parts of the model. Each method to train and adapt the system is a particular solution of the third problem (Rabiner 1989).

Example 4: Starner et al. (1994) used a HMM-based system developed for speech recognition for recognition of handwriting in order to demonstrate the similarities between the problems and the suitability of the method for both of them. The only modification to the system was the change of the front-end module – different features were used for handwriting and for speech. The system was used for recognition of cursive script written on-line. The angle of the trajectory of the pen and its changes in each data point were calculated from the x- and y-coordinates, quantized, and then used as features. A HMM consisting of seven states was used for modeling each character.

The results were good: an average of 4.2% word error rate and an average of 1.4% character error rate were achieved in the writer-dependent tasks. When a statistical bigram grammar for word pairs and additional HMMs for the contextual effects of adjacent characters were used, the recognition performance of the system improved and the word error rate dropped to 1.1%.

Example 5: In an article written by Bellegarda et al. (1994), the mathematical background of a recognition algorithm based on HMMs is well described. The algorithm is suitable for unconstrained handwriting. Its main goal is to retrieve the actual character sequence C from the handwritten sequence S. In the statistical sense, this means the finding of the character sequence C^* that minimizes the recognition error probability, or, in other words, maximizes the *a posteriori* probability of the given input sequence S. The decision criterion of the algorithm can be formulated in the following way:

$$C^* = \arg\max_C P(C|S). \tag{2.7}$$

When the Bayes rule (Schalkoff 1992) is applied and the irrelevant denominator $P(S)$ is omitted,

$$C^* = \arg\max_C P(S|C)P(C). \tag{2.8}$$

The first term $P(S|C)$, which can be called the handwriting channel, characterizes all the scribble sequences that can be generated by the model of the intended character sequence C. The last term $P(C)$, the source model, characterizes all the possible sequences of characters in the language considered.

The input sequence S consists of frames which are small equally informative portions of handwriting. A character sequence is composed of characters which can be associated with a block of frames delimited by two boundary points. In the case of discrete handwriting, the boundary points can be uniquely determined and the handwriting channel can defined in the following way:

$$P(S|C) = \prod_{j=1}^{M} P(\Phi_j|c_j), \tag{2.9}$$

where $\Phi_j = f_{b_{j-1}+1} \ldots f_{b_j}$ is a block of frames associated with the character c_j. $\{b_j\}$ are the indices of the boundary points. If the characters are connected, the boundary points are quite arbitrary and the handwriting channel is defined differently:

$$P(S|C) = \sum_{\{b_l\}} \prod_{j=1}^{M} P(\Phi_j|c_j), \tag{2.10}$$

where the summation is taken over all possible choices of the boundary points which segment S to M characters. $P(\Phi_j|c_j)$ is defined with a HMM and called the baseform $\mathcal{B}_j$ of the character c_j. It corresponds to the probability that the block Φ_j is a realization of the given baseform $\mathcal{B}_j$. The definition of the probability is:

$$P_{\mathcal{B}_j}(\Phi_j) = \sum_{\{paths\}} P(\Phi_j|path)P(path), \tag{2.11}$$

where $\{paths\}$ are all the possible routes through the associated HMM. The two probabilities needed in the previous equation are also defined by the same HMM.

2.6.2 Model matching

Recognition methods which apply model matching are based on the assumption that the handwritten characters, or some other text units, such as strokes or whole words, are distorted realizations of some ideal models. These distortions are explained by the variations of the writing style, changes in the orientation and scaling of the writing, and noise from the environment or equipment. In optical character recognition, model matching means comparison between two images but in on-line recognition, it usually means curve matching. The matched curves can consist of different functions of time, for example the preprocessed x- and y-coordinates of the pen point, pressure against the writing surface, and angle of the trajectory. Curve matching is either linear or nonlinear. In the former case, the number of data points is constant and one-to-one correspondence can easily be defined between the points of two curves. In the latter case, the definition of the correspondence between the points is not so straight-forward.

An unknown pattern is recognized by finding the model which minimizes some distance measure, or alternatively maximizes some similarity measure, between them. Many different distance and similarity measures have been suggested and successfully applied. In the following examples, some of them are described in more detail.

Example 6: Wilfong et al. (1996) have developed an on-line recognition method which is based on linear curve matching. It is suitable for user-specified alphabet which can consist of all kinds of handwritten text units or drawn symbols, for example characters, words or any arbitrary images. These hand-produced patterns are called symbol instances. Depending on its use, a symbol instance can be called a model instance or an unknown instance: the model instances form the prototype set and the unknown instances are recognized by comparing them to the prototypes. The instance symbol α is represented by the instance vector

$$v_\alpha = (x_1(\alpha), y_1(\alpha), \ldots, x_n(\alpha), y_n(\alpha)), \tag{2.12}$$

where $x_i(\alpha)$ and $y_i(\alpha)$ are the preprocessed coordinates of the ith data point and n is the total number of sampled data points. The preprocessing consists of translation normalization: the coordinates of the centroid of the instance are subtracted from the original coordinates of the data points. The centroid is the mass center of a piecewise linear curve which is formed by interpolating line segments between consecutive data points. It is defined in the following way:

$$\bar{x}_\alpha = \frac{\sum_{i=1}^{n-1} d_i \, (x_i(\alpha) + x_{i+1}(\alpha))}{2L} \tag{2.13}$$

$$\bar{y}_\alpha = \frac{\sum_{i=1}^{n-1} d_i \, (y_i(\alpha) + y_{i+1}(\alpha))}{2L}, \tag{2.14}$$

where d_i is the Euclidean distance between the ith and the $(i+1)$th data points and L is the sum over all d_is.

Two instances are regarded as equivalent if they can be derived from each other by scaling and rotation. A two-dimensional hyperplane through the origin of the instance space can be constructed for every instance. It is formed of all the equivalent instances derivable from the given instance. The distance between two instances is defined by the angle between the corresponding hyperplanes. When the angle is zero, the instances are considered to be equivalent. The hyperplane S_α consisting of all the instances equivalent with α is

$$S_\alpha = \{w_\alpha(\gamma, r) : 0 \leq r \leq \text{and} - \pi < \gamma \leq \pi\}, \tag{2.15}$$

where

$$w_\alpha(\gamma, r) = r(v_\alpha \cos(\gamma) + u_\alpha \sin(\gamma)), \tag{2.16}$$

$$u_\alpha = (-y_1(\alpha), x_1(\alpha), \ldots, -y_n(\alpha), x_n(\alpha)), \tag{2.17}$$

γ is the rotation angle, and r is the scaling factor. All hyperplanes intersect along the line where $r = 0$.

The recognition is based on minimizing the distance between the unknown instance and the model instance. Before the distance can be calculated, both instances must be modified so that they have the same number of strokes and data points. This is carried out by merging and splitting the strokes of the unknown instance and re-sampling the piecewise linear curves representing the strokes of the model instance. After performing the merging and splitting of the strokes, the relative arc lengths of the corresponding strokes of the unknown instance and the model instance are equivalent or the differences are within a given tolerance. The relative arc length of a stroke is obtained by dividing the original arc length by the sum of the arc lengths of all strokes. The resampling of the model instance is performed so that the normalized distance between adjacent points is the same as the corresponding normalized distance in the unknown instance. The normalized distance is the Euclidean distance between the points divided by the total arc length of the stroke.

The similarity measure between the unknown instance α and the model instance k derived from the angle between the corresponding hyperplanes is given by

$$\frac{(v_\alpha \cdot v_k)^2 + (v_\alpha \cdot u_k)^2}{(v_k \cdot v_k)^2}, \tag{2.18}$$

when a redundant term $(v_\alpha \cdot v_\alpha)^2$ is omitted. This measure is insensitive to the distortion caused by translation, rotation, and scaling. The similarity measure attains values from zero to one. In the case of two confusing character classes which differ only by their size or orientation, also the rotation angle γ and the scaling factor r can be used for the discrimination. They are given by

$$r = \frac{\sqrt{(v_\alpha \cdot v_k)^2 + (v_\alpha \cdot u_k)^2}}{(v_\alpha \cdot v_\alpha)} \tag{2.19}$$

and

$$\gamma = \cos^{-1} \frac{(v_\alpha \cdot v_k)}{\sqrt{(v_\alpha \cdot v_k)^2 + (v_\alpha \cdot u_k)^2}}. \tag{2.20}$$

This method was tested with upper case letters from 'A' to 'Z'. Two samples of each character were collected from eight subjects. One set of characters was used as a prototype set and the other set was used as a test set. The average recognition percentage for writer-dependent tasks was 98%.

Example 7: Wakahara and Odaka (1997) have developed a linear curve matching method for isolated characters which applies a distance measure completely different from the one used in the matching method described in the previous example. The variations between the unknown character and the model character are explained by stroke-based affine transformation (SAT) components and residual components. The SAT components are caused by affine transformations, such as translation, rotation, scaling, and shearing. They represent intraclass variations, whereas the residual components represent interclass variations. The matching method consists of three stages: deformation extraction, deformation analysis, and deformation estimation stages.

In the deformation extraction stage, deformation vector field (DVF) between the unknown character $T = (t_1, \dots, t_K)$ and the model character $R = (r_1, \dots, r_K)$ is formed. Vectors r_k and t_k are the xy-coordinate pairs of the kth data points. K is the total number of data points. Before DVF can be calculated, the model character has to be modified by stroke linkage method so that it has the same number of strokes (N) as the unknown character. After that, both characters are resampled so that the number of data points per stroke (n) is constant and the points are equidistant. DVF is a set of deformation vectors $\{d_k\}_{k=1}^{K}$ given by

$$d_k = r_k - t_k. \tag{2.21}$$

In the deformation analysis stage, DVF is decomposed into SAT components and residual components. First, the optimal affine transformation is found for each corresponding stroke pair. This is done by solving the following problem:

$$\min_{A_i, b_i} \sum_{k \in \{k_i\}} \|(A_i t_k + b_i) - r_k\|^2, \tag{2.22}$$

where i is the index of the stroke pair, A_i is a 2×2 affine transformation matrix representing rotation, scaling and shearing, b_i is a translation vector, and $\|\cdot\|$ is the Euclidean norm. $\{k_i\}_{i=1}^{n}$ is a set of data point indices of the ith stroke and its L neighboring strokes which are included because the adjacent strokes are assumed to affect each other. After A_i and b_i are found, the decomposition of DVF is carried out in the following way:

$$\begin{aligned} d_{ij} &= [(A_i t_{ij} + b_i) - t_{ij}] + [r_{ij} - (A_i t_{ij} + b_i)] \\ &= \langle d_{ij} \rangle + \varepsilon_{ij}, \end{aligned} \tag{2.23}$$

where i is the index of the stroke, j is the index of the data point, $\langle d_{i,j} \rangle$ is the SAT component, and ε_{ij} is the residual component of d_{ij}.

In the deformation estimation stage, two distances are calculated: D_1, the distance between T and R, and D_2, the distance between SAT-superimposed T and R:

$$D_1 = \frac{1}{N} \sum_{i=1}^{N} \gamma_i \sum_{j=1}^{n} |d_{ij}|/n \tag{2.24}$$

$$D_2 = \frac{1}{N}\sum_{i=1}^{N}\gamma_i\sum_{j+1}^{n}|\varepsilon_{ij}|/n. \tag{2.25}$$

The value of γ_i depends on the modifications of R performed by the stroke linkage method: if the ith stroke of R is artificially connected, $\gamma_i \leq 1$, otherwise $\gamma_i = 1$. $|\cdot|$ is the City Block norm.

The recognition is carried out hierarchically in two stages. In the first stage, the D_1-distances between T and all the model characters are calculated and the nearest 20 models are selected as candidates. In the second stage, the D_2-distances between T and the candidate models are calculated. The class of the nearest second stage candidate determines the recognition result.

Elastic matching

Elastic matching performed with a dynamic time warping (DTW) algorithm is a nonlinear matching method originally used in speech recognition. It was developed at the beginning of 1970s and was introduced as a method for recognition of handwriting in the late 1970s. Similar methods have also been used successfully in on-line signature verification (Martens and Claesen 1997), (Sato and Kogure 1982), (Wirtz 1998).

Dynamic time warping can be used in comparison of all kinds of continuous functions of a continuous parameter, typically time. It is based on the idea that the speed of the process underlying the function can vary. The effects of these variations can be omitted in the comparison of two functions if the parameter axes of the functions are warped. In this case, warping means compressing and stretching of the parameter axes locally. In practice, the functions which are originally continuous are converted into discrete sequences by sampling. A function can be sampled with regular intervals or by some other sampling method based on its nature (Sankoff and Kruskal 1983).

Time variations are especially important in signature verification. A forger can write a very similar-looking signature, at least after practice, but cannot imitate the correct writing speed and pressure because of the difficulties in observing them and the differences in muscular systems involved in the writing process (Sato and Kogure 1982). In signature verification, the whole signature is usually handled as a single unit and no segmentation to smaller units, such as words or characters, is performed. Contrary to the signature verification, in which time warping is used for extracting and measuring the variations of the generation of the similar-looking writing styles, in handwriting recognition, it is used for compensating these variations.

Elastic matching methods used for handwriting recognition are insensitive to the distortions caused by the varying writing speed, for example, the variations in the total number and distribution of the sample points in a character. Typically, writing speed slows down in sharp corners and other points of extremal curvature, at the beginning and end of the stroke, but also by hesitation and pausing of the writer. When the speed of writing is slow, the sample points are located densely, whereas quick writing produces more sparsely located points. Writing speed variations also cause distortions in the shapes of the strokes. In the elastic matching methods, the optimal time warping and point-to-point correspondence are found with dynamic programming which minimizes the shape dissimilarities. In handwriting recognition, matching can be carried out on stroke, character, or word basis.

Even though elastic matching can cope with most of the time variations, it is sensitive to the stroke order which cannot be changed only by stretching and compressing time. The variations in the number of the strokes can be handled in a similar way as in the linear curve matching methods by preprocessing. For example, accidentally broken strokes can be connected, retraces and pen lift failures can be detected and removed, or all the strokes can be connected into a single stroke in some systematic way.

Since both elastic matching and HMMs are among the state-of-the-art handwriting recognition methods and widely used, Bellegarda et al. (1994) have performed some experiments in order to compare them. Two systems, one based on elastic matching and the other on HMMs, were trained and tested with the same data. The computational complexities of the systems were approximately equal. Both the recognition performaces and CPU times consumed were compared. The system based on HMMs had a significantly lower average recognition error rate for both the writer-dependent and the writer-independent tasks. However, the system based on elastic matching performed better for some of the writers. The CPU time consumed by the HMM system compared to the time used by the elastic matching system was 50% more in the writer-dependent tasks and 30% less in the writer-independent tasks. The choice between the two methods cannot be made only on the basis of these results: the handwriting recognition problem is highly task dependent and the performance of the methods can vary considerably depending on the choice of features, the definition of distance, and the data used in the training and testing of the system.

In the following examples, some recognition systems based on elastic matching are described in more detail.

Example 8: A very basic elastic matching method was introduced and described by Tappert (1984). The method consists of three modes of operation: training, recognition, and updating. In the training mode, the prototype set is established by storing the feature vectors of the characters which the user writes according to a specific training text prompted by the system. In the recognition mode, the unknown characters are classified to the class corresponding to the best matching, or closest, prototype. The prototype set is modified in the updating mode, in which the user writes characters in a similar way as in the learning mode. The characters which are incorrectly recognized are added to the prototype set. The prototype set is further supplemented by the characters for which the ratio of the distances to the second best and the best-matching prototypes is less than a given tolerance. Due to the updating mode, the recognition system is adaptive. Before use, the system can be trained with data from several writers, and it can be updated during the use with data from a particular user when problems in recognition occur.

The recognition is carried out on character basis. The characters need to be separated spatially by the writer. Before feature extraction, the characters are preprocessed: hooks are eliminated, data which corresponds to a dot are reduced to a single point, and the data points are resampled so that they are roughly equidistant in space instead of time. The ith data point consists of the slope angle ϕ_i of the curve, the height y_i, and the normalized coordinates $\overline{x}_i$ and $\overline{y}_i$ from the mass center of the character. The distance between two data points, $p_i = (\phi_i, y_i, \overline{x}_i, \overline{y}_i)$ and $p_j = (\phi_j, y_j, \overline{x}_j, \overline{y}_j)$, is given by the following equation:

$$d(i,j) = \min\{|\phi_i - \phi_j|, 360^\circ - |\phi_i - \phi_j|\} + |y_i - y_j| + |\overline{x}_i - \overline{x}_j| + |\overline{y}_i - \overline{y}_j|. \tag{2.26}$$

The distance between the unknown character and the prototype character is found by solving the following minimization problem:

$$D = \min_{\{w(i)\}} \sum_{i=1}^{N} d(i, w(i)), \tag{2.27}$$

where $\{w(i)\}$ is the set of all feasible time warping paths defining the point-to-point correspondence between the characters, and N is the number of data points in the unknown character.

The time warping path is constrained by boundary and continuity conditions. In this case, the boundary conditions ensure that the first and last data points of the unknown and prototype are matched against each other. The continuity conditions restrict the matching of the intermediate points: the warping function is a monotonically increasing function, multiple data points of the unknown can be matched to a single point of the prototype whereas at most one point of the prototype can be matched to a single point of the unknown. Thus, all the points in the unknown are matched but stray points in the prototype can be skipped.

The distance D can be solved recursively by applying dynamic programming in the following way:

$$D(i,j) = d(i,j) + \min\{D(i-1,j), D(i-1,j-1), D(i-1,j-2)\}, \tag{2.28}$$

where $D(i,j)$ is the minimum cumulative distance to the point (i,j). At the beginning of the recursion, $D(1,1)$ is set to $d(1,1)$ and $D(i,j)$ is set to infinite elsewhere. All the possible paths from the point (i,j) to the point (N,M) are examined recursively. M is the number of the points of the prototype. The optimal time warping path is determined by the minimizations performed at the each step of the recursive algorithm. Term $D(i-1,j-2)$ in the equation (2.28) corresponds to the skipping of the $(j-1)$ th point of the prototype and term $D(i-1,j)$ means that multiple points, at least the $(i-1)$th and ith points, of the unknown are matched to the jth point of the prototype.

The recognition performance of this basic method was tested with alphanumeric characters. The average recognition result for six writer-dependent tasks was 94% when one prototype set contained all character classes, 98% when the three character categories ('A'-'Z', 'a'-'z', and '0'-'9') had their own prototype sets and were recognized separately, and nearly 100% when a dictionary was used.

Example 9: Li and Yeung (1997) have described a method also based on dynamic time warping in which boundary conditions less strict than in the previous example are used. Virtual sink and source points are augmented to the prototypes and the distances between the sink, or the source, and any other point are set to zero. The time warping path is band-limited which means that only a limited number of points at the beginning and end of the unknown can be matched to the sink and source points. Otherwise, symmetric continuation conditions are used. According to them, multiple points of the prototype can be matched to a single point of the unknown and vice versa, all the points are matched, and the time warping function is monotonically increasing. These conditions allow the omission of the points at the end and beginning of the curve and thus eliminate some hooks and continuations.

Li and Yeung (1997) have also proposed another method for the adaptation of the prototype set. The adaptation does not increase the number of the prototypes. In the method, all successfully recognized characters are added to the prototype set and the prototype which belongs to the same class and has the lowest value of the prototype evaluation function $J(C)$ is removed. The prototype evaluation function is defined in the following way:

$$J(C_{mn}) = \sum_{i=1}^{N_m} D(C_{mn}, C_{mi}) + \min_{l,l\neq m}\{D(C_{mn}, C_{lj})|C_{lj} \in C_l^r\}, \tag{2.29}$$

where C_{mn} is the nth prototype of class m, N_m is the number of prototypes belonging to class m, $D(A, B)$ is the distance between prototypes A and B, and C_l^r is the current set of prototypes for class l. The aim of the technique is to produce a constant-sized prototype set with small intraclass distances and large interclass distances.

Example 10: In her master's thesis, Scattolin (1995) proposed a modified version of the elastic matching method described in Example 8. The optimal dynamic time warping path $w(i)$ is found in the exactly similar way. The main difference between the methods is the definition of the distance between the characters: D is the weighted sum of the point distances and it is given by the following minimization problem:

$$D = \min_{\{w(i)\}} \sum_{i=1}^{N} \omega_i d(i, w(i)), \tag{2.30}$$

where the weights ω_i are calculated separately for each point of the prototype and $d(i, j)$ are squared Euclidean distances between data points which have been normalized by subtracting the mass center of the character. The values of the weights should be high if the corresponding points are important discriminators between the classes. The weights are calculated according to the recognition results of the training set, for example in the following fay:

$$\omega_i = \frac{\overline{dtot}(i, E)}{\overline{dtot}(i, C)}, \tag{2.31}$$

where E is a subset of the training set and consists of the characters which are incorrectly recognized by the given prototype, whereas the subset C consists of the correctly recognized characters. Function $\overline{dtot}(i, S)$ is the average distance between the ith data point of the prototype and the data points of the characters belonging to the set S which are matched against it. $\overline{dtot}(i, S)$ is given by the equation:

$$\overline{dtot}(i, S) = \frac{1}{size(S)} \sum_{k\in S} d(i, w^k(i)), \tag{2.32}$$

where $w^k(t)$ is the optimal time warping path corresponding to the character k. The weights are calculated iteratively, for example in the following way:

$$\omega_i^{n+1} = \sqrt{\omega_i^n \frac{\overline{dtot}(i,E)}{\overline{dtot}(i,C)}}, \tag{2.33}$$

where n is the counter of the iteration round. Sets E and C are formed according to the recognition results of the training set obtained with the weight values calculated in the previous iteration round. The initial values of the weights can be set to unity.

The method was tested with Arabic numbers and multi-writer data. The use of the weights increased the recognition rate of the writer-independent task from 85.22% to 88.67%.

Example 11: A physics-based shape metamorphosis method for nonlinear curve matching and handwriting recognition has been proposed by Pavlidis et al. (1997). The method, originally used in computer-based animation, was developed by Sederberg and Greenwood (1992). The basic idea of the method is that mutually similar shapes undergo a minor metamorphosis compared to mutually different shapes when they are transformed into each other. The degree of morphing can be used as a matching criterion. This method can be used for the matching of all kinds of handproduced shapes.

A curve is modeled by its perceptually important points, the corner points, and by the points of low curvature between them. If the curvature is constant between two corner points, the middle point is used instead of a low-curvature point. These points are found by an algorithm originally proposed by Brault and Plamondon (1993). The linear segments interpolated between the consecutive points are considered as virtual wires made of some idealized metal. The shape morphing is performed by stretching and bending these wires. The degree of morphing corresponds to the minimum energy needed in the morphing. The minimum energy and the correspondence between the points of the initial shape S^I and the target shape S^T are found by a dynamic programming algorithm (Sederberg and Greenwood 1992).

The energy needed to stretch a wire segment is given by the equation:

$$E_s(L_I, L_T) = f_s \frac{|L_T - L_I|^2}{c_s \min(L_I, L_T) + (1 - c_s)\max(L_I, L_T)} \tag{2.34}$$

where L_I is the initial length, L_T is the target length, f_s is a stretching stiffness parameter, and c_s is a penalty parameter for segments collapsing into a point during the metamorphosis. The value of c_s can vary from zero to one.

The bending energy needed to change the angle ϕ between two wire segments is given by the equation:

$$E_b(\phi) = \begin{cases} f_b\,(\Delta\phi + m_b\Delta\phi^*)^2 + p_b & \text{, if } \phi \text{ is zero,} \\ f_b\,(\Delta\phi + m_b\Delta\phi^*)^2 & \text{, otherwise,} \end{cases} \tag{2.35}$$

where f_b is the bending stiffness, $\Delta\phi$ is the change in the angle, m_b is a penalty factor for non-monotonic angle changes, p_b is a penalty factor for angles going to zero, and $\Delta\phi^*$ is a non-negative measure for the non-monotony of the change in the angle. The purpose of all the stiffness and penalty factors is to prevent non-smooth intermediate shapes. Such shapes may result if similar parts in the initial and the target shapes are not maintained during the metamorphosis.

The total energy E_{total} needed for the shape metamorphosis from $S^I = (p_0^I, \ldots, p_{n_I}^I)$ to $S^T = (p_0^T, \ldots, p_{n_T}^T)$ is the sum of the sums over all bending and stretching energies:

$$E_{total}(S^I, S^T) = \sum_{\{segment\ pairs\}} E_s(L_I, L_T) + \sum_{\{point\ pairs\}} E_b(\phi), \tag{2.36}$$

where $\{segment\ pairs\}$ and $\{point\ pairs\}$ are the sets of all line segment and segmentation point pairs, respectively. The degree of morphing, which is the matching measure used in recognition, is defined in the following way:

$$E_a(S^I, S^T) = E_{total}(S^I, S^T) + f\ |n_I - n_T|, \tag{2.37}$$

where the factor f exploits the *a priori* knowledge that similar shapes tend to have the same number of segmentation points. Its value is found experimentally for each alphabet or other symbol set.

The method was tested with one hundred cursive words and seven hand-drawn line figures written by twelve subjects. There were four test samples for each reference sample. The recognition results of the writer-dependent tasks varied from 89.7% up to 99.0%.

2.6.3 Structural and syntactical methods

Structural and syntactical methods are used for various pattern recognition tasks in which the structure of the pattern is paramount and can be used for classification. The knowledge about the existence or absence of certain features in a pattern does not always provide enough information for the classification and some additional information on the relations between the features, or the structure of the pattern, is needed.

In syntactical methods, the features found in the patterns are regarded as primitives of a formal language. The language consists of grammatic rules which define the possible relations of the features, or in other words, define the allowable structure of the pattern. The very basic features are represented by the terminal symbols of the language. The nonterminal symbols of the language are formed from the terminal symbols according to the grammar. The recognition of patterns is carried out by parsing. Each class has its own grammatic rules and the pattern is classified to the syntactically corresponding class. The grammatic rules can be presented in two ways: as a string grammar or a graphical grammar. A string grammar defines in a one-dimensional way how a string of nonterminal symbols can be replaced by a string of nonterminal or terminal symbols. Graphical grammars can present the relationships of the features in a higher-dimensional way (Schalkoff 1992).

Syntactical methods have one major drawback: if the structure of an unknown pattern does not exactly match to any of the structures of the predefined classes, a syntactical classifier cannot say anything about the class. Not even which class would be close to the pattern, as no sensible distance measure can be defined between the pattern instances. Serious problems also arise if the representatives of two different classes can have similar structures. In such cases, some additional information or features are needed to discriminate the classes.

Example 12: A good and simple example of a syntactical method is Picture Description Language (PDL) introduced by A. C. Shawn in 1969. The primitives of PDL are straight line segments with heads and tails, four directions, and three lengths. The grammatic rules describe how the line segments can be connected to each other from their heads or tails. The grammatic rules can be presented as a tree in which the leaves correspond to the line segments and the nodes to their connections (Wang and Gupta 1991).

Structural methods are also based on an analysis of the features and their relations. They differ from the syntactical methods as the classification is not based on parsing but on matching and various decision rules. The characters can be recognized hierarchically: the candidate classes are first ruled out or selected according to lower-level features. Then, more complicated higher-level structures are used. The pattern is classified to the class which has matching structure. In case of ambiguity, decisions rules containing context and language information, or rules specific to the confusing class pairs can be used. The hierarchical matching and decision procedures can usually be described with decision trees. Two structural methods designed for on-line recognition of isolated characters are described in the following examples.

Example 13: Chan and Yeung (1998) have proposed a structural method in which the structure of the unknown pattern can be deformed if it does not directly match any of the patterns found in the model base. The method is called Elastic Structural Matching. The elementary structures found in the characters are: a line segment, loop, dot, and counter-clockwise and clockwise curves. The structures, excluding loops and dots, are associated with a quantified direction from the starting point to the end point of the corresponding segment. A character is composed of a sequence of the elementary structure types and their directions. The elements of the sequence are in the same order as the have been drawn.

The recognition is carried out on the basis of the first matching model found in the model base. First, the original pattern structure is matched against all the models. If no matching model is found, the structure of the pattern is modified. The structure is modified at three levels and the matching procedure is repeated after each level. At the first level, the primitive types are modified: a line may be changed into either of the curves, and similarly, both curves can be replaced with a line. If still no match is found, the directions of the original structures are modified: a direction can be changed into either of its neighboring directions. At the last level, both the structure types and their directions are modified simultaneously.

This method was tested with multi-writer data and the recognition rates for the digits, lower and upper case letters, and all the character classes together were 98.6%, 98.5%, 97.4%, and 97.4%. The test data was written by 150 subjects and contained one sample per class written by each subject.

Example 14: A hierarchical method based on structural features of three levels, binary decision trees, and matching was described by Kerrick and Bovik (1988). First, the low-level features of the pattern to be recognized are examined. The low-level features consist of the existence and locations of the endpoints and tees of the strokes. The tees are points where two strokes touch each other. The locations can have only four values which correspond to the quadrants of the bounding box. The low-level features are easy to extract and, as they contain both local and global structural information on the character, they are very effective for the pruning of the classes. Next, the intermediate-level features, the corners and their locations, are found by a chain code-based algorithm developed by Freeman and Davis (1977). A corner contains more global information than an endpoint or a tee as it is not so locally defined.

In the first stage of recognition, the absence or presence of particular low- and intermediate-level features in given quadrants are tested according to a binary decision tree. In the next stage, the recognition is carried out by matching the unknown pattern against the remaining candidate set. The matching is performed on stroke basis and thus the candidate set is further reduced by a parsing algorithm which attempts to establish a stroke-to-stroke correspondence between the two patterns. In the matching stage, high-level features are used. Each stroke is associated with a shape which can be either straight or curved. A straight stroke can have four orientations. The curved strokes are divided into four segments of equal length. The average chain code value is computed for each segment and the shape can be coded with four digits. Two strokes are considered similar if they both are straight and have same orientation, or, if the digits of their shape codes are the same or differ only by one unit. To handle the ambiguity of several remaining candidates after the matching, the aspect ratios of the characters are also compared and some additional class dependent feature tests are performed.

2.6.4 Neural networks

Recognition systems based on neural networks (NN) use a similar basic idea as those based on model matching methods. Instead of explicitly storing models in a model base, the information on the models is stored implicitly in the weights of the neurons. The input of the network consists of the features extracted from the patterns to be recognized and the output can be the scoring values for each class or simply the index of the best-matching class. NNs are widely used for various pattern recognition tasks. Several NN-based applications and methods for handwriting recognition are described in a journal issue edited by Guyon and Wang (1993). In most cases, NNs are used for optical character recognition and the input consists of preprocessed images or different features which are extracted from them. If the handwritten data collected on-line is converted into a image, all the dynamic information will be lost. However, no background noise needs to be removed, and the recognition rates can still be expected to be better than those for scanned images. Due to the black-box nature of some neural-network models, it is difficult to analyze their recognition process and increase the recognition performance by finding and elimating the error sources. In addition, they need relatively large training databases. HMMs have similar drawbacks compared to model matching.

Example 15: Mozayyani et al. (1998) have introduced an on-line recognition method based on a NN which uses time-dependent information extracted from the pen point loci as an input. The recognition system is able to recognize characters consisting of one stroke and written in a prespecified style. A digital tablet records the coordinates of the moving pen point. The origin of the coordinate system is moved to the first sample point where the pen is pressed down and a stroke begins. All sampled points, except the first one, are projected on the horizontal x_1-axis and vertical x_2-axis. The values and signs of changes of the projection lengths are calculated for each sample point:

$$x = (\triangle x_{left}, \triangle x_{right}, \triangle x_{up}, \triangle x_{down}). \tag{2.38}$$

The components of the above vector have non-negative real values. If the components are converted into complex numbers with constant amplitudes but time varying phases, two data points can be distinguished by their sampling times (t) in addition to their locations relative to the preceding data points (2.38). The conversion is performed in the following way:

$$\xi = (\xi_{left}, \xi_{right}, \xi_{up}, \xi_{down}), \tag{2.39}$$

$$\xi_d = \triangle x_d e^{i \tan^{-1}(\kappa t)}, \tag{2.40}$$

where κ is an inverse time constant.

The converted data points are the input of a NN consisting of special spatio-temporal neurons. Each component is fed to a group of neurons in the input layer which have no other inputs. The outputs of all the neurons in the input layer are fed to all the neurons in the hidden layer. The hidden-layer neurons send their outputs to all the ouput-layer neurons. In other words, the hidden and output layer form a Multi Layer Perceptron (MLP). The number of the output neurons is the same as the number of classes. Each different writing style of a character constitutes a separate class.

A spatio-temporal neuron works as an accumulator. The output of the neuron is zero until the absolute value of the cumulative sum of the inputs reaches a predefined threshold value. After that, the accumulated value is sent out as an output, after which both the accumulator and its output are reset to zero. The neurons of the input layer are thus able to detect line segments of certain direction and length dependent on the threshold value. The input neurons accumulate the projection differences and the MLP network operates on their outputs once all the data points of the character have arrived in the input neurons.

The time delays between the inputs can be taken into account if the values of the inputs already arrived are rotated in the following way every time a new input arrives:

$$D(\xi_d) = \xi_d e^{-\mu_S \triangle t + i \tan^{-1}(\mu_T \triangle t)}, \tag{2.41}$$

where ξ_d is the original value of an input, $\triangle t$ is the time elapsed since its arrival, and μ_S and μ_T are positive constants. The operator D attenuates the amplitude of an input and changes its phase further from zero as the time passes. Due to this operation, the neurons are less sensitive to noise as they tend to forget stray inputs with small amplitudes. The time delays are considered only in the neurons of the input layer in order to keep the model simple.

The recognition method was tested with two multi-writer databases. The first database consisted of 26 lower case Latin characters written 10 times by each of 20 subjects. Only one writing style was allowed per class. The second database was otherwise similar but the number of writers was 15 and two writing styles were allowed. Both databases were subdivided into equal-sized training and test sets. In both experiments, the input layer of the NN consisted of three neurons for each of the four direction. The initial values for the thresholds were calculated from the size of the writing area and were adjusted by experience. The weights and thresholds used in the MLP were found by a training algorithm. The hidden layers of the NN consisted of 26 neurons and the number of neurons in the output layer was 26 for the one-style and 52 for the two-styles case. The recognition rates for the whole databases were 99% after 5 000 training iterations in the one-style case and 100% after 2 000 training iterations in the two-styles case.

Chapter 3

Description of the recognition system for handwritten characters

In this chapter, the on-line recognition method for isolated characters applied in this work is described. The method is based on prototype matching. Various distance metrics can be used for the matching as the examples in the previous chapter have shown. In this chapter, some distance metrics are examined in order to find the most suitable, or in other words, the one which yields the best recognition results for our data. All the distances considered are based on DTW and are therefore suitable for nonlinear curve matching. The decision criterion of the classifiers is the k Nearest Neighbor rule (k-NN rule). The prototype set is formed by clustering character samples written by multiple subjects.

In the following section, the decision criterion and the whole matching procedure are described in more detail. Next, the creation of the prototype set is explained. The requirements for the prototype set and problems in achieving them are also discussed. Last, the distance metrics used both in clustering and matching are defined.

3.1 Classifier

The basic idea behind the classifier is very simple: a prototype set is formed to cover the various writing styles of each character class, an unknown character written by the user is matched against all the prototypes, and the classification is carried out according to the best matching, or the nearest, prototypes.

Both the prototypes and the unknown characters consist of the loci of the pen point coordinates. Only coordinate points corresponding to the movements of the pen while it is pressed down against the writing surface are used in matching. These coordinate points are preprocessed and normalized before they are sent to the classifier. The pen up movements between the pen down movements are also captured but this information is used only for segmenting the character into strokes. No other information or features besides the coordinates are used for the classification – neither a language model nor a lexicon.

3.1.1 k Nearest Neighbor rule

The classifier used in this work is based on the k Nearest Neighbor rule first introduced by Fix and Hodges (1951) and further analyzed by Cover and Hart (1967). The k-NN rule is a data-driven approach and a completely parameter free method as it does not rely on any assumptions on the underlying distributions of the character classes. The classification is based only on the known classifications of the training or design samples which form the prototype set. The classification of an unknown character is made according to the majority of its k nearest prototypes in the prototype set. The volume of the region spanned by the k nearest neighbors varies with the density of prototypes near the unknown – in the areas of high density it is small and, conversely, in the areas of low density it is large. This is the main difference compared to the other nonparametric classification methods, for example the Parzen window-method (Duda and Hart 1973), which try to estimate the class distributions with the observed frequencies of classes in constant-sized areas. The mathematical background of the k-NN rule is described next (Schalkoff 1992).

The k-NN rule is based on the following assumption: as the number of the training samples n increases to infinity, the number of training samples k_n which fall into a given region V_n also increases to infinity, or

$$\lim_{n\to\infty} k_n \to \infty, \tag{3.1}$$

so that the ratio k_n/n approaches the probability that the sample falls in that region. The value of the density function $p_n(x)$ in the neighborhood of x derived from that assumption is

$$\begin{aligned} p_n(x) &= \frac{P(\text{a sample falls in the neighborhood of } x \text{ with volume } V_n)}{V_n} \\ &= \frac{k_n/n}{V_n} \\ &\approx p(x). \end{aligned} \tag{3.2}$$

The conditional probabilities for each class ω_i can be obtained with the previous equation and the Bayes rule (Råde and Westergren 1990). According to the Bayes rule,

$$P(\omega_i|x) = \frac{P(\omega_i)P(x|\omega_i)}{P(x)} = \frac{P(x,\omega_i)}{P(x)} = \frac{P(x,\omega_i)}{\sum_{j=1}^{C} P(x,\omega_j)}, \tag{3.3}$$

where C is the number of classes.

Equation (3.2) can be used for estimating $P(x, \omega_i)$ in the following way:

$$P(x,\omega_i) \approx \frac{k_i/n_v}{V}, \tag{3.4}$$

where n_v is the total number of the training samples fallen in region V. The number of those samples which belong to the class i is k_i.

If equation (3.4) is substituted into equation (3.3), the following approximation of $P(\omega_i|x)$ will be obtained:

$$P(\omega_i|x) \approx \frac{(k_i/n)/V}{\sum_{j=1}^{C}(k_j/n)/V} = \frac{k_i}{k}. \tag{3.5}$$

As has been shown above, the k-NN rule corresponds to classification carried out by choosing the most probable class according to the conditional probabilities. Classification can be performed without making any assumption on the shapes of the class distributions or estimating their parameters. Cover and Hart (1967) have shown that any other k-NN rule, $k \neq 1$, does not have an error probability lower than the 1-NN rule if every interclass distance is greater than any intraclass distance. If the classes are overlapping in the feature space and the training set is large, a k-NN rule with $k \neq 1$ performs better than the 1-NN rule as it gives more accurate estimates of the *a posteriori* probabilities of the classes. Cover and Hart (1967) have also proved that the error probability of the k-NN rule has both upper and lower limits. If R^* is the optimal error probability obtained with the Bayes rule and exactly known class distributions, the bounds of the error probability of the k-NN rule, R, are given by the following inequality when the size of the training set approaches infinity:

$$R^* \leq R \leq R^*(2 - \frac{M}{M-1}R^*), \tag{3.6}$$

where M is the number of classes.

The drawbacks of the k-NN rule are its storage requirements and the computation time which is linearly dependent on the size of the prototype set which consists of all the training samples. The Condensed Nearest Neighbor rule (CNN rule), also introduced by Hart (1968), is a lighter version of the k-NN rule – only some of the training samples are used as prototypes. The CNN rule classifies unknown patterns in a similar manner as the k-NN rule but it is based on a consistent subset of the training samples instead of the whole training set. A consistent subset is a set of training samples which correctly classifies the remaining training samples when the 1-NN rule is applied. A consistent subset can be found by using an iterative algorithm described by Hart (1968). The algorithm selects the training samples which are located near the boundaries of the classes if the Bayes error probability is low, or in other words, classes are quite separate. But in the areas of high risk of error, almost all training samples will be selected.

The classification rule used in this work resembles the CNN rule: they differ as the prototype set is formed by a clustering algorithm instead of the original condensing algorithm. The performance of the CNN rule crucially depends on the limited set of training samples constituting the prototype set. In addition, the analysis of the error probability is more complicated: as assumption (3.1) is not valid, inequality (3.6) does not necessarily apply.

3.1.2 Character prototype pruning and ordering

Computational savings can be achieved if the character prototypes are pruned or ordered prior to the matching phase. The order of the prototypes does not naturally have any effect on the recognition accuracy. Prototype pruning means that the unknown pattern is matched against only a subset of the prototypes. Prototypes used in matching are selected according to some additional information derived from the loci of the pen movements. For example, the number of strokes or the existence of corners or some other higher level features can be examined. Prototype pruning does not necessarily affect the recognition. However, pruning can significantly decrease the time consumed in the matching procedure if features which certainly exist or are absent in some classes can be defined. Unfortunately, this is not always the case. Still, the effect of the pruning can be modest on the recognition rate but significant on the recognition time.

In this work, prototypes are pruned according to their number of strokes – only characters with the same number of strokes are matched. In fact, the matching is performed on stroke basis. The strokes are determined by the pen lifts. Arakawa (1983) and Tappert (1991) have used similar pruning methods. Tappert (1991) has examined the effects of pruning on the recognition performance of a system based on elastic matching. In addition to the number of the strokes, in that work, prototypes are pruned by the ratio of the lengths of the corresponding strokes: if the ratio does not lie between 0.5 and 2, the prototype is pruned. Also, linear matching is performed prior to the elastic matching and the prototypes further than a given threshold distance are pruned. The pruning decreases recognition performance only slightly: the average recognition error of nine writers increases from 2.7% to 2.8% as the time consumed in matching decreases 56.2%. In the light of these results, pruning of the prototypes seems to be beneficial.

In addition to pruning, in this work, the prototypes are ordered according to their categories. Each characters can be placed to one of sixteen categories depending on the rough shape of its first stroke. In particular, the category is defined by the quadrants of the coordinate plane in which the starting and ending points of the first stroke are located. First, the origin of the coordinate systems is moved to the center of the character by either one of the translation normalization operations described in Chapter 5. Next, a four-bit binary value $b_4b_3b_3b_1$ is constructed for the category in the following way:

$$
\begin{aligned}
b_1 &= (x_1(1) \geq 0) \\
b_2 &= (y_1(1) \geq 0) \\
b_3 &= (x_1(N_1) \geq 0) \\
b_4 &= (y_1(N_1) \geq 0),
\end{aligned}
$$

where N_1 is the number of data points (x_1, y_1) in the first stroke, and true value of an expression corresponds to one and false to zero bit. Thus, sixteen categories are defined. If the value of one of the bits changes, either the starting or ending point moves to a neighboring quadrant. The distance between two categories is the count of bit differences in their binary presentations and its integer value lies between zero and four. Prototypes with a low category distance to the unknown character are, naturally, matched first. The ordering of the prototypes saves computation time as the unknown character is not completely matched against all the prototypes. If the distance measure between the unknown and =a prototype exceeds the distance to the kth nearest prototype among the preceding ones the matching procedure can be interrupted, even though all the points or strokes have not been matched. In some earlier experiments, in roughly 70% of all case, the nearest prototype belonged to the same category as the unknown character.

3.2 Selection of prototype set

Only a subset of all training samples is used as the prototype set. The subset is formed by clustering the training samples and selecting one sample from each cluster to represent all the samples in that cluster. Clustering can be performed in several ways but its main aim is always to form groups of characters so that the characters belonging to same group are, in some sense, as similar as possible. In the following sections, general requirements for the prototype set are discussed and the clustering method used in this work is described.

3.2.1 Requirements for prototype sets

A set of prototypes is used instead of the whole set of training samples in order to decrease recognition time and memory consumption. The recognition error probability of the k-NN classifier cannot in general be reduced. On the contrary, in order to lower the error probability, prototypes should be added, not omitted. But, if the prototypes are cleverly selected, the negative effects can be minimal. The most important requirement for the prototype set is sufficient coverage: there should be at least one prototype per a fundamentally different writing style of a character. If the recognition method is sensitive to the number, order, and direction of the strokes, as the method used in this work, each variation needs its own prototype.

Prototypes should be separable. If two almost similar prototypes present two different character classes, confusion between them is guaranteed. This problem occurs both between- and within-writer. The latter case is easy to solve by adapting the prototype set by omitting confusing but otherwise needless prototypes. The former case is, at least to some extent, insolvable.

In addition to inseparable prototypes, maverick prototypes should also be avoided. A maverick is an atypical character which is not written as intended. It is not a good representative of any writing style. The training set should be checked for malformed characters if the prototype selection algorithm tends to pick them. Prototypes should be typical or, for example, an averaged characters of a group of similar characters.

3.2.2 Semiautomatic clustering method

Clustering means that the data is divided into groups and items belonging to same cluster are in some sense similar. To make clustering possible, some kind of similarity measure, or distance, between the items has to be defined. The choice of this measure naturally has paramount effect on the clustering results. In rough mathematical terms, clustering means finding partition P of data set H that minimizes intracluster distances and maximizes intercluster distances. These two aims can be combined into a clustering criterion function $J(P)$. Partition P consists of subsets $\{H_i\}$ of data set H, so that $\bigcup_i H_i = H$ and $H_i \cap H_j = \emptyset$ if $i \neq j$. The number of feasible partitions of n items is

$$\frac{1}{c!} \sum_{k+1}^{c} (-1)^{c-k} k^n \approx \frac{c^n}{c!}, \tag{3.7}$$

where c is the number of clusters. It is evident that exhaustive methods in which all possible partitions are tested are impractical (Schalkoff 1992).

The clustering algorithm used in this work is a semiautomatic method. This means that the number of clusters must be predefined. In the current system, the number of prototypes is the same for each class. Since the matching is performed on stroke basis, there should be at least as many prototypes per class as there are stroke number variations. The number of different writing styles per character class and stroke number variation needs to be manually examined. The number of prototypes per stroke number variation is selected so that it roughly corresponds to the respective share of all the writing styles of that character. For example, the number of writing styles found for character 'A' with one, two, and three strokes were 1, 2, and 1, respectively. If the total number of prototypes per class is set to seven, the number of corresponding prototypes is 2, 3, and 2.

After deciding the number of prototypes, training samples are divided into clusters by an automatic and iterative algorithm. Prior to the clustering, training samples are preprocessed and normalized in the same way as the unknown characters in the recognition phase. The same distance measure is also used. The clustering algorithm starts with one cluster containing all the training samples belonging to the same class and having the same number of strokes. The number of clusters is increased iteratively until the predefined amount is reached or the samples run out. The iterative algorithm starts by finding the centers of the clusters or in other words, items which yield the minimum sum of distances to all other items belonging to the same cluster. After that, all the items are ordered to an increasing series $\{x_1, x_2, \ldots, x_N\}$ according to their distances to the center items of their clusters. A new cluster is formed of the items furthest from their cluster centers. The number of items in it is determined by minimizing splitting criterion function $J(i)$ which is defined in the following way:

$$J(i) = d(x_i, \overline{x}_{old}(i)) + \max_{i \leq j \leq N} d(x_j, \overline{x}_{new}(i)), \tag{3.8}$$

where x_i is an item with order index i and $\overline{x}_{old}(i)$ is the old center item of its cluster, $\overline{x}_{new}(i)$ is the center item of the new cluster consisting of items $\{x_i, \ldots, x_N\}$, and function $d(x, y)$ is the distance measure between two items x and y. In the ideal case when only two different writing style exist, the splitting criterion function is roughly U-shaped as the first term increases and the second term tends to decrease with index i. If there are more than two writing styles, the splitting criterion function can have several local minimums. A new cluster is formed in the following way: if

$$J(i^*) = \min_i J(i), \tag{3.9}$$

items which have order index $i \geq i^*$ are included in the new cluster. Before the next round of the clustering algorithm, the cluster centers are recalculated and items are assigned to the clusters according to the cluster centers nearest to them. This is repeated until a stationarity partition is reached (Laaksonen et al. 1998a).

3.3 Metrics used for matching and clustering

As the pen position is sampled with a constant sampling frequency and the size of the characters, writing speed, and style vary, the number of data points per character is definitely not a constant. Even if the movements of the pen are resampled in order to get spatially equidistant data points, the number of data points would be nearly same only for very similar-shaped strokes of same size. Therefore, the metric defined in the character space and used as a dissimilarity measure should be defined between strokes which do not have same number of data points. In addition, the dissimilarity measure should be independent of the number of data points and, if possible, symmetric.

In this work, several distance metrics, all based on DTW and thus defined between curves of varying numbers of data points, have been tested. All the distances metrics are defined on stroke basis and are symmetric. If the numbers of strokes in two characters are different, the distance between the characters is defined to be infinite. Otherwise, it is the sum of the distances between the corresponding strokes. The continuity and boundary conditions of DTW are symmetric for all the distance metrics. The main differences between the distance metrics are the associated costs of matching data points. The distance metrics used in this work are addressed in detail in the following sections.

3.3.1 Point-to-point distance

Point-to-point distance (PP-distance) is the most simple distance metric applied: the associated cost function between two data points is their squared Euclidean distance. The distance between two strokes S_1 and S_2 and the optimal time warping path are searched for by a dynamic programming algorithm. The time warping path $P(h) = (i(h), j(h))$, $1 \leq h \leq H$, describes how the point-to-point correspondence is established: h is an order index of matching of the $i(h)$th and $j(h)$th data points of strokes S_1 and S_2, respectively.

$P(h)$ has both boundary (3.10) and continuity (3.11) constrains. The first and last data points of strokes $S_1 = (p_1(1), \ldots, p_1(N_1))$ and $S_2 = (p_2(1), \ldots, p_2(N_2))$ are matched against each other:

$$\begin{aligned} P(1) = P(i(1), j(1)) = (1, 1), \\ P(H) = P(i(H), j(H)) = (N_1, N_2). \end{aligned} \tag{3.10}$$

All data points are matched at least once and several data points can be matched against one:

$$(\triangle i(h), \triangle j(h)) = (i(h) - i(h-1), j(h) - j(h-1)) = \begin{cases} (1,0) \text{ or} \\ (0,1) \text{ or} \\ (1,1). \end{cases} \tag{3.11}$$

The associated cost d between two data points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is the squared Euclidean distance,

$$d(p_1, p_2) = (x_1 - x_2)^2 + (y_1 - y_2)^2. \tag{3.12}$$

The distance measure D_{PP} between two strokes S_1 and S_2 and the point-to-point correspondence are found by minimizing the sum of all associated costs between the matched data points:

$$D_{PP}(S_1, S_2) = \min_{P(h)} \sum_{h=1}^{H} d(p_1(i(h)), p_2(j(h))). \tag{3.13}$$

Problem (3.13) can be solved with dynamic programming in the following way:

$$D_{PP}(i,j;h) = d(p_1(i),p_2(j)) + \min \begin{cases} D_{PP}(i-1,j;h-1) \\ D_{PP}(i,j-1;h-1) \\ D_{PP}(i-1,j-i;h-1), \end{cases} \tag{3.14}$$

when $D_{PP}(i,j;0)$ s are initialized to zero

$$D_{PP}(i,j;0) = 0, \tag{3.15}$$

and the undefined distances are set to infinity

$$d(p_1(i),p_2(j)) = \infty \ \text{, if } i \leq 0, j \leq 0,\ j > N_1, \text{ or } j > N_2. \tag{3.16}$$

The solution to (3.14) is given by

$$D_{PP}(S_1,S_2) = D_{PP}(N_1,N_2;H). \tag{3.17}$$

$D_{PP}(i,j;h)$ is the distance measure of optimally matched stroke parts consisting of points $(p_1(1),\ldots,p_1(i(h)))$ and points $(p_2(1),\ldots,p_2(j(h)))$ and h is the number of required point-to-point matchings.

As both the boundary and continuity constrains and the associated cost function of matching two data points are symmetric, the whole distance metric D_{PP} is symmetric.

The matching of strokes $A = (a(0),\ldots,a(N))$ and $B = (b(0),\ldots,b(M))$ is illustrated in Figures 3.1 and 3.2. The dynamic programming algorithm described above finds the optimal time warping path using a table $T(A,B)$ holding all the distances $D(i,j;h)$ corresponding to optimal subpaths and the entering direction to each cell, $r = \{horizontal, diagonal, vertical\}$. For example, the value of the cell (n,m) is $(D_{PP}(n,m;h),r) = (D_{PP}(n-1,m-1;h-1) + d(a(n),b(m)), diagonal)$ as

$$\begin{aligned} &\min\{D_{PP}(n-1,m-1;h-1), D_{PP}(n,m-1;h-1), D_{PP}(n-1,m;h-1)\} \\ &\qquad = D_{PP}(n-1,m-1;h-1). \end{aligned} \tag{3.18}$$

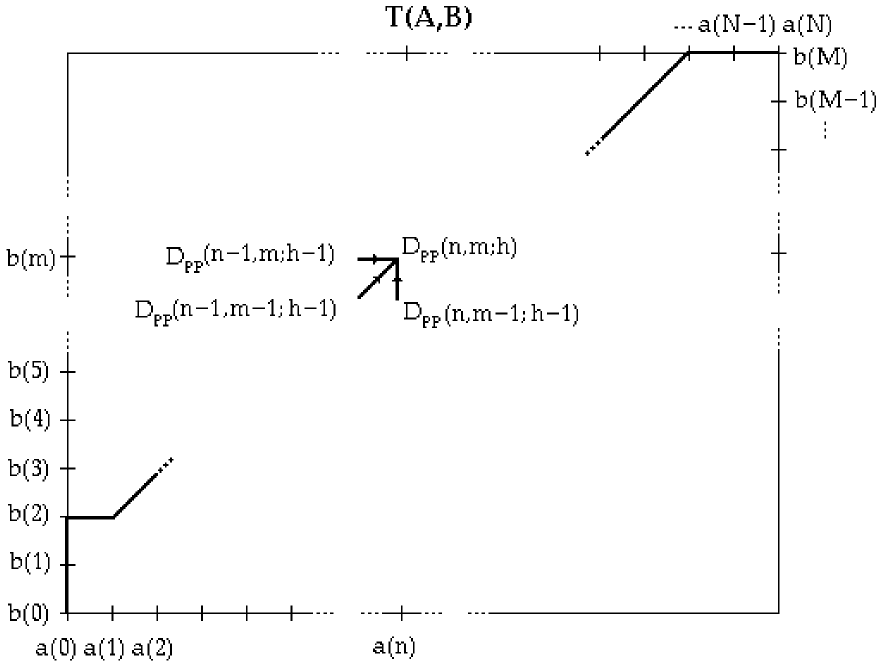


Figure 3.1: *Table $T(A, B)$ for the optimal subpaths in the point-to-point matching of strokes $A = (a(0), \ldots, a(N))$ and $B = (b(0), \ldots, b(M))$.*

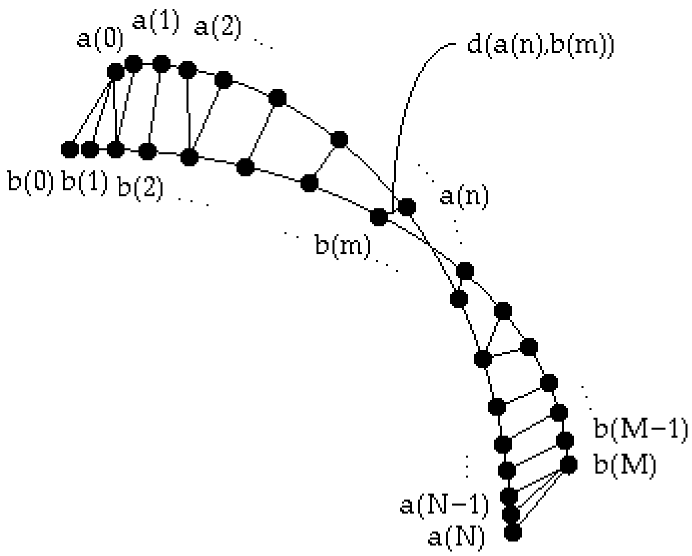


Figure 3.2: *Point-to-point matching of strokes A and B seen in the xy-coordinate plane.*

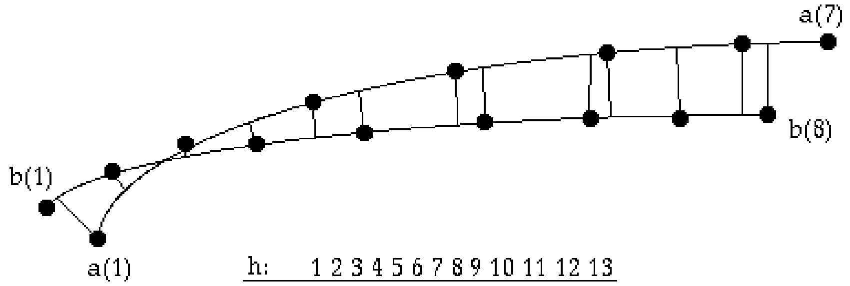
3.3.2 Point-to-line distance

Point-to-line distance (PL-distance) is a modified version of the point-to-point distance: data points can also be matched to lines interpolated between successive data points (Sankoff and Kruskal 1983). The idea behind this modification is to reduce the effects of phase differences in the sampling of two similarly-shaped curves. Considerable phase differences can occur when the sampling frequency is low and the pen point moves fast. They can cause greater increase in the PP-distance measure than slight changes in the shapes of the characters. As the sampling and writing processes are totally independent of each other, the dissimilarity measure between two similarly-shaped characters should always be small regardless how the samplings are timed. If the sampling frequency is too low, a linear interpolation between data points can cause cumbersome shapes, especially in the areas of high curvature where originally smooth trajectories are presented with piecewise linear curves. On the other hand, if sampling rate is high enough, the effects of the phase differences are meaningless and the point-to-line matching gives more degrees of freedom to the matchings of the data points.

When the PL-distance is calculated by using DTW, all points, except the first and last ones, are matched once. Lines are interpolated between adjacent data points and the data points of the other character are matched to the nearest point on these lines (see Figure 3.3). The associated cost of matching a single point is the minimum squared Euclidean distance from the point to the interpolated line. PL-matching has both boundary and continuity constrains. The continuity constrains are defined with five functions: $I(h)$, $i(h)$, $J(h)$, $j(h)$, and $r(h)$. Function $I(h)$ indicates whether the point of stroke S_1 involved in the hth matching is a data point or point on an interpolated line. Function $i(h)$ gives the index of the first data point of stroke S_1 which has not been used in the first $h-1$ matchings. Functions $J(h)$ and $j(h)$ are defined similarly for stroke S_2. Function $r(h) = [0, 1]$ gives the relative location of the interpolated point involved in the hth matching. For each matching $h = 1, \ldots, N_1 + N_2 - 2$, functions $I(h)$ and $J(h)$ indicate that 1) data point $p_1(i(h))$ and a point interpolated between data points $p_2(j(h))$ and $p_2(j(h)+1)$, or, 2) data point $p_2(j(h))$ and a point interpolated between data points $p_1(i(h))$ and $p_1(i(h)+1)$, are matched against each other. The continuity constraints consist of a set of eight conditions. For functions $I(h)$ and $i(h)$, it is required that:

1. If both $I(h)$ and $I(h+1)$ indicate data points, the data points are adjacent and thus $i(h+1) = i(h) + 1$.

2. If $I(h)$ indicates a data point and $I(h+1)$ indicates an interpolated point,



h:	1	2	3	4	5	6	7	8	9	10	11	12	13
I(h):	1	0	1	0	1	0	1	0	0	1	0	1	0
J(h):	0	1	0	1	0	1	0	1	1	0	1	0	1
i(h):	1	1	2	2	3	3	4	4	4	5	5	6	6
j(h):	1	2	2	3	3	4	4	5	6	6	7	7	8

Figure 3.3: *Point-to-line matching of two strokes.*

then $i(h+1) = i(h)$.

3. If $I(h)$ indicates an interpolated point and $I(h+1)$ indicates a data point, then $i(h+1) = i(h) + 1$.

4. If both $I(h)$ and $I(h+1)$ indicate interpolated points, then $i(h+1) = i(h)$.

Functions $J(h)$ and $j(h)$ satisfy similar conditions.

The boundary constrains are as follows: at the beginning of the strokes, the first point $p_k(1)$ of stroke S_k is matched to a point on the line interpolated between the first point $p_l(1)$ and the second point $p_l(2)$ of stroke S_l, and point $p_l(1)$ is not matched at all. Similarly, in the ends of the strokes, the last point $p_k(N_k)$ of stroke S_k is matched to a point on the line interpolated between the second last point $p_l(N_l - 1)$ and the last point $p_l(N_l)$ of stroke S_l, and point $p_l(N_l)$ is not matched at all. In both boundary conditions, $k, l \in \{1, 2\}$ and $k \neq l$. With functions $I(h)$, $i(h)$, $J(h)$, and $j(h)$, these conditions can be written in the following way:

1. $i(1) = j(1) = 1$.

2. If $I(N_1 + N_2 - 2)$ indicates a data point, then $i(N_1 + N_2 - 2) = N_1 + 1$ and $j(N_1 + N_2 - 2) = N_2$, or, if $J(N_1 + N_2 - 2)$ indicates a data point, then $j(N_1 + N_2 - 2) = N_2 + 1$ and $i(N_1 + N_2 - 2) = N_1$.

As the boundary and continuity conditions and the associated cost function of matching data points are symmetric, PL-distance is also symmetric.

The nearest point (x_i, r) to the point y on the line between points x_i and x_{i+1} is given by the following equation:

$$(x_i, r) = (1-r)x_i + rx_{i+1}. \tag{3.19}$$

The relative location parameter r is defined as follows:

$$r = \begin{cases} 1, & \text{if } \overline{r} \geq 1 \\ 0, & \text{if } \overline{r} \leq 0 \\ \overline{r} & \text{otherwise} \end{cases}, \tag{3.20}$$

where

$$\overline{r} = \frac{(y - x_i) \cdot (x_{i+1} - x_i)}{(x_{i+1} - x_i) \cdot (x_{i+1} - x_i)}. \tag{3.21}$$

The PL-distance measure is also solved with dynamic programming and the recursive equation is

$$D_{PL}(i, j, r; h) = \min \begin{cases} D_{PL}(i-1, j, r; h-1) + d(p_1(i), (p_2(j), r)) \\ D_{PL}(i, j-1, r; h-1) + d((p_1(i), r), p_2(j)) \end{cases}. \tag{3.22}$$

$D_{PL}(i, j, r; 0)$s are initialized to zero

$$D_{PL}(i, j, r; 0) = 0, \tag{3.23}$$

and the undefined distances are set to infinity

$$d(p_1(i), (p_2(j), r)) = d((p_1(i), r), p_2(j)) = \infty \text{ , if } i, j \leq 0,\ i > N_1, \text{ or } j > N_2. \tag{3.24}$$

Function $d((p_k(i), r), p_l(j))$ is the associated cost of matching point $p_l(j)$ to the closest point on the line interpolated between points $p_k(i)$ and $p_k(i+1)$.

The solution to equation 3.22 is given by:

$$D_{PL}(S_1, S_2) = D_{PL}(N_1 - 1, N_2 - 1, r; N_1 + N_2 - 2) + \min \begin{cases} d(p_1(N_1), (p_2(N_2 - 1), r)) \\ d((p_1(N_1 - 1), r), p_2(N_2)) \end{cases}. \tag{3.25}$$

3.3.3 Normalized distances

Both of the two distances described in the previous sections are defined for a pair of strokes consisting of any number of data points. However, those distances are not insensitive to the number of data points: as they are sums of the costs associated with the matchings, either point-to-point or point-to-line, they increase as the number of data points increases. Therefore, the distance between two similarly-shaped strokes, one written slowly and the other quickly, is greater than the distance between two strokes written quickly. A drawback of this feature is that the strokes are not only discriminated by their shape but also by the dynamic nature of their generation. On the other hand, it might be an advantageous feature if the system adapts to writing style of its primary user. Another feature of PP- and PL-distances is that the strokes of different length are not considered equally important. As the distance between two characters is the sum of the distances between corresponding strokes, short strokes consisting of few data points are not as important as longer strokes because they contribute less to the total distance. These two features can be avoided if the distance between strokes is normalized by the number of the matchings. Two normalized distances, which are called normalized point-to-point distance (NPP-distance) and normalized point-to-line distance (NPL-distance), are used in this work. They are defined as follows:

$$D_{NPP}(S_1, S_2) = D_{PP}(S_1, S_2)/H, \quad \max(N_1, N_2) \leq H \leq N_1 + N_2, \tag{3.26}$$

and

$$D_{NPL}(S_1, S_2) = D_{PL}(S_1, S_2)/H, \; H = N_1 + N_2 - 2, \tag{3.27}$$

where H is the number of point-to-point or point-to-line matchings.

3.3.4 Area-based distances

PL- and PP-distances do not consider or compensate for the density of the data points. Therefore, the parts where the pen point moves slowly and more data points are sampled, such as the beginnings and ends of the strokes, areas of high curvature, and corners, are emphasized. Another way, besides the normalization by the number of matchings, to avoid this is to use cost functions which are insensitive to the point density. The distances described in the previous sections use the squared Euclidean distance as a cost associated with each matching. If the area between the matched stroke parts is used as a cost function, the density of the data points does not affect the distance measure. Contrary to NPP-and NPL-distances, the lengths of the strokes are still meaningful as the size of the area between the strokes depends on the lengths of the strokes.

In this work, two area-based distances are defined. The first of them, simple-area distance (SA-distance) has similar boundary and continuity constrains as PP-distance, but the area spanned by three or four data points is used as an associated cost instead of the squared Euclidean distance. Except for the first matching, each point-to-point matching produces a triangle or a quadrilateral between the strokes. Let $S_1 = (p_1(1), \ldots, p_1(i(h)), \ldots, p_1(N_1))$ and $S_2 = (p_2(1), \ldots, p_2(j(h)), \ldots, p_2(N_2))$ be the strokes to be matched. In every new point-to-point matching, the index numbers i and j of the matched data points change in one of the following ways:

1. $(\triangle i(h), \triangle j(h)) = (1, 0)$ and a triangle spanned by $p_1(i(h))$, $p_1(i(h-1))$, and $p_2(j(h))$ is formed, or
2. $(\triangle i(h), \triangle j(h)) = (0, 1)$ and a triangle spanned by $p_1(i(h))$, $p_2(j(h-1))$, and $p_2(j(h))$ is formed, or
3. $(\triangle i(h), \triangle j(h)) = (1, 1)$ and a quadrilateral spanned by $p_1(i(h))$, $p_1(i(h-1))$, $p_2(j(h))$, and $p_2(j(h-1))$ is formed.

The areas of the triangles and quadrilaterals, and in fact, any other polygons spanned by n points, used as costs can be calculated with the following equation (Råde and Westergren 1990):

$$\begin{aligned} A_{SA}(p_1, \ldots, p_n) = |&x_1y_2 - x_2y_1 + x_2y_3 - x_3y_2 + \ldots \\ &+ x_{n-1}y_n - x_ny_{n-1} + x_ny_1 - x_1y_n|, \end{aligned} \quad (3.28)$$

where the contour of area A_{SA} is formed by connecting the points with lines in the same order as they are indexed. If the lines intersect, the equation is not valid and gives too small values. This occurs only if the strokes intersect. If the sampling rate is high, the resulting error is small. In this work, these errors are neglected in order to keep the computational complexity reasonable. SA-distance is calculated with a dynamic programming algorithm similar to the one used for PP-distance and described earlier. SA-matching is illustrated in Figure 3.4. An obvious drawback of SA-distance, illustrated in Figure 3.5, is that very sharp cusps have only minor effect on the total distance as their area is near to zero.

An alternative to the cost function defined by equation (3.28) is to weight the distance between the matched points with the distances to their adjacent points. A

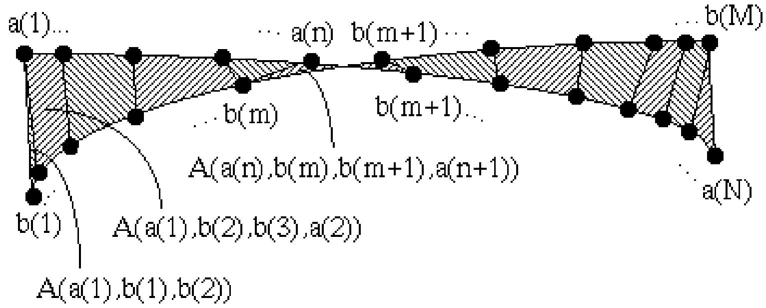


Figure 3.4: *Simple-area matching of two strokes.*

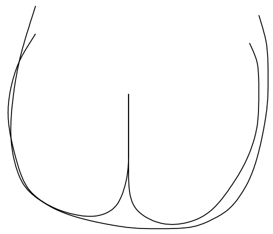


Figure 3.5: *Because the area of a sharp cusp is small, it does not have a major effect on simple-area distance. However, in the illustrated situation, it is the only feature that distinguishes letter 'w' from letter 'U'.*

cost function like this can also be seen as a kind of area. Kind-of-area distance (KA-distance) defined next is the sixth and last attempt for the dissimilarity measure between two strokes. It also has similar boundary and continuity constrains and solution as PP-distance, but the cost function, A_{KA}, is defined differently: if data points $p_1(i(h))$ and $p_2(j(h))$ are matched against each other, the associated cost is given by the following equation:

$$\begin{aligned} &A_{KA}(p_1(i(h)), p_2(j(h)); n, m) = \\ &= |p_1(i(h)) - p_2(j(h))|^n \Big(|p_1(i(h)) - p_1(i(h)+1)| + |p_1(i(h)) - p_1(i(h)-1)| + \\ &\qquad + |p_2(j(h)) - p_2(j(h)+1)| + |p_2(j(h)) - p_2(j(h)-1)|\Big)^m, \quad (3.29) \end{aligned}$$

where the notation $|\ldots|$ stands for the Euclidean norm. Strokes S_1 and S_2 are augmented prior to matching by replicating the second and next to last points: $S_i = (p_i(2), p_i(1), p_i(2), \ldots, p_i(N_i - 1), p_i(N_i), p_i(N_i - 1))$. The new starting and end points are not matched. Due to this augmentation, the cost function is defined similarly for all matchings. Kind-of-area matching between two strokes is illustrated in Figure 3.6.

Parameter $m \geq 0$ controls the effects of the point density on the total distance between the strokes. If $m = 0$, the parts of high density are emphasized. On the contrary, if $m \geq 1$, the meaning of a single point which is close to its neighboring points is reduced. Penalization of point distortions can be altered with parameter $n \geq 1$. Penalization is linear if $n = 1$. When $n > 1$, points which are far apart are penalized more harshly than points which are close to each other. When $n = 2$ and $m = 0$, KA-distance is equal to PP-distance.

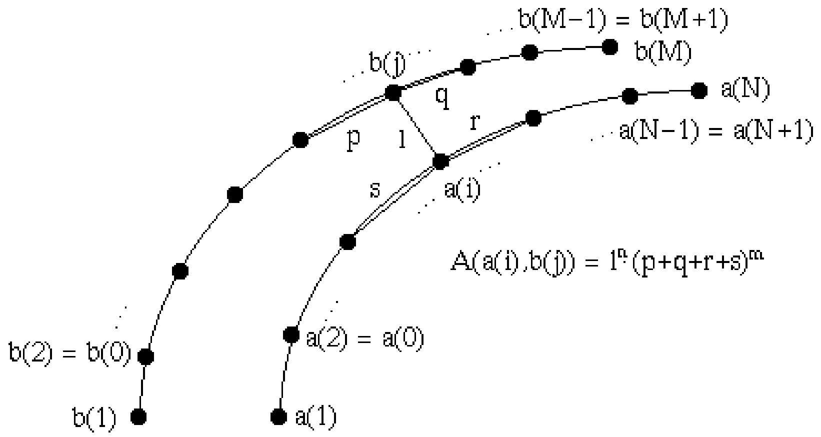


Figure 3.6: *Kind-of-area matching of two strokes.*

Chapter 4

Types of adaptation in handwriting recognition

In this chapter, various types of adaptation involved in on-line recognition of handwriting are discussed. First, some motivations for developing an adaptive recognizer and problems involved are discussed. Next, the adaptation methods used in this work are described. The last section contains some speculations on the user's response to recognition results and on the expected changes in his writing style while he is using the system.

4.1 Adaptive systems

Even though the distances defined in the previous chapter are insensitive to some writing style variations and some of the variations can be eliminated with normalization and preprocessing methods described in the following chapter, all variations cannot be accounted for by these means. The character set used in this work for forming the initial prototype set is collected from a fairly limited set of people and therefore not all, not even some quite typical, writing styles for some of the characters are covered. Thus, the recognizer can perform well only with those new writers whose writing styles resemble some of the styles included in the prototype set. Even if data from several writers is used and the number of prototypes is increased, all the existing writing styles cannot be included in the prototype set and there are always some writers having peculiar writing styles the recognizer cannot handle. On the other hand, for any single writer, most of the initial prototypes never actually take part in the classification process.

For the recognizer to be able to perform well with multiple users and thus recognize several styles of writing, it has to be adaptive. Or in other words, it should be able to learn new writing styles and forget, at least temporarily, the ones which are not used. If the system in which the recognizer has been implemented has only one user, the recognizer eventually changes from a writer-independent system into a writer-dependent one due to the adaptation. In the case of multiple users, the system should be able to identify its users by their writing styles, or by some other means, and have different prototype sets and *a priori* probabilities of the classes for each user. Schomaker et al. (1994) have proposed a method for writer and writing style classification which could be used for this purpose. In their method, a histogram of the usage of typical line segments is constructed for each writer. They found out that the histograms of individual writers were somewhat different and the histogram of a particular writer did not noticeably depend on the set of characters written.

Another problem involved with the learning process is the supervision: how does the system know whether the recognition result is correct, and if it is not, what is the correct class? This problem can be avoided if the system is taught with a separate enrollment program in which the user writes characters specified by the recognizer. The learning program can be exactly same for all writers or it can continue until a tolerable error rate is reached and concentrate on those characters with which it has problems. If the recognition rate drops during the normal use, the system could switch back to the learning mode and ask the user to write a few examples of the problematic characters. For example, Lu and Brodersen (1984), Nouboud and Plamondon (1991), and Tappert (1984) have used such adaptation methods in which the whole prototype sets are given by the user. Before starting to use the system, the user defines the character set and gives some examples of each class. During the use, the user can further refine the prototype set by adding new prototypes or removing old ones.

Alternatively, all learning can be carried out unnoticeably during the normal use and the user is assumed to confirm all the recognition results. However, the user does not necessarily notice all the mistakes immediately, if ever, or care to correct them. In addition, the user may change his mind on what he is writing and thus correct characters which were correctly recognized. Due to these facts, the system has to be able to recover from faulty training samples. The steps needed for correcting misrecognized characters have to be considered not only for the recognizer but also when designing the user interface. The work presented in this thesis has been limited to concentrate only on the methods of supervised learning applied during the normal use. All character samples are required to be correctly labeled and thus no problems caused by bad training samples arise.

An adaptive recognizer can also be constructed by combining various, not necessarily adaptive, classifiers whose recognition results together determine the class of an unknown character. The decision rule of such a committee classifier, for example a weighted voting, can be individually adjusted for each user instead of, or in addition to, the member classifiers. The accuracies of the individual classifiers are less important if the classifiers are different in the sense that they do not make same mistakes (Skalak 1997).

4.2 Adaptation strategies

In this work, experiments with on-line adaptation strategies are carried out. In all experiments, the initial prototype set is the same for all users and it consists of prototypes not written by the user himself. During the run, the prototype set is modified by adding, removing, and reshaping the prototypes according to the adaptation strategies described in the following sections.

4.2.1 Prototype addition rules

Adaptation strategy here called $Add(k)$ examines the classes of the k prototypes nearest to the input character. The classification is carried out according to the k-NN rule. The input character is added to the prototype set if any of these prototypes belongs to a wrong class, even if the classification result was correct. Problems arise, if the initial prototypes represent styles of writing which the writer uses but for different character classes. Or, if the writer uses similar writing styles for two different classes. In those cases, the adaptation strategy adds new prototypes in vain: the neighborhood of the k nearest prototypes consists of prototypes of very similar writing style but different classes no matter how many new prototypes have been added.

4.2.2 Prototype inactivation rules

Adaptation strategy $Inactivate(N, G)$ is used for inactivating those prototypes which are more harmful than useful. Some character classes tend to be confused, for example 'g' and '9', as some writers write them in exactly the same way. If the prototype set includes prototypes very similar to characters written by the user but which belong to wrong classes, they should be removed. After each recognition, *Inactivate*-strategy checks if the prototype nearest to input character should be inactivated: if its goodness value g is below a given limit G and it has been the nearest prototype for at least N times, it is removed from the set of active prototypes. The goodness value is defined as follows:

$$g = \frac{N_{corr} - N_{err}}{N_{corr} + N_{err}}, \tag{4.1}$$

where N_{corr} and N_{err} are the numbers of times when the prototype has been the nearest one and its class has been correct or incorrect, respectively. Parameters G and N control the strictness and reaction rate of the inactivation rule, respectively. Reasonable values of G are between -1 and 1.

4.2.3 Prototype modification with LVQ

When a character written by the user is basically similar to a prototype of the correct class, for example it has the same number and order of strokes, but of slightly different shape, the existing prototype can be reshaped instead of adding the input character to the prototype set. This can be performed with an adaptation strategy called $Lvq(\alpha)$ based on Learning Vector Quantization (LVQ) (Kohonen 1997). If the nearest prototype belongs to the same class as the input character, the data points of the prototype are moved toward the corresponding points in the input character. When the classes differ, the data points are moved in the opposite direction. The classification is carried out according to the 1-NN rule.

According to the original LVQ rule, the reference vector m nearest to the input vector x is updated in the following way:

$$m(t+1) = \begin{cases} m(t) + \alpha(x - m(t)), & \text{if } m \text{ and } x \text{ belong to same class} \\ m(t) - \alpha(x - m(t)), & \text{otherwise} \end{cases}, \tag{4.2}$$

where α is a positive learning coefficient. In this work, a modified version of LVQ training is used as its traditional form cannot be directly applied: prototype and input characters do not generally have same number of data points. The point-to-point correspondence is established by using DTW and the modified LVQ training is defined as follows (Laaksonen et al. 1998b):

Let $P(h) = (i(h), j(h))$ be the optimal time warping path between strokes $S_1 = (p_1(1), \ldots, p_1(N_1))$ and $S_2 = (p_2(1), \ldots, p_2(N_2))$. The PP-distance between the

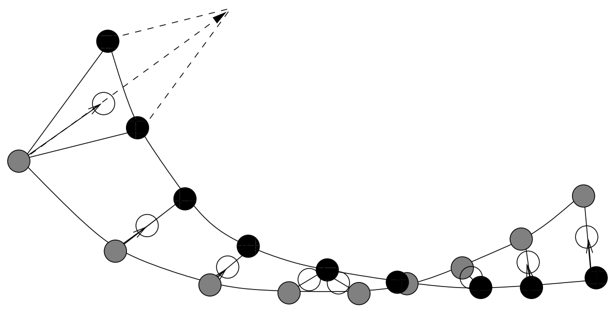


Figure 4.1: *The data points of the prototype stroke (grey circles) are moved closer to the data points of the training sample stroke (black circles) with LVQ. The new positions of the data points are marked with open circles.*

strokes is

$$D_{PP}(S_1, S_2) = \sum_{h=1}^{H} d(p_1(i(h)), p_2(j(h)), \tag{4.3}$$

where $d(p_1, p_2)$ is the squared Euclidean distance. If the data points of stroke S_2 are moved, their new locations are given by

$$S_2^{new} = \begin{cases} S_2^{old} - \alpha \frac{\partial D_{PP}(S_1,S_2)}{\partial S_2}\Big|_{S_2^{old}}, & \text{if } S_1 \text{ and } S_2 \text{ belong to the same class} \\ S_2^{old} + \alpha \frac{\partial D_{PP}(S_1,S_2)}{\partial S_2}\Big|_{S_2^{old}}, & \text{otherwise} \end{cases}, \tag{4.4}$$

where α is a positive learning coefficient and

$$\frac{\partial D_{PP}(S_1, S_2)}{\partial S_2}\Bigg|_{S_2^{old}} = -2 \begin{pmatrix} \sum_{h=1}^{H} \delta(1, j(h))(p_1(i(h)) - p_2(1)) \\ \vdots \\ \sum_{h=1}^{H} \delta(k, j(h))(p_1(i(h)) - p_2(k)) \\ \vdots \\ \sum_{h=1}^{H} \delta(N_2, j(h))(p_1(i(h)) - p_2(N_2)) \end{pmatrix}^T . \tag{4.5}$$

$\delta(i, j)$ is Kronecker's delta function. The modified LVQ training is illustrated in Figure 4.1.

4.2.4 Hybrid strategy

Adaptation strategy *Hybrid*(k, α) combines adaptation strategies *Add* and *Lvq* explained in the previous sections. The k nearest prototypes are examined. If any one of them belongs to the same class as the input character, the nearest prototype is modified with *Lvq*(α). Otherwise, the input character is added to the prototype set. The classification is carried out according to the k-NN rule. In this way, the increase in the size of the prototype set should be moderate compared with pure *Add*-strategy. Still, the system is able to learn completely new writing styles similarly to *Add*-strategy. This behavior cannot be obtained by only modifying existing prototypes with *Lvq*-strategy.

4.3 Adaptation of the user

The performance of the recognizer can change during its usage although the system is not adaptive. This is caused by the changes in the user's writing style. The writing style evolves as a novice user gets accustomed to the writing equipment and its constraints. The writer can also deliberately change his writing style on the basis of the previous recognition results in order to improve the recognition performance. According to Frankish et al. (1995), the users will not succeed in this unless they know the proper model of the recognition process and, therefore, the effects of the user adaptation are marginal.

On the other hand, according to the experiments described by Hullender (1998), the user adaptation is considerable at least at the beginning of the usage. A system, which was based on elastic matching and intended to be adaptive, was used for the collection and recognition of handwritten characters. The subjects were advised to write neatly in order to avoid recognition errors. Accidentally, the adaptation was not operating. However, the error rate did decrease. Later on, new experiments were performed with the system working as originally intended. For the first hour or so, the learning rate behaved as in the previous experiments. From these results, it can be concluded that the writer himself can remarkably affect the recognition performance.

In this work, the effects of user adaptation are examined by comparing the level and development of the recognition rate during adaptive collection with the corresponding figures for nonadaptive collection simulated with the same data.

Chapter 5

Data used in the experiments

In this chapter, the databases used in this work are discussed. Four databases have been used for creating the user-independent initial prototype sets, testing the normalization and preprocessing methods, comparing different adaptation strategies, and for estimating user adaptation. In the first section, the collection, contents, and use of the four databases are described. In the second section, various preprocessing operators which alter the sampling of pen point movements are explained. Finally, the last section describes normalization methods which reduce between- and within-writer writing style variations.

5.1 Databases

All the characters were collected with pressure sensitive Wacom ArtPad II tablet attached to a Silicon Graphics (SGI) Indy workstation (Database 1) or SGI Octane workstation (Databases 2-4). The resolution of the tablet is 100 lines per millimeter and the sampling rate is at maximum 205 data points per second. The loci of the pen point movements consists of the x- and y-coordinates, pen's pressure against the writing surface, and time stamp. The writing area was a rectangle of size $50\,\mathrm{mm} \times 20\,\mathrm{mm}$ placed at the center of the tablet. The characters were written one at a time. Writers were advised to use their natural handwriting style. All data was saved in UNIPEN format (Guyon et al. 1994). The essential details of the databases are summarized in Table 5.1.

The first database (Database 1) consists of characters which were written without any visual feedback: the trace of the pen was shown neither on the tablet nor on the screen. Therefore, the pressure level thresholding the pen movements into either

Table 5.1: *Summary of the databases used in the experiments.*

Database	*Subjects*	*Left-handed*	*Females*	*Items*	*Classes*
DB1	22	1	1	$\sim 10\,400$	'a'-'z', 'A'-'Z', 'å'-'ö', 'Å'-'Ö', '0'-'9', and special characters
DB2	8	2	5	$\sim 13\,200$	'a'-'z', 'A'-'Z', 'å'-'ö', 'Å'-'Ö', and '0'-'9'
DB3	16	2	5	$\sim 21\,200$	'a'-'z', 'A'-'Z', 'å'-'ö', 'Å'-'Ö', and '0'-'9'
DB4	8	0	5	$\sim 8\,100$	'a'-'z', 'A'-'Z', 'å'-'ö', 'Å'-'Ö', and '0'-'9'

pen-up or pen-down movements was individually set for each writer. Some of the characters were written in alphabetical order, but most of them according to a dictation of a short story. The characters of Database 1 are upper and lower case Latin characters ('A'-'Z', and 'a'-'z'), upper and lower case Scandinavian characters used in the Finnish language ('å', 'ä', 'ö', 'Å', 'Ä', and 'Ö'), digits ('0'-'9'), and some special characters ('(', ')', '/', '+', '-', '%', '$', '@','!', '?', ':', ',', and '.'). The special characters have not been used in this work. Database 1 consists of approximately 10 400 characters written by 22 subjects. The subject group includes one left-handed and one female writer. The distribution of the characters is uneven and roughly the same as in the Finnish language. The data collection was organized by Jarmo Hurri in 1997. Database 1 has been used for creating the prototypes for a user independent-recognizer.

The rest of the databases (Databases 2-4) were collected with a program which showed the pen trace on the screen and recognized the characters on-line. Characters 'a'-'z', 'A'-'Z' , 'å', 'ä', 'ö', 'Å', 'Ä', 'Ö', and '0'-'9' were written in same random order by all subjects. When a character was misrecognized, the writer had to determine whether the character was not properly written or the error could be considered as recognizer's fault. The user interface of the data collection program is illustrated in Figure 5.1. The two buttons in the bottom of the window labeled as 'Writer Mistake' and 'Recognizer Mistake' were used after each misrecognition for indicating the type of the error. The minimum writing pressure for showing the trace of the pen on the screen and detecting pen down movements was common to all writers. All the samples were checked and those which were reported as writer's mistakes or clearly incorrect were abandoned. None of the writers of Database 1 appeared in any of the Databases 2-4.

Database 2 was collected with a very poor recognizer from eight subjects. It consists

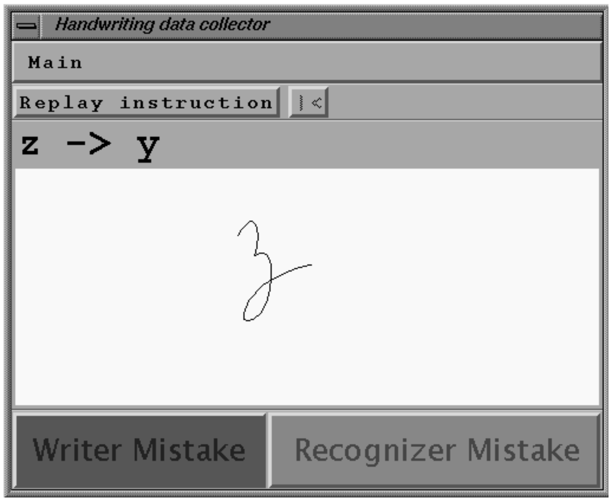


Figure 5.1: *The user interface of the data collection program used in collection of Databases 2-4.*

of approximately 13 200 characters with nearly even distribution. First five writers wrote 2 040 characters (30 per class), but the amount of labor and time consumed was found to be intolerable and so the number of samples was reduced to 1 020 for the rest of the writers. Five of the writers were female and two left-handed. Database 2 has been used for finding good preprocessing and normalization methods, the most suitable distance metric, and the best adaptation strategy.

Database 3 consists of Database 2 and characters written by additional eight writers. For them, a bit better adaptive recognizer was used. All new writers were male and right-handed. The total number of characters in the Database 3 is approximately 21 200. Database 3 has been used for comparing different adaptation strategies.

Database 4 was collected with recognizer otherwise similar to the one used for collecting Database 3 but significantly better recognition and adaptation methods were applied. Database 4 consists of approximately 8 100 characters written by eight new writers. All the writers were right-handed and five of them female. Database 4 has been used for estimating the effects of user adaptation and for verifying some of the results obtained with Database 3.

5.2 Preprocessing

Prior to clustering and matching, the characters need to be preprocessed. The preprocessing operations performed in this work are very simple as they have been mainly used for choosing a suitable sampling method and frequency. Characters were collected with a system whose properties, such as the sampling rate and resolution, are beyond the capabilities of existing hand-held devices. Therefore, it is important to examine how sensitive the recognition method is to the amount of data per character.

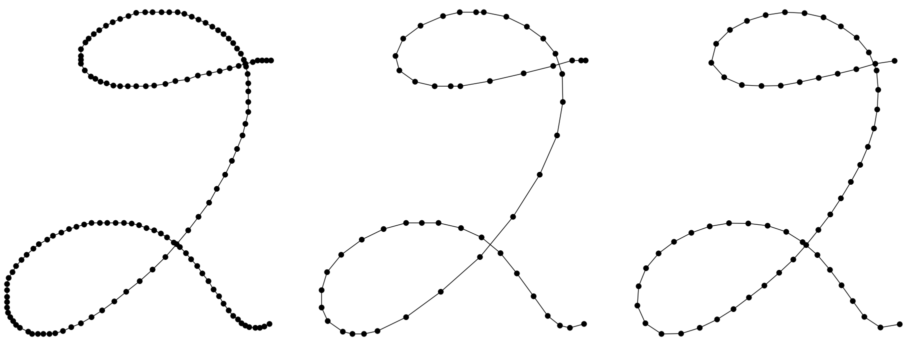
The original data consists of sample points equidistant in time and the sampling rate is relatively high. The sampling frequency is altered with either of operations called *Decimate*(n) and *Interpolate*(n). *Decimate*(n) preserves every $(n+1)$th data point and abandons the intermediate points. *Interpolate*(n) does just the opposite – it interpolates n equally spaced points between every original data point. The former operation reduces both the sampling rate and the amount of information in the data. The latter operation only increases the sampling rate as the added data points do not contain any additional dynamic information and are slightly misplaced from the actual smooth path of the pen.

In the work of Scattolin (1995) (see Example 10 on page 34), two sampling methods were compared with a recognition system based on elastic matching. According to her, better recognition results can be obtained if the implicit dynamic information of the data points is removed by making the data points equidistant in space via resampling. On that basis, in this work, an operator named *EvenlySpacedPoints*(d) is used for simulating a different sampling method. It interpolates new data points using the original points so that the resulting distance between adjacent points is a constant which depends on the original size of the character and is directly proportional to parameter d. When a character has been preprocessed with operator *EvenlySpacedPoints*(d), the distance between adjacent data points is $dl/1\,000$, where l is the length of the longer side of the bounding box. In the case of the data used in this work, the total number of data points remains practically unchanged when the value of parameter d is set to 33.

In Figure 5.2, the original and two preprocessed versions of an example character are shown. It can be seen in the figure that the original sampling rate is at least sufficient because no peculiar corners appear when adjacent data points are connected with lines.

5.3 Normalization

The writers were allowed to write anywhere on the restricted writing area. Therefore, the unknown character and the prototypes must be moved to same location before they can be matched. This is carried out by moving their center points to the origin of the coordinate system. The center point can be defined in several



(a) (b) (c)

Figure 5.2: *a) An original character and its two preprocessed versions when b) Decimate(2), and c) EvenlySpacedPoints(60) has been applied.*

ways. In the experiments this work, both the mass center of the preprocessed data points and the center of the bounding box have been applied. The bounding box is a rectangular frame drawn around the character. It is determined by the maximum and minimum values of the $x-$ and y-coordinates of the data points. Normalization method *MassCenter* moves the mass center of the character to the origin and normalization method *BoundingBoxCenter* does the same to the center of the bounding box.

The size variations in the characters are normalized with an operator called *MinMaxScaling* which scales the size of the character so that the length of the longer side of the bounding box is set to 1 000 units and the aspect ratio remains unchanged. *MinMaxScaling* has been performed in all the experiments.

Chapter 6

Experiments and their results

An adaptive on-line recognizer for handwritten characters based on the classification methods described in Chapter 3 has been constructed in the C++ programming language in Unix environment. It has been initialized with Database 1, further tuned by Databases 2 and 3, and tested with Database 4. The experiments evaluating the various options for the implementation of the recognizer are described and their results presented and discussed in the following sections. In the first section, the most suitable distance measure, preprocessing, and normalization methods for a nonadaptive recognizer are selected by using Database 2. In the next section, the most suitable parameter values for the adaptation strategies defined in Chapter 4 are found by using Database 3 as a design set. These results are then verified with Database 4. In the last section, the best adaptive recognizer found in the previous section is tested on-line and the typical errors and the effects of user adaptation are analyzed by using Database 4.

The results of all the experiments carried out in this work can be considered to be writer-independent as Database 1 used for creating the prototype set is written by subjects who do not appear in the other three databases used for testing. In all the experiments, the prototype set consists of characters which are preprocessed and normalized similarly to the characters of the test set. In addition, the distance metric used in clustering has been the same as the one used in matching.

MinMaxScaling-normalization is performed in all the experiments because the writers of Database 1 and the other three databases were different persons and they were allowed not only to use their own natural writing style but also to write the characters in any size they preferred to. The size of the characters varies both between- and within-writer, the former being most significant between the writers of Database 1 and the writers of the three other databases. One explanation for this phenomenon could be the bias caused by the enlarged images of the characters shown on the screen. Perhaps, the writers of Database 1, who did not receive any visual feedback, wrote more natural and constant-sized characters.

6.1 The best nonadaptive recognition method

All the distance metrics defined in Chapter 3 have been tested with the lower case characters ('a'-'z', 'å', 'ä', 'ö') and digits ('0'-'9') of Database 2. Several combinations of preprocessing and normalization methods have been tested for each of the six distance definitions. The results of these experiments are collected in Tables 6.1-6.6 in which each line corresponds to a single test. The firsts columns indicate which preprocessing and normalization methods were used in the test. A nonempty cell means that the corresponding method was used with the given parameter value. A bullet is used for methods which do not have any parameters. *Dec*, *Int*, *Even*, *MC*, and *BBC* stand for *Decimate*-, *Interpolate*-, *EvenlySpacedPoints*-, *MassCenter*- and *BoundingBoxCenter*-methods, respectively. Three error percentages have been calculated for each experiment: the total E_{tot}, best case E_{best}, and worst case E_{worst} error percentages. E_{tot} is the percentage of all misrecognized characters in the whole test set and E_{best} and E_{worst} are the best and worst single-writer error percentages. The tests presented in the following tables are compared by their total error percentages and the best result for each distance measure is highlighted with bold font.

Table 6.1: *Results of the nonadaptive experiments in which point-to-point distance has been used.*

Dec	*Int*	*Even*	*MC*	*BBC*	E_{tot}	E_{best}	E_{worst}
			•		16.39	6.17	31.37
				•	18.45	7.82	28.42
1			•		16.23	6.36	30.50
1				•	18.67	6.69	31.72
2			•		**15.93**	6.17	28.60
2				•	18.27	6.86	27.73
3			•		16.49	5.67	31.72
3				•	18.38	7.38	31.72
4			•		17.80	5.83	28.34
4				•	17.67	6.17	29.20
5			•		16.49	6.62	27.82
5				•	18.53	6.35	31.63
6			•		16.83	6.52	28.77

continued on next page ...

... Table 6.1 continued from previous page							
Dec	*Int*	*Even*	*MC*	*BBC*	E_{tot}	E_{best}	E_{worst}
6				•	18.49	6.36	24.87
7			•		18.39	8.06	29.90
7				•	19.73	7.20	33.02
	1		•		16.01	6.52	28.42
	1			•	18.34	7.89	28.97
	2		•		16.57	6.35	30.24
	2			•	18.92	8.40	29.14
		20	•		18.01	6.87	29.12
		20		•	19.59	9.19	28.25
		25	•		19.40	8.08	33.19
		25		•	19.49	9.36	27.21
		30	•		18.62	7.73	28.94
		30		•	19.73	9.71	26.90
		35	•		18.80	8.93	28.68
		35		•	19.08	10.14	27.79
		40	•		19.51	9.36	30.24
		40		•	18.62	9.26	26.55
		50	•		17.97	8.42	26.86
		50		•	19.31	7.22	27.93
		60	•		19.06	7.22	28.60
		60		•	20.30	9.95	29.46
		80	•		18.61	7.82	27.90
		80		•	19.75	11.15	27.93

From Table 6.1, it can be clearly seen that for point-to-point (PP) distance *MassCenter* is more suitable centering normalization than *BoundingBoxCenter* which performs better only in two stray cases. It is also evident that the sampling rate is high enough. *Interpolate*(1), which doubles the original sampling rate, decreases the error percentage only slightly, whereas the best result was obtained with *Decimate*(2), which corresponds to the use of one third of the original sampling rate. In addition, *Decimate*(6), which uses only one seventh of the original data points, gives surprisingly good results. These results are very appealing as they suggest that a lower error rate can be obtained with less data and thus the computation time can be reduced. When the results of the tests in which the data points are equally spaced with *EvenlySpacedPoints*-method are compared to the other results, it can be inferred that the implicit dynamic information is beneficial and yields better recognition accuracies. This is quite opposite to the results obtained by Scattolin (1995) which gave the inspiration for altering the original sampling method.

Table 6.2: *Results of the nonadaptive experiments in which point-to-line distance has been used.*

Dec	*Int*	*Even*	*MC*	*BBC*	E_{tot}	E_{best}	E_{worst}
			•		**17.07**	7.65	28.60
				•	18.62	8.06	27.07
1			•		17.78	8.92	26.00
1				•	20.28	8.40	30.59
2			•		18.30	10.74	31.11
2				•	20.99	6.86	29.12
3			•		19.61	10.46	28.08
3				•	22.38	9.43	34.83
4			•		20.77	11.84	32.15
4				•	21.99	9.95	32.93
5			•		21.02	8.58	30.70
5				•	21.11	8.75	34.99
	1		•		17.67	8.40	27.64
	1			•	18.20	7.38	25.22
	2		•		17.76	8.40	25.82
	2			•	18.98	8.06	27.07
		20	•		20.25	9.95	28.60
		20		•	21.76	9.43	30.85
		30	•		18.49	9.54	27.90
		30		•	22.56	9.09	31.02
		40	•		18.66	10.14	28.08
		40		•	21.02	10.29	32.67
		50	•		19.51	7.89	29.29
		50		•	22.28	10.29	30.94
		60	•		20.59	10.57	31.46
		60		•	21.75	9.61	30.24
		80	•		20.77	8.75	29.29
		80		•	22.78	13.04	30.94

According to the test results presented in Table 6.2, *MassCenter*-normalization is better than *BoundingBoxCenter*-normalization also in the case of point-to-line (PL) distance. The results obtained with data which is preprocessed by using *Interpolate*(1)-, *Interpolate*(2)-, or *Decimate*(1)-method are nearly as good as the result obtained with the original data. Otherwise, PL-distance seems to be more sensitive to *Decimate*- and *EvenlySpacedPoints*-methods than PP-distance. Also in this case, data points equidistant in time yield better results than points equidistant in space. The lowest error rate obtained with PL-distance is significantly higher than the lowest error rate for PP-distance and, in addition, three-fold amount of data points was required.

Table 6.3: *Results of the nonadaptive experiments in which normalized point-to-point distance has been used.*

Dec	*Int*	*Even*	*MC*	*BBC*	E_{tot}	E_{best}	E_{worst}
			•		17.53	8.40	28.16
1			•		**16.13**	7.55	28.25
2			•		16.79	6.35	25.73
	1		•		17.27	7.56	27.62
	2		•		17.96	7.20	28.94
		20	•		19.52	8.85	30.42
		30	•		18.30	7.90	29.55
		40	•		19.85	10.31	30.16
		50	•		17.81	9.62	29.98
		60	•		19.08	7.39	30.94
		80	•		18.73	8.53	30.58

BoundingBoxCenter-normalization was not tested with normalized point-to-point (NPP) distance at all as *MassCenter*-normalization performed clearly better with PP-distance. The lowest error rate for PP-distance was obtained with *Decimate*(1)-method. Again, using data points equidistant in time instead of space was advantageous. The best result for the NPP-distance is only slightly worse than the best result for PP-distance. However, the former result was achieved with 1.5-fold amount of data points compared to the latter result.

Table 6.4: *Results of the nonadaptive experiments in which normalized point-to-line distance has been used.*

Dec	*Int*	*Even*	*MC*	*BBC*	E_{tot}	E_{best}	E_{worst}
			•		17.93	9.78	26.95
1			•		17.27	9.09	31.54
2			•		**17.07**	8.33	25.56
3			•		18.75	6.69	30.33
	1		•		17.86	9.43	26.60
	2		•		18.17	8.85	29.03
		20	•		18.38	7.73	31.98
		30	•		17.82	7.30	30.16
		40	•		19.11	9.19	31.02
		50	•		18.95	7.65	30.68

BoundingBoxCenter-normalization was not tested with normalized point-to-line (NPL) distance either. As with PP-, NPP-, and NPL-distances, the lowest error rate was obtained by using a reduced amount of data points. In this case, *Decimate*(2) seems to be the most suitable preprocessing method. The best total error percentages for PL- and NPL-distances are the same but as the latter result can be obtained with less data points, NPL-distance can be considered to be a better dissimilarity measure. The comparison of the two sampling principles is not straightforward as the best recognition result obtained with data preprocessed with *EvenlySpacedPoints*-method is slightly better than the results obtained with original data and nearly as good as the best result obtained with data preprocessed with *Decimate*-method.

Table 6.5: *Results of the nonadaptive experiments in which simple-area distance has been used.*

Dec	*Int*	*Even*	*MC*	*BBC*	E_{tot}	E_{best}	E_{worst}
			•		**29.75**	18.52	39.97
				•	33.89	20.75	42.02
1			•		30.41	13.04	41.08
1				•	34.88	20.41	44.08
2			•		30.53	18.52	41.08
2				•	35.57	21.96	42.86
	1		•		29.95	18.56	42.97
	1			•	33.46	18.87	44.08
		20	•		34.61	18.35	43.22
		20		•	35.81	22.64	41.85
		30	•		36.09	19.73	45.28
		30		•	38.78	21.10	50.34
		40	•		38.93	22.47	46.91
		40		•	38.97	21.61	50.94
		50	•		37.98	28.50	46.36
		50		•	40.91	24.53	49.66

All the results obtained with simple-area (SA) distance are poor compared to those presented in the previous tables. Nevertheless, it can be seen from Table 6.5 that *MassCenter* is a better normalization method than *BoundingBoxCenter*. In addition, the number of data points can be reduced considerably without affecting the recognition accuracy. As before, data points sampled with a constant frequency yield better recognition results than equidistant points.

Kind-of-are (KA) distance performs as a dissimilarity measure much better than SA-distance but worse than the other distances which are not based on the area between the matched strokes. Because *Decimate*(2) was the most suitable preprocessing method for PP-distance which is equal to KA(2,0)-distance, no other preprocessing methods were tested for KA-distance. The results of the experiments with KA-distance are presented in Table 6.6. The fact that smaller values for parameter m yield lower error rates implies that the inclusion of the distances between adjacent data points in the cost function does not have such an effect on the recognition accuracy as originally assumed. On the contrary, the removal of the implicit dynamic information decreases the recognition performance instead of increasing it. Better results are obtained when the value of parameter n is set to 2 instead of 1. Therefore, it can be deduced that the cost function should be a nonlinear function of the distance between the matched points and penalize more harshly for the major distortions than the minor ones.

Table 6.6: *Results of the nonadaptive experiments in which kind-of-area distance has been used.*

n	m	*Dec*	*Int*	*Even*	*MC*	*BBC*	E_{tot}	E_{best}	E_{worst}
1	0	2			•		**18.25**	9.97	30.24
1	0	2				•	19.44	8.92	30.85
1	1	2			•		22.71	12.71	33.02
1	1	2				•	23.47	10.63	30.27
1	2	2			•		33.14	21.10	40.17
1	2	2				•	34.68	21.44	46.03
2	1	2			•		18.47	8.75	31.20
2	1	2				•	21.56	8.58	31.80
2	2	2			•		21.83	11.15	29.12
2	2	2				•	23.68	12.52	30.59

The best recognition results for each distance definition are collected in Table 6.7. According to these results, the PP-distance is chosen to be the best dissimilarity measure and to be used in all the later experiments. As well, preprocessing method *Decimate*(2) and normalization method *MassCenter* followed with *MinMaxScaling* will be exclusively used from this on. This combination gives the best average result for all the writers in Database 2, but it is the best choice only for one particular writer. In Table 6.8, the best combination of distance measure, preprocessing-, and normalization-method for each writer is presented. According to it, PP-distance and *MassCenter*-normalization are the most suitable dissimilarity measure and centering operation for a clear majority of the writers. In addition, reduction of data with *Decimate* improves recognition performance for all but one writer.

Table 6.7: *The summary best recognition results for all distance definitions.*

Distance definition	*Dec*	*Int*	*Even*	*MC*	*BBC*	E_{tot}
Point-to-point	2			•		15.93
Point-to-line				•		17.07
Normalized point-to-point	1			•		16.13
Normalized point-to-line	2			•		17.07
Simple-area				•		29.75
Kind-of-area(1,0)	2			•		18.25

Table 6.8: *The best combination of the distance measure, preprocessing, and normalization method for each writer of Database 2.*

Writer	*PP-dist.*	*NPL-dist.*	*Dec*	*Int*	*MC*	*BBC*	*Error %*
DB2:7	•		3		•		5.67
DB2:6	•		4		•		5.83
DB2:4	•		2		•		11.23
DB2:3	•		1		•		12.17
DB2:8	•		1		•		16.64
DB2:1	•			1	•		17.84
DB2:2		•	1		•		18.97
DB2:5	•		6			•	24.78

6.2 The best adaptation strategy

In this section, the experiments with adaptation strategies are described and their results are discussed. First, the most suitable parameter values are selected for each adaptation strategy using Database 3. Next, the generality of the results is verified with Database 4. Last, the evolution of the error rate during adaptation is examined more closely.

6.2.1 Parameter selection

In this section, the most suitable parameter values are searched for adaptation strategies *Add*, *Inactivate*,*Lvq*, and *Hybrid* defined in Chapter 4. The data used in the experiments consists of the lower case characters ('a'-'z','å','ä','ö') and digits ('0'-'9') of Database 3. All the results of the experiments are collected in Table 6.9. Again, a line in the table corresponds to a single test and the last three colums contain the total, best and worst error percentages. The nonempty cells in the first four columns indicate which adaptation strategies were used. In all the tests, characters were preprocessed and normalized with *Decimate*(2)-, *MinMaxScaling*-, and *MassCenter*-methods and PP-distance was used as dissimilarity measure. This was found to be the most suitable combination for the nonadaptive recognizer in the previous section.

The first test in Table 6.9 was carried out with no adaptation at all. Therefore, it can be used for judging the benefits of the adaptation strategies. The total error percentage of the nonadaptive test is considerably high, over 14%, and the worst error rate percentage would be intolerable in any practical application. However, the best error percentage is relatively low for a totally writer-independent recognizer and would most probably be accepted by a novel user.

Different adaptation strategies are compared by their total error percentages as the adaptation method should be suitable for a large set of writers. In addition, the total error percentage does not only take into account both the initial and final recognition accuracies but also the rate of learning. For every adaptation strategy, the worst error rate is considerably high because of a few writers in Database 3 whose writing styles are quite careless or not strictly consistent over the whole writing period.

The best results were obtained when adaptation strategy *Add* was used alone with parameter value $k = 4$. The results obtained when strategies *Add* and *Inactivate* were used together are almost as good. In addition, the results for *Hybrid*-strategy are fairly good. However, *Inactivate*-strategy does not work well together with *Hybrid*-strategy. The recognition result for combined *Hybrid*- and *Inactivate*-strategies improves as the value of parameter n is increased. By doing so, *Inactivate*-operation is performed more seldom. Still, the combined strategy never outperforms the result obtained with sole *Hybrid*-strategy. The results for *Lvq*-strategy are not as good as the results for the other adaptation strategies. As *Lvq*-strategy is only able to halve the total error percentage of the nonadaptive test, it is not a sufficient adaptation method when used alone. The best total error rates achieved with *Add*-, combined *Add*- and *Inactivate*-, and *Hybrid*-strategies are all approximately 3%. As the initial error rate before adaptation was considerably higher, the final error rate after adaptation is naturally lower than the total error rate.

Table 6.9: *Results of the experiments with different adaptation strategies.*

Add	*Lvq*	*Hybrid*	*Inactivate*	E_{tot}	E_{best}	E_{worst}
				14.33	3.60	28.60
1				4.61	1.37	10.50
2				3.40	1.03	9.07
3				3.09	0.60	9.25
4				**2.93**	0.51	8.90
5				3.34	0.51	9.43
6				4.00	0.69	10.85
4			(1,0)	3.19	0.69	8.54
4			(2,0)	3.00	0.51	9.07
4			(3,0)	**2.95**	0.51	9.07
4			(4,0)	2.97	0.51	9.07
4			(5,0)	2.95	0.51	9.07
4			(6,0)	2.95	0.51	8.90
	0.05			8.70	1.98	17.59

continued on next page ...

... *Table 6.9 continued from previous page*						
Add	*Lvq*	*Hybrid*	*Inactivate*	E_{tot}	E_{best}	E_{worst}
	0.10			7.47	1.71	15.86
	0.15			7.06	1.71	14.99
	0.20			6.90	1.20	14.90
	0.25			6.85	1.03	16.12
	0.30			**6.81**	1.37	16.03
	0.35			7.26	1.37	17.42
	0.40			1.54	1.54	19.24
		(3,0.15)		3.52	0.77	8.19
		(3,0.20)		3.36	0.69	7.83
		(3,0.30)		**3.06**	0.69	7.65
		(3,0.35)		3.18	0.60	7.47
		(3,0.40)		3.29	0.60	8.01
		(4,0.15)		3.67	0.77	9.96
		(4,0.20)		3.53	0.69	9.25
		(4,0.30)		3.21	0.69	8.90
		(4,0.35)		3.36	0.60	8.36
		(4,0.40)		3.54	0.60	9.43
		(3,0.30)	(1,0)	6.46	1.55	13.52
		(3,0.30)	(2,0)	5.36	2.07	11.39
		(3,0.30)	(3,0)	4.95	2.07	9.79
		(3,0.30)	(4,0)	4.76	2.05	10.32
		(3,0.30)	(5,0)	4.74	2.05	10.14
		(3,0.30)	(6,0)	4.27	1.71	9.79
		(3,0.30)	(8,0)	4.17	1.71	8.90
		(3,0.30)	(10,0)	3.93	0.86	8.72
		(3,0.30)	(12,0)	3.90	1.29	8.19
		(3,0.30)	(14,0)	3.43	0.69	7.83
		(3,0.30)	(16,0)	**3.33**	0.69	7.83

In Table 6.10, the best recognition results and the corresponding adaptation strategies are presented for each writer. If the lowest error rate was obtained with several strategies, only the strategy having the lowest tendency to add prototypes is mentioned. It is evident that there are clear variations between the writers and the selection of the most suitable adaptation strategy depends on the writer as did the selection of the distance measure and preprocessing and normalization methods. If the adaptation strategy could have been selected individually for each writer, the avarage total, best, and worst error percentages would have been 2.79%, 0.51%, and 7.47%, respectively.

Table 6.10: *The best adaptation strategy yielding the lowest error percentage for each writer of Database 3.*

Writer	*Add*	*Hybrid*	*Inactivate*	*Error %*
DB3:16	4		(2,0)	0.51
DB3:2	3			1.03
DB3:3	4			1.03
DB3:4	4		(1,0)	1.20
DB3:15	4		(2,0)	1.20
DB3:7	4		(1,0)	1.55
DB3:11	4			1.71
DB3:14	4		(3,0)	1.72
DB3:1		(3,0.30)		2.57
DB3:9		(3,0.30)		3.43
DB3:13	4		(2,0)	3.47
DB3:8	4		(1,0)	3.48
DB3:10	4		(2,0)	3.62
DB3:5		(3,0.30)		4.32
DB3:12		(3,0.20)		6.27
DB3:6		(3,0.35)		7.47

6.2.2 Verification of the results

The results of the six adaptation strategies in Table 6.9 highlighted with bold font have been verified with Database 4. The results are presented in Table 6.11, where E_{design} is the total error rate for the design set, Database 3. E_{tot}, E_{best}, and E_{worst} are defined as earlier and E_{final} is the average error rate for the last 200 characters of each writer. Abbreviation *INP* stands for the increase in the number of prototypes in percentages.

The recognition results obtained with the design set, Database 3, and the verification set, Database 4, are quite close to each other. The total error percentage of the nonadaptive test is only slightly lower for the verification set and, thus, the results of the two sets can be reasonably compared. The error percentages of all the tests are higher, especially in the case of *Lvq*-strategy, for the verification set. In addition, the difference between *Add*- and *Hybrid*-strategies is more evident: clearly better recognition results can be achieved with *Add*-strategy. On the other hand, the number of prototypes increases significantly. The probabilities that a new sample will be added to the prototype set are illustrated in Figures 6.1 and 6.2 for both *Add*- and *Hybrid*-strategies. The probabilities are average values over the eight writers of Database 4 and are calculated by using a forwarded moving window of 50 samples. It

Figure 6.1: *Probability that a new sample will be added to the prototype set when adaptation strategy Add(4) is used.*

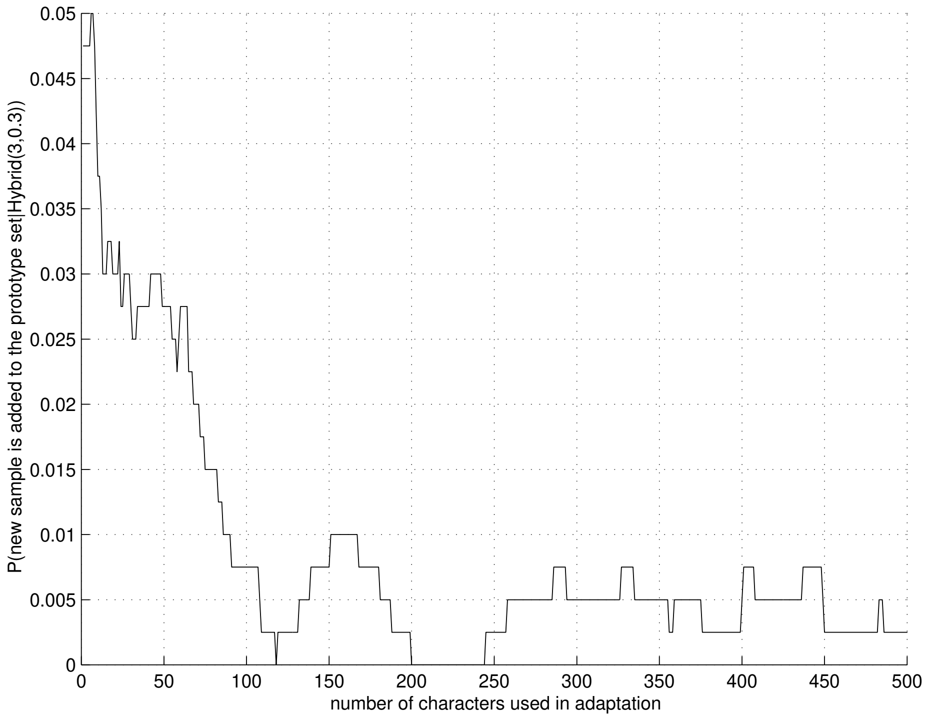


Figure 6.2: *Probability that a new sample will be added to the prototype set when adaptation strategy Hybrid(3,0.3) is used.*

can be seen from both of the figures that more prototypes are added at the beginning of adaptation. In the case of *Add*-strategy, the initial probability is about 0.7 and the probability at the end of the test is also remarkably high – approximately every fifth sample is added to the prototype set. For *Hybrid*-strategy, the initial probability is about 0.05 and the final probability is less than 0.01. As a result, the final number of prototypes is only 2% higher than the initial, whereas for strategy *Add* it is 66% higher.

Add-strategy applied together with *Inactivate*-strategy works better than *Add*-strategy used alone for the verification set. The effects of the *Inactivate*-strategy seem to be more prominent for the last 200 characters as the relative difference between the final errors is larger than between the total errors. However, *Inactivate*-strategy is rarely used, in total 40 times with *Add*- and 11 times with *Hybrid*-strategy, and it does not really limit the size of the prototype set.

Table 6.11: *Verification of the recognition results of selected adaptation strategies.*

Add	*Lvq*	*Hybrid*	*Inact.*	E_{design}	E_{tot}	E_{best}	E_{worst}	E_{final}	*INP*
				14.33	14.13	5.64	31.76	14.06	0
4				2.93	3.12	0.51	6.20	1.81	66
	0.3			6.81	9.89	2.42	28.27	8.63	0
4			(3,0)	2.95	3.04	0.51	5.34	1.56	65
		(3,0.3)		3.06	4.18	0.85	8.73	2.50	2
		(3,0.3)	(16,0)	3.33	4.26	0.85	9.08	2.75	2

6.2.3 Evolution of the error rate

The evolution of the error rates during the experiments is illustrated in Figures 6.3-6.6. The figures correspond to the first, second, third, and fifth experiments in Table 6.11. No figures are presented for the fourth and sixth experiment as their results are so similar to the results of the second and fifth experiments. The error rate is calculated by using a centered moving window of 101 characters for each individual writer in the verification set. In all the four figures, same colors are used for same writers. All the plots in one figure are scaled similarly but they are raised upward in order to avoid overlaps. The error rate calculated for the first window position and the zero level are indicated with grid lines for each of the plots.

The evolution of the error rate of the nonadaptive runs are shown in Figure 6.3. It can be seen that the recognition accuracy can vary considerably during the runs even without any system adaptation. However, the error rates do not tend to decrease permanently but rather oscillate. In this case, the oscillations are caused only by the inconsistencies in the writing styles as the recognition system has remained unchanged. These inconsistencies can be either a characteristic of the writing style or they can be caused by the tiresome data collection.

Figure 6.4 corresponds to the adaptation runs in which strategy *Add* has been used alone. The error rate is considerably decreased during the first 101 characters for all the writers. The adaptation does not prevent the error rate from oscillating, which is especially prominent for the two writers plotted with black and magenta. Compared to the runs for the other writers, the characters of these two writers were quite well recognized even without adaptation. Apparently, their writing styles are somewhat inconsistent and have changed during the data collection. It seems that the effects of the inconsistencies are more prominent if the writing style is otherwise easy to recognize.

The plots of error rates for the runs in which adaptation strategy *Lvq* has been applied are shown in Figure 6.5. The error rates for the first 101 characters have decreased but not as much as in the case of *Add*-strategy. *Lvq*-strategy is a considerably slow learning method and it seems to cause some oscillations of higher frequency not observed with *Add*-strategy. These oscillations could probably be avoided by using a decreasing learning parameter instead of a constant one.

The error percentages for the experiments in which adaptation strategy *Hybrid* has been used are plotted in Figure 6.6. Compared to *Add*-strategy, *Hybrid* strategy is slower. Otherwise, their results are quite similar. Thus, the selection between *Add*- and *Hybrid*-strategies is a trade-off between the required rate of learning and final accuracy, and, the available memory and computation resources.

6.3 On-line recognition experiments

In the collection of Database 4, adaptation strategy *Add*(4) was applied. This can be considered as a genuine on-line test of the adaptation strategy. The corresponding recognition results are presented and analyzed in the following sections. As the results were obtained with all the character classes, not only with digits and lower case letters, they cannot be directly compared to the previous results. They, however, give valuable insight to the problems which emerge when there are more classes. In addition, some of the classes are quite easily confused even by humans when no contextual information is available.

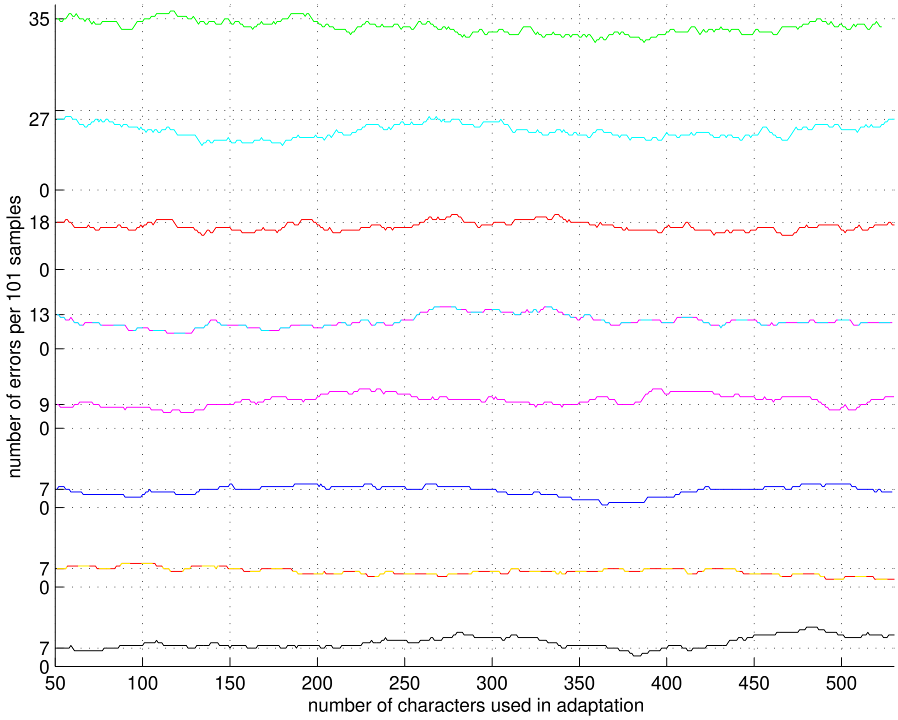


Figure 6.3: *Evolution of the error rate during the runs in which no adaptation strategy has been applied. A plot corresponds to a single writer. The shift between the plots is 30 units.*

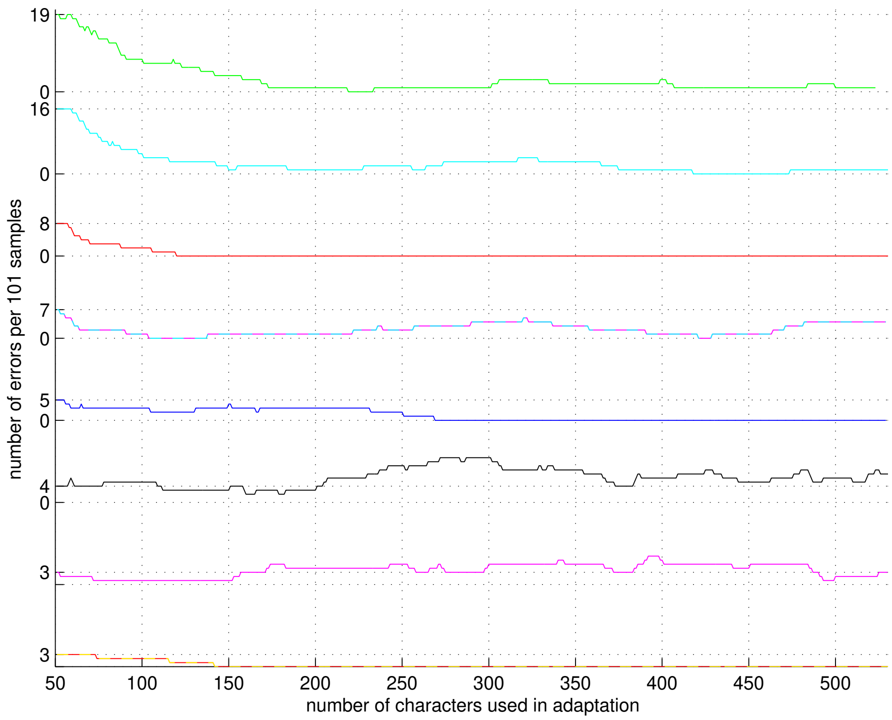


Figure 6.4: *Evolution of the error rate during the runs in which adaptation strategy Add(4) has been applied. A plot corresponds to a single writer. The shift between the plots is 20 units.*

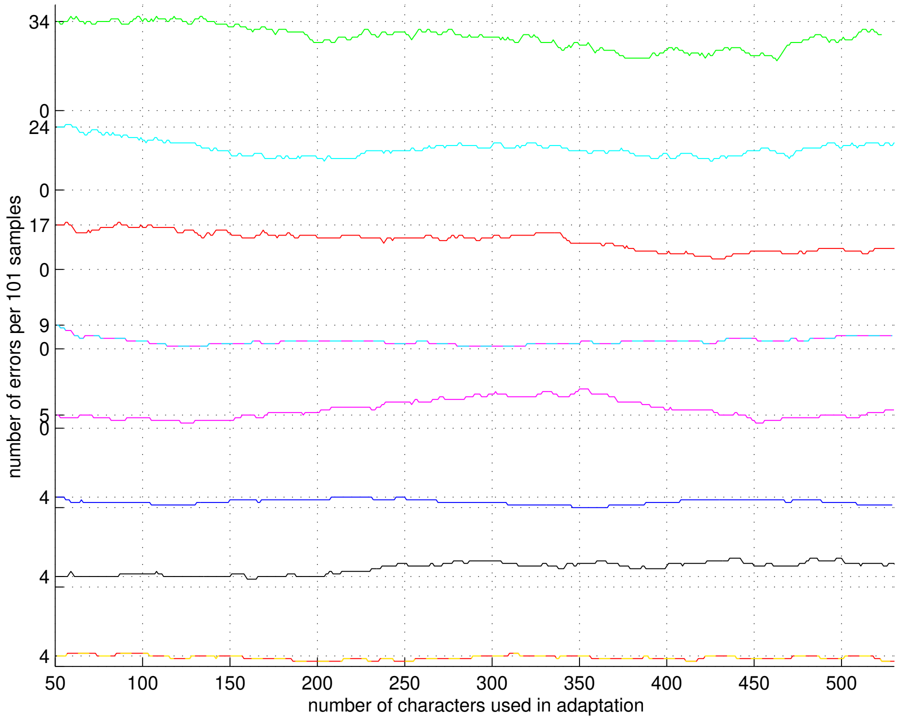


Figure 6.5: *Evolution of the error rate during the runs in which adaptation strategy Lvq(0.3) has been applied. A plot corresponds to a single writer. The shift between the plots is 30 units.*

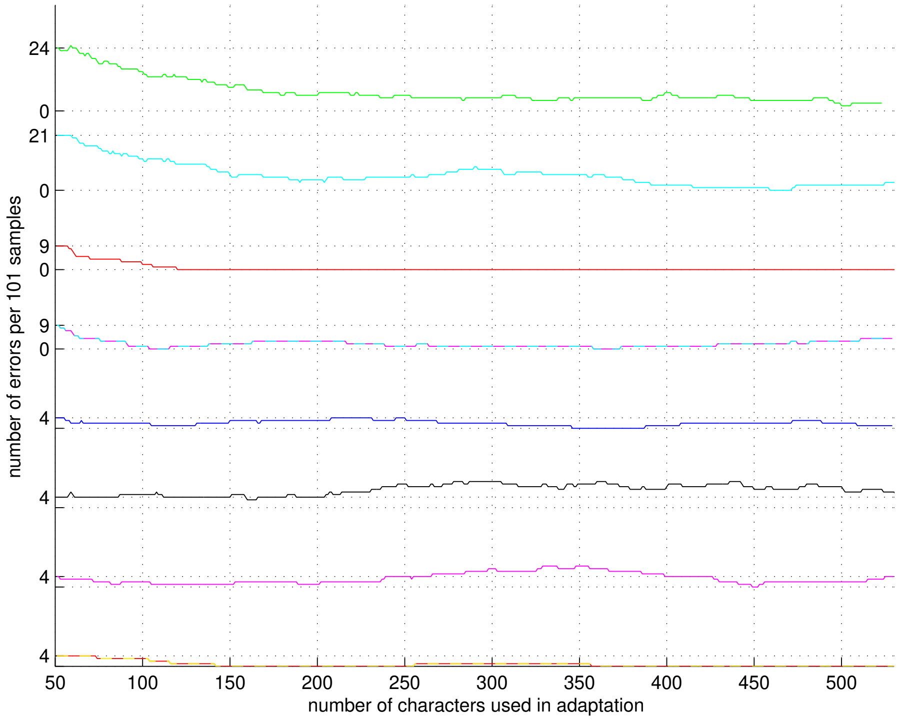


Figure 6.6: *Evolution of the error rate during the runs in which adaptation strategy Hybrid(3,0.3) has been applied. A plot corresponds to a single writer. The shift between the plots is 30 units.*

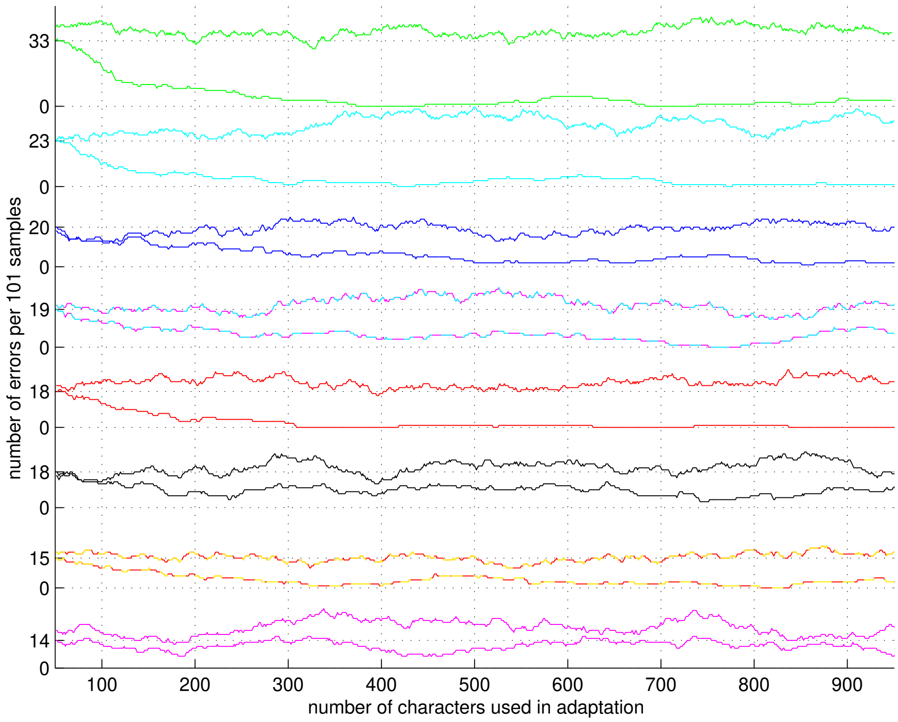


Figure 6.7: *Evolution of the error rate in the true collection of Database 4 with adaptation strategy Add(4) and in the nonadaptive simulation. A plot corresponds to a single writer. The shift between the plots is 30 units.*

6.3.1 Results

The evolution of the error rate during the adaptive data collection is illustrated in Figure 6.7 for all the writers of Database 4. Again, the error rate calculated for the first centered moving window of 101 characters and the zero level are indicated with grid lines. In addition, the error rate for the nonadaptive simulation of the data collection is plotted in order to demonstrate the effects of the adaptation method. The simulation is carried out simply by re-recognizing the characters in the same order as the have been collected but with a nonadaptive system.

It can be seen that the system adaptation improves the recognition accuracy for all the writers as the plots corresponding to the true adaptive data collection always stay beneath the plots illustrating the nonadaptive simulations. The error rates stabilize to some constant level or a narrow interval only in the case of three writers plotted with green, cyan, and red. For all the other writers, the error rate oscillates significantly. This oscillatory behavior can be explained by the inconsistencies in the writing styles as it is more prominent for those writers whose nonadaptive plots also oscillate considerably, for example the writers plotted with magenta and black. In addition, the shapes of the corresponding plots seem to correlate especially at the beginning of the run.

Some interesting figures for both the true and simulated data collections are shown in Table 6.12. In the table, there is one line per writer and one line for the average values of all the eight writers. $E_{tot,A}$ and $E_{tot,NA}$ stand for the total error rates of the true adaptive and simulated nonadaptive data collections, respectively. $E_{final,A}$ and $E_{final,NA}$ are the respective error percentages for the last 200 characters, INP_A is the increase in the number of prototypes in percentages, and *color* is the color used for the writer in all the figures. IA_A and ILR_A are the initial error percentage and initial learning rate (change in the error rate per training sample) of the true data collection. They both are approximated by fitting a second order polynomial to the cumulative sum of the errors for the first 200 characters. IA_A and ILR_A are the first and second derivatives of the fitted polynomial at the first character, respectively.

Following facts can be seen from Table 6.12: 1) A low final error rate for the adaptive experiment does not necessarily follow from a relatively good total error rate for the nonadaptive experiment. The initial accuracy of the recognizer is determined by the initial writer-independent prototype set. It is high for those writers whose writing style is covered by the initial prototype set. On the other hand, the benefits of the adaptation, and thus the final accuracy, depend on the consistency and nature of the writing style. 2) The initial rate of learning is high if the initial accuracy is low. 3) The increase in the size of the prototype set is more prominent if the final error rate is high.

Table 6.12: *Results for the true collection of Database 4, in which adaptation strategy Add(4) has been applied, and for the nonadaptive simulation.*

Writer	*color*	$E_{tot,A}$	$E_{tot,NA}$	$E_{final,A}$	$E_{final,NA}$	IA_A	ILR_A	INP_A
DB4:1	black	9.51	19.92	8.00	21.50	21.60	-0.08	98
DB4:2	red&yell.	4.61	15.59	2.50	18.00	18.91	-0.08	61
DB4:3	mag.&c.	6.94	21.11	6.00	18.50	21.26	-0.09	84
DB4:4	green	6.31	37.27	2.00	36.50	49.33	-0.28	84
DB4:5	blue	6.55	18.87	2.00	20.50	18.89	-0.05	74
DB4:6	cyan	4.87	30.62	1.00	31.50	38.54	-0.26	77
DB4:7	magenta	11.25	20.14	10.00	17.50	16.01	-0.05	106
DB4:8	red	3.25	22.27	0.50	24.50	26.12	-0.14	61
avg	-	6.66	23.19	4.00	23.56	26.42	-0.13	81

6.3.2 User adaptation

The error rates in the simulated nonadaptive data collections do not decrease monotonically during the collection. Evidently, the writers cannot change their writing

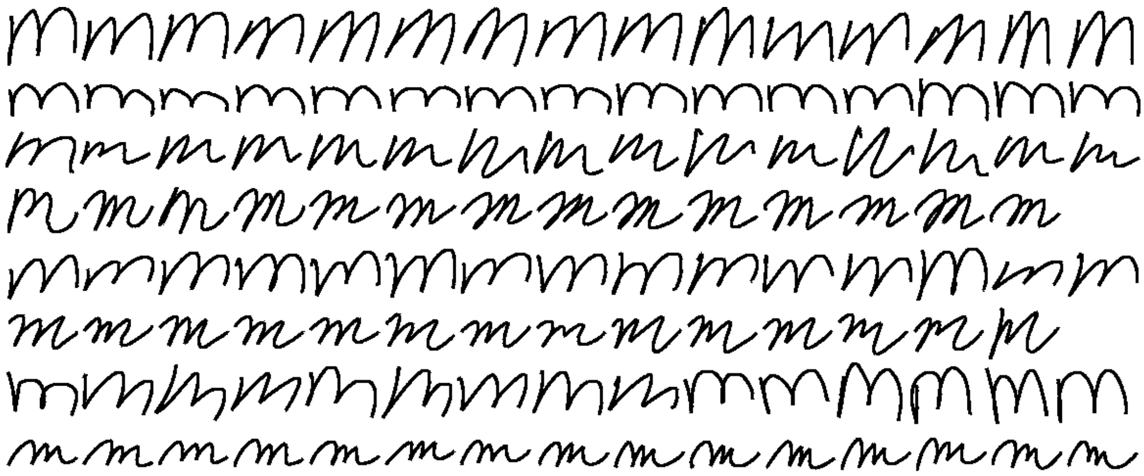


Figure 6.8: *All 'm'-characters of Database 4. Every line contains the samples by one writer, in the order DB4:1, ... ,DB4:8. The characters are in the same order as they were written.*

styles so that the characters would more closely remind the initial writer-independent prototypes and thus be better recognized. On the contrary, the error rate increases. As it can be seen from Table 6.12, the error percentage for the last 200 characters is on average slightly higher than the total error rate for all characters. This can be caused by the fact that when the user notices that the system can learn to recognize even quite sloppily written characters, he does not anymore try to write the characters as carefully as he did in the beginning.

To illustrate this phenomenon, all the 'm'-characters excluding those which were reported to be writer mistakes are shown in Figure 6.8. They are in the same order as they were written. Characters which are at same row are written by same person. It can be seen that only writer DB4:2 (second row) and writer DB4:8 (bottom most row) have maintained their style of writing character 'm' unchanged during the whole test. In the case of writer DB4:3 (third row), the change in the writing style is the most evident: the last characters are clearly less carefully written than the first ones. This also applies to writers DB4:5 (fifth row) and DB4:6 (sixth row). Similar changes in the writing styles can also be observed by examining samples of the other character classes.

Figure 6.9 illustrates how the samples reported as writer mistakes are distributed in time. The average number of writer mistakes for all the eight writers is approximately ten, which roughly corresponds to one percent of all samples. The dotted line represents the even distribution and the solid plot is the average of the cumulative sums of the writer mistakes. Clearly, the distribution is not even: the writers seem to have made errors more often in the middle of the collection than in the beginning or end.

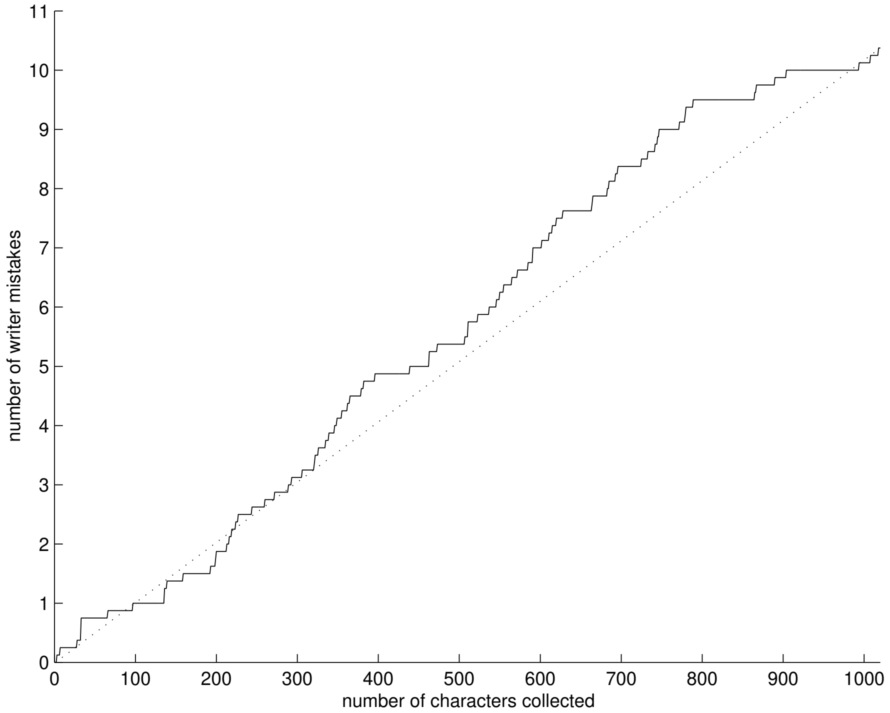


Figure 6.9: *Average cumulation of the writer mistakes in the collection of Database 4. The dotted line represents the even distribution.*

6.3.3 Typical errors

In the next analysis, the characters were divided into three groups: lower and upper case letters and digits. The error rates shown in Table 6.13 are denoted with superscript d, u, and l for the digits, upper and lower case letters, respectively. Superscript *all* is used for the error rates which are calculated by combining the recognition results for the three character groups. Abbreviations *TDC* and *SDC* stand for the true and simulated collections of Database 4. Subscript *tot* again means that the error rate is calculated over the whole test. Superscripts $1°$, $2°$, $3°$, and $4°$ correspond to different types of errors, namely: $1°$) a confusion between the lower and upper case versions of same letter, or $2°$) between a number and letter, or $3°$) between the lower and upper case versions of different letters, and $4°$) all other mistakes.

From Table 6.13 it can be seen that, as a group, digits are best and lower case letters worst recognized. In the light of these results, it was a good choice to use lower case letters instead of upper case letters in the previous experiments as they impose a more challenging problem. The error probability for digits is naturally the lowest as the group contains least classes. The relative proportions of misrecognitions in the three character groups are roughly same for the adaptive and nonadaptive experiments. Therefore, the initial prototypes of the three character groups can be considered to be equally representative.

Table 6.13: *Percents of different error types occurred during the true collection of Database 4 in which adaptation strategy Add(4) has been applied (TDC) and during the nonadaptive simulation (SDC).*

Test	E_{tot}^{all}	E_{tot}^{d}	E_{tot}^{l}	E_{tot}^{u}	$E_{tot}^{1°}$	$E_{tot}^{2°}$	$E_{tot}^{3°}$	$E_{tot}^{4°}$
TDC	6.66	4.10	7.74	6.46	2.97	1.28	0.42	2.00
SDC	23.19	12.80	25.06	24.93	7.86	4.23	2.75	8.34

In both the adaptive and nonadaptive experiments, the 1°-type error is the most probable of the errors in which two character groups are confused. This was not an unexpected results as many of the subjects used very similar shapes but different sizes for the lower and upper case versions of a letter and the size information is lost due to *MinMaxScaling*-normalization. The following character pairs were very prone to 1°-type error in the adaptive experiments: 's' and 'S' (44 cases), 'z' and 'Z' (42), 'c' and 'C' (40), and 'x' and 'X' (36). The most typical examples of the 2°-type error were: '1' and 'I' (11), '9' and 'g' (11), '2' and 'Z' (9), '1' and 'l' (9), and '0' and 'O' (8). The 3°- and 4°-type errors were more evenly distributed over the classes and thus no typical errors can be clearly pointed out. The character pairs mentioned above for the adaptive experiments were also among the most troublesome in the nonadaptive experiments. The classification method and the adaptation strategy do not seem to be sufficient for separating these pairs – there is no sense in having and no gain in adding almost identical prototypes for two different classes. Consequently, the problem of confusing character groups should be solved by some other means. One such approach will be described in the following section.

6.4 Separate prototype sets for character groups

The easiest and a commonly-used way to avoid confusions between character groups is to demand that the user specifies the group in which the input character belongs. In such systems, the prototype sets of the groups can be kept separated. For example, specific input areas can be used for each character group. It is less convenient for the user but can be, at least partially, justified by the reduced error probability. In the next two tables, the results for simulated adaptive ($Add(4)$) and nonadaptive collections of Database 4 are shown. It is now assumed that the group of the character is known prior to its recognition. All the notations have similar meanings as in the previous tables. Also in this case, subscript *final* means that the error percentage is calculated for the last 200 samples of each writer.

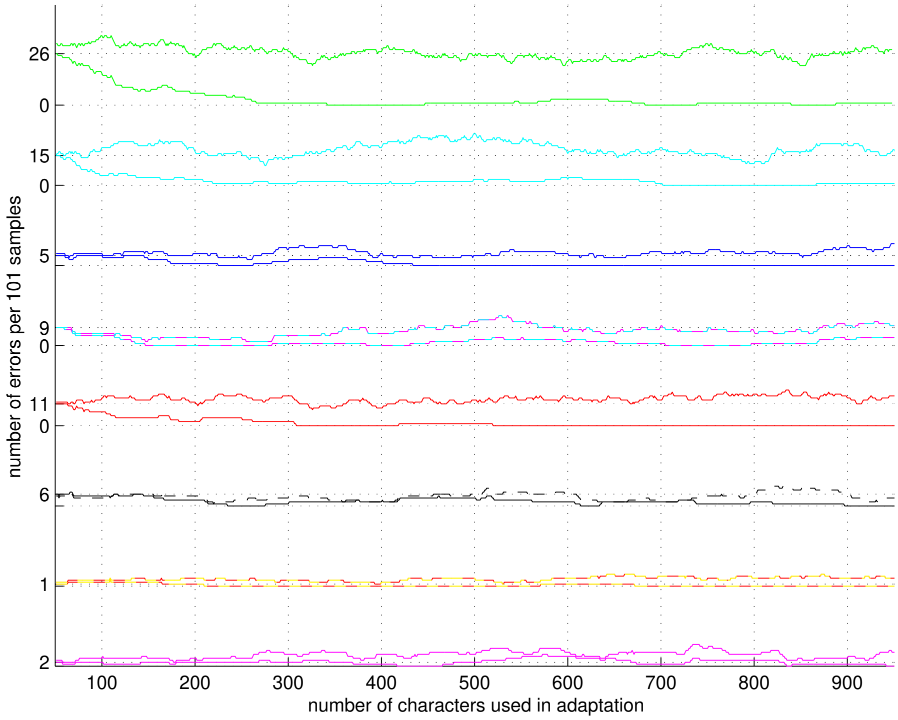


Figure 6.10: *Evolution of the error rate in the simulated collections of Database 4 in which separate prototype sets have been used for the three character groups. The lower plots are obtained with adaptation strategy Add(4) and the upper plots with a nonadaptive recognizer. A plot corresponds to a single writer. The shift between the plots is 20 units.*

The figures in Table 6.14 are the combined results of the three character groups and they can be compared with the figures in Table 6.12. It is evident that the use of separate prototype sets for the character groups is beneficial: the error rates for the majority of the writers are at least halved. In addition, the final numbers of prototypes are significantly lower. Detailed results for the character groups are presented in Table 6.15. The evolution of the error rates is illustrated in Figure 6.10. The lower plots correspond to the adaptive tests. In the case of the writer plotted with black, dashed line is used for the nonadaptive test for the sake of clarity. It can be seen by comparing Figures 6.7 and 6.10 that the initial accuracies clearly improve but the rates of learning remain quite similar when separate prototype sets are used for the character groups.

Table 6.14: *Simulated collection of Database 4 in which adaptation strategy Add(4) and multiple prototype sets have been used. The results have been calculated by combining the results for separate character groups.*

Writer	$E^{all}_{tot,A}$	$E^{all}_{tot,NA}$	$E^{all}_{final,A}$	$E^{all}_{final,NA}$	INP^{all}_{A}
DB4:1	2.38	4.76	0.50	6.00	51
DB4:2	0.39	3.33	0.00	4.00	26
DB4:3	2.08	7.23	2.50	7.50	51
DB4:4	4.31	26.35	0.50	24.00	66
DB4:5	1.19	6.36	0.00	7.50	36
DB4:6	2.98	17.89	0.50	16.50	59
DB4:7	2.07	5.73	1.00	5.50	49
DB4:8	1.87	13.30	0.00	14.50	48
avg	2.16	9.83	0.63	10.69	48

Table 6.15: *Simulated collections of Database 4 in which adaptation strategy Add(4) and multiple prototype sets have been used. Separate results for the three character groups are shown.*

Writer	$E^{d}_{tot,A}$	$E^{l}_{tot,A}$	$E^{u}_{tot,A}$	$E^{d}_{tot,NA}$	$E^{l}_{tot,NA}$	$E^{u}_{tot,NA}$	INP^{d}_{A}	INP^{l}_{A}	INP^{u}_{A}
DB4:1	0.00	2.32	3.27	0.67	5.10	5.84	17	61	53
DB4:2	0.00	0.46	0.46	0.00	3.91	3.91	13	31	27
DB4:3	0.00	3.96	0.93	0.00	8.62	8.37	19	68	45
DB4:4	0.00	6.37	3.53	1.35	41.04	20.24	19	81	66
DB4:5	0.68	1.62	0.93	6.08	3.02	9.81	21	39	38
DB4:6	1.33	4.86	1.65	1.33	25.69	15.80	17	80	52
DB4:7	1.35	2.08	2.31	11.49	4.86	4.62	30	52	53
DB4:8	0.00	1.84	2.55	0.00	18.43	12.76	11	49	53
avg	0.42	2.93	1.95	2.60	13.78	10.13	18	56	48

Chapter 7

Conclusions

7.1 Comments on the literature survey

In literature, handwritten character recognition problem has been approached in various ways. Several methods originally developed and used for other classification tasks, especially speech recognition, have been applied. In this work, the methods have been divided into groups of statistical, model matching, structural and syntactical, and neural network approaches. The most popular, recent, and unique methods suitable for on-line recognition and adaptation have been more closely described in the examples. Two state-of-art methods, namely Hidden Markov Models and elastic matching, along with neural network approaches seem to be at the center of attention of current research. However, hardly anything can be said about the methods used in the most recent commercial application as there are only few and very cursory publications about them. In addition, in only a few articles the actual use of the recognition system and users' responses are addressed.

The selection between the various methods which all have been reported to be very promising is difficult: they have been tested with different data, characters sets, and constraints on the writing, and some of the results are writer-dependent and the others writer-independent. All these factors make the comparison between the results presented in this work and in other research works quite unreasonable. However, if all these differences are ignored, the results of the nonadaptive experiments carried out in this work seem to be somewhat worse than those reported in the literature for basically similar applications of elastic matching. On the other hand, after adaptation, a recognition accuracy high enough to be acceptable for real-world applications has been attained for most of the writers.

7.2 Benefits of the implicit dynamic information

In this work, a classifier based on elastic matching has been created. First, a writer-dependent classifier was constructed by choosing the combination of preprocessing and normalization methods and dissimilarity measure most suitable for a set of writers. The major conclusion made in this phase of the work was that the implicit dynamic information which is contained in data points sampled with a constant frequency is beneficial. The recognition accuracy improves if not only the shapes of the characters but also the implicit dynamic features, such as the velocity and acceleration, of their generation are considered in the dissimilarity measure.

This is an interesting result as explicit dynamic features have mainly been used in signature verification problems. In the case of handwriting recognition, dynamic information is usually used only for segmentation and data points are preprocessed so that they are evenly spaced in the spatial domain. In the experiments, the dissimilarity measure and centering normalization most sensitive to the data point distribution, namely point-to-point distance and *MassCenter*-normalization, were found to be better than the dissimilarity measures and centering normalization insensitive to the point density. These results speak for the benefits of the implicit dynamic information.

7.3 Performances of the adaptation strategies

The writer-independent classifier can be changed into a writer-dependent one by using the adaptation strategies defined in Chapter 4. Adaptation strategy *Add* adds input characters into the prototype set and the recognition accuracy increases considerably fast. For digits and lower case letters, the improvement of the recognition accuracy is evident after 50 input characters and the error probability reaches its minimum during the first 200 input characters. The increase in the size of the prototype set is considerable, especially at the beginning of the experiment. The increase is significant even at the end of the test even though the recognition accuracy is not improving anymore.

Adaptation strategy *Lvq* does not change the size of the prototype set as it reshapes the initial prototypes according to the writer-dependent input characters. The reshaping is performed according to a modified LVQ training rule. *Lvq*-strategy does improve the recognition results but not enough when used alone. The beneficial features of adaptation strategies *Add* and *Lvq* are combined in *Hybrid*-strategy: a recognition accuracy nearly as high as with *Add*-strategy can be reached and the increase in the size of the prototype set is remarkably smaller.

In addition, adaptation strategy *Inactivate* was used for reducing errors caused by confusing prototypes. It worked well with *Add*- but not with *Hybrid*-strategy. In both cases, it was actually quite rarely used as all input characters were correctly labeled and all bad samples had been cleaned out from the databases by hand.

In the case of digits and lower case letters of Database 4, adaptation strategy *Add* was able to drop the error rate from 14% to 1.8%. In the case of all the characters of Database 4, the error rate dropped from 23% to 4.0% due to the adaptation. When separate prototype sets were used for the digits and lower and upper case letters of Database 4, the final error rates of the nonadaptive and adaptive classifier were 9.8% and 0.63%, respectively.

7.4 User adaptation

An important phenomenon which can be observed in the true on-line test of adaptation strategy *Add* is the user's adaptation to the performance of the recognition system. Instead of trying to improve the recognition results by himself by writing more carefully and consistently, the user tends to demand more and more from the system as it learns and the writing style becomes more careless. In this case, the users probably had no other motivation than to get off the tiresome test as soon as possible. User adaptation might be quite different in a more realistic experiment in which the user has to perform some real tasks, for example, write a letter, and the time consumed in the test directly depends on the recognition accuracy.

7.5 Ideas for future work

During this work, some new problems and research ideas have arisen. For example, the clustering algorithm which has been used for selecting prototypes leaves certainly a lot to be desired. When the clusters are manually examined, it can be seen that the algorithm does not always succeed in separating two clearly different writing styles even if the number of the clusters is increased. The performance of the writer independent classifier depends crucially on the prototype set and a better recognition accuracy could thus be achieved by developing a better prototype selection method. The current clustering algorithm is only semiautomatic and requires some *a priori* knowledge on the existing writing styles. Perhaps, a two dimensional Self-Organizing Map (SOM) (Kohonen 1997) could be used instead. A SOM is able to order its units so that similar items are mapped close to each other. The writing styles would appear as clusters if all the characters were placed on such a map. The prototypes could then be manually selected from the centers of the clusters. If the prototype selection is carried out in this way, *a priori* knowledge is established automatically and a representative prototype per writing style can be found.

The opinions of the subjects about the data collection system varied a lot and quite independently of the recognition performance. The writers having a high initial accuracy seemed to be more annoyed at the recognition mistakes made in the end of the data collection than the writers with a low initial accuracy. Most of the writers did not know the basic mechanisms the system was based on and thus did not always see any reason for the mistakes. Therefore, some of them were really annoyed by the fact that the system was able to learn some character classes very well but had serious problems with classes that seemed easier from the writer's point of view. It will be very interesting to test the recognition method and hear the user's opinions of it with some portable device and in more realistic tasks. Before that, the supervision method of the learning process, the user interface, all error situations, and the system's sensitivity to bad learning samples must be considered more deeply.

References

Arakawa, H. (1983). On-line recognition of handwritten characters – alphanumerics, hiragana, katakana, kanji. *Pattern Recognition 16*(1), 9–16.

Bellegarda, E. J., J. R. Bellegarda, D. Nahamoo, and K. S. Nathan (1994). A fast statistical mixture algorithm for on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 16*(12), 1227–1233.

Bellegarda, E. J., J. R. Bellegarda, D. Nahamoo, and K. S. Nathan (1995). A discrete parameter HMM approach to on-line handwriting recognition. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pp. 2631–2622. IEEE.

Brault, J.-J. and R. Plamondon (1993). Segmenting handwritten signatures at their perceptually important points. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*(9), 953–957.

Chan, K.-F. and D.-Y. Yeung (1998, August). A simple yet robust structural approach for recognizing on-line handwritten alphanumerical characters. In *Proceedings of the Sixth International Workshop on Frontiers in Handwriting Recognition*, pp. 229–238.

Comerford, R. (1998, May). Pocket computers ignite OS battle. *IEEE Spectrum*, 43–48.

Cover, T. M. and P. E. Hart (1967, January). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory 13*(1), 21–27.

Duda, R. O. and P. E. Hart (1973). *Pattern Classification and Scene Analysis.* John Wiley & Sons.

Fix, E. and J. L. Hodges (1951). Discriminatory analysis—nonparametric discrimination: Consistency properties. Technical Report Number 4, Project Number 21-49-004, USAF School of Aviation Medicine, Randolph Field, Texas.

Frankish, C., R. Hull, and P. Morgan (1995). Recognition accuracy and user acceptance of pen interfaces. In *Proceedings ACM CHI'95 Conference on Human Factors in Computing Systems.*

Freeman, H. and L. S. Davis (1977). A corner-finding algorithm for chain-coded curves. *IEEE Transactions on Computers*, 287–303.

Fujisaki, T., T. E. Chefalas, J. Kim, C. C. Tappert, and C. G. Wolf (1991). On-line run-on character recognizer: Design and performance. *International Journal of Pattern Recognition and Artificial Intelligence 5*(1&2), 123–137.

Gardner, W. D. (1997). Putting pen to smart paper. *IBM Research magazine* (3).

Govindan, V. K. (1990). Character recognition – a review. *Pattern Recognition 23*(7), 671–683.

Guyon, I., L. Schomaker, R. Plamondon, M. Liberman, and S. Janet (1994). Unipen project of on-line data exchange and recognizer benchmark. In *Proceedings of International Conference on Pattern Recognition*, pp. 29–33.

Guyon, I. and P. S. P. Wang (Eds.) (1993, August). *International Journal of Pattern Recognition and Artificial Intelligence, Special issue: Advances in pattern recognition systems using neural networks*, Volume 7.

Guyon, I. and C. Warwick (1996). Joint EC-US survey of the state-of-the-art in human language technology. http://www.cse.ogi.edu/CSLU/HLTsurvey.htm.

Hart, P. E. (1968, May). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory 14*(3), 515–516.

Hu, J., M. K. Brown, and W. Turin (1996). HMM based on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*(10), 1039–1045.

Hu, J., S. G. Lim, and M. K. Brown (1998, August). HMM based writer independent on-line handwritten character and word recognition. In *Proceedings of the Sixth International Workshop on Frontiers in Handwriting Recognition*, pp. 143–155.

Hullender, G. (1998, August). A discussion with Greg Hullender from Microsoft at the Sixth International Workshop on Frontiers in Handwriting Recognition.

Kassel, R. H. (1995, June). *A comparison of approaches to on-line handwritten character recognition*. Ph. D. thesis, Massachusetts Institute of Technology.

Kerrick, D. D. and A. C. Bovik (1988). Microprocessor-based recognition of handprinted characters from a tablet input. *Pattern Recognition 21*(5), 525–537.

Kohonen, T. (1997). *Self-Organizing Maps*, Volume 30 of *Springer Series in Information Sciences*. Springer-Verlag. Second Extended Edition.

Kuklinski, T. T. (1984). Components of handprint style variability. In *Proceedings of the 7th International Conference on Pattern Recognition*, pp. 924–926.

Laaksonen, J., J. Hurri, E. Oja, and J. Kangas (1998b). Comparison of adaptive strategies for on-line character recognition. In *Proceedings of International Conference on Artificial Neural Networks*.

Laaksonen, J., J. Hurri, E. Oja, and J. Kangas (1998a, August). Experiments with a self-supervised adaptive classification strategy in on-line recognition of isolated handwritten latin characters. In *Proceedings of Sixth International Workshop on Frontiers in Handwriting Recognition*, pp. 475–484.

Li, X. and D.-Y. Yeung (1997). On-line handwritten alphanumeric character recognition using dominant points in strokes. *Pattern Recognition 30*(1), 31–44.

Loy, W. and L. Landay (1982, July). An on-line procedure for recognition of handprinted alphanumeric characters. *Pattern Recognition 4*(4), 422–427.

Lu, P.-Y. and R. W. Brodersen (1984). Real-time on-line symbol recognition using a DTW processor. In *Proceedings of International Conference on Pattern Recognition*, Volume 2, pp. 1281–1283.

Martens, R. and L. Claesen (1997). Dynamic programming optimization for on-line signature verification. In *Proceeding of International Conference on Document Analysis and Recognition*, pp. 653–656. IEEE.

Mel, M. W., S. M. Omohundro, A. D. Robinson, S. S. Skiena, K. H. Thearling, L. T. Young, and S. Wolfram (1988, June). Tablet: personal computer in the year 2000. *Communications of the ACM 31*(6), 639–648.

Mozayyani, N., A. Baig, and G. Vaucher (1998). A fully-neural solution for on-line handwritten character recognition. In *Proceedings of International Joint Conference on Neural Networks*, Volume 2, pp. 160–164.

Nathan, K., J. R. Bellegarda, D. Nahamoo, and E. J. Bellegarda (1993). On-line handwriting recognition using continuous parameter hidden markov models. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, Volume 5, pp. 121–124.

Neisser, U. and P. Weene (1960). A note on human recognition of hand-printed characters. *Information and Control* (3), 191–196.

Nouboud, F. and R. Plamondon (1990). On-line recognition of handprinted characters: survey and beta tests. *Pattern Recognition 23*(9), 1031–1044.

Nouboud, F. and R. Plamondon (1991). A structural approach to on-line character recognition: System design and applications. *International Journal of Pattern Recognition and Artificial Intelligence 5*(1&2), 311–335.

Odaka, K., H. Arakawa, and I. Masuda (1982). On-line recognition of handwritten characters by approximating each stroke with several points. *IEEE Transactions on Systems, Man and Cybernetics 12*(6), 898–903.

Parizeau, M. and R. Plamondon (1994). Machine vs humans in a cursive script reading experiment without linguistic knowledge. In *Proceedings of International Conference on Pattern Recognition*, Volume 2, pp. 93–98.

Pavlidis, I., R. Singh, and N. P. Papanikolopoulos (1997). An on-line handwritten note recognition method using shape metamorphosis. In *Proceedings of International Conference on Document Analysis and Recognition*, Volume 2, pp. 914–918. IEEE.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pp. 267–295.

Råde, L. and B. Westergren (1990). *BETA Mathematics Handbook for Science and Engineering, 2nd edition*. Chartwell-Bratt Ltd.

Rigoll, G., A. Kosmala, J. Rottland, and C. Neukirchen (1996). A comparison between continuous and discrete density hidden markov models for cursive handwriting recognition. In *Proceedings of International Conference on Pattern Recognition*, Volume 2, pp. 205–209. IEEE.

Sankoff, D. and J. B. Kruskal (1983). *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison.* Addison-Wesley.

Sato, Y. and K. Kogure (1982). Online signature verification based on shape, motion, and writing pressure. In *Proceedings of International Conference on Pattern Recognition*, Volume 2, pp. 823–826. IEEE.

Scattolin, P. (1995, November). Recognition of handwritten numerals using elastic matching. Master's thesis, Concordia University, Canada.

Schalkoff, R. (1992). *Pattern recognition: statistical, structural and neural approaches.* John Wiley & Sons, Inc.

Schomaker, L. (1998, June). From handwriting analysis to pen-computer applications. *Electrics & Communication Engineering Journal*, 93–101.

Schomaker, L., G. Abbink, and S. Selen (1994). Writer and writing-style classification in the recognition of online handwriting. In *IEE Colloquium (Digest) Proceedings of the European Workshop on Handwriting Analysis and Recognition: European Perspective.*

Sederberg, T. and E. Greenwood (1992). A physically based approach to 2-D shape blending. *Computer Graphics 26*(2), 25–34.

Skalak, D. B. (1997, May). *Prototype selection for composite nearest neighbor classifier.* Ph. D. thesis, University of Massachusetts Amherst.

Srihari, S. N. (1985). *Computer text recognition and error correction*, Chapter Introduction, pp. 1–4. IEEE Computer Society Press.

Starner, T., R. S. J. Makhoul, and G. Chou (1994). On-line cursive handwriting recognition using speech recognition methods. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, Volume 5, pp. 125–128. IEEE.

Suen, C. Y. (1979, November). A study on man-machine interaction problems in character recognition. *IEEE Transactions on Systems, Man, and Cybernetics 9*(11), 732–737.

Tappert, C., C. Y. Suen, and T. Wakahara (1990). The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*(8), 787–808.

Tappert, C. C. (1984). Adaptive on-line handwriting recognition. In *Proceedings of International Conference on Pattern Recognition*, pp. 1004–1007. IEEE.

Tappert, C. C. (1991). Speed, accuracy, and flexibility trade-offs in on-line character recognition. *International Journal of Pattern Recognition and Artificial Intelligence 5*(1&2), 79–95.

Veltman, S. R. and R. Prasad (1994). Hidden markov models applied to on-line handwritten isolated character recognition. *IEEE Transactions on Image Processing 3*(3), 314–318.

Wakahara, T. and K. Odaka (1997, December). On-line cursive kanji character recognition using stroke-based affine transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*(12), 1381–1385.

Wang, P. S.-P. and A. Gupta (1991). An improved structural approach for automated recognition of handprinted character. *International Journal of Pattern Recognition and Artificial Intelligence 5*(1&2), 97–121.

Ward, J. R. and B. Blesser (1985, September). Interactive recognition of handprinted characters for computer input. *IEEE Computer Graphics and Applications 9*(5), 24–37.

Ward, J. R. and T. Kuklinski (1988, May/June). A model for variability effects in handprinting with implications for design of handwriting character recognition systems. *IEEE Transactions on Systems, Man, and Cybernetics 18*(3), 438–451.

Wilfong, G., F. Sinden, and L. Ruedisueli (1996). On-line recognition of handwritten symbols. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*(9), 935–940.

Wirtz, B. (1998, March). Method for the dynamic verification of an autograph character string on the basis of a reference autograph character string. United States Patent, patent number 5,730,468.

Yuen, H. (1996). A chain coding approach for real-time recognition of on-line handwritten characters. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, Volume 6, pp. 3426–3429. IEEE.